A Methodology for Performance Modeling of Distributed Event-Based Systems

Samuel Kounev
Computer Laboratory
University of Cambridge, United Kingdom
skounev@acm.org

Kai Sachs
Department of Computer Science
Technische Universität Darmstadt, Germany
sachs@dvs.tu-darmstadt.de

Jean Bacon
Computer Laboratory
University of Cambridge, United Kingdom
jmb@cl.cam.ac.uk

Alejandro Buchmann
Department of Computer Science
Technische Universität Darmstadt, Germany
buchmann@dvs.tu-darmstadt.de

Abstract

Distributed event-based systems (DEBS) are gaining increasing attention in new application areas such as transport information monitoring, event-driven supply-chain management and ubiquitous sensor-rich environments. However, as DEBS increasingly enter the enterprise and commercial domains, performance and quality of service issues are becoming a major concern. While numerous approaches to performance modeling and evaluation of conventional request/reply-based distributed systems are available in the literature, no general approach exists for DEBS. This paper is the first to provide a comprehensive methodology for workload characterization and performance modeling of DEBS. A workload model of a generic DEBS is developed and operational analysis techniques are used to characterize the system traffic and derive an approximation for the mean event delivery latency. Following this, a modeling technique is presented that can be used for accurate performance prediction. The paper is concluded with a case study of a real life system demonstrating the effectiveness and practicality of the proposed approach.

1 Introduction

Distributed event-based systems (DEBS) based on the publish/subscribe communication paradigm were originally motivated by the need for decoupled and asynchronous dissemination of information in large-scale event-driven applications such as stock trading, Internet-wide news distribution, air traffic control and dissemination of auction bids. More recently, however, DEBS have been gaining increasing attention in many other industry domains including manufacturing, transportation, health-care and supply chain management. With the advent of ambient intelligence and ubiquitous computing, many new applications of DEBS have been proposed, for example, in the areas of transport information monitoring [3], event-driven supply chain management [26, 1], ubiquitous sensor-rich environments [9, 24] and location-based services [15, 10].

Novel event-based applications, however, pose some serious performance and scalability challenges. For example, the next generation of event-driven supply chain management based on RFID technology will be highly reliant on scalable and efficient backend systems to support the processing of acquired real-time data and its integration with enterprise applications and business processes [25]. The performance and scalability of the event-based middleware used to process real-time event data will be of crucial importance for the successful adoption of such applications in the industry.

To avoid the pitfalls of inadequate Quality of Service (QoS), it is essential that event-based systems are subjected to a rigorous performance and scalability analysis before they are put into production. The analysis should include a detailed characterization of the expected system workload as well as its anticipated development over time. Furthermore, methods are needed to estimate the system performance as a function of its configuration and the workload it is exposed to. Common performance metrics of interest are the expected event notification latency as well as the utilization and message throughput of the various system components (event brokers, network links, etc). Obtaining such information is essential in order to determine the optimal system topology, configuration and capacity that would provide adequate QoS to applications at a reasonable cost. Moreover, given the dynamics of most DEBS applications, it is important that the performance of the system is continuously monitored and analyzed during operation to help anticipate changes in the workload and take proactive measures to ensure that QoS requirements are continuously satisfied.

While numerous approaches for modeling conventional distributed systems and evaluating their performance and scalability are available in the literature, no general methodology has been proposed for DEBS. Current work on performance modeling and evaluation of DEBS is targeted at specific implementations or applications and is highly specialized. This paper is the first to provide a comprehensive methodology for workload characterization and performance modeling of DEBS that is applicable to a wide range of systems. The methodology helps to identify and eliminate bottlenecks and ensure that systems are designed and sized to meet their QoS requirements. The methodology is based on operational analysis and Queueing Petri Nets (QPNs). We first use analytical analysis techniques to find the utilization of system components and derive an approximation for the mean event delivery latency. We then show how more detailed performance models based on QPNs can be built to provide more accurate performance prediction. After presenting the methodology, we present a case study in which the methodology is successfully applied in the context of a supermarket supply chain application. The case study demonstrates the effectiveness and practicality of our modeling approach.

The rest of this paper is organized as follows. In Section 2, we define a workload model of a generic DEBS and then present our methodology for workload characterization and performance modeling. First analytical analysis techniques are used to characterize the utilization of system components and derive an approximation for the mean event delivery latency. Following this, more detailed and accurate performance models based on QPNs are developed. In Section 3, we present a modeling case study with a real life system. Following this, we review related work in Section 4 and present some concluding remarks in Section 5.

2 Modeling Methodology for DEBS

A generic distributed event-based system (DEBS) is normally composed of a set of nodes deployed in a distributed environment and exchanging information through a set of communication networks. Clients of the system are either publishers or subscribers depending on whether they act as producers or consumers of information. Publishers publish information in the form of *events* which are commonly structured as a set of attribute-value pairs. Subscribers express their interest in specific events through *subscriptions*. Most generally, subscriptions are defined as a set of constraints on the content of events. The constraints are specified using a subscription language. A published event is said to *match* a subscription if it satisfies all constraints of the subscription on the event attributes. The main task of the system is to deliver published events to all subscribers that have issued matching subscriptions.

Depending on the subscription model, DEBS can be classified into topic-based (channel-based and subject-based being variants thereof), content-based, type-based, XML-based or concept-based. Different subscription models have different expressive power. Highly expressive models enable subscribers to precisely specify the events they are interested in, however, the more expressiveness, the higher the system overhead for matching events. The typical architecture of DEBS can be decomposed into four logical layers: network layer, overlay layer, event routing layer and event matching layer. A detailed overview of these layers as well as the techniques used to implement them can be found in [6].

Modeling DEBS is challenging because of the complete decoupling of communicating parties, on the one hand, and the dynamic changes in the system structure and behavior, on the other hand. When a request is sent in a traditional request/reply-based distributed system, it is sent directly to a given destination which makes it easy to identify the system components and resources involved in its processing. In contrast to this, when an event is published in a DEBS, it is not addressed to a particular destination, but rather routed along all paths that lead to subscribers with matching subscriptions. It is hard to know in advance which system nodes will be involved in delivering the event. There are numerous event routing algorithms and they all have different implementation variants leading to different routing behavior. Moreover, depending on the subscriptions that exist, individual events published by a given publisher might be routed along completely different paths visiting different sets of system nodes. Therefore, it is hard to partition events into workload classes that have similar resource usage. Another difficulty stems from the fact that every time a new subscription is created or an existing one is modified, or when nodes join or leave the system, this might lead to significant changes in the workload. Thus, the dynamics of DEBS necessitate that workload characterization be done much more frequently in order to reflect the changes in the system configuration and workload.

For the sake of an example, without loss of generality, let us consider a distributed event-based system implemented as a network of event brokers arranged in the topology depicted in Figure 1. We assume that brokers communicate using TCP sockets. Formally, the system can be represented as a 5-tuple $G = (N, C, P, S, E)$ where:

N $= \{n_1, n_2, ..., n_{|N|}\}$ is the set of system nodes.

C $= \{c_1, c_2, ..., c_{|C|}\}$ is the set of connections btw. nodes.

P $= \{p_1, p_2, ..., p_{|P|}\}$ is the set of publishers.

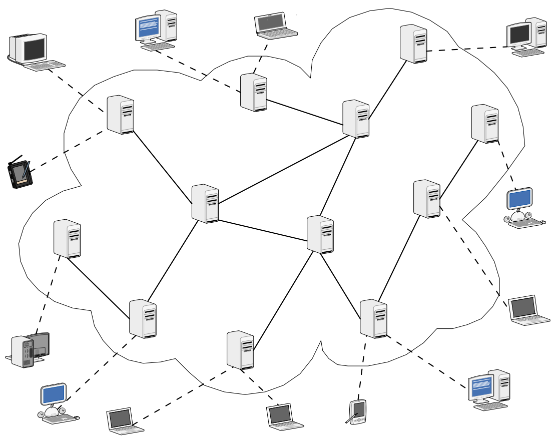

Figure 1. System topology.

S $= \{s_1, s_2, ..., s_{|S|}\}$ is the set of subscribers.

E $= \{e_1, e_2, ..., e_{|E|}\}$ is the set of event types.

We will use the following additional notation:

$H_P(r)$ is the id of the system node that publisher p_r is connected to.

$H_S(l)$ is the id of the system node that subscriber s_l is connected to.

$H_C^L(q)$ is the id of the system node on the "left side" of connection c_q. The left side is defined as the side of the node with lower id.

$H_C^R(q)$ is the id of the system node on the "right side" of connection c_q.

B_q is the bandwidth of the underlying network corresponding to connection c_q.

M_q^t is the size of the message that has to be transferred (taking protocol overhead into account) when an event of type e_t is sent over the network corresponding to connection c_q.

$\nu_{i,j}^{t,k}$ is the probability that an event of type e_t, published by publisher p_k, is forwarded to system node n_j after visiting system node n_i. If $i = j$, $\nu_{i,j}^{t,k} \overset{def}{=} 0$.

$\lambda^{t,k}$ is the rate at which events of type e_t are published by publisher p_k.

$\lambda_j^{t,k}$ is the rate at which events of type e_t, published by publisher p_k, arrive at node n_j.

λ_j^t is the total rate at which events of type e_t (published by any publisher) arrive at node n_j.

τ_q^t is the rate at which events of type e_t (published by any publisher) are sent over connection c_q.

$S_{t,j}^{CPU}$ is the mean CPU service time of an event of type e_t at node n_j.

$S_{t,j}^{I/O}$ is the mean disk I/O service time of an event of type e_t at node n_j.

$S_{t,q}^{NET}$ is the mean network service time when an event of type e_t is sent over the network link corresponding to connection c_q.

$\delta_{i,j}$ is the Kronecker function, i.e., $\delta_{i,j} = 1$ if $i = j$ and $\delta_{i,j} = 0$ if $i \neq j$.

We can consider the events published in the system as basic components of the workload. Events can be partitioned into workload classes based on their type. However, events of the same type published by different publishers could have completely different routing behavior and resource consumption. Therefore, to make the workload classes more homogeneous in terms of resource consumption, we further partition them based on the publisher.

2.1 Analysis of the Event Routing Behavior

To determine the routing behavior of events in the system, we suggest conducting some experiments in a small testing environment. Brokers are configured according to the desired topology (Figure 1), however, instead of being deployed on separate servers distributed over a WAN, they are deployed on *a few* locally hosted servers connected to a LAN. Ideally, all brokers should be deployed on a single machine, however, this might be impossible due to technical reasons. Note that co-located brokers still use TCP/IP to exchange messages and event routing at the overlay and event routing layers is done in exactly the same way that it would be done in the target environment. Subscriptions are set up according to the target workload and experiments are conducted to estimate the routing probabilities $\nu_{i,j}^{t,k}$ for $1 \leq i \leq |N|$, $1 \leq j \leq |N|$ and $i \neq j$. We assume that the system has been instrumented to monitor the event traffic and extract the routing probabilities. In each experiment, a subset of the overall set of publishers are emulated by generating events of the respective types. The event publication rates need not be equal to the target publication rates $\lambda^{t,k}$ and they should be chosen in such a way that the load injected does not exceed the capacity of the testing environment. If content-based routing is used, it must be ensured that events with content representative of the target workload are generated when conducting the experiments, i.e., the distributions of the event attributes must match their distributions in the target workload.

The following relationship between the routing probabilities $\nu_{i,j}^{t,k}$ and the arrival rates $\lambda_j^{t,k}$ holds for $j = 1, 2, ..., |N|$:

$$\lambda_j^{t,k} = \lambda^{t,k} \delta_{j,H_P(k)} + \sum_{i=1}^{|N|} \lambda_i^{t,k} \nu_{i,j}^{t,k} \tag{1}$$

Dividing both sides of Eq. (1) by $\lambda^{t,k}$ we obtain:

$$\frac{\lambda_j^{t,k}}{\lambda^{t,k}} = \delta_{j,H_P(k)} + \sum_{i=1}^{|N|} \frac{\lambda_i^{t,k} \nu_{i,j}^{t,k}}{\lambda^{t,k}} \tag{2}$$

The ratio $\frac{\lambda_j^{t,k}}{\lambda^{t,k}}$ is equal to the mean number of visits to node n_j of an event of type e_t published by publisher p_k. This ratio is known as *visit ratio* or *relative arrival rate* and we will denote it as $V_j^{t,k}$. Thus, from Eq. (2) the following relationship between the visit ratios and routing probabilities follows:

$$V_j^{t,k} = \delta_{j,H_P(k)} + \sum_{i=1}^{|N|} V_j^{t,k} \nu_{i,j}^{t,k} \tag{3}$$

Solving the above simultaneous equations enables us to derive the visit ratios based on the measured routing probabilities.

2.2 Estimation of the Event Service Times

The next step is to determine the service times of events at the system resources. This includes all resources used by the system to deliver published events. The two types of resources that we have to consider are the CPUs of the system nodes and the networks used for communication between nodes. In addition, secondary storage devices at the system nodes (e.g., disk drives) might have to be considered if they are used by the event-based middleware, e.g., for reliable delivery.

There are different approaches to estimate the service times of events at the CPU of a system node. First, the system node can be instrumented to directly measure the CPU usage when processing events. Another approach which does not require instrumentation is to estimate the service times based on measured CPU utilization and event throughput. Assume that the considered system node is deployed on a separate dedicated machine running on a given reference hardware configuration if possible similar to the node's configuration in the target environment. By injecting events of the respective type and measuring the throughput X_j^t of events at the node and the machine CPU utilization U_j^{CPU}, we can derive the mean service time $S_{t,j}^{CPU} = U_j^{CPU} / X_j^t$.

This obvious relationship follows from the Utilization Law [16]. Once the CPU service times of events at the considered system node on the reference hardware configuration have been estimated, they can be used as reference values to extrapolate the service times to the node's configuration in the target environment. Benchmark results (e.g., SPECcpu2006 or SPECjbb2005) for the respective hardware configurations can be exploited in such extrapolations. If the chosen reference configuration is similar to the target configuration, no extrapolation is necessary. If, in addition to the CPUs, further resources at the system node are used when delivering events (e.g., secondary storage devices), the mean service times at these resources can be estimated based on the measured resource utilization in exactly the same way as shown above for the CPUs. In certain cases, techniques can be employed that help to estimate the service times without the need to do any measurements on the system [23]. Such techniques are based on analyzing how the system components are implemented at the code level.

The mean network service time $S_{t,q}^{NET}$ can be calculated based on the available bandwidth and the size of the message that has to be transferred (taking protocol overhead into account) when an event of type e_t is sent over the network corresponding to connection c_q: $S_{t,q}^{NET} = M_q^t / B_q$.

2.3 System Operational Analysis

If we look at the CPUs of the nodes in our system as M/M/1 queues, we can use the following relationship from queueing theory to obtain an *approximation* for the mean response time $R_{t,j}^{CPU}$ of events of type e_t at the CPU of node n_j:

$$R_{t,j}^{CPU} = \frac{S_{t,j}^{CPU}}{1 - U_j^{CPU}} \tag{4}$$

From the Utilization Law it follows that:

$$U_j^{CPU} = \sum_{t=1}^{|E|} \lambda_j^t S_{t,j}^{CPU} = \sum_{t=1}^{|E|} \left(\sum_{k=1}^{|P|} \lambda_j^{t,k} \right) S_{t,j}^{CPU} \tag{5}$$

Similarly, an *approximation* for the mean response time $R_{t,j}^{I/O}$ of events of type e_t at the disk I/O subsystem of node n_j can be obtained. The arrival rates $\lambda_j^{t,k}$ can be derived by solving the system of simultaneous equations (1). Using the same approach, we can estimate the utilization of the network links in the system and the network response times. The rate at which events of type e_t (published by any publisher) are sent over connection c_q can be computed as follows:

$$\tau_q^t = \sum_{k=1}^{|P|} \left(\lambda_l^{t,k} \nu_{l,r}^{t,k} \right) + \sum_{k=1}^{|P|} \left(\lambda_r^{t,k} \nu_{r,l}^{t,k} \right) \tag{6}$$

where $l = H_C^L(q)$ and $r = H_C^R(q)$. Assuming that connections use dedicated network links, the utilization U_q^{NET} of the network link corresponding to connection c_q is given by:

$$U_q^{NET} = \sum_{t=1}^{|E|} \tau_q^t S_{t,q}^{NET} \tag{7}$$

An *approximation* for the mean response time $R_{t,q}^{NET}$ of events of type e_t at connection c_q is then given by:

$$R_{t,q}^{NET} = \frac{S_{t,q}^{NET}}{1 - U_q^{NET}} \tag{8}$$

If multiple connections are sharing a network link, the utilization of the network link due to each of these connections must be taken into account when computing the mean response times at the connections. Assume for example that connections $c_{q_1}, c_{q_2}, ..., c_{q_m}$ all share a single physical network link. The relative utilization of the link due to connection c_{q_i} is given by:

$$U_{q_i}^{NET} = \sum_{t=1}^{|E|} \tau_{q_i}^t S_{t,q_i}^{NET} \tag{9}$$

The total utilization of the network link can be computed by summing up the relative utilizations due to the connections that share it:

$$U_{q_1,\dots,q_m}^{NET} = \sum_{i=1}^{m} U_{q_i}^{NET} \tag{10}$$

An *approximation* for the mean response time R_{t,q_i}^{NET} of events of type e_t at connection c_{q_i} can then be obtained by substituting $U_{q_1,\dots,q_m}^{NET}$ for U_q^{NET} in Eq. (8). Now that we have approximations for the mean response times of events at the system nodes and network links, we can use this information to derive an approximation for the mean event delivery latency. In order to do that we need to capture the paths that events follow on their way from publishers to subscribers.

Definition 1 (Delivery Path) *A delivery path of an event is every ordered sequence of nodes $(n_{i_1}, n_{i_2}, ..., n_{i_m})$ without repetitions that is followed by the event upon its delivery to a subscriber (the event is published at node n_{i_1} and delivered to a subscriber at node n_{i_m}).*

Event delivery paths can be determined by monitoring the system during the experiments conducted to measure the routing probabilities $\nu_{i,j}^{t,k}$ (Section 2.1). Every delivery path can be seen as a vector $\vec{w} = (n_{i_1}, n_{i_2}, ..., n_{i_m})$ whose elements are system nodes.

Definition 2 (Dissemination Tree) *The set W of all delivery paths of an event will be referred to as the dissemination tree of the event.*

Let $W^{t,k}$ be the union of the dissemination trees of all events of type e_t published by publisher p_k. By definition $W^{t,k} = \emptyset$, if publisher p_k does not publish any events of type e_t. Let W^t be the union of the dissemination trees of all events of type e_t irrespective of the publisher, i.e., $W^t = \bigcup_{k=1}^{|P|} W^{t,k}$ is the set of all delivery paths of events of type e_t. Let $Q(i,j)$ for $1 \leq i < j \leq |N|$ be the id of the connection between nodes n_i and n_j assuming that such a connection exists.

Definition 3 (Mean Delivery Latency) *If $\tilde{W} \subseteq W^t$, the mean delivery latency $L_t(\tilde{W})$ of an event of type e_t over the set of delivery paths $\tilde{W}$ is defined as the average time it takes to deliver an event of type e_t over a randomly chosen path from $\tilde{W}$.*

If $\tilde{W}$ includes a single delivery path $\vec{w} = (n_{i_1}, n_{i_2}, ..., n_{i_m})$, an approximation for $L_t(\tilde{W})$ can be computed as follows:

$$L_t(\{\vec{w}\}) = \left(\sum_{r=1}^{m} R_{t,i_r}^{CPU} + \sum_{r=1}^{m} R_{t,i_r}^{I/O} \right) + \sum_{r=1}^{m-1} R_{t,Q(i_r,i_{r+1})}^{NET} \tag{11}$$

If $\tilde{W}$ includes multiple delivery paths $\{\vec{w}_1, \vec{w}_2, ..., \vec{w}_h\}$, we have:

$$L_t(\{\vec{w}_1, \vec{w}_2, ..., \vec{w}_h\}) = \frac{\sum_{k=1}^{h} L_t(\{\vec{w}_k\})}{h} \tag{12}$$

2.4 Performance Model Construction and Evaluation

If approximate results are not enough and accurate performance prediction is required, a more detailed performance model must be built. One possibility is to model the system using a queueing network, where system nodes and networks are represented as queues and events are represented as jobs served at the queues. Modeling the system in this way would result in a non-product form queueing network. This is because every time an event arrives at a system node, it might be forwarded to multiple other nodes, resulting in forking of multiple asynchronous tasks. Even though extended queueing networks make it possible to model the forking of asynchronous tasks, existing analysis techniques for this type of models (for example [17]) are rather restrictive and only provide approximate results.

An alternative approach is to model the system using a QPN, where system nodes and networks are represented as queueing places and events are represented as tokens. The forking of asynchronous tasks is much easier to model in this case. Whenever an event is forwarded to multiple system nodes, a transition can be used to create an instance of the event, i.e., an event token, at each of the queueing places corresponding to the target nodes. Modeling the system using QPNs provides a number of important benefits. QPN models provide excellent expressiveness and allow the integration of hardware and software aspects of system behavior into the same model [20]. This can be exploited to model the individual system nodes at a higher level of detail, capturing both hardware and software contention aspects. Furthermore, the knowledge of the structure and behavior of QPNs can be exploited for fast and efficient simulation [21]. This, on the one hand, ensures that models of realistically sized systems can be analyzed. On the other hand, it allows us to have service times with non-exponential distributions, thus improving the model's representativeness. For an introduction to modeling using QPNs, the reader is referred to [20, 19].

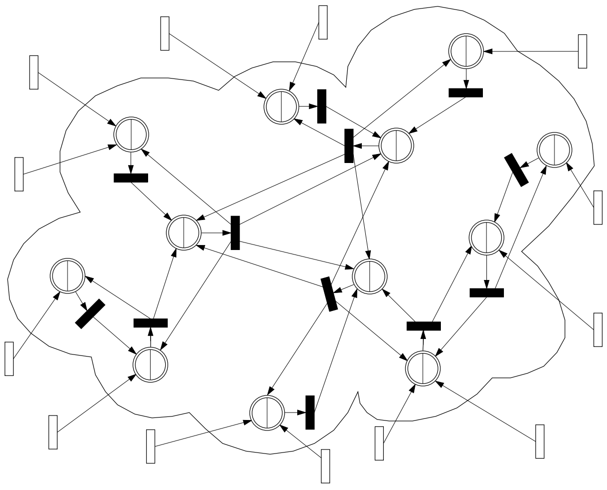

Figure 2. High-level system model.

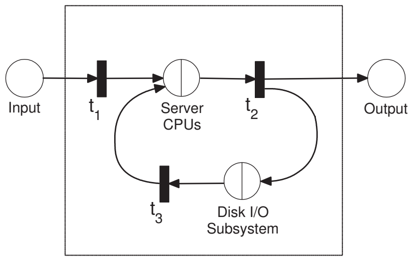

Figure 3. Node QPN model.

Figure 2 shows a QPN model of the system topology in Figure 1. In this model, we ignore the network, assuming that network delays are negligible. We will later show how the model can be extended to include contention for network resources. Each system node is modeled using a nested QPN (represented as a subnet place). The latter can be made as detailed as required to accurately capture the internal behavior of the node. In the simplest case, a system node can be modeled with a single queueing place representing the CPU of the node, assuming that this is the only resource used by the workload. Figure 3 shows an example of a nested QPN model of a system node. Two queueing places are used, the first one representing the node CPUs and the second one representing the node's disk I/O subsystem. Events are modeled using tokens and transitions are used to move events among nodes as they are routed in the system. Every system node has a single output transition. Event publications are modeled using timed transitions. We will use the following notation:

π_i is the subnet place corresponding to node n_i.

φ_i is the output transition of place π_i.

ϕ_k is the timed transition used to model event publications by publisher p_k.

To distinguish between events with different resource consumption and routing behavior, a separate token color $x^{t,k}$ is defined for every combination of event type e_t and publisher p_k that publishes events of this type ($\lambda^{t,k} > 0$). The token color $x^{t,k}$ is defined for place π_i, if and only if events of type e_t, published by publisher p_k, visit the place, i.e., $\lambda_i^{t,k} > 0$.

Every timed transition ϕ_k has a separate firing mode η_k^t for each event type e_t published by the publisher it represents:

$$\eta_k^t : \emptyset \rightarrow \pi_{H_P(k)}\{1'x^{t,k}\} \tag{13}$$

This definition specifies that whenever the transition fires in mode η_k^t, no tokens are removed from any place and only one $x^{t,k}$ token is deposited in place $\pi_{H_P(k)}$. The firing delay $\rho(\eta_k^t)$ of this mode is set to the reciprocal of the rate at which events of type e_t are published by publisher p_k: $\rho(\eta_k^t) = 1/\lambda^{t,k}$.

Since the firing delays of timed transitions in QPNs are assumed to be exponentially distributed, the above approach to modeling event publications results in Poisson event arrivals. If, however, the distribution of the time between successive event publications is not exponential, a different approach can be used. Instead of a timed transition, a queueing place and an immediate transition are used to model event publications as shown in Figure 4. The queueing place has an integrated queue with an Infinite Server (IS) scheduling strategy. The queue service time distribution is equal to the distribution of the time between successive event publications. For every event type e_t published by the publisher, there is a single event token $x^{t,k}$ in the initial marking of the place. After an event token is served at the IS queue, the immediate transition fires, moving the token back to the queue and depositing a copy of it in the system node the publisher is connected to. The latter corresponds to an event publication.

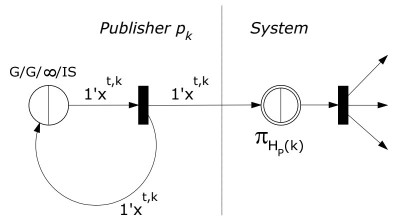

Figure 4. Modeling non-Poisson event publications.

We now show how the firing weights of the immediate transitions φ_i for $i = 1..N$ must be set in order to achieve the desired routing behavior. We use the notation

$$\mu : A_1\{c_1{}'x_1\} \wedge A_2\{c_2{}'x_2\} \wedge ... \wedge A_n\{c_n{}'x_n\} \rightarrow B_1\{d_1{}'y_1\} \wedge B_2\{d_2{}'y_2\} \wedge ... \wedge B_m\{d_m{}'y_m\}$$

to denote a transition mode μ in which $c_i \times x_i$ tokens are taken from place A_i for $i = 1..n$ and $d_j \times y_j$ tokens are deposited in place B_j for $j = 1..m$. If an event modeled by token color $x^{t,k}$ visits the system node corresponding to place π_i, from there it can possibly be forwarded to every node n_j such that $\nu^{t,k}_{i,j} > 0$. Assuming that there are m such nodes, let us denote the set of their id's as $\zeta = \{j_1, j_2, ..., j_m\} = \{j : \nu^{t,k}_{i,j} > 0\}$. For every subset of these nodes $\sigma = \{l_1, l_2, ..., l_r\} \subseteq \{j_1, j_2, ..., j_m\}$, we define a firing mode μ^σ_i of transition φ_i as follows:

$$\mu^\sigma_i : \pi_i\{1'x^{t,k}\} \rightarrow \pi_{l_1}\{1'x^{t,k}\} \wedge \pi_{l_2}\{1'x^{t,k}\} \wedge ... \wedge \pi_{l_r}\{1'x^{t,k}\} \tag{14}$$

which means that a $x^{t,k}$ token is taken from place π_i and a $x^{t,k}$ token is deposited in each of the places $\pi_{l_1}, \pi_{l_2}, ..., \pi_{l_r}$, as shown in Figure 5. This corresponds to node n_i forwarding an arriving event of type e_t, published by publisher p_k, to nodes $n_{l_1}, n_{l_2}, ..., n_{l_r}$. In order to achieve the desired routing behavior, the firing weight $\psi(\mu^\sigma_i)$ of the mode is set as follows:

$$\psi(\mu^\sigma_i) = \prod_{g \in \sigma} \nu^{t,k}_{i,g} \prod_{g \in \zeta \setminus \sigma} (1 - \nu^{t,k}_{i,g}) \tag{15}$$

To explain this, let us consider the action of node n_i forwarding an arriving event to node n_{l_h} as an "event" in terms of probability theory. For $h = 1..r$, we have r events and their probabilities are given by $\nu^{t,k}_{i,l_h}$. At the same time for each $g \in \zeta \setminus \sigma$ we can consider the action of node n_i *not* forwarding an event to node n_g as an event. We have $m - r$ such events in total and their probabilities are given by $(1 - \nu^{t,k}_{i,g})$. If we assume that all these events are independent, the probability of all of them occurring at the same time would be equal to the product of their probabilities. Thus, Eq. (15) can be interpreted as the probability of forwarding an arriving event of type e_t, published by publisher p_k, to nodes n_{l_h} for $h = 1..r$ and no other nodes. Even though in reality the independence assumption might not hold, it is easy to see that by setting the firing weights as indicated above, routing behavior with equivalent resource consumption is enforced.

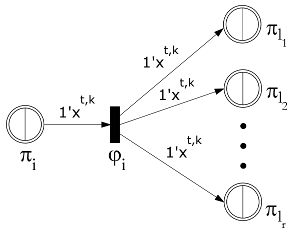

Figure 5. Firing of transition φ_i in mode μ^σ_i

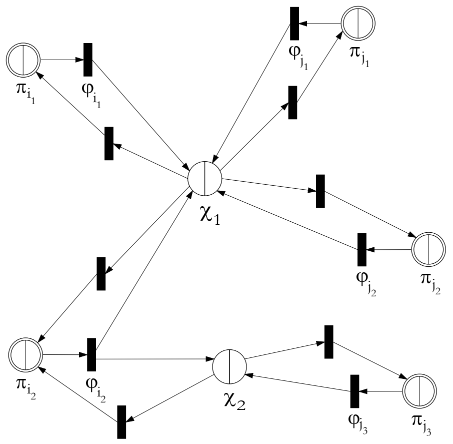

Figure 6. Modeling network connections.

The model we presented is focused on capturing resource contention inside the system nodes, however, it can easily be extended to also capture contention for network resources. Network links can be modeled using queueing places that event tokens visit when they are sent from one place to another. Figure 6 shows an example of how networks can be modeled. Nodes n_{i_1} and n_{i_2} (represented with places π_{i_1} and π_{i_2}) communicate with nodes n_{j_1} and n_{j_2} over a network (e.g., a LAN) modeled using queueing place χ_1. Node n_{i_2} communicates with node n_{j_3} through another network modeled using queueing place χ_2. Depending on the size of QPN models, different methods can be used for their analysis, from product-form solution methods [7] to optimized simulation techniques [21].

3 Case Study

Consider a scenario in which DEBS is used to manage the interactions among participants in a supermarket supply chain [26]. The participants involved are the supermarket company, its stores, its distribution centers and its suppliers. Since most of the interactions in the supply chain are asynchronous in nature, they can be implemented using DEBS. Some examples of services that can be handled by the system are supermarket order and shipment handling, inventory management in supermarkets and distribution centers, automated tracking of goods as they move through the supply chain and dissemination of new product announcements, price updates, special offers and discount information from the company headquarters to the supermarkets. Here we consider the dissemination of *requests for quotes* sent when goods in a distribution center are depleted and an order has to be placed to refill stock. A request for quote is published as an event in the system and it is automatically delivered to all suppliers that supply the respective types of goods. It is assumed that suppliers have subscribed to all events belonging to the product groups/categories they are concerned about. We have implemented the dissemination of requests for quotes using the SIENA publish/subscribe system [12] enhanced with self-monitoring functionality. We instrumented the system to monitor and collect the event publication rates and routing probabilities needed for characterizing the workload. We used a hierarchical topology with 15 brokers, 8 publishers and 16 subscribes. Brokers were communicating over a Gigabit LAN. The deployment topology is depicted in Figure 7.

Following our methodology presented in the previous section, we first characterized the workload and then built a QPN model of the system and used it to predict the system performance under load. Given that the network utilization was very low, it was omitted from the model, assuming that the network service times did not have any significant impact on the overall system performance. We employed the QPME tool (Queueing Petrinet Modeling Environment) to build and analyze the model [22]. QPME greatly simplified the task by providing a user-friendly graphical user interface for constructing QPN models and an optimized simulation engine for steady-state analysis [21]. We considered a number of workload and configuration scenarios varying the publication rates and the system topology. For lack of space, here we only consider two of the scenarios we analyzed. Tables 1 and 2 show the results for these scenarios. The broker throughput and the event delivery latency for a randomly selected subscriber are shown. As we can see, the predictions are pretty accurate and reflect the real system behavior. The results for the rest of the subscribers as well as for the other scenarios we considered were of similar accuracy to the ones presented here. The model analysis was done using SimQPN [21] and took less than 2 minutes on a standard PC hardware. A more detailed and comprehensive analysis of the performance and scalability of the proposed techniques will be presented in a follow up paper.

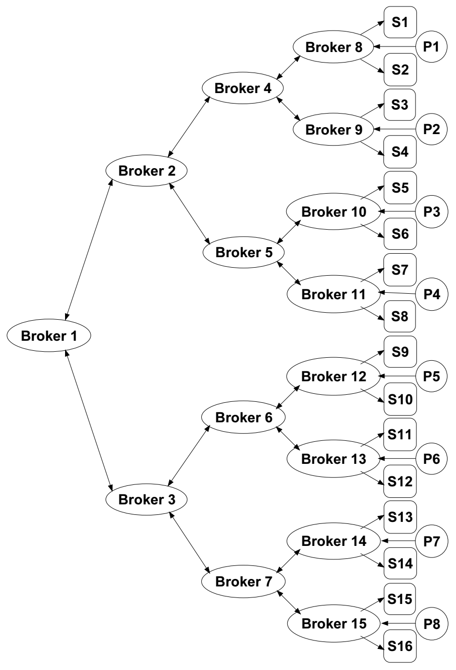

Figure 7. Broker topology

Table 1. Broker throughput (messages / sec)

	Scenario 1		Scenario 2	
Broker	Model	Measured	Model	Measured
1	94.66	93.46	61.88	62.11
2	94.65	96.15	61.88	62.11
3	89.93	89.29	59.28	59.17
4	90.40	89.29	58.27	57.80
5	83.42	84.03	56.42	56.18
6	85.24	84.75	56.35	56.18
7	71.90	71.94	48.63	48.54
8	78.91	79.37	51.12	51.28
9	67.15	68.03	43.49	43.48
10	67.14	67.11	47.01	46.95
11	59.54	59.88	41.72	41.67
12	58.26	58.82	40.01	40.16
13	73.09	72.46	48.23	48.08
14	56.35	57.47	38.49	38.46
15	63.11	63.29	42.97	42.92

Table 2. Delivery latency (ms)

	Scenario 1		Scenario 2	
Publisher	Model	Measured	Model	Measured
1	9.48	8.98	24.60	26.71
2	19.01	18.56	24.79	25.93
3	28.82	27.27	7.90	9.05
4	29.03	27.79	16.39	17.59
5	38.34	37.01	32.61	35.20
6	38.00	37.77	32.63	35.52
7	39.06	38.12	33.27	36.25
8	38.71	37.87	33.28	35.47

4 Related Work

To the best of our knowledge, there is currently very limited work on performance modeling of DEBS. In the following, we briefly discuss some related work on approaches to modeling DEBS and then consider some more general work on QoS in publish/subscribe systems. In [4, 27] a computational model of a publish/subscribe notification service is proposed, where the latter is abstracted as a black box connecting all participants in the computation. Based on the computational model, a probabilistic model for measuring the effectiveness of the notification service in delivering publications to the set of the interested subscribers is developed. The effectiveness of the notification service is studied as a function of the subscription and diffusion delays. While some interesting results are presented, the proposed model is way too coarse grained and it is based on the assumption that the subscription and diffusion delays are known which is not realistic to expect. In [5], the authors present an attempt to formally model a publish/subscribe communication system as a classical distributed computation. The authors formalize the concept of information availability and model a few properties of the computation, namely completeness and minimality, that capture the expected behavior of a publish/subscribe system from an application viewpoint. The protocol-level requirements for managing availability and providing basic QoS properties under very simplified conditions are discussed. In [18], a stochastic analysis of self-stabilizing routing algorithms for publish/subscribe systems is presented. The analysis is based on continuous time birth-death Markov Chains and investigates the characteristics of systems in equilibrium. Closed analytical solutions for the sizes of routing tables, for the overhead required to keep the routing tables up-to-date, and for the leasing overhead required for self-stabilization are presented. The proposed modeling approach, however, does not provide means to predict the event delivery latency and it is rather limited in terms of generality.

A general overview of relevant QoS metrics in the context of distributed and decentralized publish/subscribe systems can be found in [8]. In [2] it is advocated that QoS attributes in publish/subscribe systems should be managed in a uniform way with regard to other event attributes such as type or content. The authors propose a model for QoS-aware publications and subscriptions in which QoS-related properties are decoupled from event type and content. In a follow up paper published in [11], the authors present an architecture of a distributed QoS-aware publish/subscribe broker. The broker, called IndiQoS, leverages on existing network-level QoS reservation mechanisms to automatically select QoS-capable paths. The approach, however, concentrates on QoS at the network level and does not consider contention for processing resources at the broker level. In [14] an overview of the QoS aspects of publish/subscribe middleware is given. Two industrial standards for publish/subscribe middleware, the Java Message Service and the Data Distribution Service are described and their QoS-related features are discussed. Some general guidelines for designing a benchmark suite for evaluating distributed publish/subscribe systems are presented in [13], however, no specific implementation or measurement results are provided.

5 Concluding Remarks

This paper presented the first comprehensive methodology for workload characterization and performance modeling of DEBS. We developed a workload model of a generic DEBS and used operational analysis techniques to characterize the system traffic and derive an approximation for the mean event delivery latency. We then showed how more detailed performance models based on QPNs can be built and used to provide more accurate performance prediction. Finally, we presented a case study demonstrating the effectiveness and practicality of our methodology in the context of a real-world scenario. The advantage of the proposed approach is that it is both practical and general, and it can be readily exploited for performance evaluation of DEBS.

As part of our future work, we plan to apply the techniques proposed in this paper to develop a framework for designing self-managed DEBS that dynamically reconfigure themselves to ensure that QoS requirements are continuously met. The first step will be to integrate our workload characterization technique with online monitoring data and use it to automate the model generation and analysis process so that performance prediction can be carried out on-the-fly in an autonomic fashion. The task of managing the system performance will be delegated to a centralized *QoS Controller* that periodically receives the latest monitoring data from system nodes and updates the internal workload model. Using the techniques presented in this paper, the QoS controller will be able to predict the system performance on-the-fly based on observed trends in the user behavior (e.g., forecast changes in event publication rates). Thus, violations of the application QoS requirements will be anticipated and changes in the system configuration will be undertaken in a proactive manner to avoid breaking the SLAs.

6 Acknowledgments

This work was funded by the German Research Foundation (Deutsche Forschungsgemeinschaft) as part of the project ”Performance Modeling and Evaluation of Event-based Systems” under grant KO 3445/1-1.

References

[1] J. Abbott, K. B. Manrodt, and P. Moore. From Visibility to Action: Year 2004 Report on Trends and Issues in Logistics and Transportation. Technical report, March 2005. http://www.ca.capgemini.com/DownloadLibrary/requestfile.asp?ID=432.

[2] F. Araújo and L. Rodrigues. On QoS-Aware Publish-Subscribe. In *Proc. of the 2002 Intl. Workshop on Distributed Event-Based Systems*, pages 511–515, 2002.

[3] J. Bacon, A. Beresford, D. Evans, D. Ingram, N. Trigoni, A. Guitton, and A. Skordylis. TIME: An open platform for capturing, processing and delivering transport-related data. In *Proc. of the 5th IEEE Consumer Communications and Networking Conf. (CCNC), Las Vegas*, 2008.

[4] R. Baldoni, R. Beraldi, S. T. Piergiovanni, and A. Virgillito. On the modelling of publish/subscribe communication systems. *Concurrency and Computation: Practice and Experience*, 17(12):1471–1495, Oct. 2005.

[5] R. Baldoni, M. Contenti, S. T. Piergiovanni, and A. Virgillito. Modeling Publish/Subscribe Communication Systems: Towards a Formal Approach. In *Proc. of the Eighth Intl. Workshop on Object-Oriented Real-Time Dependable Systems (WORDS-2003)*, pages 304–311. IEEE, 2003.

[6] R. Baldoni and A. Virgillito. Distributed Event Routing in Publish/Subscribe Communication Systems: a Survey. Technical Report 15-05, Dipartimento di Informatica e Sistemistica, Università di Roma “La Sapienzia”, 2005.

[7] F. Bause and P. Buchholz. Queueing Petri Nets with Product Form Solution. *Performance Evaluation*, 32(4):265–299, 1998.

[8] S. Behnel, L. Fiege, and G. Mühl. On Quality-of-Service and Publish/Subscribe. In *5th Intl. Workshop on Distributed Event-based Systems*, 2006.

[9] G. Biegel and V. Cahill. A Framework for Developing Mobile, Context-aware Applications. In *Proc. of 2nd IEEE Intl. Conf. on Pervasive Computing and Communications*, 2004.

[10] I. Burcea and H.-A. Jacobsen. L-ToPSS - Push-oriented location-based services. In *In Proc. of the 4th VLDB Workshop on Technologies for E-Services (TES’03)*, 2003.

[11] N. Carvalho, F. Araujo, and L. Rodrigues. Scalable QoS-Based Event Routing in Publish-Subscribe Systems. In *Proc. of the Fourth IEEE Intl. Symposium on Network Computing and Applications*, 2005.

[12] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, 19(3):332–383, Aug. 2001.

[13] A. Carzaniga and A. L. Wolf. A Benchmark Suite for Distributed Publish/Subscribe Systems. Technical report, Dept. of Computer Science, University of Colorado, 2002.

[14] A. Corsaro, L. Querzoni, S. Scipioni, T. S. Piergiovanni, and A. Virgillito. Quality of Service in Publish/Subscribe Middleware. 8, July 2006.

[15] G. Cugola and J. E. M. de Cote. On introducing location awareness in publish-subscribe middleware. In *25th IEEE Intl. Conf. on Distributed Computing Systems Workshops*, pages 377–382, 2005.

[16] P. J. Denning and J. P. Buzen. The Operational Analysis of Queueing Network Models. *ACM Computing Surveys*, 10(3):225–261, 1978.

[17] P. Heidelberger and K. Trivedi. Queueing Network Models for Parallel Processing with Asynchronous Tasks. *IEEE Transactions on Computers*, 31(11):1099–1109, 1982.

[18] M. A. Jaeger and G. Mühl. Stochastic Analysis and Comparison of Self-Stabilizing Routing Algorithms for Publish/Subscribe Systems. In *Proc. of the 13th IEEE Intl. Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS’05)*, 2005.

[19] S. Kounev. *Performance Engineering of Distributed Component-Based Systems - Benchmarking, Modeling and Performance Prediction*. Shaker Verlag, ISBN: 3832247130, December 2005.

[20] S. Kounev. Performance modeling and evaluation of distributed component-based systems using queueing petri nets. *IEEE Transactions on Software Engineering*, 32(7):486–502, 2006.

[21] S. Kounev and A. Buchmann. SimQPN - a tool and methodology for analyzing queueing Petri net models by means of simulation. *Performance Evaluation*, 63(4-5):364–394, May 2006. doi:10.1016/j.peva.2005.03.004.

[22] S. Kounev, C. Dutz, and A. Buchmann. QPME - Queueing Petri Net Modeling Environment. In *Proc. of the 3rd Intl. Conf. on Quantitative Evaluation of SysTems (QEST)*, 2006.

[23] D. A. Menascé and H. Gomaa. A Method for Desigh and Performance Modeling of Client/Server Systems. *IEEE Transactions on Software Engineering*, 26(11), Nov. 2000.

[24] P. R. Pietzuch. *Hermes: A Scalable Event-Based Middleware*. PhD thesis, Computer Laboratory, Queens’ College, University of Cambridge, February 2004.

[25] K. Sachs. Evaluation of Performance Aspects of the SAP Auto-ID Infrastructure. Master’s thesis, Dept. of Computer Science, Darmstadt University of Technology, 2004.

[26] K. Sachs, S. Kounev, J. Bacon, and A. P. Buchmann. Workload Characterization of the SPECjms2007 Benchmark. In K. Wolter, editor, *EPEW*, volume 4748 of *LNCS*, pages 228–244. Springer Verlag, 2007.

[27] A. Virgillito. *Publish/Subscribe Communication Systems: From Models to Applications*. PhD thesis, Universita La Sapienza, 2003.