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

Vasilis Sourlas
University of Thessaly, Greece
CERTH-ITI
vsourlas@inf.uth.gr

Georgios S. Paschos
University of Thessaly, Greece
CERTH-ITI
gpasxos@gmail.com

Petteri Mannersalo
VTT Technical Research
Center of Finland
petteri.mannersalo@vtt.fi

Paris Flegkas
University of Thessaly, Greece
CERTH-ITI
pflegkas@inf.uth.gr

Leandros Tassiulas
University of Thessaly, Greece
CERTH-ITI
ltassiulas@inf.uth.gr

ABSTRACT

This paper considers cache dimensioning in the context of publish/subscribe (pub/sub) systems. We assume that each broker is equipped with a limited capacity cache and it decides upon a policy for caching and prioritizing messages. By using a request mechanism defined on top of the native pub/sub communication, a client may also request earlier published information. To study the *survival time* of published messages, a Markovian system model capturing the essential dynamics is defined. The model has a modular generic form which admits a variety of different policies and thus enables the calculation of their performance. For systems without message replication between the caching brokers, the distribution of message survival time is found using matrix analytic methods for solving absorbing Markov chains. For the general problem with messages copied from caches, we propose a heuristic approximation based on estimating the mean rate of copies. The approximate model is evaluated by a discrete event simulator and it is shown that for a wide set of parameters, the approximation provides a good basis for dimensioning the caches in the content-based pub/sub systems.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Miscellaneous; H.3 [**Information Systems**]: Information Storage and Retrieval

General Terms

Measurement, Performance, Theory, Verification

Keywords

analytical modeling, Markov chains, pub/sub systems, model verification and validation

SAC'11 March 21-25, 2011, TaiChung, Taiwan.

1. INTRODUCTION

The pub/sub paradigm is a 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]. This holds for all clients that are active and therefore their subscriptions are known to the system at the 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 time of an interesting message. In pub/sub systems it is not possible for a new client to retrieve previously published messages that match his/her subscription. Therefore, enabling caching for retrieval of past information (old content) is one of the most challenging problems in pub/sub networks.

Fiege et al. [2] 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. Li et al. [3] and Singh et al. [4] consider a historic data retrieval pub/sub system where databases are connected to various brokers, each associated with a filter to store particular information. This approach is based on predefined caching points in the network. A caching mechanism where the messages are stored opportunistically is introduced by Sourlas et al. [5]. Our work is in line with [5]; each broker in the network is a potential caching point for each published message and the main purpose of caching is to preserve the information over time instead of making information available in nearer space as in traditional caching schemes.

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 at the same time provides a means 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. With this goal in mind, we study opportunistic caching and

the interrelation between the client dynamics and several modular policies for arbitrating caching.

To our knowledge, there exists no prior work involving analytical models for dynamic pub/sub networks. In this paper, 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 focus on survival time of an arbitrary message, a metric that characterizes the availability of a message in the system cache.

We propose several policies, for caching, requesting and prioritizing messages. We build a multidimensional Markov model which captures the behavior of all these policies allowing for direct comparison as well as providing intuition for modes of operation. When copying between brokers is not allowed (equivalently when we consider only one broker), we find the distribution of message survival time by analyzing a two dimensional absorbing Markov process. However, even for this simple case the state space is very large and we provide a further approximation to the system reducing the state space significantly while having a minimal loss in accuracy. For the general problem where copying messages from one cache to another is allowed, we propose a heuristic approximation based on estimating the mean rate of copies. This approximate model is evaluated by a discrete event simulator. It is shown that for a wide set of parameters, the approximation provides a good basis for dimensioning the caches in the content-based pub/sub systems. 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 priority policy resembles the Least Recently Used (LRU) policy (see [8]) but it is modified to fit the pub/sub architecture.

2. THE PUB/SUB SYSTEM WITH CACHES

2.1 Legacy pub/sub architecture

The legacy pub/sub system uses the subscription forwarding routing strategy [6]. According to this network architecture, the brokers form an overlay tree and each client selects one broker, called hereinafter the serving broker, and connects to it.

At *subscription time* a client sends a `Subscribe()` message containing the corresponding subscription filter to the serving broker. The filter is a binary function which applied to 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 client. 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 pointing to a neighboring broker on the shortest 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 as well as any other communication events are propagated instantaneously through the whole network.

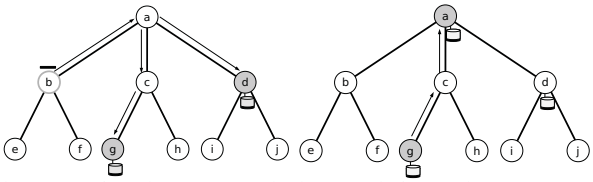

Figure 1: Caching and retrieving old information. The grey brokers have interested clients and the arrows denote the publication tree.

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].

2.2 A Caching mechanism in pub/sub

Each broker is equipped with a cache of size k (i.e. assuming equal message sizes, each cache fits exactly k messages). In the spirit of being pub/sub friendly, we introduce a simple mechanism for interaction with the cache. A client may issue a `Request()` message using a filter similarly to the subscription procedure. The `Request()` is spread across the network as a `Subscribe()` message would, i.e. multicast to all brokers matching the filter. The brokers search the cache and respond with a `Response()` message in case of a successful matching.

For better understanding consider the following example of Figure 1. First, broker `b` publishes a message which travels through the publication tree (arrows in Figure 1) and reaches brokers `g` and `d` which at the publication time have client subscriptions in their tables that match the published message (they are indicated grey). These two brokers decide opportunistically to cache the message, while brokers `a`, `b` and `c` decide not to cache the message according to an implied caching policy. Next, all clients interested in this content previously attached to broker `d` leave, and thus `d` becomes white again. Later, a client interested in this content (both old and new) enters the network through broker `a`. Using the caching mechanism, the new client at broker `a`, issues a `Request()` message which is forwarded to the brokers. Brokers `a` and `g`, having client subscriptions, search their caches for a match. Broker `g`, finds the old message and issues a `Response()` message. Finally, the interested client receives the cached message and broker `a` decides to cache the message.

The above example shows 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` had 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 or moved towards the end of it, providing more space since it is not possible to be found. The question then is whether popular files can find their way of extending their survival time only by exploiting the rate of requests for them.

2.3 Modular policies of the caching mechanism

The caching mechanism operates based on modular policies. Following we explain the role of each one and present the example policies that we will consider in this paper.

1. *Caching policy*: When a message crosses a broker, the caching policy determines whether the broker will opportunistically

store the message in the cache.

2. *Request policy*: Determines the way that the `Request()` message spreads throughout the network, reaching several brokers along the way.
3. *Priority policy*: Re-orders the messages according to client dynamics.

Caching policies: A straightforward policy is the one where a broker stores the traversing message in its cache whenever there exists at least one interested client attached to it (as in the example). We call this caching policy `sel`, standing for selective. Also, it is possible that the brokers always store the traversing messages resulting in a very dynamic caching system. We call this policy `all` and we study it solely for performance reference. Note that by caching everywhere in the propagation path causes extra overhead that might not be acceptable for the pub/sub system. In this paper we focus only on these two policies, although it is possible to define other policies for caching as well, e.g. random caching.

Request policies: A common policy is similar to the example above, where the `Request()` message is propagated along the subscription tree. We refer to this policy as the `sub` request policy. Alternatively, it is possible to flood the `Request()` message in the network (`fld` policy). This guarantees that any message cached is retrieved at the cost of higher overhead.

Priority policies: The priority policy is a way to differentiate the messages in a cache based on their usability as well as to provide better system performance. As a common principle for all policies, a new message is always positioned at the top of the cache and the last message (in the bottom) is dropped. In order to introduce priority, we propose cache management with regenerations and degradations. In particular, the client activity is monitored locally at each serving broker. Upon an arrival of a client whose subscription matches a given message in the cache, the message is *regenerated*, i.e. 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 and the message is transferred to the end of the cache. This way, we prioritize messages based on the locality of current interest. This policy is called `prt`. By using the priority policy `prt` it should be possible to differentiate between popular and unpopular messages and extend the survival time of the popular messages at the expense of the unpopular ones. As a reference, the plain policy without regeneration and degradation is also considered (called `nop`). When using this policy, the cache operates in a First-Come-First-Serve (FCFS) manner.

In this work, we are specifically interested in the `sel-sub-prt` policy and compare it against `sel-fld-prt`, `sel-sub-nop` and the yardstick `all-fld-nop`. The policy `sel-sub-***` is used for the example in Figure 1.

3. SYSTEM MODEL

We consider a pub/sub network with n brokers. 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, the client arrivals are modeled by a Poisson process. For a given content m and broker i, the clients with subscriptions matching to m arrive with a rate $\lambda_c(m,i) \equiv \lambda_c$ and depart with a rate $\mu_c(m,i) \equiv \mu_c$. Thus we assume that the client dynamics are all the same for all brokers and messages[1]. Consequently, the population sizes of clients interested in m are given by $\mathbf{X}_m(t) \doteq (X_{m,1}(t),\ldots,X_{m,n}(t))$, where $X_{m,i}(t) \stackrel{distr}{=} X(t)$ is a birth-death process of the interested clients in broker i. Given the above rates, the stationary state probability vector for $X(t)$ is $\pi_j = \pi_0 \frac{\rho_c^j}{j!}$, where $j = 0,1,\ldots.$ $\pi_0 = e^{-\rho_c}$ and $\rho_c = \frac{\lambda_c}{\mu_c}$.

Similarly, we assume that all the new messages are arriving in the system with equal rate λ_b following a Poisson process as well. The arrival of a message corresponds to a publishing which happens only once for each message. Each message survives in the caches of the system for some random time that depends on the caching, request and priority policies, as well as the client dynamics, arrival rate of the messages and the cache size. If a 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 and consider the process $\mathbf{Y}_m(t) \doteq (Y_{m,1}(t),\ldots,Y_{m,n}(t))$, where $Y_{m,i}(t) \in \mathcal{C} = \{0,1,\ldots,k\}$ is the position of message m in cache i. Here location 1 corresponds to the top of a cache and the zero point corresponds to the fact that the message is not stored in this cache. Assume that message m is published at $t = 0$ and consider the process Y_m on $t \geq 0$. The state $\mathbf{0} = \{0,\ldots,0\} \in \mathcal{C}^n$ corresponds to the absorbing state. Therefore, the message could be lost before stored in any cache. The main performance metric studied is the mean survival time (MST) which is given by

$$\mathrm{MST} \doteq \mathbb{E}\{T_m \,|\, \mathbf{Y}_m(0) \neq \mathbf{0}\}\, \mathbb{P}(\mathbf{Y}_m(0) \neq \mathbf{0})$$

where $T_m = \sup\{t > 0 :\ \mathbf{Y}_m(t) \neq \mathbf{0}\}$ denotes the survival time of the message m. We also define $P_{\mathrm{loss}} \doteq \mathbb{P}(\mathbf{Y}_m(0) = \mathbf{0})$ as the probability of initial loss of the message.

By Little's law $E(T) = E(N)/\lambda_b$, where N is the number of different messages in the system. Thus, increasing the number of different messages stored simultaneously increases also MST. It also gives analytical bounds for MST in a homogeneous system. Since the number of different messages varies between k (all caches having an identical content) and nk (all caches storing different messages), we have

$$\frac{k}{\lambda_b} \leq MST \leq \frac{nk}{\lambda_b}. \tag{1}$$

In order to get deeper performance results, we need to analyze $(\mathbf{X}_m(t), \mathbf{Y}_m(t))$ in detail. Unfortunately, it is not a Markov process due to the dependence on which messages have nonzero client population. On the other hand, the full system $(\mathbf{X}_j(t), \mathbf{Y}_j(t),\ j \in \mathbb{Z})$ is Markovian but too complicated to be studied directly. Thus we will provide approximate approaches in the following sections.

The caching policy determines the initial state probabilities. The `all` policy, is the only one for which we have $P_{\mathrm{loss}} = 0$. In `sel` policy, there is always a probability that the set of brokers, to which interested clients are directly subscribed, is empty. In that case we have a loss event with a probability $P_{\mathrm{loss}} = \pi_0^n = e^{-\rho_c n}$. The caching policy also affects time to absorption since selective caching will reduce the contention at the cache.

The request policy determines the request overhead and the message retrieval efficiency. Particularly, the `fld` request policy ensures the retrieval of a degraded message but requires the request to visit the whole network, while the subscription based selective policy (`sub`) retains the principles of the pub/sub, since the request is propagated towards the brokers with interested clients. 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, less messages in the system and smaller MST.

[1] Here different rates can be used to model message popularity. In this paper, this is studied only via simulation.

The priority policy differentiates the survival time of messages by bringing popular messages to the top of the cache and thus extending the sojourn time of these messages 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.

4. THE SINGLE BROKER CASE

In this section we consider the simplified network composed of one broker ($n = 1$) which has k cache slots. This model captures also the case where the network consists of more than one broker, but no message copies from cache to cache are allowed. Then the brokers are behaving independently, i.e., the state probabilities are just products of the single broker state probabilities.

First note Equation (1) implies $MST = k/\lambda_{\mathrm{b}}$, independently of the caching mechanisms. In order to study the distribution and more importantly to prepare the basis for the multi-broker ($n > 1$) case, 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 $\mathbf{Q}$. The matrix $\mathbf{Q}$ 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. The size of $\mathbf{Q}$ can be reduced to $((k+1)k+1) \times ((k+1)k+1)$ because there are $k(k+1)$ transient states in the chain and we can group all k absorbing states (states of the type $\mathbf{s} = \{i, 0\}, j \in \mathcal{C}$) into one. Typically, the elements $q_{\mathbf{s},\mathbf{s}}$ of the main diagonal are defined by $q_{\mathbf{s},\mathbf{s}} = -\sum_{\hat{\mathbf{s}} \neq \mathbf{s}} q_{\hat{\mathbf{s}},\mathbf{s}}$.

There are three events that may affect the state of the Markov process, (1) a caching event is triggered by the publication of a message, (2) a regeneration event is triggered by the arrival of a client interested in a message m and (3) a degradation event is triggered by the departure of a client such that the new state for this message becomes $X_m(t+\Delta t) = 0$. Accordingly there are seven types of transitions that can take place in system which focuses on a particular message m.

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 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).
3. 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.
4. The event of regeneration of an alive message (other than m) retains the number of alive messages and may move 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 a degraded message (other than m) increases the number of alive messages by one and may move message m one cache slot further depending on the original position of message m (before or after the regenerated message).
6. 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).
7. The event of degradation of an alive message (other than m) decreases the number of alive messages by one and may move message m one cache slot towards to the top depending on the original position of message m (before or after the degraded message).

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

$$q_{\hat{i},\hat{j}} = \lim_{\tau \to 0} \frac{\mathbb{P}(W(t+\tau) = \hat{\mathbf{s}} | W(t) = \mathbf{s})}{\tau}, \forall t$$

by omitting the first index of q for presentation reasons. Then for any policy we get:

$$\begin{aligned}
t\,(1) &: q_{0,0} = \lambda_{\mathrm{h}}^{\mathrm{p}} && j = k, i \leq k \\
t\,(1,4) &: q_{k,j+1} = \max\{0, i-j\}\,\lambda_{\mathrm{g}} + \lambda_{\mathrm{h}}^{\mathrm{p}} && i = k, j < k \\
t\,(1,5) &: q_{i+1,j+1} = \min\{k-i, k-j\}\,\lambda_{\mathrm{g}} + \lambda_{\mathrm{h}}^{\mathrm{p}} && i < k, j < k \\
t\,(2) &: q_{i,1} = \lambda_{\mathrm{g}} && j \leq i \leq k, 2 \leq j \leq k \\
t\,(3) &: q_{i+1,1} = \lambda_{\mathrm{g}} && i < j, j \leq k \\
t\,(4) &: q_{i,j+1} = \max\{0, i-j\}\,\lambda_{\mathrm{g}} && i < k, j < k \\
t\,(5) &: q_{i+1,j} = (j-i-1)\,\lambda_{\mathrm{g}} && i < j, 2 \leq j \leq k \\
t\,(6) &: q_{i-1,k} = \mu_{\mathrm{d}} && j \leq i \leq k, j \leq k \\
t\,(7) &: q_{i-1,j-1} = \min\{i, j-1\}\,\mu_{\mathrm{d}} && 2 \leq i \leq k, 2 \leq j \leq k \\
t\,(7) &: q_{i-1,j} = (i-j)\,\mu_{\mathrm{d}} && j < i \leq k, 2 \leq j < k \\
& q_{\hat{i},\hat{j}} = 0 && \text{otherwise,}
\end{aligned} \tag{2}$$

where in general $i, j \in \mathcal{C}$ ($i, j \geq 1$ where not explicity denoted) and $k \geq 2$, μ_{d} is the rate of message degradation, λ_{g} is the rate at which messages are regenerated and $\lambda_{\mathrm{h}}^{\mathrm{p}}$ 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-dependent). In particular,

1. For the `sel` caching policy, the messages are stored in all brokers with interested clients, $\lambda_{\mathrm{h}}^{\mathrm{sel}} = \lambda_{\mathrm{b}}(1-\pi_0)$.
2. For the `all` caching policy, the messages are stored in all brokers, $\lambda_{\mathrm{h}}^{\mathrm{all}} = \lambda_{\mathrm{b}}$.

For the rate of regeneration we have either $\lambda_{\mathrm{g}} = \lambda_{\mathrm{c}}$ in case the regeneration policy is used, or $\lambda_{\mathrm{g}} = 0$ otherwise. The rate of message degradation is the total rate of transitions of the type $\{X_m(t+\Delta t) = 0 | X_m(t) \neq 0\}$ in the underlying birth-death process $X_m(t)$. Therefore we have $\mu_{\mathrm{d}} = \frac{\lambda_{\mathrm{c}}\pi_0}{1-\pi_0}$.

The studied system is a continuous time Absorbing Markov Process (AMP) (see, e.g., [9],[10]). We renumber the states in the generator matrix so that the transient (the non-absorbing) states come first. So if there are i absorbing states and j transient states, the generator matrix will have the following canonical form:

$$\mathbf{Q} = \begin{pmatrix} \mathbf{Tr} & \mathbf{tr} \\ \mathbf{0}^{ij} & \mathbf{0}^{ii} \end{pmatrix},$$

where $\mathbf{Tr}$ is a j-by-j matrix of transition rates among transient states, $\mathbf{tr}$ is a j-by-i matrix of transition rates from transient to absorption states and $\mathbf{0}^{ij}$ a i-by-j matrix of zeros. Then

$$\mathbb{P}(MST < x) = 1 - \boldsymbol{\phi} e^{\mathbf{Tr}x} \mathbf{e}_j, \qquad MST = -\boldsymbol{\phi} \mathbf{Tr}^{-1} \mathbf{e}_j, \tag{3}$$

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

The state probability vector for the transient states $\boldsymbol{\phi}^{\mathrm{p}} = \{\phi^{\mathrm{p}}(\mathbf{s})\}$ corresponds to states where message m is positioned at the first slot, i.e., $\mathbf{s} = (i, 1)$, $i = 1, \ldots, k$. Note that the cache is always full with k different messages and for the corresponding stationary client birth-death processes it holds $\mathbb{P}(X(t) = 0) = \pi_0$. For caching policies $\mathrm{p} = \mathtt{sel}, \mathtt{all}$,

$$\phi^{\mathrm{p}}(\{i, 1\}) = \binom{k-1}{i-1} \pi_0^{k-i} (1-\pi_0)^{i-1}, \quad 1 \leq i \leq k.$$

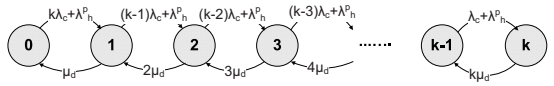

Figure 2: CTMC capturing the evolution of the number of alive messages in the single broker scenario.

The above model can be further applied to other policies for caching, request and priority as long as the event rates used in the Markovian model are tractable. Finding the distribution for the message sojourn time, i.e., applying Equation (3), requires manipulation of the reduced transition matrix **Tr**. This can be done numerically by any mathematical software after the transition rates in Equation (2) are defined.

5. REDUCING THE STATE SPACE

A two-dimensional Markov process with $|\mathcal{S}| = k(k+1)+1$ states was developed in Section 4. However, in order to use this model for a larger network ($n > 1$), it is important to reduce if possible the state space. In this section we propose a further approximation which has a state space of $|\mathcal{S}| = k+1$ states while incurring very small errors in comparison to the original model. The approach is based on stationary analysis of the number of alive messages and an AMP which utilizes this stationarity assumption.

In particular, we consider first a stationary CTMC which captures the evolution of the number of alive messages in the cache. Figure 2 depicts this Markov process and its transitions. The stationary state probabilities are given by

$$\psi_i = \begin{cases} \dfrac{1}{1+\sum_{n=1}^{k}\left(\frac{\prod_{j=0}^{n}\left((k-j)\lambda_{\mathrm{c}}+\lambda_{\mathrm{h}}^{\mathrm{p}}\right)}{j!\mu_{\mathrm{d}}^{j}}\right)} & i=0 \\ \dfrac{\prod_{j=0}^{l}\left((k-j)\,\lambda_{\mathrm{c}}+\lambda_{\mathrm{h}}^{\mathrm{p}}\right)}{j!\mu_{\mathrm{d}}^{j}}\psi_0 & 0<i\le k, \end{cases}$$

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

Define U_i as the number of alive messages in front of message m when m is at slot i. Then assuming independence between the state of message m and the number of alive message, gives

$$\mathbb{E}[U_i] = \sum_{j=0}^{k-1}\left(\psi_j \cdot \min(j, i-1)\right).$$

Moreover, the probability that message m is alive when it is in the i-th slot of the cache is given by

$$\pi_i^{alive} = \begin{cases} 0, & i=0, \\ \sum_{j=i}^{k}\psi_j & \text{otherwise.} \end{cases}$$

Using $\boldsymbol{\psi}$, $\mathbb{E}[U_i]$ and π_i^{alive} we can approximate statistically the number of alive messages and use the state space $\mathcal{S} = \mathcal{C}$ to denote the position of message m in the cache. In this model, 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 regeneration of message m moves message m to state one (at the top of the cache).

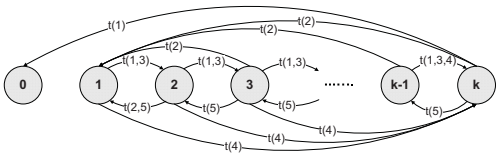

Figure 3: Markov chain for the approximated single broker scenario.

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 slot further (closer to the end of the cache).
4. The event of degradation of message m (m should be alive) moves message m at the bottom of the cache (slot k).
5. 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.

For any policy, the transition rates from state i to state j, with $i, j \in \mathcal{C}$, are given by

$$\begin{aligned} t(1):&q_{k,0} = \lambda_{\mathrm{h}}^{\mathrm{p}} \\ t(1,3):&q_{i,i+1} = (k-i)\cdot\lambda_{\mathrm{g}} + \lambda_{\mathrm{h}}^{\mathrm{p}} && 0<i\le k-2 \\ t(1,3,4):&q_{k-1,k} = \lambda_{\mathrm{g}} + \lambda_{\mathrm{h}}^{\mathrm{p}} + \pi_{k-1}^{alive}\cdot\mu_{\mathrm{d}} \\ t(2):&q_{i,1} = \lambda_{\mathrm{g}} && 2<i\le k \\ t(2,5):&q_{2,1} = \mathbb{E}[U_i]\cdot\mu_{\mathrm{d}} + \lambda_{\mathrm{g}} \\ t(4):&q_{i,k} = \pi_i^{alive}\cdot\mu_{\mathrm{d}} && 0<i\le k-2 \\ t(5):&q_{i,i-1} = \mathbb{E}[U_i]\cdot\mu_{\mathrm{d}} && 2<i\le k \\ &q_{ij} = 0 && \text{otherwise.} \end{aligned} \quad (4)$$

Figure 3 shows the transitions of the approximated single broker scenario. For caching policies p =`sel, all`, the initial state probability vector for the transient states is given by

$$\phi^{\mathrm{p}}(i) = \begin{cases} \pi_0 & i=1 \\ 0 & \text{otherwise.} \end{cases}$$

6. THE MULTI-BROKER CASE

Consider a network composed of n brokers with message copying allowed. In this case, the message can be cached initially in a subset of brokers and later copied to other brokers as well. The message disappears from the network only when it is not cached in any broker of the network. We start by making the similar approximation as used for the single broker scenario in the previous section. The state space is $\mathcal{S} = \mathcal{C}^n$ and state vector $\mathbf{s} = \{s_1, s_2, \ldots, s_n\} \in \mathcal{S}$ indicates 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, regeneration and degeneration events of a cached message. In addition to those we should cope with message copies which occur together with the events $\{X(t+\Delta t) = 1 | X(t) = 0\}$. We assume that all events concerning client activity affect only one dimension of the state, i.e., when a message is requested and sent, it will not be cached in the cache of any intermediate broker other than the serving one and only the newly arrived client will receive it. Thus, for each transition from state $\mathbf{s}$ to state $\hat{\mathbf{s}}$, we get $\hat{s}_i = s_i$, $\forall i \neq j$, where j is the dimension where the change is taking place.

For events which depend only on client activities, 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). We collect these transitions in matrix $\mathbf{Q}_1$. However, a special care is required when

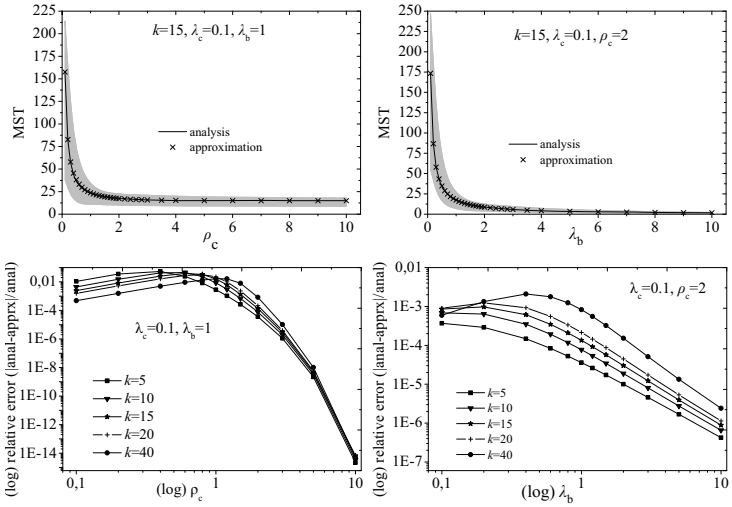

Figure 4: Single broker: (up) The gray area represents the 50% of data–data between the 1st and 3rd quartile. (down) relative error between analysis and approximation for several parameter settings.

$s_j = 0$. In such cases, a message may be copied from another cache reachable by the chosen request policy. Thus we get

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

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

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

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

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

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. In the following, the transitions in case of selective caching policy are shown. 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

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

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

The information required to compute the rate of message copies λ_{cp}^{p} is the random process $N_i(t)$ defined as the number of messages in the network that differ from those cached in broker i. Process $N_i(t)$ is hidden from our Markov chain and thus we need to rely on an approximation. Process $N_i(t)$ 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 $N_i(t)$ is not stationary and its behavior depends greatly on the ratio of λ_c/λ_b.

In this work we make a gross approximation of $N_i(t)$. In particular, we assume that at each time epoch all messages in the network have identical replication pattern as that of message m. On the average, this is, of course, true because of i.i.d. client dynamics and message arrivals. Let $R_m(\mathbf{s})$ be the number of copies of m cached in the network, i.e., it counts the total number of non-zero elements in $\mathbf{s}$ and thus is state dependent but known at each state. We make the approximation that $\tilde{N}(\mathbf{s}) = \left\lfloor k\left(\frac{n}{R_m(\mathbf{s})} - 1\right)\right\rfloor$. The intuition behind this approximation is driven by simulations where we observed that the number of replicas in the network has very small variance and has quasi-stationary behavior soon after the publication of each message.

Then depending on the request policy we have

- For the flooding request policy we get $\lambda_{cp}^{\text{fld}}(\mathbf{s}) = \tilde{N}(\mathbf{s}) \cdot \lambda_c$.
- For the subscription-based selective request policy we should calculate how many out of the $\tilde{N}(\mathbf{s})$ messages could be copied. Defining $p_{\text{fail}}(\mathbf{s}) = \pi_0^{R_m(\mathbf{s})}$ as the probability of a different message not to be copied, we get $\lambda_{cp}^{\text{sub}}(\mathbf{s}) = \tilde{N}(\mathbf{s})(1-p_{\text{fail}}(\mathbf{s}))\lambda_c$.

The transitions from (4) in $\mathbf{Q}_1$ that are affected from λ_{cp}^{p} and their new form are

$$\begin{aligned} t(1) &: q_{k,\mathbf{0}} = \lambda_{cp}^{p}(\mathbf{s}) \\ t(1,3) &: q_{i,i+1}(\mathbf{s}) = (k-i)\cdot\lambda_g + \lambda_{cp}^{p}(\mathbf{s}), \qquad 0 < i \leq k-2, \\ t(1,3,4) &: q_{k-1,k}(\mathbf{s}) = \lambda_g + \pi_{k-1}^{alive}\cdot\mu_d + \lambda_{cp}^{p}(\mathbf{s}) \end{aligned} \tag{5}$$

Also, the initial probability vector for the transient states, i.e., $\mathbf{s} \neq \mathbf{0}$, is given by

$$\phi(\mathbf{s}) = \begin{cases} (1-\pi_0)^{\sum_i s_i}\pi_0^{n-\sum_i s_i}, & s_i = 0,1, \forall i \in \{1,\ldots,n\} \\ 0, & \text{otherwise.} \end{cases}$$

7. 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 proposed approximations yield accurate results and compare the various proposed policies. The metric for comparison is chosen to be the mean survival time (MST), where survival time is the time from the publication of the message until it disappears 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.

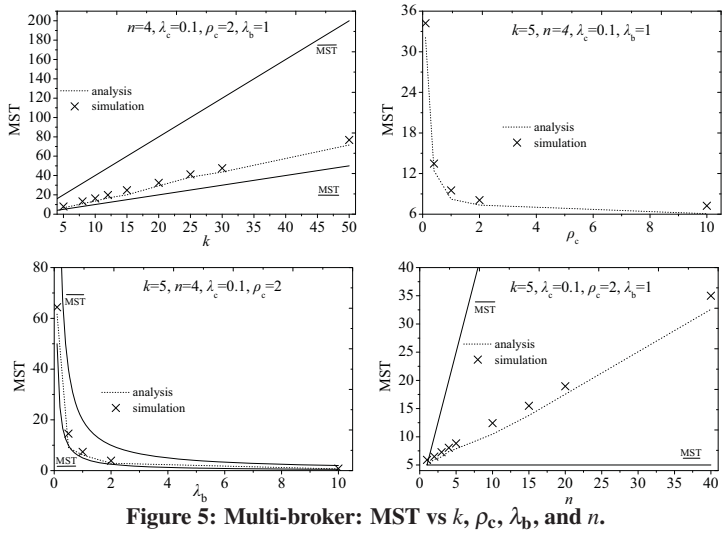

Figure 5: Multi-broker: MST vs k, ρ_c, λ_b, and n.

We consider a network with n brokers, with each broker having a cache capable of storing k messages. New messages are published with rate λ_b; 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.

7.1 Validation of the single-broker models

In order to validate our models we examine the `sel-sub-prt` policy. Apart from MST, we are also interested in the relative error between the the original model (we call it analysis) and the one with the reduced state space (we call it approximation). The MST and the relative error are both random variables and we estimate their mean by simulating 50k of observations. We set out two experiments, one varying the client intensity ρ_c ($\lambda_c = 0.1$, $k = 15$ and $\lambda_b = 1$) and one varying the rate of newly published messages λ_b ($\lambda_c = 0.1$, $k = 15$ and $\rho_c = 2$).

The gray area in Figure 4 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 also showcased for several scenarios in Figure 4. 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 is well motivated and worthwhile.

7.2 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 consider the `sel-sub-prt` policy as before. We set out 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_b and one varying n, the number of brokers in the network. We define as $\overline{\text{MST}}$ and $\underline{\text{MST}}$ the upper and lower bound respectively obtained from (1).

Figure 5 depicts MST for several experiments. Note that MST decreases fast with λ_b and ρ_c and increases linearly with n and k. 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 where the assumption for uniform replication is less accurate. On the other hand, the accuracy seems to be rather invariant to λ_b, ρ_c.

7.3 Performance evaluation of the proposed policies

In this section we compare four different combinations of policies: 1) `sel-sub-prt` policy, 2) `sel-fld-prt` policy, 3) `sel-sub-nop` policy and 4) `all-fld-nop` policy.

First we compare the policies based on P_{loss}, the probability that a published message is not cached in the system. The only policy which guarantees that all messages have positive survival time (or $P_{\text{loss}} = 0$) is the `all-fld-nop` policy that requires flooding of the published messages. In Figure 6 we observe this phenomenon. The pub/sub caching mechanism proposed bears always a number of messages with zero survival time. This is a price to pay for the opportunistic way of caching used. The rest of the policies behave similarly. P_{loss} is quite small when $\rho_c > 2$ or $n > 3$ in the shown examples.

Next, in Figure 6 we set four experiments showing MST vs k, ρ_c, λ_b and n as before, comparing the proposed policies. First note, that the `all-fld-nop` policy performs strictly worse in almost all settings. This is expected since no coordination is used in that policy. Next, the gain from using priority (compare `prt` policies with `nop`) 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.

Last we compare the proposed policies in the case of multiclass messaging using simulations. In particular, we define two classes of messages with clients subscribing to them arriving with rates λ_{pop} and λ_{unpop} correspondingly with $\lambda_{\text{pop}} > \lambda_{\text{unpop}}$ indicating the popularity. Figure 7 shows the MST for the two classes of messages when `sel-sub-prt` and `sel-sub-nop` are used. First note that the system promotes the differentiation of the classes by reducing significantly the unpopular messages survival time and allowing the use of cache space by the popular ones. Next, comparing the two policies it is possible to quantify the gain from the prioritization inside the cache and the implicit prioritization yield by the selective caching policy. From Figure 7, we conclude that the mechanism of selective caching provides an important tool for differentiating the

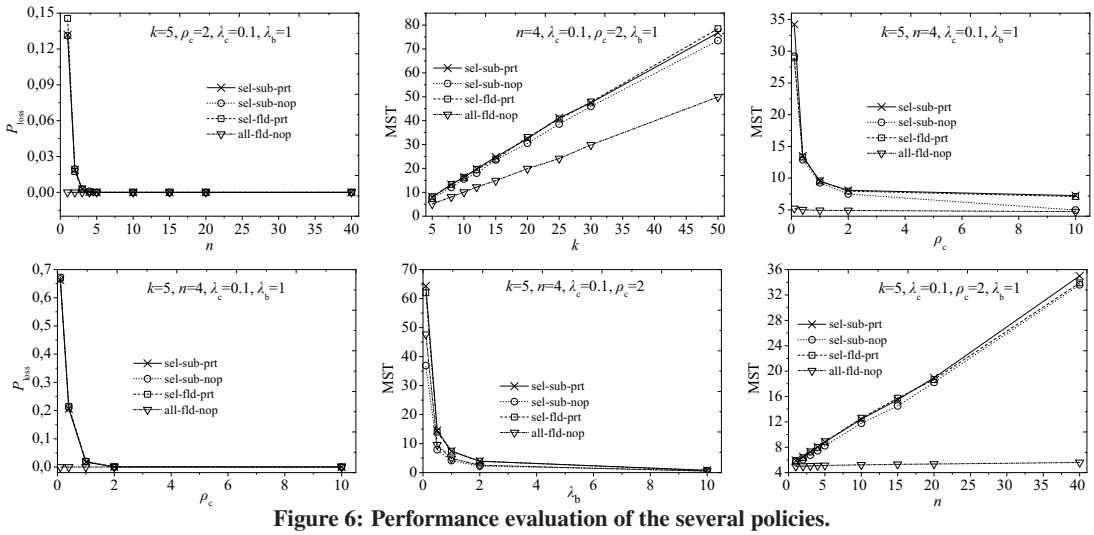

Figure 6: Performance evaluation of the several policies.

messages belonging to popular and unpopular classes by forbidding the caching of unpopular messages. When the ratio of popularity is small, however, the prioritization in the cache has a significant effect (up to 40% of improvement for ratio of 2).

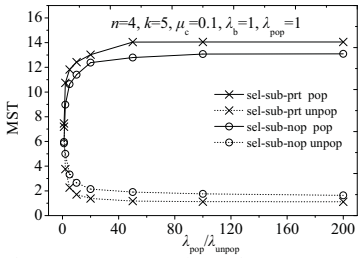

Figure 7: Performance evaluation of the proposed policies in case of multiclass messages.

8. CONCLUSION AND FUTURE WORK

We proposed a stochastic model that captures the dynamics of a pub/sub system with caches. In this system, messages 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 messages in the system, making them available to more clients requesting them. The proposed stochastic models are based mostly on the transient behavior of continuous-time Markov chains and utilize several approximations well motivated by experiments. We use the analytical models to compare rational policies and understand their performance. As a mode of operation we propose the `sel-sub-prt` policy which is lightweight, opportunistic, pub-sub–friendly and yields the most efficient results for cache operation and handling. Future work can be steered in different directions, improving the model and the approximations, modeling multiclass messaging (popular and unpopular messages) and using utility theory to identify optimal policies.

Acknowledgment

V. Sourlas' work was supported by the Greek Ministry of National Education and Religious Affairs (E.S.P.A.- "HRAKLEITOS II"), P. Mannersalo's work was supported by TEKES as part of the Future Internet program of TIVIT (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT). This work has also been supported by the European Commission through the FP7 PURSUIT program.

9. REFERENCES

[1] Baldoni R., Contenti M., Piergiovanni S.T. and Virgillito A., "Modeling publish/subscribe communication systems: towards a formal approach," Proceedings of the 8th International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2003), pp. 304 – 311, 2003.

[2] 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.

[3] 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.

[4] 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.

[5] Sourlas V., Paschos G. S., Flegkas P. and Tassiulas L., "Caching in content-based publish/subscribe systems," in Proc. of IEEE Globecom 2009, Honolulu, HI, USA, 2009.

[6] Carzaniga A., Rosenblum D. and Wolf A., "Design and evaluation of a wide-area event notification service," ACM Trans. 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.