Modeling the dynamics of caching in content-based publish/subscribe systems

Vasilis Sourlas*, Georgios S. Paschos*, Petteri Mannersalo†, Paris Flegkas* and Leandros Tassiulas*

*Department of Computer & Communication Engineering University of Thessaly, Greece,

†VTT Technical Research Center of Finland

Email: (vsourlas, gpasxos, pflegkas, leandros) @uth.gr, petteri.manersalo@vtt.fi

Abstract

We study cache dimensioning in the context of publish/subscribe (pub/sub) systems. The clients are assumed to join and leave the network and request previously published information. Each broker is equipped with a limited capacity cache and decides upon a policy for caching, dropping or prioritizing messages. We are interested to model such a dynamic system in order to quantify the time that a message survives in the system, the *survival time*. We begin with the case where replication of messages is not allowed. In this case, we yield an exact solution based on an absorbing Markov chain and by utilizing this we are able to determine numerically the distribution of the message survival time. Also, we provide an approximation which reduces the state space significantly while having a minimal loss in accuracy. Next, we deal with the general problem of allowing replicas of messages and thus considering copy events as time evolves. We propose a heuristic approximation for estimating the mean rate of copies

and we evaluate it using a discrete event simulator. We show that for a wide set of parameters, the approximation can provide a good solution for dimensioning the caches in the content-based pub/sub systems.

I. INTRODUCTION

The pub/sub paradigm is an important network architecture that makes communications scalable and content driven. Autonomous components (clients) interact with publication events (messages) by subscribing to classes of events they are interested in. The mediation routers (brokers) are responsible for collecting subscriptions and forwarding the messages to clients. In pub/sub systems the forwarding of a message is determined entirely by the client, using expressions (filters) that allow sophisticated matching on the event content. Multicasting notification events or even large files is significantly facilitated by this process.

In a pub/sub system, any message is guaranteed to reach all interested destinations, (completeness property) [1]. The above holds for all clients that are active and therefore their subscriptions are known to the system at publish time. However, there are cases where clients join and leave the system (e.g. mobile environment), and it is then possible that a client joins the network after the publication instance of an interesting message. In pub/sub systems it is not possible for a new subscriber to retrieve previously published messages that match his/her subscription. Therefore, enabling caching for retrieval of past information is one of the most challenging problems in pub/sub networks.

The authors in [3] propose a caching mechanism for wireless ad-hoc networks based on buffers that offers a way to integrate data repositories all over the network. Their approach operates within the class of applications that initiate normal operation after having monitored a sequence of events. In [4] and [5] the authors propose a historic data retrieval pub/sub system

where databases are connected to various brokers, each associated with a filter to store particular information. The work there is based on predefined caching points in the network. A different and more general approach is attempted in [2] where the messages are stored opportunistically. Each broker of the network is a potential caching point for each published message. The latter approach, which is more general, is analyzed by means of measurements and simulations. In [2] a different aspect of caching is proposed; preserving the information over time instead of making information available in nearer space as in traditional caching schemes found in literature.

In all of the above approaches as well as in this paper, it is assumed that the brokers, being part of the core network, are endorsed with the task of storing information. The strategical goal is to find a policy that can coexist with the pub/sub principles while in the same time provides a sort of differentiation among popular and unpopular files, that is the available caching space is efficiently used in order to maximize the time that popular messages survive in the system. To achieve this goal, we use opportunistic caching (as in [2]) and we study the interrelation between the client dynamics and the dimensioning of the broker caches.

To our knowledge, there exists no prior work involving analytical models for dynamic pub/sub networks. We assume that clients arrive to the pub/sub network according to a Poisson process, stay in the system for an exponentially distributed time duration and are interested in receiving any message (old or new) matching their subscription. Messages arrive according to a Poisson process as well, and the brokers must decide whether to cache them or not. When the caches are full, caching a message means that another message must be dropped. We propose several policies, for caching, prioritization and dropping of messages.

In order to study the effect of several policies, we investigate an exact solution based on absorbing Markov processes as well as a lightweight approximation with obscured information. We

validate our models using discrete-event simulations and show that for a large set of parameters, the approximation is a useful tool for dimensioning the pub/sub caching problem. Finally, we compare the proposed policies and find out that a simple prioritization mechanism can produce the desired differentiation while the system remains agnostic to the popularity of each particular message. The policy resembles the Least Recently Used (LRU) policy but it is reformed to fit the pub/sub architecture.

II. The pub/sub system with caches

A. The pub/sub legacy architecture

We consider a pub/sub system that uses the subscription forwarding routing strategy [6]. In such a network architecture, the brokers form an overlay tree similar to the one in Figure 1. The clients instead (not visible in the same Figure) select one broker and connect to it (we call it serving broker).

At subscription time a client may send a `Subscribe()` message containing the corresponding subscription filter to the serving broker. The filter is a binary function which applied on a message yields 1 if the client is interested in the message and 0 otherwise. The serving broker, inserts the filter in a Subscription Table (ST) together with the identifier of the subscriber. Also, it propagates the subscription to all of its neighboring brokers, which in turn store the filter using this time the identifier of the serving broker. Finally, the forwarding procedure goes on until the filter is inserted into all tables of the network with each entry indicating a neighboring broker in a path towards the serving broker. This scheme is usually optimized by avoiding subscription forwarding of the same event pattern in the same direction exploiting *coverage* relations among filters.

Requests to unsubscribe from an event pattern are handled and propagated analogously to subscriptions, although at each hop, entries in the subscription table are removed rather than inserted. In this paper we assume that `Subscribe()` and `Unsubscribe()` messages are propagated instantaneously through the whole network.

At publication time a client may send a `Publish()` message to the serving broker which in turn, for any entry in the ST matching the message, forwards the message to the corresponding client or broker. Following, all brokers repeat the same procedure until the message is forwarded to all intended clients. For an in-depth discussion of the pub/sub architecture see [7].

B. Enabling caching in pub/sub systems

In this section, we give a short description of the caching scheme which is similar to [2]. Each broker is equipped with a cache of size K. By assuming equal message sizes, it is equivalent to consider that each cache can fit K messages. In general it is possible to consider a push/pull approach for caching. That is, at publication time, we select a number of brokers opportunistically and we explicitly push the message in their caches. When a client is interested in a message, he/she should request (pull) the message from the cache. As discussed above, however, pub/sub systems have a particular way of solving the routing problem, which is very efficient in cases of multicasting content. In the spirit of retaining this edge in performance, push/pull mechanism should be avoided. Thus, in this work, we are interested to study caching techniques that do not require extra communications other than what a native pub/sub system would use.

We introduce the following caching mechanism;

1) Whenever a message traverses a broker, the broker opts for caching the message based on the *caching* policy.

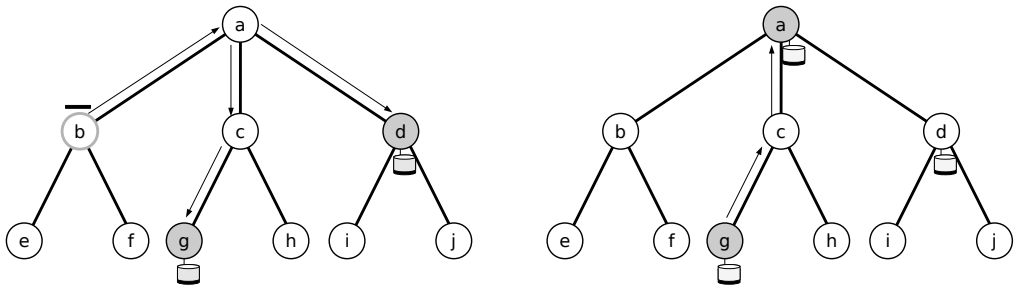

Fig. 1. Caching and retrieving old information. Grey brokers have interested clients. We describe the caching of a message in broker g and d (left) and the retrieval and caching (right) from g to a.

2) When a client interested in old content appears in the network, apart from subscribing, he/she issues a request by sending a `Request()` message. This message is propagated as if it was a publish message, i.e. it follows the subscription tree[1].
3) A broker upon receiving the `Request()` message checks in the ST for subscriptions matching the filter carried by the request message. For every matched broker subscription, the broker forwards the `Request()` message. If there is at least one matching client subscription, the broker searches in its cache for messages matching the initial filter.
4) If a matching message is found, a `Response()` message is propagated backwards the request path.

For a motivating example, consider the policy "cache in a broker with interested clients"; those brokers are depicted as gray in Figure 1. The publication tree is designated with arrows. Broker b publishes a message which travels through the publication tree and reaches brokers g and d which at the publication instance have client subscriptions in their tables that match the

[1]Here we assume that the request has the same form with a message–it can be evaluated by the filter functions. Equivalently we can think of messages and filters as sets where naturally the size of messages, filters, requests would be in descending order.

published message. According to the aforementioned policy these two brokers decide to cache the message, while brokers b, a and c decide not to cache the message. Later, a client interested in this content (both old and new) enters the network through broker a (thus a is also gray now). Meanwhile, the clients interested in this content previously attached to broker d have now left the network and thus broker d is white again. Using the caching mechanism, the new client at broker a, issues a `Request()` message which is forwarded to broker a, c and g (since these three brokers are part of the publication tree that corresponds to gray content). Brokers a and g, having client subscriptions, search their caches for a match. Broker g, finds the black message and issues a `Response()` message. Finally, the interested client receives the cached message.

The above example showcases how it is possible to avoid exhaustive search techniques for caching and fetching messages in pub/sub systems. There is a certain risk involved, however. If for example clients from broker g have left the network as well, the message cached in g and d would not be discovered. On the other hand, this mechanism provides a tool for efficient cache handling. For example, the message cached in broker d could have been erased from cache providing more space since it is not possible to be found. Alternatively it is possible to prioritize messages that are reachable against those that are not reachable. All these options are explored via several introduced policies.

1) Caching policies: At publication time of a message, a *caching policy* selects a number of brokers and assigns them as the caching points. We may restrict our interest to those brokers having clients interested to the particular message. We will therefore investigate the following caching policies

- Save in all interested brokers (same as in [2])–*selective caching* (sel).
- Save in all brokers–*flood caching* (fld).

- Save in a single interested broker (using for example a random distributed algorithm in order to select the broker along the publication tree)–*single caching* (sin).

Note, that when the flood caching is activated, the message must be forwarded to the whole network, thus extending the number of transmissions from those belonging to the publication tree to the whole network. This extra overhead might not be acceptable for the pub/sub system. Therefore this policy is advocated only as a performance reference for the other policies.

2) Request policies: The *request policy* dictates how the request message is propagated in the network. In [2], the request message is propagated along the subscription tree created by entries in the subscription table that match the request. This is a meaningful procedure that retains the principles of pub/sub concept. On the other hand, it is possible to flood the network with requests in order to make sure that any message cached is retrieved at the cost of higher overhead.

- Request based on subscription–*subscription-based selective request policy* (sub).
- Request propagated to the whole network–*flooding request policy* (fld).

3) Priority policies: The *priority policy* is a way to differentiate the messages in a cache based on their usability as well as provide better efficiency for the performance of the whole system. In order to introduce priority we propose a policy with regenerations and degradations. In particular, the client activity is monitored at each broker. Upon the arrival of a client whose subscription matches a given message, a message regeneration is activated; the message is transferred to the front of the cache. Instead, whenever the broker is left without clients with interest (at the precise moment that the last client interested in this message leaves the broker) a message degradation takes place; the message is transferred to the end of the cache. This way, we prioritize messages based on current interest. Thus we consider the following policies for prioritization.

- Prioritize with regenerations and degradations–*priority policy* (prt).

- No regenerations and degradations–*plain policy* (nop).

When a message is cached, it is always positioned at the front of the cache and thus no placement policy is considered. Also, as a dropping policy we always use: "drop the last". These two policies are a necessary reference for the priority policies.

In [8] many policies can be found, classified in three directions; traditional, key based and cost-based. In this paper we deal with traditional policies. The rest two directions are left for future work.

Several combinations of the above policies can be considered. In this work, however, we are interested to study the sel–sub–prt policy and compare it against sel–fld–prt, sel–sub–nor and the yardstick sin–fld–nor.

III. System model

Assume a pub/sub network having a set of brokers $\mathcal{N} = \{1, 2, \ldots, N\}$. Clients arrive in each broker requesting content (old and new) and stay in the system for some time until they disappear. By assuming a potentially infinite population of clients and modelling the arrivals using a Poisson process, we focus on a given content m and let the clients with subscriptions matching m to arrive with a rate $\lambda_c(m, i) \equiv \lambda_c$ and depart with a rate $\mu_c(m, i) \equiv \mu_c$, implying that for all different messages and brokers, initially we assume that client dynamics are the same[2]. Consequently, the population size of clients subscribed to a given broker i and interested in m is actually a Birth-Death process $N(t)$.

Given the above rates, one can actually calculate the stationary probability vector for this process as

[2]Here different rates can be used to model message popularity but we do not deal with this problem in this paper.

$$\pi_j = \pi_0 \frac{\rho_c^j}{j!} \quad j = 1, 2, \ldots$$

where $\pi_0 = e^{-\rho_c}$ and $\rho_c = \frac{\lambda_c}{\mu_c}$.

Similarly, we assume that all the messages arrive at the system with equal rate λ_m following a Poisson process as well. The arrival of a message models its publishing which happens only once for each message. Each message survives in the caches of the system for some random time T that depends on the caching and priority policies, as well as the client dynamics, arrival rate of the messages and the cache size. If the message disappears from all caches at one instance, then clearly it is impossible to be recovered and thus it is lost forever. We then say that the message is *absorbed*.

Let each broker be equipped with a cache of size K. Then a message m can be in one of the following cache points $\mathcal{C} = \{0, 1, \ldots, K\}$, with all nonzero points indicating the position of the message in the cache (with 1 corresponding to the top of the cache) and the zero point corresponding to the fact that m is not stored in this cache. Apart from the position of the message under examination, we might be interested in the number of messages that are not degraded (we call them alive since $N(t) > 0$ for them). In any case let $\mathcal{S}$ be the state space of a Markov process depicting our system. When the message is initially published, depending on the state of the Birth-Death processes as well as the caching policy, the Markov process starts at an initial state $s_{init} \in \mathcal{S}$. Note that $\mathbf{0} \in \mathcal{S}$ is actually an absorbing state since the message has disappeared from all the caches and will not be published again. Therefore, depending on the caching policy it might be that the message is lost before stored in any cache. Thus we will be interested in calculating the probability that the survival time is zero and the expected time to absorption given that the survival is nonzero. Mathematically this is defined as

$$P_{loss} \doteq \mathbb{P}(s_{init} = \mathbf{0}),$$

$$MST \doteq \mathbb{E}\{T(\boldsymbol{\phi}) | s_{init} \neq \mathbf{0}\},$$

where *MST* stands for *Mean Survival Time* and $T(\boldsymbol{\phi})$ is the time it takes to reach the state $\mathbf{0}$ for the first time given a stationary probability vector $\boldsymbol{\phi}$ for the initial state. The states belonging to $\mathcal{S} \setminus \{\mathbf{0}\}$ are all transient states if we allow copying of message from cache to cache. We are going to use a standard tool for solving such problems in a continuous Absorbing Markov Process (AMP) [9],[10]. We renumber the states in the generator matrix so that the transient (the non-absorbing) states come first. So if there are m absorbing states ($m = 1$ in our case) and n transient states, the generator matrix will have the following canonical form:

$$\mathbf{Q} = \begin{pmatrix} \mathbf{T} & \mathbf{t} \\ \mathbf{0}^{mn} & \mathbf{0}^{mm} \end{pmatrix}, \tag{1}$$

where $\mathbf{T}$ is a n-by-n matrix of transition rates, $\mathbf{t}$ is a n-by-m matrix of transition to absorption rates and $\mathbf{0}^{mn}$ a m-by-n matrix of zeros. We know that $\mathbf{T}\mathbf{e}_n + \mathbf{t}\mathbf{e}_m = \mathbf{0}^{n1}$, where $\mathbf{e}_n$ is a column vector of ones with n elements. The later equation states that the sum of all elements of any row of the generator matrix is equal to zero. We also know that the matrix $\mathbf{T}$ is invertible. From [10] we find

$$\mathbb{P}(T < x) = 1 - \boldsymbol{\phi} e^{\mathbf{T}x} \mathbf{e}_n.$$

$$\mathbb{E}\{T(\boldsymbol{\phi})\} = -\boldsymbol{\phi}\mathbf{T}^{-1}\mathbf{e}_n, \tag{2}$$

where $\mathbf{T}^{-1}$ is the inverse matrix, $e^{\mathbf{T}x}$ is the exponential of the matrix and $\boldsymbol{\phi}$ is the row vector with the stationary probabilities of the initial state.

The role of caching policy is to determine the initial state probabilities. The fld policy, is the only one for which we have $P_{loss} = 0$. In the selective and single policies, there is always a probability that the set of brokers, to which interested clients are directly subscribed, is empty. In both cases we have a loss event with a probability $P_{loss} = \pi_0^N = e^{-\rho_c N}$. The caching policy also affects time to absorption since selective caching will reduce the contention at the cache.

The role of priority policy is clearly to differentiate the survival time of messages by bringing popular messages to the top of the cache and thus extending the sojourn time of these message in the transient states. It is then of interest to see whether such an approach, together with the properties of a pub/sub system, is enough to provide a priority mechanism for popular and unpopular messages.

The role of the request policy is to determine the request overhead and the message retrieval efficiency. Particularly, the flooding request policy ensures the retrieval of a degraded message but requires the request to visit the whole network, while the subscription based selective policy retains the principles of the pub/sub, since the request is propagated towards the brokers with interested subscribers. Another effect of the request policy relates to copying messages. Since the flooding request policy always discovers cached messages, the copying rate is higher, leading to higher replication degree. Sometimes, this can increase the absorption time but under some circumstances it can also decrease it due to increased contention.

IV. The single broker case

In this section we present the Markov chain that models the case where the network has only one broker ($N = 1$) and K cache slots. This model captures also the case where the network is

made of more than one broker, but no message copies from cache to cache is allowed.

We model both the position of a given message m in the cache as well as the number of messages with interested clients (*alive* messages), with a two dimensional continuous-time Markov chain (CTMC) with state space $\mathcal{S} = \mathcal{C} \times \mathcal{C}$ and generator matrix $\mathbf{Q}$. $\mathbf{Q}$ contains the transition rates $q_{\mathbf{s},\hat{\mathbf{s}}}$ from any state $\mathbf{s}$ to any other state $\hat{\mathbf{s}}$, where $\mathbf{s}, \hat{\mathbf{s}} \in \mathcal{S}$, and $\mathbf{s} = \{i, j\}$ is the state where we have i alive messages and message m is stored in the j^{th} slot of the cache.

$\mathbf{Q}$ is a $((K+1)\,K+1) \times ((K+1)\,K+1)$ matrix. This is because there are $K(K+1)$ transient states in the chain and we group all K absorbing states (states of the type $\mathbf{s} = \{i, 0\}, j \in \mathcal{C}$) into one. The elements $q_{\mathbf{s},\mathbf{s}}$ of the main diagonal are defined by $q_{\mathbf{s},\mathbf{s}} = -\sum_{\hat{\mathbf{s}} \neq \mathbf{s}} q_{\hat{\mathbf{s}},\mathbf{s}}$ (the property of any generator matrix).

There are seven types of transitions that take place in our chain.

1) The event of caching a new message (other than m) in the given broker increases the number of alive messages by one and moves message m one cache slot further.
2) The event of degradation of an alive message (other than m) decreases the number of alive messages by one and may moves message m one cache slot towards to the top depending on the original position of message m (before or after the degraded message).
3) The event of regeneration of a dead message (other than m) increases the number of alive messages by one and may moves message m one cache slop further depending on the original position of message m (before or after the regenerated message).
4) The event of regeneration of an alive message (other than m) retains the number of alive messages and may moves message m one cache slot further depending on the original position of message m (before or after the regenerated message).
5) The event of regeneration of message m when message m was alive retains the number

of alive messages and moves message m to state one (at the top of the cache).

6) The event of regeneration of message m when message m was degraded increases the number of alive messages by one and moves message m at the top of the cache.

7) The event of degradation of message m decreases the number of alive messages by one and moves message m at the bottom of the cache (slot K).

In the following we formulate the transition rates according to the above intuitive connection to our system. We define the transition rate from state $\mathbf{s} = \{i, j\}$ to state $\hat{\mathbf{s}} = \{\hat{i}, \hat{j}\}$ as $q_{\hat{i},\hat{j}} = \lim_{\tau \to 0} \frac{\mathbb{P}(X(t+\tau)=\hat{\mathbf{s}}|X(t)=\mathbf{s})}{\tau}$, $\forall t$, by omitting the first index of q for presentation reasons. Then for any policy we get

$$
\begin{cases}
Tr\,(1) : q_{i+1,j+1} = \min\{K-i, K-j\}\,\lambda_{reg} + \lambda^{p}_{cch} & 0 \le i \le K-1, 1 \le j \le K-1 \\
Tr\,(2) : q_{i+1,j} = (j-i-1)\,\lambda_{reg} & 0 \le i \le j-1, 2 \le j \le K \\
Tr\,(3) : q_{i+1,1} = \lambda_{reg} & 0 \le i \le j-1, 1 \le j \le K \\
Tr\,(4) : q_{\{i,K\},\mathbf{0}} = \lambda^{p}_{cch} & 1 \le i \le K \\
Tr\,(5) : q_{i-1,j-1} = \min\{i, j-1\}\,\mu_d & 2 \le i \le K, 2 \le j \le K \\
Tr\,(6) : q_{i-1,j} = (i-j)\,\mu_d & j+1 \le i \le K, 2 \le j \le K-1 \\
Tr\,(7) : q_{i-1,K} = \mu_d & j \le i \le K, 1 \le j \le K \\
Tr\,(8) : q_{i,j+1} = \max\{0, i-j\}\,\lambda_{reg} & 0 \le i \le K-1, 1 \le j \le K-1 \\
Tr\,(9) : q_{i,1} = \lambda_{reg} & j \le i \le K, 2 \le j \le K \\
Tr\,(10) : q_{\{K,j\},\{K,j+1\}} = \max\{0, i-j\}\,\lambda_{reg} + \lambda^{p}_{cch} & 1 \le j \le K-1 \\
Tr\,(11) : q_{\hat{i},\hat{j}} = 0 & \text{otherwise},
\end{cases}
\tag{3}
$$

where in general $i, j \in \mathcal{C}$ and $K \geq 2$. μ_d is the rate of message degradation, λ_{reg} is the rate at which messages are regenerated and λ_{cch}^{p} is the rate at which message m is *pushed* in the cache by one slot when a new published message is cached in the broker (p indicates that this rate is policy-depended). Figure 2 shows the Markov chain for the single broker scenario.

Also, the stationary probability vector for the initial state of this Markov process $\boldsymbol{\phi} = \{\phi(\mathrm{s})\}$ is nonzero for all the states where message m is positioned at the first slot and given by the binomial distribution

$$\phi\left(\{i,j\}\right) = \begin{cases} \binom{K-1}{i-1} \pi_0^{K-i} (1-\pi_0)^{i-1} & 1 \leq i \leq K, j = 1 \\ 0 & \text{otherwise,} \end{cases}$$

where we have assumed that the cache is always full with K different messages the birth-death processes of which are independent and exhibit stationarity with the probability of $N(t) = 0$ being π_0.

Solving the aforementioned Markov chain requires the inversion of the reduced transition matrix $\mathbf{T}$ (obtained from $\mathbf{Q}$ by removing the row and column that contain the absorption state), which in general turns out to be impossible to obtain in a closed-form manner and thus we will rely on numerical results solely. An interesting exception is when we let μ_d and λ_{reg} go to zero where the final mean time to absorption is

$$\boldsymbol{MST} = \frac{K}{\lambda_{cch}^{p}}.$$

The first condition is implied if $\rho_c >> 1$ (or a degraded message is not moved at end of the cache) and the second condition is true if we are not using regenerations meaning that the position of the message in the cache and the number of alive messages do not play any particular role.

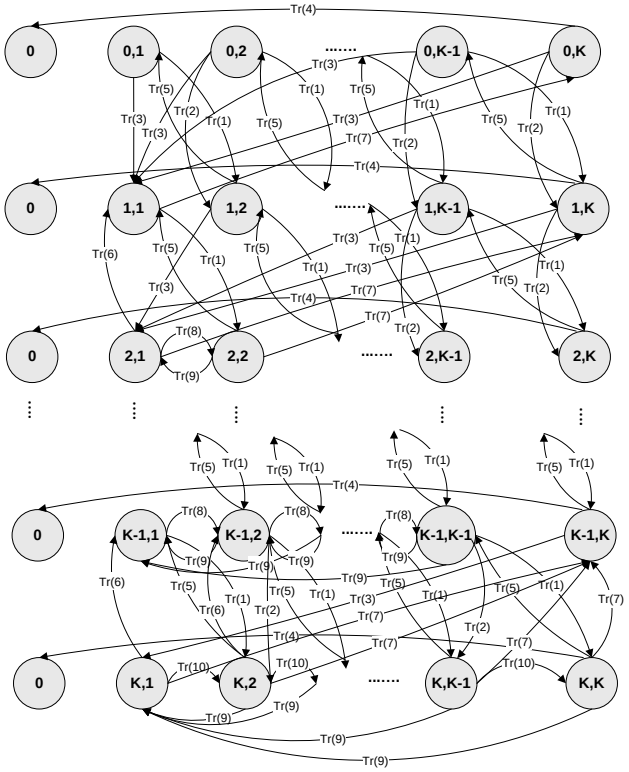

Fig. 2. Markov chain for the single broker scenario.

A. Alterations for different policies

Now we are interested to study how the several possible policies may affect the actual transition rates in our model. The transitions due to pushing message m because a new message other than m is stored in the cache depends on the caching policy. In particular for λ_{cch}^{p} we have

1) For the selective caching policy, the messages are stored in all brokers with interested clients, $\lambda_{cch}^{sel} = \lambda_m(1 - \pi_0)$.
2) For the flood caching policy, the messages are stored in all brokers, $\lambda_{cch}^{fld} = \lambda_m$.
3) For the single caching policy, the messages are stored in one randomly selected broker, $\lambda_{cch}^{sin} = \frac{\lambda_m}{N}$.

For the rate of regeneration we have either $\lambda_{reg} = \lambda_c$ in case regeneration policy is used, or $\lambda_{reg} = 0$ otherwise.

The rate of message degradation is the rate at which the underlying Birth-Death process for the clients of each message is transiting from any state $N_i, i \neq 0$ to state N_0. Therefore we have $\mu_d = \frac{\lambda_c \pi_0}{1-\pi_0}$.

V. The approximated single broker case

In order to analyse the case of one broker with K slots we developed in section IV a two-dimensional Markov chain with $|\mathcal{S}| = K(K+1)+1$ states. Since we are planning to use this model for a larger network ($N > 1$), it is important to reduce if possible the state space. Indeed, in this section we propose an approximate method which reduces the state space to $|\mathcal{S}| = K+1$ states while incurring very small errors in comparison to the exact solution. The approach is based on stationary analysis of the number of alive messages and an AMP which utilizes the stationary distribution of it.

In particular, we consider first a CTMC which captures the evolution of the number of alive message in the cache. Figure 3 depicts this Markov chain and its transitions. Solving this chain produces the stationary probabilities

$$\psi_i = \begin{cases} \dfrac{1}{1 + \sum_{n=1}^{K} \left(\frac{\prod_{j=0}^{n} \left((K-j)\lambda_c + \lambda_{cch}^p \right)}{j! \mu_d^j} \right)} & i = 0 \\ \dfrac{\prod_{j=0}^{l} \left((K-j)\,\lambda_c + \lambda_{cch}^p \right)}{j! \mu_d^j} \psi_0 & 0 < i \leq K, \end{cases}$$

where ψ_i the stationary probability of having i alive messages in the cache of the broker.

We also define as Y_i the number of alive messages in front of message m when m is at slot

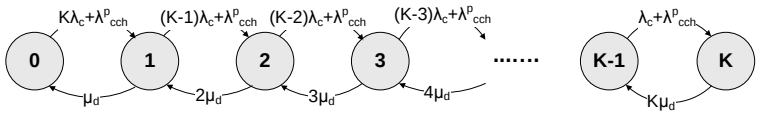

Fig. 3. Markov chain for tracking the number of the alive messages (messages with interested clients) in the single broker scenario.

i. Then the expected number $\mathbb{E}[Y_i]$ of alive messages in front of message m is given by

$$\mathbb{E}[Y_i] = \sum_{k=0}^{K} (\psi_k \cdot \min(k, i)).$$

Moreover we define the probability π_i^{alive} that message m is alive when it is in the i-th slot of the cache as

$$\pi_i^{alive} = \begin{cases} 0 \\ \sum_{k=i}^{K} \pi_k^{lv}. \end{cases}$$

Using $\boldsymbol{\psi}$, $\mathbb{E}[Y_i]$ and π_i^{alive} we can approximate statistically the number of alive messages. Thus we define the state space $\mathcal{S} = \mathcal{C}$ modeling the position of message m in the cache. In this case , there are five types of transitions that can take place in the chain that tracks the position of message m in the cache.

1) The event of caching a new message (other than m) in the given broker moves message m one cache slot further and thus triggers an increase in the state by one.
2) The event of degradation of an alive message in front of message m moves message m one cache slot towards to the top of the cache.
3) The event of regeneration of a message which is before message m (closer to the bottom of the cache) moves message m one cache slop further (closer to the end of the cache).
4) The event of regeneration of message m moves message m to state one (at the top of the cache).

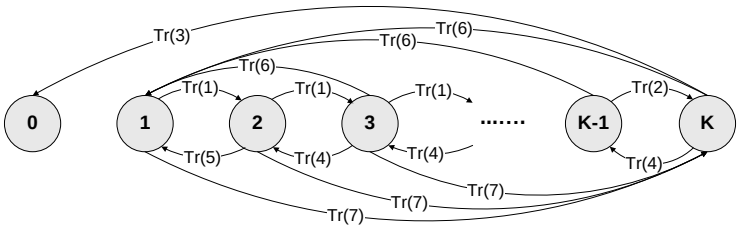

Fig. 4. Markov chain for the approximated single broker scenario.

5) The event of degradation of message m (message m should be alive) moves message m at the bottom of the cache (slot K).

We define the transition rate from state s_i to state s_j as $q_{i,j} = \lim_{\tau \to 0} \frac{\mathbb{P}(X(t+\tau)=s_j | X(t)=s_i)}{\tau}$, $s_i, s_j \in \mathcal{S}$. Then for any policy we get

$$\begin{cases} Tr\,(1) : q_{i,i+1} = (K-i) \cdot \lambda_{reg} + \lambda_{cch}^{p} & 0 < i \leq K-2 \\ Tr\,(2) : q_{K-1,K} = \lambda_{reg} + \lambda_{cch}^{p} + \pi_{K-1}^{alive} \cdot \mu_d & \\ Tr\,(3) : q_{K,\mathbf{0}} = \lambda_{cch}^{p} & \\ Tr\,(4) : q_{i,i-1} = \mathbb{E}\left[Y_i\right] \cdot \mu_d & 2 < i \leq K \\ Tr\,(5) : q_{2,1} = \mathbb{E}\left[Y_i\right] \cdot \mu_d + \lambda_{reg} & \\ Tr\,(6) : q_{i,1} = \lambda_{reg} & 2 < i \leq K \\ Tr\,(7) : q_{i,K} = \pi_i^{alive} \cdot \mu_d & 0 < i \leq K-2 \\ Tr\,(8) : q_{ij} = 0 & \text{otherwise}, \end{cases} \tag{4}$$

Figure 4 shows the Markov chain for the approximated single broker scenario. Also, the stationary probability vector for the initial state of this Markov process is given by

$$\phi\,(i) = \begin{cases} 1 & i = 1 \\ 0 & \text{otherwise}, \end{cases}$$

VI. THE MULTI-BROKER CASE

Here we study the case where the network is composed of N brokers and message copying is allowed. In this case, the message can be cached in one broker initially and be copied to other brokers later on. The message disappears from the network only when it is not cached in any broker of the network.

In order to model this system we start by making the approximation we used for the single broker scenario in the previous section. The state space is defined as $\mathcal{S} = \mathcal{C}^N$). Consider the state $\mathbf{s} = \{s_1, s_2, \ldots, s_N\}$, $s_i \in \mathcal{C}$, with the value of s_i indicating that the m message is positioned at the s_i^{th} slot in the cache of broker i.

As before, we have caching events that model the publish of a message (m or another), regeneration events that model the arrival of a client in a broker with no interest ($N(t+\Delta t) = 1|N(t) = 0$) and degradation events that model the departure of a client in a broker with only one interested client ($N(t+\Delta t) = 0|N(t) = 1$). In addition to that we should cope with message copies which occur together with the events $N(t+\Delta t) = 1|N(t) = 0$.

Due to independent increments of the multidimensional birth-death process, the events involving client dynamics (second and third above) affect only one dimension of the state. Thus, for each transition from state $\mathbf{s}$ to state $\hat{\mathbf{s}}$, we get $s_i = \hat{s}_i$, $\forall i \neq j$, where j is the dimension where the change is taking place. For those events we write $q_{\mathbf{s},\hat{\mathbf{s}}}(j) = q_{s_j,\hat{s}_j}$ where, $q_{s_j,\hat{s}_j}$ is calculated using (4) and we collect them in $\mathbf{Q}_1$ as before. Special care is required when $s_j = 0$. In such cases, a message copy takes place from another cache reachable by the chosen request policy. Thus we get

$$q_{\mathbf{s},\hat{\mathbf{s}}}(j) = q_{s_j=0,\hat{s}_j=1} = f^{\textit{request}}(\lambda_c)\,,$$

where $\mathbf{s} \neq \mathbf{0}$ and $f^{request}(\lambda_c)$ depends on the request policy used and the prioritization or not of the cached message.

- For the flooding request policy we get $f^{request}(\lambda_c) = \lambda_c$
- For the subscription-based selective request policy we get

$$f^{request}(\lambda_c) = \prod_{k=1}^{N-1} (1 - \pi_{s_k}^{alive}) \mathbb{1}_{\{s_k > 0\}}.$$

The rest of the transitions in $\mathbf{Q}_1$ take place independently in each dimension and they are given from (4) with the exception of the transitions reflecting the message copies (we call this additional pushing rate as λ_{cp}^p), which is presented in the following section.

Next we deal with events that reflect the publication of a message other than m. These events can cause a change in multiple dimensions of the state space. Specifically, in each broker where message m is stored in the cache, a new message may appear and cause a push of message m. The caching event in this case depends on the caching policy. Following we show the transitions in case of selective caching policy. Consider the set $\breve{\mathcal{S}} = \{\mathbf{s}_2 \in \mathcal{S} : \mathbf{s}_2 = \mathbf{s}_1 + \mathbf{u}, \mathbf{s}_1 \in \mathcal{S} \setminus \{\mathbf{0}\}, \mathbf{u} \in \{0,1\}^N\}$. For all $\mathbf{s} \in \mathcal{S} \setminus \{\mathbf{0}\}$ and $\hat{\mathbf{s}} \in \breve{\mathcal{S}}$ we write

$$q_{\mathbf{s},\hat{\mathbf{s}}} = (1 - \pi_0)\lambda_m. \tag{5}$$

By collecting the above transition rates in $\mathbf{Q}_2$, we finally obtain the generator matrix as $\mathbf{Q} = \mathbf{Q}_1 + \mathbf{Q}_2$.

A. The λ_{cp}^p approximation

The information required for computing the rate of message copies λ_{cp}^p is the random process $Z_i(t)$ defined as the number of messages in the network that differ from those cached in broker i. $Z_i(t)$ is hidden from our Markov chain and thus we will rely on an approximation. $Z_i(t)$ takes

values in $[0,(N-1)K]$ with the minimum attained when all caches contain exactly the same messages and the maximum attained when all messages are cached exactly once. Note that $Z_i(t)$ is not stationary and it behaviour depends greatly on the ratio of λ_c/λ_m.

In this work we make a gross approximation of $Z_i(t)$ as a first approach. In particular, we assume that all messages in the network have identical replication pattern. Since we have defined a symmetric case for the arrival of different messages and clients, this approach is expected to bring a good result. Let r_m be the number of m copies cached in the network. We make the assumption that $\tilde{Z} = \left\lfloor K\left(\frac{N}{r_m}-1\right)\right\rfloor$. Then depending on the requesting policy we have

- For the flooding request policy we get $\lambda_{cp}^{fld} = \tilde{Z}\cdot\lambda_c$.
- For the subscription-based selective request policy we should calculate how many out of the $\tilde{Z}$ messages could be copied. We define as $p_{fail} = \pi_0^{r_m}$ the probability of a different message not to be copied.

$$\lambda_{cp}^{sub} = \tilde{Z}(1-p_{fail})\lambda_c.$$

The transitions from (3) that are affected from λ_{cp}^{p} and their new form are

$$\begin{cases} Tr\,(1) : q_{i,i+1} = (K-i)\cdot\lambda_{reg} + \lambda_{cch}^{p} + \lambda_{cp}^{p} & 0 < i \leq K-2 \\ Tr\,(2) : q_{K-1,K} = \lambda_{reg} + \lambda_{cch}^{p} + \pi_{K-1}^{alive}\cdot\mu_d + \lambda_{cp}^{p} \\ Tr\,(3) : q_{K,\mathbf{0}} = \lambda_{cch}^{p} + \lambda_{cp}^{p} \end{cases} \tag{6}$$

Also, the stationary probability vector for the initial state of this Markov process is given by

$$\phi\,(\mathbf{u}) = \begin{cases} \prod_{j=1}^{N} b_j(\mathbf{u}) & ,j \in 1,\ldots,N \\ 0 & \text{otherwise,} \end{cases}$$

where $\mathbf{u}\in\{0,1\}^N$, $b_j(\mathbf{u}) = (1-\pi_0)^k\pi_0^{N-k}$ and

$$k = \sum_{j=1}^{N}\mathbf{u}$$

VII. MODEL VALIDATION

In this section we present results from the proposed models set side by side with discrete event simulations of a pub/sub system with caches. The goal is to validate the models, show that the approximations proposed yield accurate results and compare the several proposed policies. The metric for comparison is chosen to be the mean survival time (MST), i.e. the time from the publication of the message until it is disappeared from the network. The larger the MST the better indicating that a message survives longer in the system cache and thus it is available for longer time. Message availability is not only a function of the MST, but in this work we rely solely on MST for comparison.

For reasons of simulation, we consider a network with N brokers (for some experiments $N = 1$) with each broker having a cache capable of storing K messages. Initially we assume that the cache is empty while new messages are published with rate λ_m; we use a unique identifier for each message and a different independent birth-death process with birth rate λ_c and death rate μ_c representing the clients interested in this message.

A. *Validation of the Single-Broker models*

In order to validate our models we evaluate only the following combination of policies: "Selective caching policy combined with priority policy". Apart from MST, we are also interested in the relative error between the exact analytical model and the approximation. The MST and the relative error are both random variables and we estimate their mean by simulating 50k of observations. We set two experiments, one varying the client intensity ρ_c ($\lambda_c = 0.1$, $K = 15$ and $\lambda_m = 1$) and one varying the rate of new published messages λ_m ($\lambda_c = 1$, $K = 15$ and $\rho_c = 2$).

Figures in 5 depict the MST produced by the exact model (analysis) and the approximate

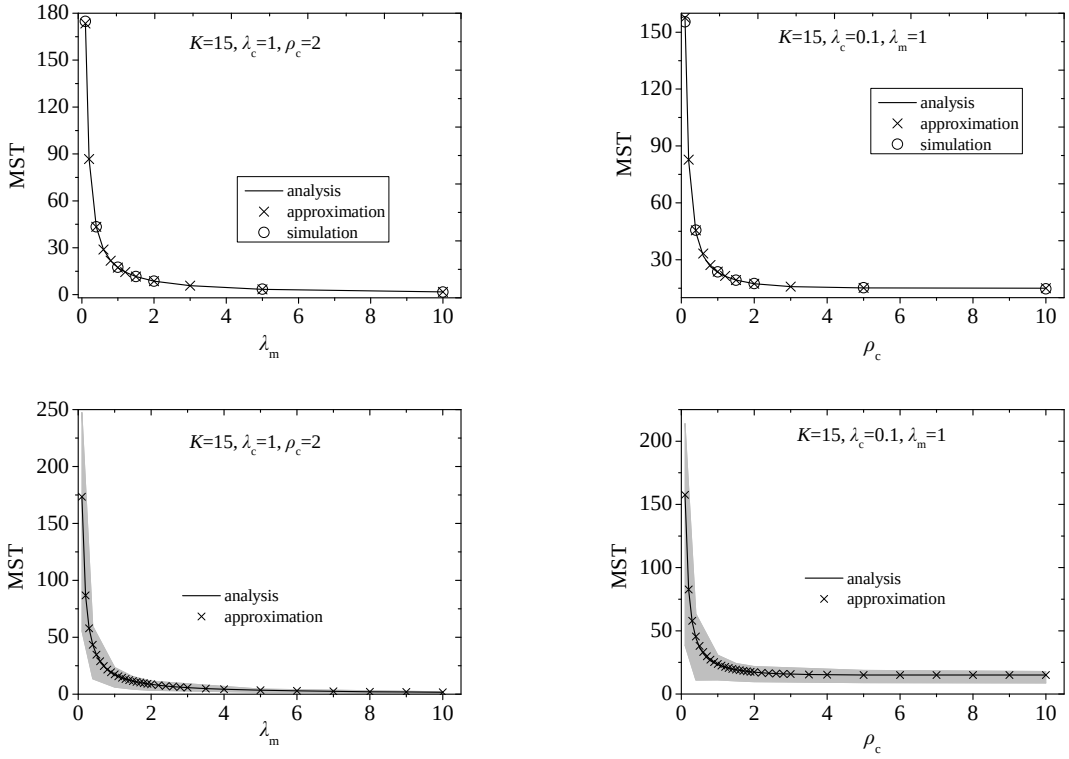

Fig. 5. Single broker: (top) MST for simulation, analysis and approximation; (bottom) The grey area represents the 50% of data–are between the 1st and 3rd quartile.

model (approximation). In the second set of Figures, the gray area represents the area between the 1st and 3rd quartile and thus the 50% of the mass of data. From these Figures we extract the conclusion that the proposed models predict very accurately the MST for several settings. Also, the approximation brings a very small error.

The relative error between analysis and approximation is showcased for several scenarios in Figures 6. The relative error is always smaller than 1% and in many cases is lower than 10^{-4}. This implies that approximating the number of alive messages in the case is well motivated and worthwhile.

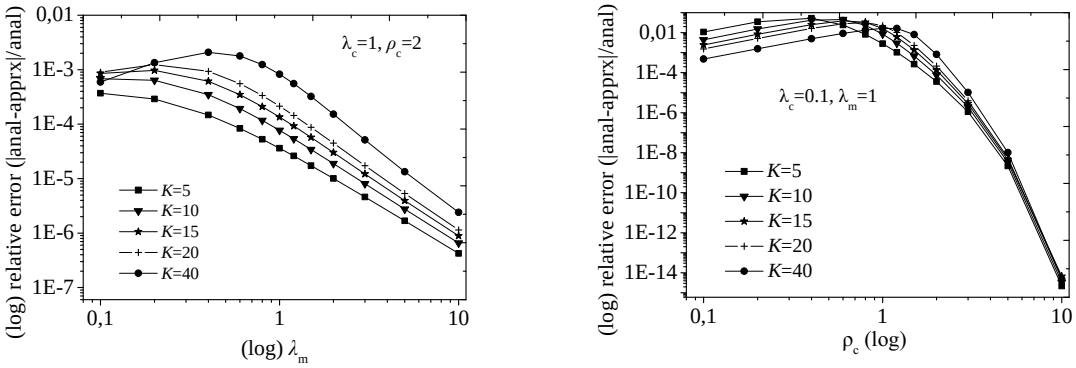

Fig. 6. Single broker: relative error between analysis and approximation for several parameter settings.

B. Validation of the Multi-Broker model

In this section we validate the proposed model for the multi-broker scenario using the discrete event simulator. For validation reasons we only consider the "Selective caching policy combined with priority policy and subscription-based selective request policy" as before. We set four experiments, one varying the number of cache slots K, one varying the client intensity per broker ρ_c, one varying the rate of message publication λ_m and one varying N, the number of brokers in the network.

Figures in frame 7 depict MST varying several settings. Note that MST decreases exponentially with λ_m and $\in rho_c$ and increases linearly with N and K, a behavior somewhat expected. Also it is notable that despite the approximations that we have used, the model is close to simulation having an error of at most 10% in the shown cases. The error seems to increase when N and K increase when the assumption for uniform replication is getting far from reality. On the other hand, the accuracy seems to be rather invariant to λ_m and ρ_c. Note that N is kept small so the size of the matrix to be inverted fits the limits of the mathematical package we used.

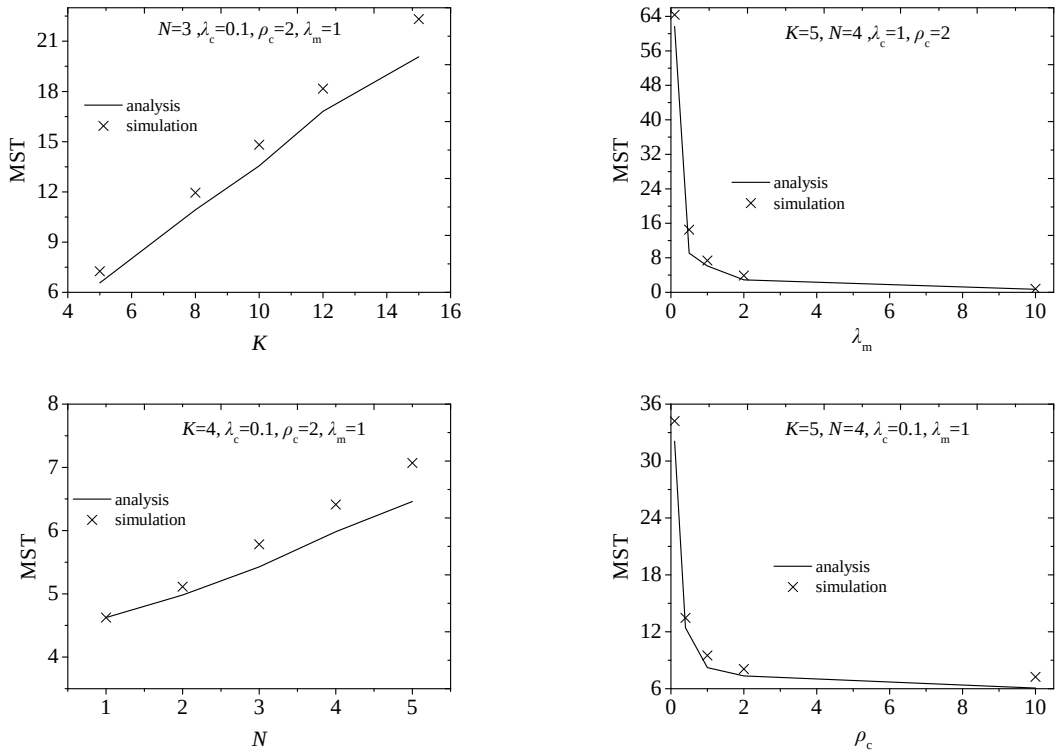

Fig. 7. Multi-broker: MST vs K, λ_m, N and ρ_c.

C. Performance evaluation of the proposed policies

In this section we compare four different combinations of policies. Particularly we compare the following combinations[3]:

- sel-sub-prt policy
- sel-fld-prt policy
- sel-sub-nor policy
- fld-fld-nor policy

[3]see section II-B1 for a description of these policies.

First we compare the policies based on P_{loss}, the probability that a published message is not cached in the system. A message that is not cached in the system, can be thought as a message with MST equal to zero. The only policy which guarantees that all messages have positive MST is fld-fld-nor policy that requires flooding of the published messages. In Figures 8 we observe this phenomenon. The pub/sub caching mechanism proposed bears always a number of messages with zero MST. This is a price to pay for the opportunistic way of caching used. The rest of the policies behave similarly. P_{loss} is significantly small when $\rho_c > 2$ or $N > 3$ in the shown examples. Whenever we have enough many brokers with $X(t) > 0$, the message is guaranteed to be cached at least once.

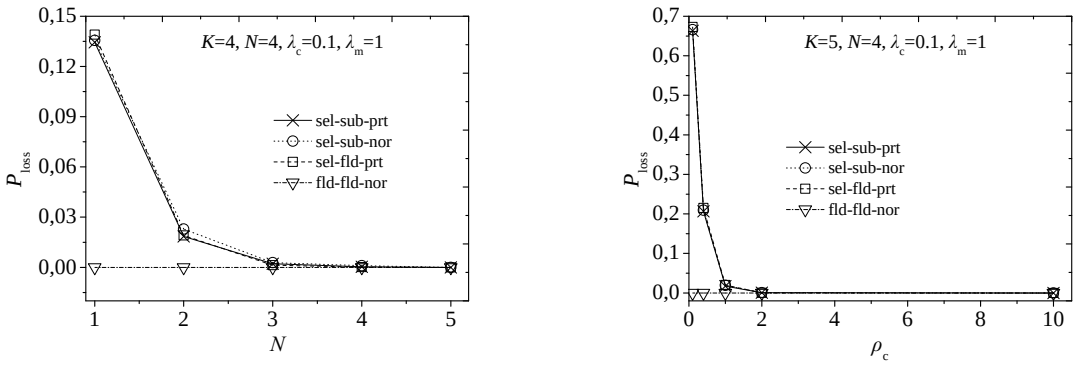

Fig. 8. Multi-broker: P_{loss} vs N and ρ_c.

Next, in Figures 9 we set four experiments of MST vs K, ρ_c, λ_m and N as before, comparing the proposed policies. First note, that the fld-fld-nor policy performs strictly worse in all settings. This is expected since no coordination is used in this policy. Next, the gain from using priority (compare prt policies with nor) is evident in most cases. Notably, priority gain increases importantly when N increases and slowly when K increases. Also for small ρ_c, a case of interest, the priority gain is very small while the gain from selective caching is very large indicating

that selective caching can provide an important differentiation tool in case of a large distributed system with non-uniform interest. From the same figure, note that for large ρ_c, a priority gain is retained while selective caching does not offer that much.

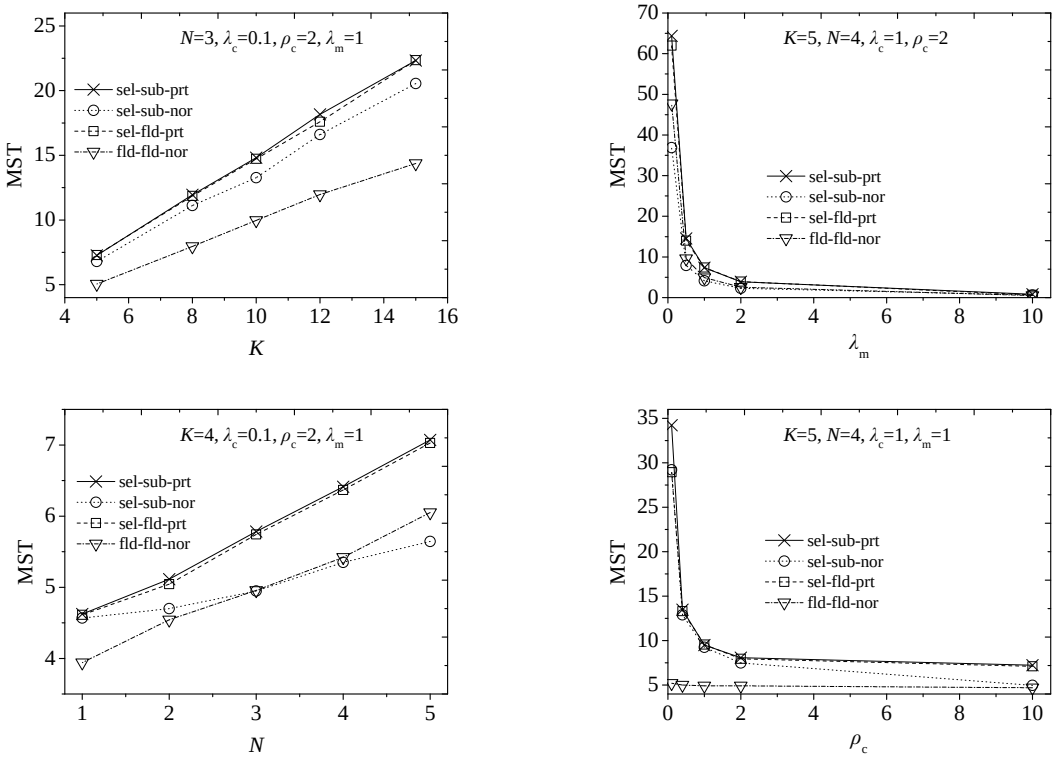

Fig. 9. Performance evaluation of the several policies: MST vs K, λ_m, N and ρ_c.

VIII. Conclusion And Future Work

We proposed a stochastic model that captures the dynamics of a pub/sub system with caches. In this system, messaged are published and cached opportunistically at the core nodes called brokers. Later, clients enter the system through a broker and request old messages. An efficient system policy should utilize the caches in order to increase the survival time of the message in the system, making it available to clients requesting it in this period of time. The proposed model are based

mostly on the transient behavior of continuous Markov chains and utilize several approximations well motivated by experiments. We use the analytical model to compare rational policies and understand their performance. Future work can be stirred in different directions, improving the model and the approximations, modeling multiclass messaging (popular and unpopular messages) and using utility theory to identify optimal policies.

REFERENCES

[1] Baldoni R., Contenti M., Piergiovanni S.T. and Virgillito A., "Modeling publish/subscribe communication systems: towards a formal approach," Proceedings of the Eighth International Workshop on Object-Oriented Real-Time Dependable Systems, 2003. (WORDS 2003), Issue 15-17, pp. 304 – 311, Jan 2003.

[2] Sourlas V., Paschos G. S., Flegkas P. and Tassiulas L., "Caching in content-based publish/subscribe systems," to appear in IEEE Globecom 2009 Next-Generation Networking and Internet Symposium.

[3] Cilia M., Fiege L., Haul C., Zeidler A., and Buchmann A. P., "Looking into the past: enhancing mobile publish/subscribe middleware," Proceedings of the 2nd international Workshop on Distributed Event-Based Systems (DEBS 2003), pp. 1–8, San Diego, California, 2003.

[4] Li G., Cheung A., Hou S., Hu S., Muthusamy V., Sherafat R., Wun A., Jacobsen H., and Manovski S., "Historic data access in publish/subscribe," Proceedings of the 2007 inaugural international Conference on Distributed Event-Based Systems (DEBS 2007), pp. 80–84, Toronto, Canada, 2007.

[5] Singh J., Eyers D. M., and Bacon J., "Controlling historical information dissemination in publish/subscribe," Proceedings of the 2008 Workshop on Middleware Security (MidSec 2008), pp. 34–39, Leuven, Belgium, 2008.

[6] Carzaniga A., Rosenblum D. and Wolf A., "Design and evaluation of a wide-area event notification service," ACM Transaction On Computer Systems, vol. 19, pp. 332–383, 2001.

[7] Eugster P. Th., Felber P. A., Guerraoui R. and Kermarrec A. M., "The many faces of publish/subscribe," ACM Computing Surveys, vol. 35, pp. 114–131, 2003.

[8] Wang J., "A survey of web caching schemes for the Internet," ACM SIGCOMM Computer Communication Review, 29 (5), pp. 36–46, 1999.

[9] Latouche G., Ramaswami V., "Introduction to Matrix Analytic Methods in Stochastic Modeling," SIAM, Philadelphia, 1999.

[10] Neuts M., "Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach", Johns Hopkins, 1994.