Supporting End-to-end Scalability and Real-time Event Dissemination in the OMG Data Distribution Service over Wide Area Networks

Akram Hakiri, Pascal Berthou, Andrew Gokhalec, D.C. Schmidtc, Thierry Gayraud

Supporting End-to-end Scalability and Real-time Event Dissemination in the OMG Data Distribution Service over Wide Area Networks

Akram Hakiri[a,b], Pascal Berthou[a,b], Aniruddha Gokhale[c], Douglas C. Schmidt[c], Gayraud Thierry[a,b]

[a] *CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France*
[b] *Univ de Toulouse, UPS, LAAS, F-31400 Toulouse, France*
[c] *Institute for Software Integrated Systems, Dept of EECS*
Vanderbilt University, Nashville, TN 37212, USA

Abstract

Assuring end-to-end quality-of-service (QoS) in distributed real-time and embedded (DRE) systems is hard due to the heterogeneity and scale of communication networks, transient behavior, and the lack of mechanisms that holistically schedule different resources end-to-end. This paper makes two contributions to research focusing on overcoming these problems in the context of wide area network (WAN)-based DRE applications that use the OMG Data Distribution Service (DDS) QoS-enabled publish/subscribe middleware. First, it provides an analytical approach to bound the delays incurred along the critical path in a typical DDS-based publish/subscribe stream, which helps ensure predictable end-to-end delays. Second, it presents the design and evaluation of a policy-driven framework called Velox. Velox combines multi-layer, standards-based technologies—including the OMG DDS and IP DiffServ—to support end-to-end QoS in heterogeneous networks and shield applications from the details of network QoS mechanisms by specifying per-flow QoS requirements. The results of empirical tests conducted using Velox show how combining DDS with DiffServ enhances the schedulability and predictability of DRE applications, improves data delivery over heterogeneous IP networks, and provides network-level differentiated performance.

Keywords: DDS services, Schedulability, QoS Framework, DiffServ.

1. Introduction

Current trends and challenges. Distributed real-time and embedded (DRE) systems, such as video surveillance, on-demand video transmission, homeland security, online stock trading, and weather monitoring, are becoming more dynamic, larger in topology scope and data volume, and more sensitive to end-to-end latencies [1]. Key challenges faced when fielding these systems stem from how to distribute a high volume of messages per second while dealing with requirements for scalability and low/predictable latency, controlling trade-offs between latency and throughput, and maintaining stability during bandwidth fluctuations. Moreover, assuring end-to-end quality-of-service (QoS) is hard because end-system QoS mechanisms must work across different access points, inter-domain links, and within network domains.

Over the past decade, standards-based middleware has emerged that can address many of the DRE system challenges described above. In particular, the OMG's *Data Distribution Service* (DDS) [2] provides real-time, data-centric publish/subscribe (pub/sub) middleware capabilities that are used in many DRE systems. DDS's rich

Email addresses: hakiri@laas.fr (Akram Hakiri), berthou@laas.fr (Pascal Berthou), a.gokhale@vanderbilt.edu (Aniruddha Gokhale), d.schmidt@vanderbilt.edu (Douglas C. Schmidt), gayraud@laas.fr (Gayraud Thierry)

QoS management framework enables DRE applications to combine different policies to enforce desired end-to-end QoS properties.

For example, DDS defines a set of *network scheduling policies* (*e.g.*, end-to-end network latency budgets), *timeliness policies* (*e.g.*, time-based filters to control data delivery rate), *temporal policies* to determine the rate at which periodic data is refreshed (*e.g.*, deadline between data samples), *network priority policies* (*e.g.*, transport priority is a hint to the infrastructure used to set the priority of the underlying transport used to send data in the DSCP field for DiffServ), and other policies that affect how data is treated once in transit with respect to its reliability, urgency, importance, and durability.

Although DDS has been used to develop many scalable, efficient and predictable DRE applications, the DDS standard has several limitations, including:

- **Lack of policies for processor scheduling.** DDS does not define policies for processor-level packet scheduling *i.e.*, it provides no standard means to designate policies to schedule IP packets. It therefore lacks support for analyzing end-to-end latencies in DRE systems. This limitation makes it hard to assure real-time and predictable performance of DRE systems developed using standard-compliant DDS implementations.

- **End-to-end QoS support.** Although DDS policies manage QoS between publisher and subscribers, its control mechanisms are available only at end-systems. Overall response time and pubsub latencies, however, are also strongly influenced by network behavior, as well as end-system resources. As a result, DDS provides no standard QoS enforcement when a DRE system spans multiple different interconnected networks, *e.g.*, in wide-area networks (WANs).

Solution approach → End-system performance modeling and policy-based management framework to ensure end-to-end QoS. This paper describes how we enhanced DDS to address the limitations outlined above by defining mechanisms that (1) coordinate scheduling of the host and network resources to meet end-to-end DRE application performance requirements [3] and (2) provision end-to-end QoS over WANs composed of heterogeneous networks comprising networks with different transmission technologies over different links managed by different service providers that support different technologies (such as wired and wireless network links). In particular, we focus on the end-to-end timeliness and scalability dimensions of QoS for this paper, while referring to these properties simply and collectively as "QoS."

To coordinate scheduling of host and network resources, we developed a performance model that calculates each node's local latency and communicates it to the DDS data space. This latency is used to model each end-system as a schedulable entity. This paper first defines a pub/sub system model to verify the correctness and effectiveness of our performance model and then validates this model via empirical experiments. The parameters found in the performance model are injected in the framework to configure the latency budget DDS QoS policies.

To provision end-to-end QoS over WANs composed of heterogeneous networks, we developed a QoS policy framework called *Velox* to deliver end-to-end QoS for DDS-based DRE systems across the Internet by supporting QoS across multiple heterogeneous network domains. Velox propagates QoS-based agreements among heterogeneous networks involving the chain of inter-domain service delivery. This paper demonstrates how those different agreements can be used together to assure end-to-end QoS service levels: : the QoS characterization is done from the application, and notifies the upper layer about its requirements, which adapt the middleware's service to them using the DDS QoS settings. Then, the middleware negotiates the network QoS with Velox on behalf of the application. Figure 1 shows the high-level architecture of our solution.

We implemented the two mechanisms described above into the Velox extension of DDS and then used Velox to evaluate the following issues empirically:

- How DDS scheduling overhead contributes to processing delays, which is described in Section 3.2.2.

- How DDS real-time mechanisms facilitate the development of predictable DRE systems, which is described in Section 3.2.4.

- How DDS QoS mechanisms impact bandwidth protection in WANs, which is described in Section 3.3.2.

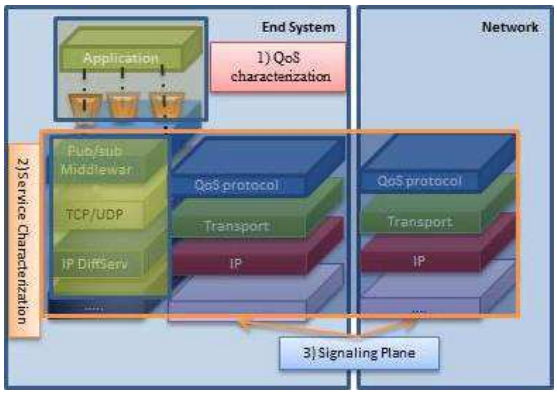


Figure 1: End-to-end Architecture for Guaranteeing Timeliness in OMG DDS

- How customized implementations of DDS can achieve lower end-to-end delay, which is described in Section 3.3.3.

The work presented in this paper differs from our prior work on QoS-enabled middleware for DRE systems in several ways. Our most recent work [4, 5] only focused on bridging OMG DDS with the *Session Initiation Protocol* (SIP) to assure end-to-end timeliness properties for DDS-based application. In contrast, this paper uses the Velox framework to manipulate network elements to use mechanisms, such as DiffServ, to provide QoS properties. Other earlier work [6] described how priority- and reservation-based OS and network QoS management mechanisms could be coupled with CORBA-based distributed object computing middleware to better support dynamic DRE applications with stringent end-to-end real-time requirements in controlled LAN environments. In contrast, this paper focuses on DDS-based applications running WANs.

We focused this paper on DDS and WANs due to our observation that many network service providers allow clients to use MPLS over DiffServ to support their traffic over the Internet, which also is also the preferred approach to support QoS over WANs. We expect our Velox technique is general enough to support end-to-end QoS for a range of communication infrastructure, including CORBA and other service-oriented and pub/sub middleware. We emphasize OMG DDS in this paper since prior studies have showcased DDS in LAN environments, so our goal was to extend this existing body of work to evaluate DDS QoS properties empirically in WAN environments.

Paper organization. The remainder of this paper is organized as follows: Section 2 conducts a scheduling analysis of the DDS specification and describes how the Velox QoS framework manages both QoS reservation and the end-to-end signaling path between remote participants; Section 3 analyzes the results of experiments that evaluate our scheduling analysis models and the QoS reservation capabilities of Velox; Section 4 compares our research on Velox with related work; and Section 5 presents concluding remarks and lessons learned.

2. The Velox Modeling and End-to-end QoS Management Framework

This section describes the two primary contributions of this paper:

- **The performance model of DDS scheduling**. This contribution describes the end-system that hosts the middleware itself and analyzes its capabilities and drawbacks in terms of scheduling capabilities and timeliness used by DDS on the end-system and across the network.

- **The Velox policy-based QoS framework**. This contribution performs the QoS negotiation and the resource reservation to fulfill participants QoS requirements across WANs.

This performance model is evaluated according the queuing systems and the values provided this analytical model are used to configure the QoS DDS Latency policy in XML file at end-system (shown later in Figure 20). Those values are used by Velox to configure the session initiation at setup phase. Together, these contributions help analyze an overall DRE system from both the user and network perspectives.

2.1. An Analytical Performance Model of the DDS End-to-end Path

Below we present an analytical performance model that can be used to analyze the scheduling activities used by DDS on the end-system and across the network.

2.1.1. Context: DDS and its Real-time Communication Model

To build predictable DDS-based DRE systems developers must leverage the capabilities defined by the DDS specification. For completeness we briefly summarize the OMG DDS standard to outline how it supports a scalable and QoS-enabled data-centric pub/sub programming model. Of primary interest to us are the following QoS policies and entities defined by DDS:

- *Listeners and WaitSets* receive data asynchronously and synchronously, respectively. Listeners provide a callback mechanism that runs asynchronously in the context of internal DDS middleware thread(s) and allows applications to wait for the arrival of data that matches designated conditions. WaitSets provide an alternative mechanism that allows applications to wait synchronously for the arrival of such data. DRE systems should be able to control the scheduling policies and the assignments of the scheduling policies, even for threads created internally by the DDS middleware.

- The DDS *deadline* QoS policy establishes a contract between data writers (which are DDS entities that publish instances of DDS topics) and data readers (which are DDS entities that subscribe to instances of DDS topics) regarding the rate at which periodic data is refreshed. When set by datawriters, the deadline policy states the maximum deadline by which the application expects to publish new samples. When set by data readers, this QoS policy defines the deadline by which the application expects to receive new values for the Topic. To ensure a datawriter's offered value complies with a data reader's requested value, the following inequality should hold:

$$offered\_deadline \leq requested\_deadline \quad (1)$$

- The DDS *latency_budget* QoS policy establishes guidelines for acceptable end-to-end delays. This policy defines the maximum delay (which may be in addition to the transport delay) from the time the data is written until the data is inserted in the reader's cache and the receiver is notified of data's arrival. It is therefore used as a local urgency indicator to optimize communication (if zero, the delay should be minimized).

- The DDS *time_based_filter* QoS policy mediates exchanges between slow consumers and fast producers. It specifies the minimum separation time for application to indicate it does not necessary want to see all data samples published for a topic, thereby reducing bandwidth consumption.

- The DDS *transport_priority* QoS policy specifies different priorities for data sent by datawriters. It is used to schedule the thread priority to use in the middleware on a per-writer basis. It can also be used to specify how data samples use *DiffServ Code Point* (DSCP) markings for IP packets at the transport layer.

We consider these QoS policies in our performance model described in Section 2.1.3 since they meet the DDS *request/offered* framework for matching publishers to subscribers. These policies can also be used to control the end-to-end path by simultaneously matching DDS data readers, topics, and data writers.

2.1.2. Problem: Determining End-to-end DDS Performance at Design-time

The OMG DDS standard is increasingly used to deploy large-scale applications that require scalable and QoS-enabled data-centric pub/sub capabilities. Despite the large number of QoS policies and mechanisms provided by DDS implementations, however, it is not feasible for an application developer to determine at design-time the expected end-to-end performance observed by the different entities of the application. There are no mechanisms in standard DDS to provide an accurate understanding of the end-to-end delays and predictability of pub/sub data flows, both of which are crucial to application operational correctness.

These limitations stem from shortcomings in DDS to control the following scheduling and buffering activities in the end-to-end DDS path:

- **Middleware-Application interface.** DDS provides no mechanisms to control and bound the overhead on

the activities at the interface of the DDS middleware and the application. This interface is used primarily by (1) data writers to publish data from the application to the middleware and (2) data readers to read the published data from the middleware into the application space. Developers of DDS application have no common tools to estimate the performance overhead at this interface.

- **Processor scheduling.** When application-level data is transiting through the DDS middleware layer, it must be (de)serialized, processed for the QoS policies, and scheduled for dispatch over the network (or read from the network interface card). Since DDS does not dictate control over the scheduling of the processor and I/O resources during this part of the critical path traversal, it is essential to analyze scheduling performance and effectiveness of a DDS-based system, particularly where real-time communication is critical.

- **Network scheduling.** Although DDS provides mechanisms to control communication-related QoS, these mechanisms exist only at an end-system. Consequently, there is no mechanism to bound the delay incurred over the communication channels.

The consequence of these limitations is that developers of DDS applications have no common analysis techniques or tools at their disposal to estimate the expected performance for their DDS-based applications at design-time.

2.1.3. Solution Approach: Developing an Analytical Performance Model for DDS

One approach to resolving the problems outlined in Section 2.1.2 would be to empirically measure the performance of the deployed system. Depending on the deployment environments and QoS settings, however, different performance results will be observed. Moreover, empirical evaluation requires fielding an application in representative deployment environment. To analyze DDS capabilities to deliver topics in real-time, therefore, we present a stochastic performance model for the end-to-end path traced via a pub/sub data flow within DDS. This model is simple but powerful to express the performance constrains without adding complexity to the system. In more complicated models, one common solution is to look for a canonical form to reduce the complexity and hold the power of the expression of the model to permit powerful analysis techniques for validating the quality of service. The model presented in this paper is well suited for the context of LAN as well as WAN context and does not require any additional complexity because it can express the behavior of the system easily and allows powerful metrics to evaluate the performance of the system.

Figure 2 shows the different data timeliness QoS policies described below, along with the time spent at different scheduling entities in the critical path.

Model Assumptions. We assume knowledge of the following system parameters to assist the analysis of processor scheduling:

- Each job requires some CPU processing to execute in the minimum possible time t_i, meaning that a job_i can be executed at the cost of a slower execution rate.

- There is sufficient bandwidth to support all data transfer at the defined rate without losing data packets.

- The CPU scheduler can preempt jobs that are currently being executed and resume their execution later.

- The service times for successive messages have the same probability distribution and all are mutually independent.

- The publish rate λ (which is the rate at which messages are generated) is governed by a Poisson process and events occur continuously and independently at a constant average arrival rate λ, having exponential distribution with average arrival time $\frac{1}{\lambda}$.

- The service rate μ (which is the rate at which published messages arrive at the subscriber) has an exponential distribution and is also governed by a Poisson process.

- The traffic intensity per CPU ρ (which is the normalized load on the system) defines the probability that the processor is busy processing pub/sub messages. The utilization rate of the processor is defined as the ratio $\rho = \frac{\lambda}{\mu}$.

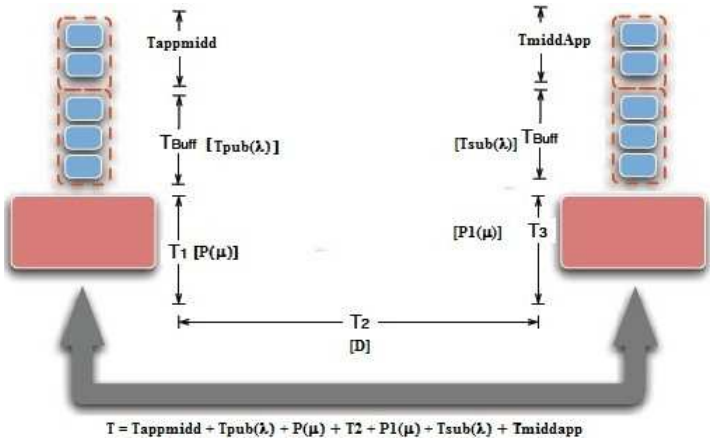


Figure 2: End-to-End Data Timeliness in DDS

- Pub/Sub notification cost per event message T_{am}: cost required by the application to provide event publish message or retrieve event subscribe message. This parameter is divided into two parts:(1) T_{appmid}: amount of time for even source application to provide the message to the middleware even broker system, and (2) T_{midapp}: amount of time required by application to retrieve message from the reader's cache to relay the messages to the displayer. Those parameters are experimentally evaluated using high performance time stamp included within the application source code.

- The pub/sub cost per event message $T_{ps}(\lambda)$ (which is the store and forward cost required for pub/sub messages). This parameter is divided into two parts: (1) $T_{pub}(\lambda)$, which is the store and forward cost for DDS to send data from the middleware to the network interface, and (2) $T_{sub}(\lambda)$, which is the cost to retrieve data after CPU processing at the subscriber's middleware. These parameters are evaluated using the Gilbert Model [7] (one of the most-commonly applied performance evaluation models), as shown in Figure 2.

- The effective processing time for pub/sub message for a given DDS message $P_{ps}(\mu)$ (which is the time cost required by processes executing on a CPU, possibly being preempted by the scheduler, until they have spent their entire service time on a CPU, at which point they leave the system). We assume that $P(\mu)$ has the same value on the publisher as on the subscriber and we note them $P(\mu)$ and $P1(\mu)$, respectively, as shown in Figure 2.

- The Network time delay D (which is the packet delivery time delay from the first bit leaves the network interface controller of the transmitter until the least is received). The network delay is measured using high-resolution timer synchronization based on NTP protocol [8]. This parameter is shown by T in Figure 2.

Analytical Model. Having defined the key scheduling activities along the pub/sub path, we need a mechanism to model these activities. If the CPU scheduler is limited by a single bottleneck node, job processing can be modeled in terms of a single queuing system. As shown in Figure 3, the DDS scheduler shown is a single queuing system that consist of three components: (1) an arrival process for messages from N different data writers with specific statistics, (2) a buffer that can hold up to K messages, which are received in first-in/first-out (FIFO) order, and (3) the output of the CPU (process complete) with a fixed rate f_s bits/s. We assume that discarded messages are not considered in this model, a message is ready for delivery to the network link when processing completes, and messages can have variable length, all of which apply

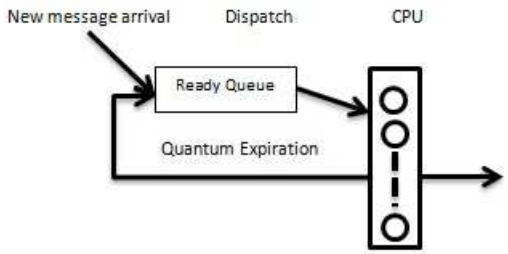


Figure 3: Single Processor Queuing System Model

for asynchronous data delivery in DDS.

Although these assumptions may not apply to all DRE systems, they enable us to derive specific behaviors via our performance model since jobs frequently arrive and depart (*i.e.*, are completed or terminated) at irregular intervals and have stochastic processing times, thereby allowing us to obtain the empirical results presented in Section 3.2.2. As mentioned above, our performance model is based on the *Gilbert Model* due to its elegance and the high-fidelity results it provides for practical applications [9]. This model simplifies the complexity of the schedulability problem by providing a first-order Markov chain model, shown in Figure 4.

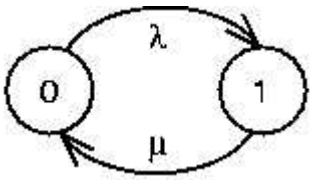


Figure 4: The Markov Model for Processor Scheduling

The Markov model shown in Figure 4 is characterized by two states of the system with random variables that change through time: State 0 ("waiting"), which means that data are being stored in the DDS middleware message queue, and State 1 ("processing"), which means that the job is being processed on the CPU scheduler. In addition, two independent parameters, P_{01} and P_{10}, represent state transition probabilities. The steady-state probabilities for the "waiting" and "processing" states are given, respectively, by 2, as follows:

$$\pi_0 = \frac{P_{10}}{P_{10} + P_{01}}; \;\; \pi_1 = \frac{P_{01}}{P_{10} + P_{01}} \tag{2}$$

Recall that P_{01} and P_{10} are the derived from the Markov transition matrix, which the general format is given by equation 3. As described in Figure 4, because we have an ergodic process, $P00 = 0$, $P01 = \lambda$, $P10 = \mu$, and $P11 = 0$, therefore we also can note that $\pi_0 = \frac{\lambda}{\lambda+\mu}$; $\pi_1 = \frac{\mu}{\lambda+\mu}$.

$$P = \begin{bmatrix} 0 & P01 \\ P10 & 0 \end{bmatrix} \tag{3}$$

From the expectation of the overall components described above, the overall time delay for the performance model is described by the following relation 4:

$$T = T_{appmid} + T_{pub}(\lambda) + P(\mu) + D + P1(\mu) + T_{sub}(\lambda) + T_{midapp} \tag{4}$$

Where, $\pi_0 = T_{pub}(\lambda) = T_{sub}(\lambda)$ and $\pi_1 = P(\mu) = P1(\mu)$.

$$\overline{T} = \frac{\overline{N}}{\lambda} = \frac{1}{\lambda} \times \frac{\rho}{1-\rho} \tag{5}$$

According to Little formula, $(T_{pub}(\lambda))$ can be written as $\frac{1}{\lambda}$, because the general form as is given by equation 5 and we consider the waiting for only one message per DDS topic, the number of messages in each Topic is $\overline{N} = 1$ ($\overline{T} = \frac{1}{\lambda}$) is considered for only one DDS topic including one message (with variable size).

Costs of Publish/Subscribe Network Model. The stochastic performance modeling approach described in Section 2.1.3 has lower complexity than a deterministic approach that strives to schedule processor time optimally for DDS-based applications. Since the service time for DDS messages is independent, identically distributed, and exponentially distributed, the scheduler can be modeled as a standard M/M/1 queuing system, such as the Little inter-arrival distribution [10].

We assume the time cost for communication between the application and the DDS middleware can be evaluated experimentally. In particular, the $T_{ps}(\lambda)$ cost can be evaluated using a stochastic Markov model. In this case, $T_{pub}(\lambda)$ is the store-and-forward cost for data writers to publish DDS messages to a CPU scheduler and $T_{sub}(\lambda)$ is the cost for the DDS middleware to retrieve these messages at the subscriber.

Network latency is comprised of propagation delay, node delay, and congestion delay. DDS-based end-systems also add processing delays, as described above.

We therefore assume the following network parameters

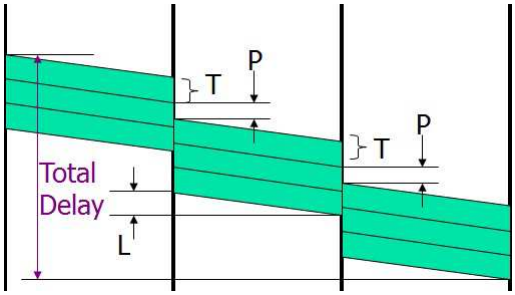


Figure 5: Timing Parameters in Datagram Packet Switching

shown in Figure 5 to analyze the network scheduling:

- M: number of hops
- P: Per-hop processing delay (s)
- L: link propagation delay (s)
- T: packet transmission delay (s)
- N: message size (packets)
- The total delay T_{tot} = total propagation + total transmission + total store and forward + total processing, as described relation by the following 6:

$$T_{tot} = M \times L + N \times T + (M-1) \times T + (M-1) \times P \quad (6)$$

The parameters described in relation 6 are used together with the delay parameters in the relation 4 to calculate the end-to-end delay. Our focus is on the delay elapsed from the time the first bit was sent by the network interface on the end-system to the time the last bit was received, which corresponds to the T_{tot} delay, as shown by T2 ([D]) in Figure 2.

Note that the performance analysis involves gathering formal and informal data to help define the behavior of a system. Its power does not reside in the complexity of the model, but in its power to express the system constraints. The model presented here allows expressing all of the constraints without adding complexity to the system. In more complicated models, one common solution is to look for a canonical form to reduce the complexity and hold the power of the expression of the model to permit powerful analysis techniques for validating the quality of service. The model presented in this paper is well suited for the context of LAN as well as WAN context and does not require any additional complexity because it can express the behavior of the system easily and allows powerful metrics to evaluate the performance of the system.

2.2. Architecture of the End-to-end Velox QoS Framework

Below we describe the architecture of the Velox QoS management framework, which enhances DDS to support QoS provisioning over WANs by enabling DRE systems to select an end-to-end QoS path that fulfills applications requirements. Requirements supported by Velox include per-flow traffic differentiation using DDS QoS policies, QoS signaling, and resource reservation over heterogeneous networks.

2.2.1. Context: Supporting DDS over WANs

Implementations of DDS have predominantly being deployed in local area network (LAN) environments. As more DRE systems become geographically distributed, however, it has become necessary for DDS to operate over wide area networks (WANs) consisting of multiple autonomous systems that must be traversed by published messages. In turn, the WAN topologies imply that DDS traffic must be routed over core network routers in addition edge routers, as well as support multiple different type of network technologies and links with different capacities.

Integrated Services (IntServ) [11] are viable in small- to medium-size LANs, but have scalability problems in large-scale WANs. Differentiated Services (DiffServ) [12] provide diverse service levels for flows having different priorities requiring lower delays under variable bandwidth. Moreover, various network technologies composing an end-to-end path have different capabilities in terms of bandwidth, delay, and forwarding capabilities, which makes it hard to apply a single unified solution for all network technologies.

Any technique for assuring end-to-end QoS for DDS-based DRE systems must optimize the performance and scalability of WAN deployments over fixed and wireless access technologies and provide network-centric QoS

provisioning. It is therefore necessary to reserve network resources that will satisfy DRE system requirements. Likewise, traffic profiles must be defined for each application within a DRE system to ensure they never exceed the service specification while ensuring their end-to-end QoS needs are met.

2.2.2. Problem: Dealing with Multiple Systemic Issues to Support DDS in WANs

Challenge 1: Heterogeneity across WANs. To operate over WANs—and support end-to-end QoS—DDS applications must be able to control network resources in WANs. DDS implementations must therefore shield application developers from the complexity of communication mechanisms in the underlying network(s). This complexity is amplified due to different network technologies (*e.g.*, wired and wireless) that comprise the WANs.

Each technology exposes different QoS management mechanisms for which QoS allocation is performed differently; their complexity depends on resource reservation mechanisms for the underlying network technology (*e.g.*, Ethernet, Wimax, WiFI, Satellite, etc.). DDS application developers need an approach that encapsulates the details of the underlying mechanisms. Likewise, they need a uniform abstraction to manage complexity and ensure DDS messages can be exchanged by publishers to subscribers with desired QoS properties.

Challenge 2: Signaling and Service Negotiation Requirements. Even if there is a uniform abstraction that encapsulates heterogeneity in the underlying network elements (*e.g.*, links and routers), when QoS mechanisms must be realized within the network the underlying network elements require specific signaling and service negotiations to provision the desired QoS for the applications. It is therefore important that any abstraction DDS provides to application developers also provides the appropriate hooks needed for signaling and service negotiations.

Challenge 3: Need for Admission Control. Signaling and service negotiation alone is insufficient. In particular, care must be taken to ensure that data rates/sizes do not overwhelm the network capacity. Otherwise, applications will not achieve their desired QoS properties, despite the underlying QoS-related resource reservations. A call setup phase is therefore useful to prevent oversubscription of user flow, protect traffic from the negative effects of other competent traffic, and ensure there is sufficient bandwidth for authorized flows.

Challenge 4: Satisfying Security Requirements. Admission control cannot be done for all transmitted traffic, which means that user traffic must be identified and allowed to access some restricted service. Only users that have registered for the service are allowed to use it (Authentication). Moreover, available resources may be overprovisioned due to their utilization by unauthorized users that are not granted to require and receive a specific service (Authorization). Even if a particular authenticated user should have to secured resources controlled by the system, the system should be able to verify the correct user is charged for the correct session, according to the resources reserved and delivered (Accounting).

2.2.3. Solution Approach: A Layer 3 QoS Management Middleware

Figure 6 shows Velox, which provides an end-to-end path for delivering QoS assurance across heterogeneous autonomous systems build using DDS at the network layer (which handles network routing and addressing issues in layer 3 of the OSI reference model). Each path corresponds to a given set of QoS parameters—called *classes of services*—controlled by different service providers. The Velox framework is designed as session service platform over DiffServ-based network infrastructure, as shown in Figure 6. The remainder of this section explains how Velox is designed to address the challenges described in Section 2.2.2.

Resolving Challenge 1: Masking the Heterogeneity via MPLS tunnels. Challenge 1 in Section 2.2.2 stemmed from complex QoS management across WANs due to heterogeneity across network links and their associated QoS mechanisms. Ideally, this complexity can be managed if there exists a uniform abstraction of the end-to-end path, which includes the WAN links. Figure 7 depicts how Velox implements an end-to-end path abstraction using a *Multi Protocol Label Switching* (MPLS) tunnel [13]. This tunnel enables aggregating and merging different autonomous systems from one network domain (AS1 in Figure 7) to another (AS5 in Figure 7), so that data crosses core domains more transparently.

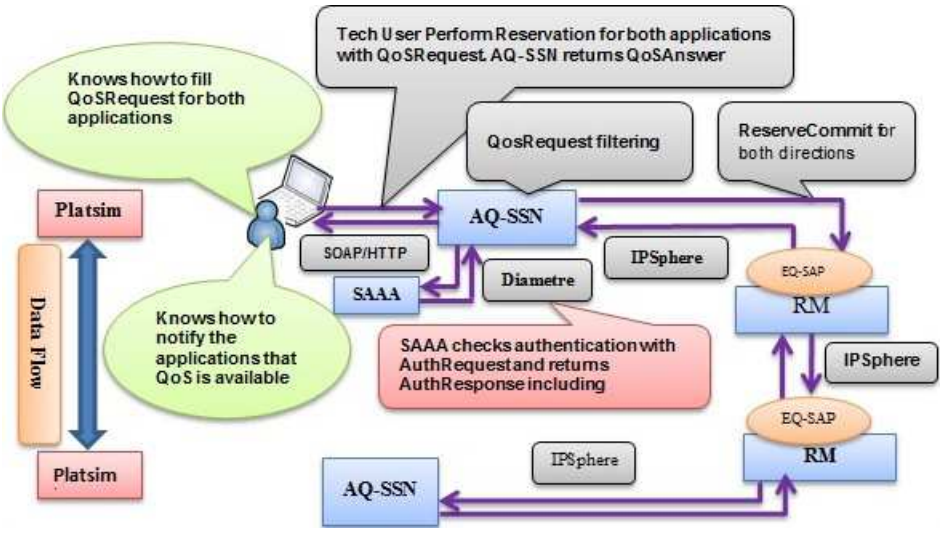


Figure 6: Velox Framework Components

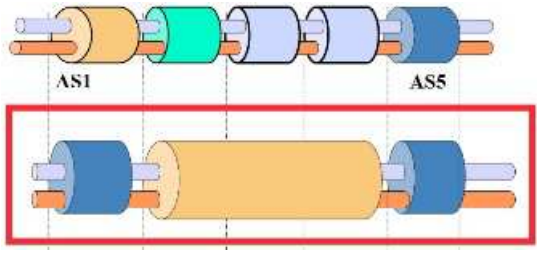


Figure 7: End-to-end path with MPLS tunnel

To ensure the continuity of the per-hop behavior along a path, *Network Layer Reachability Information* (NLRI) [14] is exchanged between routers using an NLRI field to convey information related to QoS. The Velox computation algorithm determines a path based on QoS.

Resolving Challenge 2: Velox Signaling and Service Negotiation. Challenge 2 in Section 2.2.2 is resolved using the Velox *Signaling and Service Negotiation* (SSN) capability. After an end-to-end path (tunnel) is established, the Velox SSN enables the sending of a QoS request from the service plane using a web interface to the first resource manager via a service-level agreement during session establishment. This resource manager performs QoS commitment and checks if there is a suitable end-to-end path fulfilling the QoS requirements in terms of classes of services.

The Velox SSN function coordinates the use of the various signaling mechanisms (such as end-to-end, hop-by-hop, and local) to establish QoS-enabled end-to-end sessions between communicating DDS applications. To ensure end-to-end QoS, we decompose the full multi-domain QoS check into a set of consecutive QoS checks, as shown in Figure 6. The QoS path on which the global behavior will be based therefore establishes the transfer between the remote entities involved, which must be controlled to ensure end-to-end QoS properties.

Figure 8 shows the architecture for the caller application trying to establish a signaling session. The caller sends a "QoSRequest" (which includes the required bandwidth, the class of service, the delay, etc.) to the SSN, as shown in Figure 8. In turn, the callee application uses the *establishSession* service exposed by the web service interface. The following components make up the Velox SSN capability:

- *AQ-SSN (Application QoS)* allows callers to contact the callee side and negotiate the session parameters.
- *Resource Manager (RM)* handles QoS requests solicited by the control plane and synchronizes those requests with the service plane for handshaking QoS

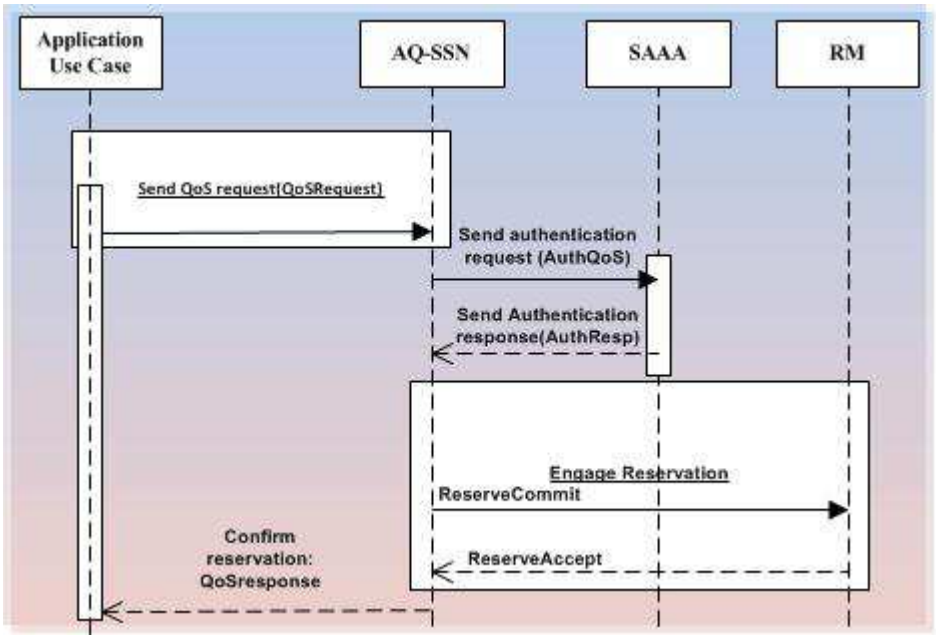


Figure 8: Velox Signaling Model

invocation among domains using the IPsphere *Service Structuring Stratum* (SSS) [1] signaling bus with the *Next Steps in Signaling* (NSIS) [15] protocol to establish, invoke, and assure network services.

After the QoSRequest has been performed, the *performReservation* service exposed by AQ-SSN attempts to reserve network resources. AQ-SSN requests network QoS using the EQ-SAP (Service Access Point) interface on top of the resource manager. After QoS reservation has completed at the network level, the response will be notified to AQ-SSN, which returns a QoSAnswer to the caller. Since there is one *reserveCommit* request for each unidirectional flow, if the *reserveCommit* operation fails, the AQ-SSN must trigger the STOP request for the rest of the flows belonging to the same session that were reserved previously.

Resolving Challenge 3: Velox Call Admission Control and Resources Provisioning. Challenge 3 in Section 2.2.2 is addressed by the Velox *Connection Admission Control* (CAC) capability. The CAC functionality is split into

- A domain CAC that manages the admission in each domain, and is called accordingly as the Inter-domain CAC, Intra-domain CAC, and Database CAC.
- An End-to-end CAC that determines a path with a specified QoS level.

When the resource manager receives the *reserveCommit* request from AQ-SSN it checks whether the source IP address of the flow belongs to its domain. The AQ-SSN then performs resource reservations for the new submitted call to the system in either a hop-by-hop manner or a single-hop related to a domain, as shown in the control plane in Figure 9. During the setup phase of a new call, therefore, the associated QoS request will be sent via the signaling system to each domain (more precisely to each resource manager) being on the path from source to destination. Not all requests will be serviced due to network overload. To solve the resulting problems, the end-to-end

[1] `http://www.tmforum.org/ipsphere`

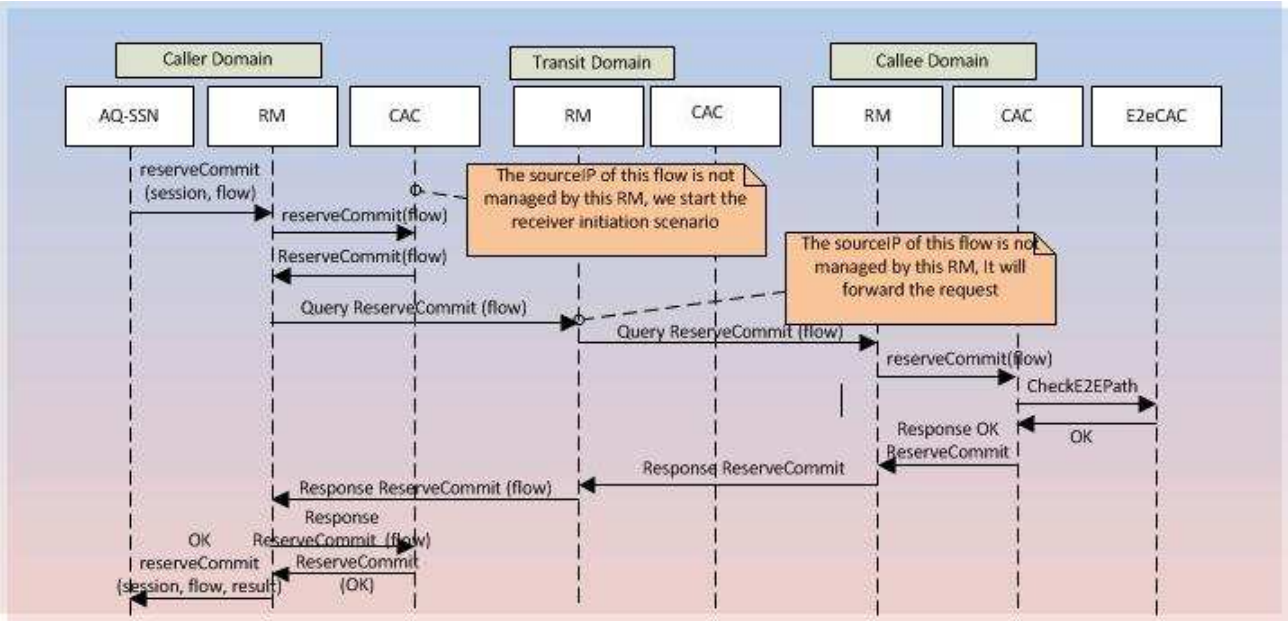


Figure 9: Velox Resource Reservation Model

Velox *connection admission control* (CAC) capability is used for intra-domain, inter-domain, and end-to-end path.

For the intra-domain CAC, the existence of a QoS path internal to the domain (*i.e.*, between the ingress router and the egress router) is then checked by the Velox resource manager. If the QoS parameters are fulfilled, the intra-domain call is accepted, otherwise it is rejected. For the intra-domain CAC, the resource manager checks whether the QoS requirements in the inter-domain link (between the two BR routers of two different autonomous systems) can be fulfilled. If the link can accept the requested QoS, the call is accepted, otherwise it is rejected. For the end-to-end CAC, Velox first checks the existence of the end-to-end path via the *Border Gateway Protocol* table. If this check does not find an acceptable QoS path, the CAC result is negative.

Finally, if the three CACs accept the call, the first resource manager forwards the call to the subsequent resource manager in the next domain. This manager is deduced from the information given when the first resource manager selects the appropriate path. The network resources of each domain are fully available by each call passing the domain. As a result, no *a priori* resource reservations are required. To reserve the resources for a new call, therefore, Velox needs to reserve the resources inside the MPLS end-to-end tunnel and need not perform per-flow resource reservations in transit domains.

Resolving Challenge 4: Security, Authentication, Authorization, and Accounting. Challenge 4 in Section 2.2.2 is addressed by the Velox *Security, Authentication, Authorization, and Accounting* (SAAA) capability. Velox's SAAA manages user access to network resources (authentication), grant services and QoS levels to requesting users (authorization), and collects accounting data (accounting). AQ-SSN then checks user authentication and authorization using SAAA and will optionally filter some QoSRequests according to user rights via the Diameter protocol [16], which is an authentication, authorization, and accounting (AAA) protocol for computer networks.

The Velox SSN module coordinates the session among end-users. The SSN module asks CAC whether or not the network can provide enough resources to the requesting application. It manages the session data, while CAC stores the session status, and it links events to the relevant

session, translating network events (faults, resource shortage, etc) into session events. The Velox SSN notifies its CAC of user authorizations, after having authenticated the user with AAA. The SSN is also responsible for shutting down the session if faults have occurred. These CAC decisions are supported by knowledge of the network configuration and the current monitoring measurements and fault status.

3. Analysis of Experimental Results

This section presents experimental results that evaluate the Velox framework in terms of its timing behavior, overhead, and end-to-end latencies observed in different scenarios. We first use simulations to evaluate how the performance model described in Section 2.1 predicts end-system delays and then compare these simulation results with those obtained in an experimental testbed. We also evaluate the impact of increasing the number of topics on DDS middleware latency and then evaluate the client-perceived latency with the increasing size of topic data, where the number of topics is fixed. We next evaluate the latency incurred when increasing the number of subscribers involved in communication and compare the results with the empirical study. Finally, we demonstrate how the network QoS provisioning capabilities provided by the Velox framework described in Section 2.2 significantly reduce end-to-end delay and protect end-to-end application flows.

3.1. Hardware and Software Testbed and Configuration Scenario

The performance evaluations reported in this paper were conducted in the Laasnetexp testbed shown in Figure 10. Laasnetexp consists of a server and 38 dual-core machines that can be configured to run different operating systems, such as various versions of Windows and Linux [17]. Each machine has four network interfaces per machine using multiple transport protocols with varying numbers of senders, receivers and 500 GB disks. The testbed also contains four Cisco Catalyst 4948-10G switches with 24 10/100/1000 MPS ports per switch and three Juniper M7i edge routers connected to the RENATER network [2].

To serve the needs for the emulations and real network experiments, two networks have been created in Laasnetexp: a three-domain real network (suited for multi-domain experiments) with public IP addresses belonging to three different networks, as well as an emulation network. Our evaluations used DiffServ QoS, where the QoS server was hosted on the Velox blade.

In our evaluation scenario, a number of real-time sensors and actuators sent their monitored data to each other so that appropriate control actions are performed by the military training and Airbus Flight Simulators we used. Figure 10 shows several simulators deployed on EuQoS5-EuQoS8 blades communicating based on the RTI DDS middleware implementation [3]. To emulate network traffic behavior, we used a traffic generator that sends UDP traffic over the three domains with configurable bandwidth consumption. To differentiate the traffic at the edge router, the Velox framework described in Section 2.2 manages both QoS reservations and the end-to-end signaling path between endpoints.

3.2. Validating the Performance Scheduling Model

Section 2.1 described an analytical performance model for the range of scheduling activities along the end-to-end critical path traced by a DDS pub/sub flow. We now validate this model by first conducting a performance evaluation using real conditions and estimating the time delays in the analytical performance model. We then compare these simulation results with actual experimental results conducted in the testbed described in Section 3.1. The accuracy of our performance model is evaluated by the degree of similarity of these results.

We apply the approach above because some parameters in our analytical formulation are only observable and not controllable (*i.e.*, measurable). To obtain the values for these observable parameters so they can be substituted into the analytical model, we conducted simulation/-emulation studies. These studies estimated the values by measuring the time taken from when a request was submitted to the DDS middleware by a publisher applica-

[2] `http:www.renater.fr`
[3] `www.rti.com`

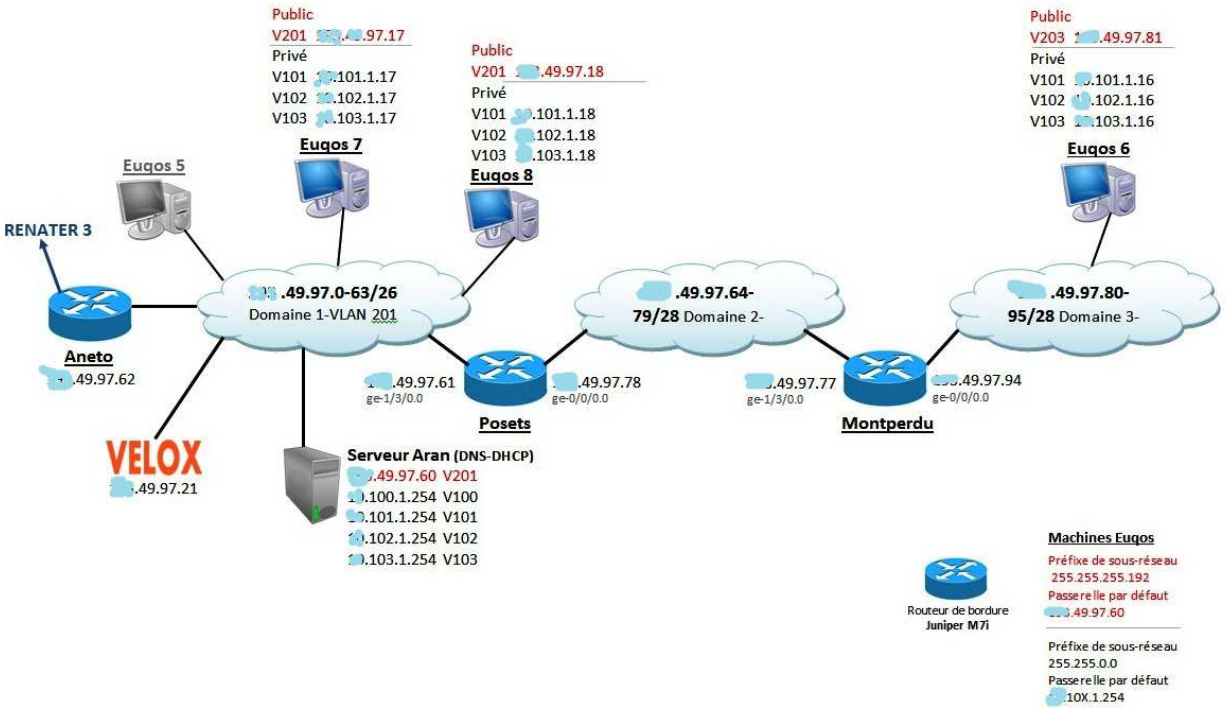


Figure 10: Laasnetexp testbed

tion calling a "*write()*" data writer method until the time the subscriber application retrieves data by invoking the "*read()*" data reader method. We first analyze the results and then validate the analytical model as a whole.

3.2.1. Estimating the Publish and Subscribe Activity at the Middleware-Application Interface in the Pub/Sub Model

Rationale and approach. One component of our performance model (see Equation 4 in Section 2.1.3) includes the event notification time T_{am}. This time measures how long an application takes to provide the published event to the middleware (called T_{appmid}) or the time taken to retrieve a subscribed event from the middleware (called T_{midapp}). We estimate these modeled parameters by comparing the overall time using our analytical model with the empirically measured end-to-end delay encountered in the LAN environment shown by VLAN "V101" in Figure 10 and described in Section 3.1. Since the LAN environment provides deterministic and stable results, the impact of the network can easily be separated from the results. We can therefore pinpoint the empirical results for the delays in the end-systems and compare them with the analytically determined bounds.

We implemented a high accuracy time stamp function in the application using the Windows high-resolution method `QueryPerformanceCounter()` to measure the time delay required by the application to disseminate topic data to the middleware event broker system. The publisher application writes five topics using the reliable DDS QoS setting, where each topic data size ranges between 20 and 200 bytes and the receiver subscribes to all topics. Increasing the number of topics and their respective data sizes enables us to analyze their impact on end-to-end latency in the performance model. The Reliability QoS policy configures the level of reliability DDS uses to communicate between a data reader and data writer.

Results and analysis. Figure 11 shows the time delay measured at both the publisher and subscriber applications, respectively, T_{appmid} and T_{midapp}. As shown in Figure 11 (note the different time scales for the publisher and subscriber sides), the time required by the application to retrieve topics from the DDS middleware broker is larger than the time required to publish the five topics. The subscriber application takes ~50μs to retrieve data by invoking the "*read()*" data reader method and displaying the results on the user interface. Likewise, publisher application takes ~ 1μs to transmit the request to the DDS middleware broker by a invoking a "*write()*" data writer method.

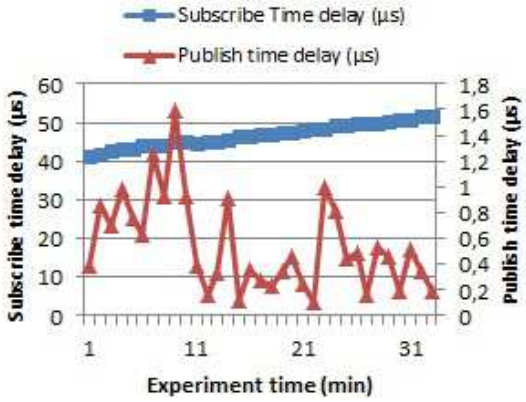


Figure 11: Time Delay for the Publish/Subscribe Event

The DDS Reliability QoS policy has a subtle effect on data reader caches because data readers add samples and instances to their data cache as they are received. We therefore conclude that the time required to retrieve topic data from the data reader caches contributes to the majority of time delay observed by a subscriber. Figure 12a further analyzes the impact of number of topics on the time delay for a subscriber event. This figure shows the cumulative time delay required to push up all six samples of topic data from the DDS middleware to the application we called T_{midapp} in the previous section (the experiment was conducted for a 30 minute duration).

As shown in Figure 12a, T_{midapp} is linearly proportional to number of topics. For example, the amount of time required by the application to retrieve a message from the reader's cache to relay the events to the display console for only a single topic remains close to 9μs for all samples.

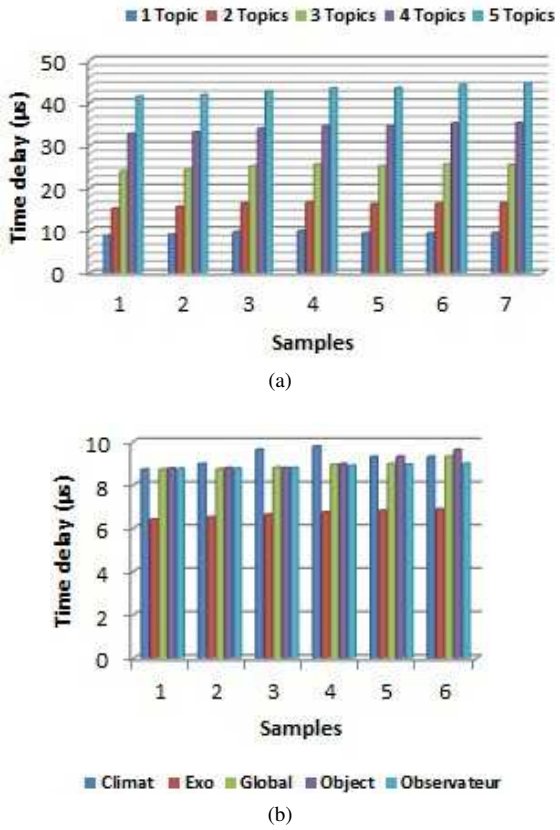


Figure 12: Impact of the number of DDS Topics on the Time Delay for publish/subscribe event

When the number of topics increases, T_{midapp} increases, respectively, *e.g.*, for 2 topics T_{midapp} = 15 μs, for 3 topics T_{midapp} = 24μs, for 4 topics T_{midapp} = 34 μs, and for 5 topics T_{midapp} = 42μs.

A question stemming from these results is what is the impact of the data size on $T_{appmidd}$ and T_{midapp}? To answer this question, we analyze Figure 12b, which shows the T_{midapp} for each topic. To retrieve topic "Climat" (which is 200 bytes in size) the required T_{midapp} is close to 9μs for all samples (and also for all experiments). Likewise, to retrieve topic "Exo" (which is 20 bytes in size) the required T_{midapp} remains close to 6μs. Finally, the T_{midapp} for topic "Object" (which is 300 bytes in size) remains close to 9μs. These results reveal that the size of the data has little impact on T_{midapp}.

Figure 13 shows the time delay required by the publish application to send every topic to the DDS middleware. Indeed, to push "Climat"Topic into the DDS mid-

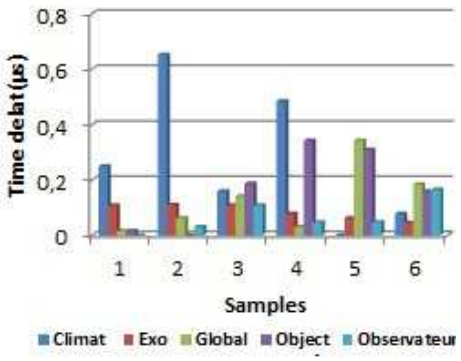


Figure 13: the Time Delay for publish event

dleware the required $T_{appmidd}$ is between0.2μs and 0.6μs. The $T_{appmidd}$ of the "Exo" Topic is close to 0.1μs, Topic "Object" has $T_{appmidd}$ value between to 0.1μs and 0.4μs, and the "Global" and "Observateur" Topics have $T_{appmidd}$ smaller then 0.4μs and 0.2μs, respectively. These results reinforce those provided by Figure 11; we therefore conclude that most of the time between the application and the DDS middleware is spent on the subscriber and not on the publisher.

To summarize, the pub/sub notification time-per-event (which corresponds to the cost required by the application to provide event publish message or retrieve event subscribe message) depends largely on the number of topic data exchanged between remote participants. The time-per-event is relatively independent of the size of each topic instance. Moreover, the time delay required by an application to retrieve a message from the reader's cache to relay the events to the display console T_{midapp} is greater than T_{appmid}, which is the time for publisher application to provide the message to the DDS middleware.

3.2.2. Estimating the CPU Scheduling Activities in the Analytical Model

Rationale and approach. To evaluate the scheduling model, we refer to Figure 14 that describes the CPU scheduling. During the experimentation, the traffic intensity per CPU refers to the utilization rate of the processor as the ratio $\rho = \frac{\lambda}{\mu}$, which is on average equal to 0.1

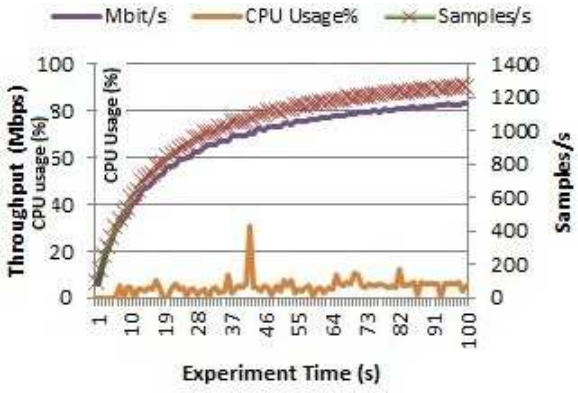


Figure 14: Impact of increasing the Topic samples on the utilization rate of the CPU

(10% in Figure 14), which illustrate that the service rate of the CPU remains constant when the topic samples increases during the experiments. That is, The pub/sub cost per event $T_{ps}(\lambda)$ for the DDS middleware is the store-and-forward cost required for an event publish and subscribe message. It remains undefined, however, both at the publisher ($T_{pub}(\lambda)$) and the subscriber ($T_{sub}(\lambda)$). These parameters were therefore empirically evaluated using the Gilbert model described in Section 2.1.3.

Results and analysis. The data collected from the trace files shows that the DDS middleware sends data at publish rate λ equal to 12,000 packets per second (pps). The average inter-arrival time $\frac{1}{\lambda}$ to the CPU is equal to 83.3μs. Moreover, using the utilization rate of the processor, the average service time $\frac{1}{\mu}$ is equal to 8μs.

When an event is generated, it is assigned a timestamp and is stored in the DDS store-and-forward queue. Processes enter this queue and wait for their turn on a CPU for an average delay of 83.3μs. They run on a CPU until they have spent their service time, at which point they leave the system and are routed to the network interface (NIC interface). A process is selected from the front of the queue when a CPU becomes available. A process executes for a set number of clock cycles equivalent to the service time of 8μs.

From the above discussion, the average arrival time $\frac{1}{\lambda}$ is ten times greater than the average service time $\frac{1}{\mu}$. Processes spend most of their time waiting for CPU avail-

ability. Referring to the relation 2 in Section 2.1.3, the steady-state probabilities for the "waiting" and "processing" states are 0.9 and 0.1, respectively.

3.2.3. Estimating the Network Time Delay in the Analytical Model

Rationale and approach. To evaluate end-to-end network latency and determine each of its components discussed above (*i.e.*, the DDS pub/sub notification time per event T_{am} and DDS pub/sub cost-per-event T_{ps}), we empirically evaluate both the transmission delay and propagation delay. We are interested only in the delay "D" elapsed from the time the first bit was sent to the time the last bit was received (*i.e.*, we exclude the time involved in T_{am} and T_{ps}).

Results and analysis. Table 1 shows the different parameters and their respective values used to evaluate the network delay empirically. This model emulates the behavior of two remote participants in the same Ethernet LAN. In this configuration, the average time delay "D" is 171.84μs.

Table 1: Empirical Evaluation of Network Time Delay

Parameters	Value
M: number of hops	2
P: Per-hop processing delay (μs)	5
L: link propagation delay (μs)	0.5
T: packet transmission delay (μs)	82.92
N: message size (packets)	1
Pkt: Packet size	8192 bits
D: Total delay (μs)	171.84

3.2.4. Comparing the Analytical Performance Model with Experimental Results

Rationale and approach. We now compare our analytical performance model (Section 2.1) with the results obtained from experiments in our testbed (Section 3.1). We first calculate the end-to-end delay "ED" provided by the performance model and given by relation 4 in Section 2.1.3, by summing the DDS pub/sub notification time per event T_{am}, the DDS pub/sub cost-per-event $T_{ps}(\lambda)$, the effective processing time per DDS pub/sub message $P_{ps}(\mu)$, and the average time delay "D". We then compare "ED" with empirical experiments shown in Figure 15, which indicate the time required to publish topic data until they are displayed at the subscriber application.

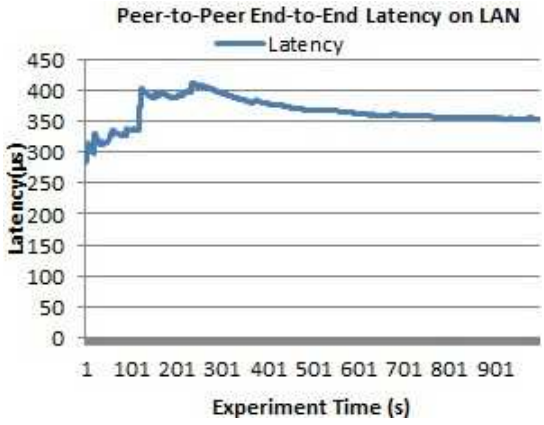


Figure 15: Experimental end-to-end latency for Pub/Sub events over LAN

Results and analysis. The experimental results in Figure 15 show that the end-to-end delay is ~350μs. In addition, the results provided by our performance model described in Table 2 are consistent with those provided by the experiments, *i.e.*, the end-to-end latency provided by the performance model is 306.74μs. We believe the values are acceptable because rather than taking into account the percentage (14%), the 44 microseconds is not noticeable because it is due to hardware ASIC processing at the network physical node. These evaluations show that the results obtained from the analytical model are similar to those obtained using empirical measurements, which demonstrates the effectiveness of our performance model

Table 2: Evaluation of the End-to-End Delay (ED)

Parameters	Value
$T_{midapp}(\mu s)$	42
$T_{appmid}(\mu s)$	1.6
$T_{pub}(\lambda) + T_{sub}(\lambda) = \frac{1}{\lambda}(\mu s)$	83.3
$P(\mu) + P1(\mu) = \frac{1}{\mu}(\mu s)$	8
D (μs)	171.84
ED (μs)	306.74

to estimate the different time delay components described above. The slight discrepancy between those results stems from the simplified assumptions made with the first-order Markov model, which is not completely accurate. We believe the slight discrepancy is acceptable because rather than taking into account the percentage difference (14%) which may appear large, the 44 microseconds is not noticeable because it is due to hardware ASIC processing at the network physical node.

3.2.5. Impact of Increase in Number of Subscribers

Rationale and approach. We conducted experiments with a large number of clients and measured the communication cost by varying the number of clients. We leverage and compared our experimental results of the end-to-end latency delay with the empirical study found in [18], where the authors suggested a function $S(n)$ to evaluate the effect of distributing messages for several subscribers.

The experiments were conducted by increasing the number of subscribers, so we used only one publisher that sent data to respectively 1, 2, 4, and 8 subscribers and plot the end-to-end delay taken from trace files, as shown in Figure 16. The results in this figure show that the latency

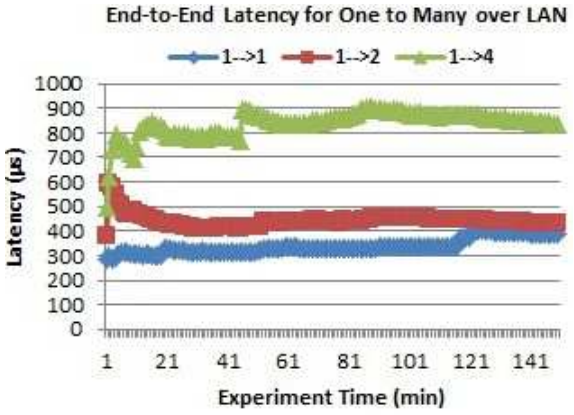


Figure 16: End-to-end latency for one publisher to many subscribers

for one-to-one communication (single publisher sending topic data to a single consumer) is ~400μs.

Results and analysis. As the number of subscribers increased, the moving average delay (the time from sending a topic from the application layer to its display on the subscriber) increased proportionally with the respect to the number of subscribers. The moving average delay remained ~600μs for two subscribers, became ~900μs for 4 subscribers, and remained ~1400μs when the number of subscribers was 8.

Our results confirm the results provided in [18], where the moving average delay is proportionally affected by the number of clients declaring their intention to receive data from the same data space. The publisher can deliver events with low cost when it broadcasts events to many subscribers with an impact factor between $\frac{1}{n}$ and 1.

In summary, when using DDS as a networking scheduler, the required time delay to distribute topic data is determined at least by the number of topics and the number of readers. In the case of the number of topics, our experiments described above showed that the time delay for sending data from the application to the DDS middleware increases with the number of topics. Those experiments have been conducted for different DDS middleware vendor implementations including RTI DDS [4], OpenSplice DDS [5] and CoreDX DDS [6].

Based on these results, we recommend sending larger data size packets with fewer topics instead of using a large number of topics. DDS middleware defines the `get_matched_subscriptions()` method to retrieve the list of data readers that have a matching topic and compatible QoS associated with the data writers. Having a greater number of topics, however, allows dissemination of information with finer granularity to select set of subscribers. Likewise, reducing the number of topics by combining their types results in more coarse-grained dissemination with a larger set of subscribers receiving unnecessary information. Application developers must therefore make the right tradeoffs based on their requirements.

3.3. Evaluation of the Velox Framework

Below we present the results of experiments conducted to evaluate the performance of the Velox framework described in Section 2.2. These results evaluate the Velox premium service, which uses the DiffServ expedited forwarding per-hop-behavior (PHB) model [19] whose char-

[4] `www.rti.com/products/dds`
[5] `www.prismtech.com/opensplice`
[6] `www.twinoakscomputing.com/coredx`

acteristics of low delay, low loss, and low jitter are suitable for voice, video, and other real-time services. Our future work will evaluate the assured forwarding PHB model [20] [21] that operators can use to provide assurance of delivery as long as the traffic does not exceed some subscribed rate.

3.3.1. Configuration of the Velox Framework

To differentiate the traffic at the edge router, the Velox server manages both QoS reservations and the end-to-end signaling path between endpoints.[7] Velox can manage network resources in a single domain and multi-domain network. In a multi-domain network, Velox behaves in point-to-point fashion and allows users to buy, sell, and deploy services with different QoS (*e.g.*, expedited forwarding vs. assured forwarding) between different domains. Velox can be configured using two types of services: the network service and the session service, as shown in Figure 17 and described below:

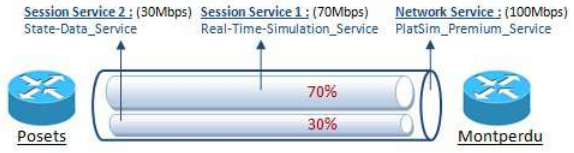


Figure 17: Resource Reservation Inside the MPLS End-to-End Tunnel

- *Network services* define end-to-end paths that include one or more edge routers. When the network session is created, the overall bandwidth utilization for different sessions are assigned to create communication channels that allow multiple network sessions to use this bandwidth. Moreover, it is possible to create several network sessions, each one having its bandwidth requirements among the end-to-end paths.

- *Session services* refer to a type of DiffServ service included within the network session. Service sessions create end-to-end tunnels associated with specific QoS parameters (including the bandwidth, the latency, and the class of service) to allow different applications to communicate with respect to those parameters. For example, bandwidth may be assigned to each session (shown in Figure 17) and allocated by the network service. Velox can therefore call each service using its internal "Trigger" service described next.

- *Trigger service* initiates a reservation of bandwidth available for each session of a service, as shown in Figure 18. When the network service and session services are ready for use, the trigger service propagates the QoS parameters among the end-to-end paths that join different domains.

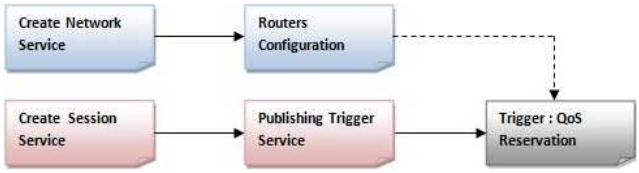


Figure 18: Trigger Service QoS Configuration

3.3.2. Evaluating the QoS Manager's QoS Provisioning Capabilities

Rationale and approach. The application is composed of various data flows. Each flow has its own specific characteristics, so we need to group them into categories (or media), taking into account the nature of the data (homogeneity) as described in Figure 19. Then, we analyze those application's flows to define and specify their network QoS constraints to enhance the interaction between the application layer, the middleware layer and the network layer. Therefore, We associate a set of middleware QoS policies (History, Durability, Reliability, Transport-priority, Latency-budget, Time-based-filter, Deadline, etc.) by media to classify them into 3 traffic classes, each class of traffic has its specific DDS QoS policies, then map them to specific IP services.

The application used for our experiments is composed of three different DDS topics. Table 3 shows how topics with different DDS QoS parameters allow data trans-

[7] Performance evaluation of the functions of Velox is not presented in this paper because we address the impact (from the point of view the network QoS) of mapping the DDS QoS policies to the network (routing and QoS) layer with the help of the MPLS tunneling.

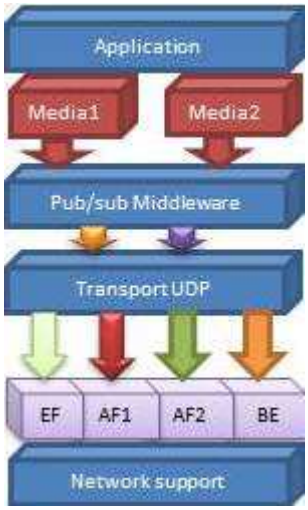


Figure 19: Mapping the application flow requirements to the network through the DDS middleware

Topic Data	Requirements	QoS DDS	DSCP Field
Continuous Data	Constantly updating data	best-effort	12
	Many-to-many delivery	keys, multicast	
	Sensor data, last value is best	keep-last	
	Seamless failover	ownership, deadline	
State Information	Occasionally changing persistent data	durability	34
	Recipients need latest and greatest	history	
Alarms & Events	Asynchronous messages	liveliness	46
	Need confirmation of delivery	reliability	

Table 3: Using DDS QoS for End-Point Application Management

fer with different requirements. As shown in the table, continuous data is sent immediately using best-effort reliability settings and written synchronously in the context of the user thread. The data writer will therefore send a sample every time the *write()* method is called. State information should deliver only previously published data samples (the most recent value) to new entities that join the network later.

Asynchronous data are used to send alarms and events asynchronously in the context of a separate thread internal to the DDS middleware using a flow controller. This controller shapes the network traffic to limit the maximum data rates at which the publisher sends data to a data writer. The flow controller buffers any excess data and only sends it when the send rate drops below the maximum rate. When data is written in bursts—or when sending large data types as multiple fragments—a flow controller can throttle the send rate of the asynchronous publishing thread to avoid flooding the network. Asynchronously written samples for the same destination is coalesced into a single network packet, thereby reducing bandwidth consumption.

Figure 20 describes the overall architecture for mapping the application requirements to network through the middleware: the DDS QoS policies provided by the middleware to the network (Transport-priority, latency-budget, deadline) are parsed from an XML configuration files. The Transport-priority QoS policy is processed by the application layer at the terminal nodes according to the value of this QoS policy, then translated by the middleware to IP packet DSCP marking; the Latency-budget is considered very roughly at the terminal nodes, only; and the "Deadline QoS policy" allows adapting the production profile to the subscriber request.

This solution improves the effectiveness of our approach to enhance the interaction between the application and the middleware and the network layer. The data produced using the local DDS service must be communicated to the remote DDS service and vice versa. The networking service provides a bridge between the local DDS service and a network interface. The application must dimension the network properly, *e.g.*, a DDS client performs a lookup and assigns a QoS label to the packet to identify all QoS actions performed on the packet and from which

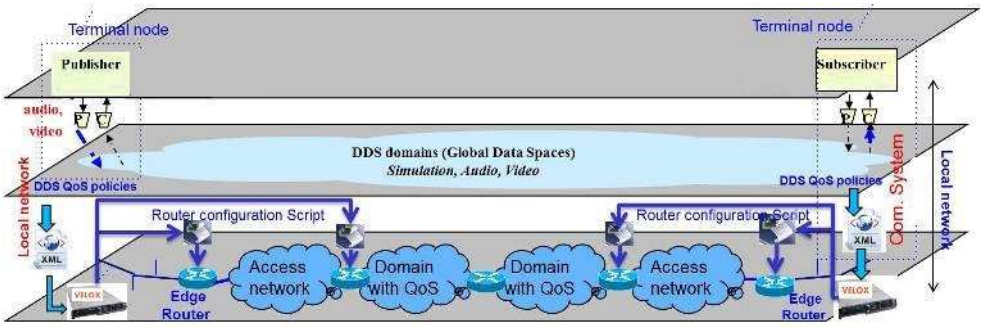


Figure 20: QoS Guaranteed Architecture

queue the packet is sent. The QoS label is based on the DSCP value in the packet and decides the queuing and scheduling actions to perform on the packet.

An edge router selects a packet in a traffic stream based on the content of DSCP packet header (described in column 3 in Table 3) to check if the traffic falls within the negotiated profile. If it does, the packet is marked to a particular DiffServ behavior aggregate. The application then uses the DDS transport_priority policy to define the aggregated traffic the domain can handle separately. Each packet is marked according to the designated service level agreement (SLA).

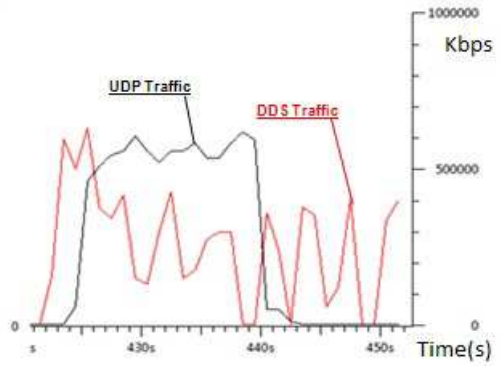
Since Velox supports QoS-sensitive traffic reliably to support delay- and jitter-sensitive applications, QoS requirements for a flow can be translated into the appropriate bandwidth requirements. To ensure queuing delay and jitter guarantees, it may be necessary to ensure that the bandwidth available to a flow is higher than the actual data transmission rate of this flow. We therefore identified two flows and used them to evaluate the impact of Velox on the bandwidth protection as follows: (1) a real-time traffic generated by the application using expedited forwarding DiffServ service with priority level 46 and (2) UDP best-effort traffic using Jperf traffic generator (`iperf.sourceforge.net`).

We performed two variants of this experiment. The first variant uses UDP network background load of forward and reverse bandwidth. For this configuration, the Velox resource manager does not provide any QoS management for the large-scale network, as the default configuration of routers uses only two queues with 95% for best-effort packets and 5% for network control packets, *i.e.*, all traffic traversing the network goes through a single best-effort queue. Subsequently, we begin sending a DDS flow at 500 Kbps followed by a UDP flow at 600 Kbps injected from Jperf to congest the queue and observe the behavior of the DDS flow.

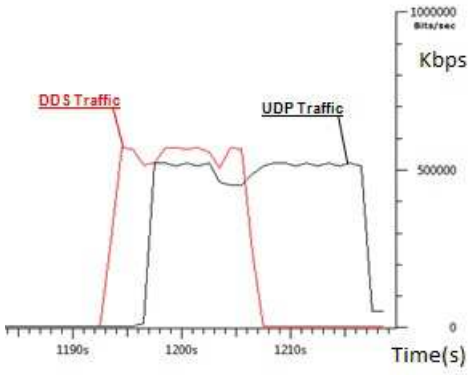
The second variant also used the UDP perturbing traffic, but we enabled Velox for QoS management. The Velox resource manager configured the edge router queues to support 40% best-effort traffic, 30% expedited forwarding traffic and 20% assured forwarding traffic, and 5% for network control packets.

Results and analysis. Figure 21a shows the results of experiments when deployed applications were (1) configured without any network QoS class and (2) sending DDS flow competing with UDP background traffic. These results show the deterioration of the flow behavior as it cannot maintain a constant bandwidth expected by the DDS application due to the disruption by the UDP background flows.

Figure 21b shows the results of experiments when the deployed applications were (1) configured with expedited forwarding network QoS class and (2) sending DDS flows competing with UDP background traffic. These results show that irrespective of heavy background traffic, the bandwidth experienced by the DDS application using the expedited forwarding network class is protected against background perturbing traffic.



(a) without QoS



(b) with QoS

Figure 21: Impact of the QoS provisioning Capabilities on the bandwidth protection

3.3.3. *Evaluating the Impact of the Velox QoS Manager Capabilities on Latency*

Rationale and approach. Velox provides network QoS mechanisms to control end-to-end latency delay between distributed applications. The next experiment evaluates the overhead of using it to enforce network QoS. As described in Section 2.2, DDS provides deployment-time configuration of middleware by adding DSCP markings to IP packets. When applications invoke remote operations, the Velox QoS Server intercepts each request and uses it to reserve the network QoS resources for each call. It reserves these resources by configuring the edge router queues with the priority level extracted from the DSCP field (*e.g.*, expedited forwarding, assured forwarding, etc).

We used WANem (`wanem.sourceforge.net`) to emulate realistic WAN behaviors during application development/testing over our LAN environment. WANem allows us to conduct experiments in real environments to assess performance with and without QoS mechanisms. These comparisons enabled us to measure the impact of change with the QoS mechanisms provided by Velox.

This experiment had the following variants:

- We started one-to-one communication between endpoints, followed by sending perturbing UDP background traffic, and
- We increased the number of senders and receivers applications to evaluate their impact on transmission delay.

To measure the one way delay between senders and receivers, we used the Network Time Protocol (NTP) [22] to synchronize all applications components with one global clock. We then ran application components that overloaded the network link and routers to perform extra work and applied policies to instrument IP packets with the appropriate DSCP values.

Results and analysis. Figure 22 shows the end-to-end delivery time for distributed DDS applications over a WAN without applying any QoS mechanisms. Figure 22 also shows the impact of using the Velox QoS server, which shows the latency delay measured when applying QoS mechanisms to use-case applications. These results indicate that the end-to-end delay measured without QoS management is more than twice as large than the delay measured when applying QoS management at the edge routers. A closer examination shows that WANem incurs roughly an order of magnitude more effort than Velox to provide QoS assurance for end-to-end application flows.

3.3.4. *Evaluating QoS Manager Capabilities for One-to-Many Communications*

Rationale and approach. This experiment evaluates the potential of the Velox framework to handle increases in the number of DDS participants (we do not consider

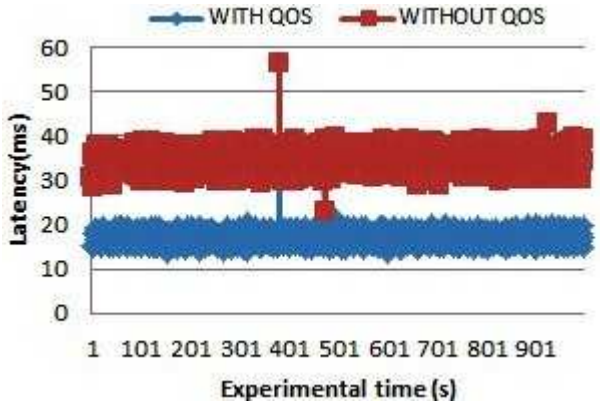


Figure 22: Impact of the QoS provisioning Capabilities on the end-to-end delay

WANem here). We measured the moving average delay between DDS applications distributed over the Internet. We configured the DiffServ implementation in the edge router of each network, as described in Section 3.1. We then used DDS-based traffic generator applications to send DDS topics via the Velox QoS service's expedited forwarding mechanisms at 500 kbps. Each DDS flow was sent from one or more remote publishers from IP domain 1 managed by the "Montperdu" edge router (shown in Figure 10) to one and/or many subscribers in IP domain 2 managed by "Posets" edge router.

The experiments in this configuration had the following variants:

- We started one publisher sending data in the direction of two remote subscribers and then measured the worst-case end-to-end latency between them,
- We used the same publisher and increased the number of subscribers, *i.e.*, we added two more subscribers to analyze the impact of competing flows arriving from distributed applications on the Velox QoS server, and
- We increased the number of participants to obtain eight subscribers in competition for receiving a single published expedited forwarding QoS flow from the EuQoS6 machine.

The bandwidth utilization was limited to 1 Mbps for all experiments so it would be consistent with the number of participants tested.

Results and analysis. The end-to-end delay shown in Figure 23 includes the latency curves for 1-to-2, 1-to-4, and 1-to-8 configurations. When a single publisher sent DDS

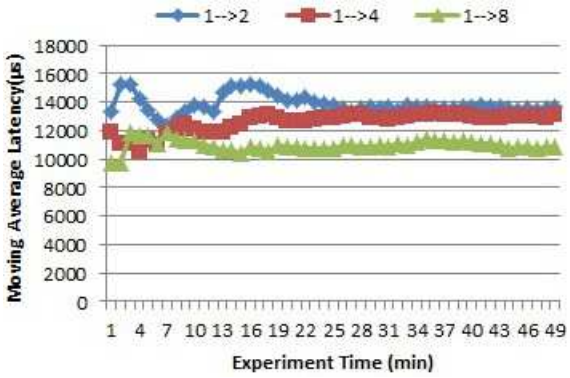


Figure 23: Impact of Competing DDS Flows on End-to-End Delay

topic data to several subscribers we found the latency values for different configurations remained ~13 ms. In particular, the average latency is ~13 ms for the 1-to-2 variant and the average latency is ~12 ms for the 1-to-4 and 1-to-8 variants.

Based on these results, we conclude that the number of subscribers affects end-to-end latency. In comparison with communication over a LAN, the increase in the number of subscribers in the WAN adds more jitter to the overall system. This jitter remains perceivable for the WAN configuration since communication is measured in milliseconds. Additional experiments conducted over a WAN for other configurations—including more than 30 distributed application subscribers—indicated an end-to-end delay of ~15 ms.

3.3.5. Evaluating QoS Manager Capabilities for Many-to-One Communications

Rationale and approach. This experiment is the inverse of the one in Section 3.3.4 since we considered two expedited forwarding QoS competing flows sent by two remote publishers to reach a single subscriber. We increased

the number of published QoS flow by increasing the number of participants to 4 and 8 publishers, respectively. Figure 24 shows the many-to-one latency obtained from trace files, where each sending DDS application uses the expedited forwarding QoS class supported by Velox.

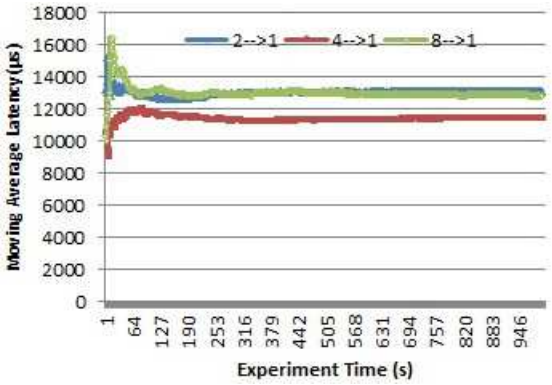


Figure 24: Impact of Competing DDS Flows on End-to-End Delay

Results and analysis. As shown in Figure 24, the end-to-end latency is ~13ms when two publishers sent DDS topics to a single DDS subscriber. The delay is ~13ms when we considered 4 and 8 publishers sending data to a single DDS subscriber. The increased number of publishers does not significantly affect the end-to-end delay during the experiments. In particular, all data packets marked with DSCP value 46 are processed with the same priority in the edge router. The Velox framework can configure edge router queues to support the expedited forwarding of packets with high priority.

3.3.6. Evaluating QoS Manager Capabilities for Many-to-Many communications

Rationale and approach. This experiment evaluates the impact of increasing number of participants on both publishers and subscribers. We started with 2-to-2 communication where two publishers send DDS topic data to both two remote subscribers. We then increased the number of participants to have 4-to-4 and 8-to-8 communication, respectively. Figure 25 shows the many-to-many configuration using the expedited forwarding QoS class supported by Velox.

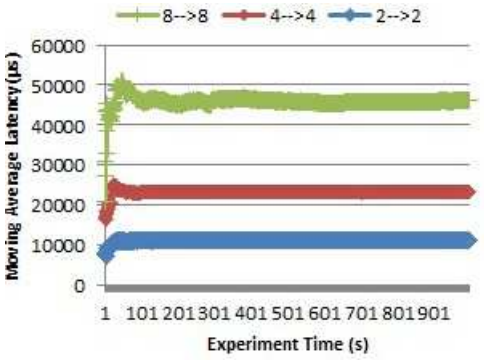


Figure 25: Impact of the competing flows on the end-to-end delay

Results and analysis. The latency experienced for many-to-many communication shows a time delay of ~14 ms for the 2-to-2 configuration. The latency increases to ~22 ms for the 4-to-4 configuration and ~45 ms for the 8-to-8 configuration. By setting the DDS reliability QoS policy setting to "reliable" (*i.e.*, the samples were guaranteed to arrive in the order published), Velox helps to balance time-determinism and data-delivery reliability.

The latency for the 8-to-8 configuration is higher than the 2-to-2 and 4-to-4 values because the data writers maintain a send queue to hold the last "X" number of samples sent. Likewise, data readers maintain receive queues with space for consecutive "X" expected samples. Nevertheless, the end-to-end latency for the 8-to-8 configuration is acceptable because DDS ensures the one-way delay for applications in DRE systems is less than 100 ms.

4. Related work

Conventional techniques for providing network QoS to applications incur several key limitations, including a lack of mechanisms to (1) specify deployment context-specific network QoS requirements and (2) integrate functionality from network QoS mechanisms at runtime. This section compares the Velox QoS provisioning mechanisms for DiffServ-enabled networks with related work. We divide the related work into general middleware-based QoS management solutions and those that focus on network-level QoS management.

4.1. QoS Management Strategies in Middleware

Different QoS properties are essential to provide each operation the right data at the right time, and hence the network infrastructure should be flexible enough to support varying workloads at different times during the operations [23], while also maintaining highly predictable and dependable behavior [24]. Middleware for adaptive QoS control [25] [26] was proposed to reduce the impact of QoS management on the application code, which was extended in the HiDRA project [27] for hierarchical management of multiple resources in DRE systems [28]. Many middleware-based technologies have also been proposed for multimedia communications to achieve the required QoS for distributed systems [29] [30].

QoS management in content-based pub/sub middleware [31] allows powerful content-based routing mechanisms based on the message content instead of IP-based routing. Likewise, many pub/sub standards and technologies (*e.g.*, Web Services Brokered Notification [32] and the CORBA Event Service [33]) have been developed to support large-scale data-centric distributed systems [34]. These standards and technologies, however, do not provide fine-grained and robust QoS support, but focus on issues related to monitoring run-time application behavior. Addressing these challenges requires end-system QoS policies to control the deployment and the self-adaptation of resources to simplify the definition and deployment of network behavior [35]. Besides, many pub/sub middleware [36] have been proposed for real-time and distributed systems to ensure both performance and scalability in QoS-enabled components for DRE systems, as well as for Web-enabled applications.

For example, [37] proposed a reactive QoS-aware service for DDS for embedded systems to refactor the DDS RTPS protocol. This approach scales well for DRE systems comprising on-board DDS applications, however, it does not provide any analyses about the schedulability of the occurring events, and how it can impact the behavior of the system end-to-end. In addition, we developed container-based pub/sub services in the context of OMG's Lightweight CORBA Component Model (LwCCM) [38]. We argue this solution is restricted to few number of QoS policies. It provides only two QoS settings that can be mapped into 2 network services that can be used in the context of mono-domain network. The solution provided in this paper benefits from the rich set of DDS QoS policies that we used in the context in multi-domain network. This allows defining more flexible classes of services to fit the application requirements. In addition,[39] presented a benchmark of DDS middleware regarding it timeliness performance. Authors studied the DDS QoS properties in the context of Best-Effort network. Our concerns is using the DDS QoS policies that allows controlling the QoS proprieties end-to-end. Our work addresses the QoS-based network architecture which help us to mark out the latency experienced in the network.

In [40] authors presented the integration of the DDS middleware with SOA and web-service into a single framework to allow teams collaboration over the Internet. Since this solution allow the interoperability between heterogeneous applications, however, the end-to-end QoS can be guaranteed because the additional latencies added by the web interfaces. Likewise, in [41] the authors proposed a redirection proxy on top of DDS to support adaptation to mobile networks. Even if this architecture adds a Mobile DDS client implemented in mobile device, the Mobile DDS Clients are expected to run in single network domains in wireless networks with connectivity guarantees, which is not the case in heterogeneous networks. We argue that using a redirecting proxy can have several shortcomings when applied to real-time communication. In particular, our solution benefits from the mapping between the application layer and the middleware layer to improve the QoS constraints required by the each data flow. Without using either redirection proxy or mobile agent, therefore, each flow in our solution has a specific requirement that allows grouping them into different classes of traffic, where each class has its specific DDS QoS policies that we mapped to a specific IP services.

In [42], authors presented a broker-like redirection layer to allow P2P data dissemination between remote participants. We argue that even if we use brokers, we will still need to use our solution because even the brokers will be geographically distributed, and our approach should apply even if we have brokers.

To assess the adequate QoS supply chain management application, authors in [43] presented a queuing Petri net methodology for message-oriented event-driven systems. Such a system is composed of interconnected business products and services required to the end user. Petri nets are well suited to analyze the performance of Flexible

Manufacturing System (FMS) which involves measuring the production rate, machine utilization, kanban scheduling, etc. In this model, the transportation times are included in the transitions times. In comparison with our analysis model, this one differs from ours on three points: first, the FMS application does not require any real-time constraints when putting it in production; even if some cases require this, the queuing Petri net is not the best choice to analyze the performance of the system, but the timed petri net is more appropriate for this purpose. Thus, TINA (TIme petri NetAnalyzer) [8] is a toolbox developed in our lab which allows analyzing real-time system using time petri nets. Second, DDS is not a message-oriented middleware (*e.g.*, JMS), even if DDS topics are similar to messages, DDS is a data-centric middleware. DDS and JMS are based on fundamentally different paradigms with respect to data modeling, dataflow routing, discovery, and data typing. Finally, the analytical model presented in this paper is based on queuing theory to perform analysis of real-time constraints in our application. The model differs from the petri net model in the way the performance analysis is inferred from the model and how they can be applied in telecommunication system.

The OMG's Data Distribution Service (DDS) defines several timing parameters (*e.g.*, deadline, latency_budget) that are suitable for network scheduling rather than the data processing in the processor since those QoS parameters are used to update the topic production profile. For example, the deadline QoS manages the write updates between samples, while latency_budget QoS can control the end-to-end latency. DDS QoS policies thus effectively make the communication network a schedulable entity [44]. In contrast, DDS does not provide policies related to scheduling in the processor.

Despite a range of available middleware-based QoS management solutions, there has heretofore been a general lack of tools to analyze the predictability and timeliness of these solutions. Verifying these solutions formally requires performance modeling techniques (such as those described in Section 2.1) to empirically validate QoS in computer networks. Our performance modeling approach can be used to specify both the temporal non-determinism of weakly distributed applications and the temporal variability of the data processing when using DDS middleware. DDS middleware can use the results of our performance models to control scheduling policies (*e.g.*, earliest deadline first, rate monotonic, etc.) and then assign the scheduling policies for threads created internally by the middleware.

4.2. Network-level QoS Management

Prior middleware solutions for network QoS management [45] focus on how to add layer 3 and layer 2 services for CORBA-based communication [46] [47]. A large-scale event notification infrastructure for topic-based pub/sub applications has been suggested for peer-to-peer routing overlaid on the Internet [48]. Those approaches can be deployed only in a single-domain network, however, where one administrative domain manages the whole network. Extending these solutions to the Internet can result in traffic specified at each end-system being dropped by the transit network infrastructure of other domains [38].

It is therefore necessary to specify the design for network QoS support and session management that can support the diverse requirements of these applications [49], which require differentiated traffic processing and QoS, instead of the traditional best-effort service [40] provided by the Internet. Integrating signaling protocols (such as SIP and H.323) into the QoS provisioning mechanisms has been proposed [50] with message-based signaling middleware for the control plane to offer per-class QoS. Likewise, a network communication broker [51, 46] has been suggested to provide per-class QoS for multimedia collaborative applications. This related work, however, supports neither mobility service management nor scalability since it adds complicated interfaces to both applications and middleware for the QoS notification. When an event occurs in the network, applications should adapt to this modification [52], *e.g.*, by leveraging different codecs that adapt their rates appropriately.

Authors in [42, 53] have provided a framework [9] that address the reliability and the scalability of DDS communication over P2P large-scale infrastructure. This work, however, is based on the best-effort QoS mechanisms of

[8] `http://projects.laas.fr/tina`

[9] `http://lia.deis.unibo.it/Research/REVENGE/index.html`

the network and omits the fact that if the network is unable to provide the QoS provisioning and the resource allocation, there will be no guarantees that the right data will be transmitted at the right time.

Our earlier DRE middleware work [54] has focused on priority reservation and QoS management mechanisms that can be coupled with CORBA at the OS level to provide flexible and dynamic QoS provisioning for DRE applications. In the current work, our Velox framework provides an architecture that extends the best-effort QoS properties found in prior work. In particular, our solution considers application flows requirements and maps them into the DDS layer to allow end-to-end QoS provisioning. We therefore integrate QoS along two key dimensions: (1) the *horizontal direction* between different adjacent layers in the network stack (application, middleware, and network), and (2) the *vertical direction* between homologous layers (layers at the same level of the OSI model).

To address limitations with related work, the Velox framework described in Section 2.2 need not modify existing applications to achieve the benefits of assured QoS. Velox uses QoS provisioning mechanisms based on MPLS over DiffServ QoS-based architectures. These mechanisms were widely discussed in several papers in networking including admission control mechanism with COPS [55], NSIS [56] protocol and MPLS mapping over WAN [57].

Experiments were conducted with one MPLS tunnel because if there are any traffic engineering tunnels to the BGP next hop, and if one or more of those is available for use by the packet in question, one of these tunnels is chosen. This tunnel will be associated with an MPLS label, the "tunnel label". The tunnel label gets pushed on the MPLS label stack, and the packet is forwarded to the tunnels next hop.

In this paper, DDS applications may be connected to one Service Provider (SP) and/or many SPs, but this is made transparent to the application since it supposes there is a *Service Level Agreement* (SLA) between the SPs (RFC 4364: BGP/MPLS IP VPNs) [57]. Using one MPLS tunnel is therefore sufficient for the application to distribute DDS Topics end-to-end, so there is no need to manage multiple MPLS tunnels is required (the SLA allows data distribution among multiple ASs transparently).

Velox allows end users to request a specific QoS-guaranteed connectivity independent of the chosen applications, thereby achieving net neutrality requirements. Moreover, Velox does not alter the decentralized Internet model since it relies on bi-lateral agreements among neighboring domains. Velox's policy-based management also enables the integration of services in a single session that requires network resources, such as a bandwidth and controllable end-to-end delay. It provides the mapping of the DDS session between the application at the service plane and the underlying network. The Velox QoS server thus addresses mobility issues by installing the required QoS allocation scheme for DDS sessions using per-flow QoS reservation in edge routers.

5. Conclusions

Although DDS implementations have been used to develop many scalable, efficient, and predictable DRE applications, the DDS standard has several limitations, including lack of processor scheduling and end-to-end QoS support. This paper describes how we addressed these limitations by (1) analyzing DDS scheduling capabilities to deliver DDS samples on an end-system using a performance model and validating the accuracy of this model via both simulation results and empirical experiments and (2) developing the Velox policy-based management framework to provide end-to-end QoS provisioning for DDS-based applications by controlling network resources, such as a bandwidth and end-to-end delay.

We learned following lessons from developing and evaluating our performance model and the Velox framework:

• **Ontology-based mechanisms are needed to automate DDS QoS configurations.** The results in this paper underscore the importance of integrating QoS policies at both the end-system and network levels to supply users with the right data at the right time, support varying workloads at different times during operations, and maintain predictable behavior. Velox currently supports end-to-end QoS provisioning for DDS applications over multi-domain networks, but lacks a robust means to ensure semantic consistency of QoS policies end-to-end. In particular, a DDS QoS configuration developed for one scenario in one operating environment may be suboptimal for different scenarios in different operating environments. Our future work will therefore focus on adding

ontology-based mechanisms [58, 59] to Velox that automate the assured configuration of DDS QoS policies, thereby adapting network resources to better meet application requirements.

For example, for effective QoS-based service selection DDS applications require a tool to determine which transport policy should be provided to the network to decide which DiffServ service (AF, EF, etc.) best fits application requirements. The use of ontologies makes it possible to detect the appropriate *Service Level Agreement* (SLA) to perform matching between the DDS application needs and the service offered in real-time. This tool can be implemented as an adjacent layer to the DDS middleware to detect actual behavior of the network service.

• **History Cache latency must be taken into account when calculating the overall delay.** This paper described the specification and the evaluation of the performance model to calculate the time latency at an end-system and then conducted experimentations to verify the effectiveness of this approach. The fidelity and effectiveness of the delay computation model can be improved when all software layers and hardware artifacts are accurately considered in the model.

For example, the results from Section 3.2 show discrepancies between the delay predicted by our analytical model versus the experimentally validated model. These discrepancies stem from simplifying assumptions made when integrating DDS with the application layer, *e.g.*, we did not consider the history cache that keeps DDS topics in the case of durable data writers and readers nor did we consider the DLRL (Data Local Reconstruction Layer) layer of DDS. Accurate modeling can help improve time delay calculations required by the history cache in the latency calculation, both in terms of performance and memory consumption. Our future work will therefore consider the case were durable writers and readers keep their history cache in permanent storage and study its impact on the end-to-end latency.

• **A constrained application protocol can help support QoS provisioning for resource-constrained Internet devices.** This paper explained how the web-service QoS provisioning provided by Velox can control network elements by enforcing policy control mechanisms, negotiating QoS, and coordinating the data and signaling paths to perform resource reservation. Velox does not, however, address the QoS requirements of resource-constrained applications over the Internet that have power and energy limitations. In particular, to allow machine-to-machine (M2M) communication, a special data format should replace existing HTML and XML-based formats with REST-based formats that work with existing Internet solutions. Our future work will therefore develop cross-layer DDS-based solutions that support efficient and compact energy transmission methods, energy harvesting and service-layer architecture for M2M that includes bindings for both HTTP/REST and CoAP (Constrained Application Protocol) [60] for constrained battery-powered devices.

• **Large deployments of publish/subscribe applications involve brokers.** For scalability reasons, large deployments of publish/subscribe applications often involve brokers at various network and middleware levels [61, 62, 6, 38]. Although this paper focused on end-to-end timeliness of event dissemination between a publisher and a subscriber without involving any brokers, our approach should seamlessly apply to a system comprising event brokers.

• **Additional dimensions of QoS require refinements in the solution.** The paper addresses the end-to-end timeliness issues in large-scale networks, which relies on information sharing between different participants among shared data spaces. Since DDS is deployed in mission-critical and/or enterprise DRE systems for its ultra-low latency benefits, the framework should support high information assurance requirements without overloading the overall system. A minimum and efficient use of cryptography is thus required to enhance the integrity of the information dissemination. In particular, security policies to control and restrict access to information to only the authorized recipients should be included to the framework to prevent denial of service attacks and ensure the non-repudiation of information.

Our future work will therefore develop security policies to allow authentication, authorization, access control, and secure transport. These policies can be defined via the *Security Assertion Markup Language* (SAML) to allow exchanging public/private keys as part of the DDS QoS policies, *e.g.*, within a confidential DDS topics that may be transported using the *Datagram Transport Layer Security* (DTLS) protocol. Additional work is also needed to support reliability of the event dissemination and tolerance to failures.

References

[1] J. White, B. Dougherty, R. Schantz, D. C. Schmidt, A. Porter, A. Corsaro, R&D Challenges and Solutions for Highly Complex Distributed Systems: a Middleware Perspective, the Springer Journal of Internet Services and Applications special issue on the Future of Middleware 2 (3).

[2] OMG-DDS, Data distribution service for real-time systems specification, ddsv1.2, `http://www.omg.org/spec/DDS/1.2/`.

[3] G. Kartik, K. Kyoung-Don, Coordinated allocation and scheduling of multiple resources in real-time operating systems, in: Proc. OSPERT, June 2007.

[4] A. Hakiri, B. Pascal, A. Gokhale, G. Thierry, D. C. Schmidt, J. Hoffert, SIP-based QoS Support Architecture and Session Management for DDS-based Distributed Real-time and Embedded Systems, in: Poster at ACM Distributed Event-based Systems (DEBS '11), ACM, Yorktown Heights, NY, USA, 2011, pp. 389–390.

[5] A. Hakiri, A. Gokhale, D. C. Schmidt, B. Pascal, J. Hoffert, G. Thierry, A SIP-based Network QoS Provisioning Framework for Cloud-hosted DDS Applications, in: 1st International Symposium on Secure Virtual Infrastructures (DOA-SVI'11), Springer LNCS, Crete, Greece, 2011, pp. 507–524.

[6] R. E. Schantz, J. P. Loyall, C. Rodrigues, D. C. Schmidt, Y. Krishnamurthy, I. Pyarali, Flexible and adaptive qos control for distributed real-time and embedded middleware, in: Middleware, Vol. 2672 of Lecture Notes in Computer Science, Springer, Rio de Janeiro, Brazil, 2003, pp. 374–393.

[7] E. N. Gilbert, Capacity of a burst-noise channel, Bell System Technical Journal (1960) 1253–1265.

[8] M. David, M. Jim, B. Jack, K. William, Network time protocol version 4: Protocol and algorithms specification, IETF, RFC 5905.

[9] A. Konrad, B. Y. Zhao, A. D. Joseph, Determining model accuracy of network traces, Journal of Computer and System Sciences (2006) 1156–1171.

[10] P. Gao, S. Wittevrongel, K. Laevens, D. De Vleeschauwer, H. Bruneel, Distributional littles law for queues with heterogeneous server interruptions, Electronics Letters 46 (2010) 763–764.

[11] J. Wroclawski, The use of rsvp with ietf integrated services, IETF RFC 2210.

[12] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, An architecture for differentiated service, IETF RFC 2475.

[13] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, J. McManus, Requirements for traffic engineering over mpls, IETF RFC 2702.

[14] F. L. Faucheur, E. Rosen, Advertising IPv4 Network Layer Reachability Information with an IPv6 Next Hop, RFC 5549 (Proposed Standard) (May 2009).
URL `http://www.ietf.org/rfc/rfc5549.txt`

[15] R. Hancock, G. Karagiannis, J. Loughney, S. V. den Bosch, Next Steps in Signaling (NSIS): Framework, RFC 4080 (Informational) (Jun. 2005).
URL `http://www.ietf.org/rfc/rfc4080.txt`

[16] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, J. Arkko, Diameter base protocol, IETF RFC 3588.

[17] P. Owezarski, P. Berthou, Y. Labit, D. Gauchard, LaasNetExp: a generic polymorphic platform for network emulation and experiments, 4th International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities.

[18] O. Sangyoon, K. Jai-Hoon, F. Geoffrey, Real-time performance analysis for publish/subscribe systems, Future Generation Computer Systems 26 (2009) 318 – 323.

[19] B. Davie, A. Charny, J. Bennet, K. Benson, J. L. Boudec, W. Courtney, S. Davari, V. Firoiu, D. Stiliadis, An Expedited Forwarding PHB (Per-Hop Behavior), RFC 3246 (Proposed Standard) (Mar. 2002).
URL `http://www.ietf.org/rfc/rfc3246.txt`

[20] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, Assured Forwarding PHB Group, RFC 2597 (Proposed Standard), updated by RFC 3260 (Jun. 1999).
URL http://www.ietf.org/rfc/rfc2597.txt

[21] D. Grossman, New Terminology and Clarifications for Diffserv, RFC 3260 (Informational) (Apr. 2002).
URL http://www.ietf.org/rfc/rfc3260.txt

[22] D. Mills, J. Martin, J. Burbank, W. Kasch, Network Time Protocol Version 4: Protocol and Algorithms Specification, RFC 5905 (Proposed Standard) (Jun. 2010).
URL http://www.ietf.org/rfc/rfc5905.txt

[23] D. C. Schmidt, D. L. Levine, S. Mungee, The design of the tao real-time object request broker, Computer Communications 21 (1998) 294–324.

[24] J. Chen, M. Diaz, L. Llopis, B. Rubio, J. M. Troya, A survey on quality of service support in wireless sensor and actor networks: Requirements and challenges in the context of critical infrastructure protection, J. Netw. Comput. Appl. (2011) 1225–1239.

[25] G. Duzan, J. Loyall, R. Schantz, R. Shapiro, J. Zinky, Building adaptive distributed applications with middleware and aspects, 2004, pp. 66–73.

[26] S. P. Mahambre, S. Kumar-Madhu, U. Bellur, A taxonomy of qos-aware, adaptive event-dissemination middleware, IEEE Internet Computing 11 (2007) 35–44.

[27] N. Shankaran, X. D. Koutsoukos, D. C. Schmidt, Y. Xue, C. Lu, Hierarchical control of multiple resources in distributed real-time and embedded systems, in: ECRTS, 2006, pp. 151–160.

[28] W. Duangdao, N. Klara, G. Xiaohui, X. Dongyan, 2k: An integrated approach of qos compilation and reconfigurable, component-based run-time middleware for the unified qos management framework, 2001, pp. 373–394.

[29] M. Valls, A. Alonso, J. Ruiz, A. Groba, An architecture of a quality of service resource manager middleware for flexible embedded multimedia systems, in: Software Engineering and Middleware, LNCS, 2003, pp. 36–55.

[30] K. Nahrstedt, H. H. Chu, S. Narayan, Qos-aware resource management for distributed multimedia applications, J. High Speed Netw. 7 (1998) 229–257.

[31] A. Farroukh, E. Ferzli, N. Tajuddin, H. A. Jacobsen, Parallel event processing for content-based publish/subscribe systems, in: ACM. DEBS '09, 2009, pp. 1–8.

[32] OASIS, Web services brokered notification version 1.3, http://www.oasis-open.org/.

[33] CROBA-OMG, Common object request broker architecture (corba/iiop), 3.1,, http://www.omg.org/spec/CORBA/3.1/.

[34] J. A. Dianes, M. Diaz, B. Rubio, Using standards to integrate soft real-time components into dynamic distributed architectures, Comput. Stand. Interfaces (2012) 238–262.

[35] C. D. Gill, J. M. Gossett, D. Corman, J. P. Loyall, R. E. Schantz, M. Atighetchi, D. C. Schmidt, Integrated adaptive qos management in middleware: A case study, Real-Time Syst. 29 (2005) 101–130.

[36] P. T. Eugster, P. A. Felber, R. Guerraoui, A. M. Kermarrec, The many faces of publish/subscribe, ACM Comput. Surv. 35 (2003) 114–131.

[37] X. Lu, X. Li, T. Yang, Z. Liao, W. Liu, H. Wang, Qos-aware publish-subscribe service for real-time data acquisition, Business Intelligence for the Real-Time Enterprise 27 (2009) 29–44.

[38] J. Balasubramanian, S. Tambe, B. Dasarathy, S. Gadgil, F. Porter, A. Gokhale, D. C. Schmidt, Netqope: A model-driven network qos provisioning engine for distributed real-time and embedded systems, in: RTAS' 08: Proceedings of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium, IEEE Computer Society, Los Alamitos, CA, USA, 2008, pp. 113–122. doi:10.1109/RTAS.2008.32.

[39] C. Esposito, S. Russo, D. D. Crescenzo, Performance Assessment of OMG Compliant Data Distribution Middleware, in: Ipdps'08, 2008, pp. 1–8.

[40] Y.-H. Wang, S.-H. Yang, A. Grigg, J. Johnson, A dds based framework for remote integration over the internet, in: 7th Annual Conference on Systems Engineering Research, CSER, 2009.

[41] K.-J. Kwon, C.-B. Park, H. Choi, A Proxy-based Approach for Mobility Support in the DDS System, in: 6th IEEE International Conference on Industrial Informatics, IEEE, 2008.

[42] A. Corradi, L. Foschini, A dds-compliant p2p infrastructure for reliable and qos-enabled data dissemination, in: IPDPS, 2009, pp. 1–8.

[43] K. Sachs, S. Kounev, A. Buchmann, Performance modeling and analysis of message-oriented event-driven systems, Software & Systems Modeling (2012) 1–25.

[44] J. Schlesselman, G. Pardo-Castellote, B. Farabaugh, Omg data-distribution service (dds): architectural update, in: IEEE, MILCOM 2004, 2004, pp. 961–967.

[45] E. S. Richard, P. L. Joseph, R. Craig, C. S. Douglas, K. Yamuna, I. P., Flexible and adaptive qos control for distributed real-time and embedded middleware, 2003, pp. 374–393.

[46] B. Dasarathy, S. Gadgil, R. Vaidyanathan, K. Parmeswaran, B. Coan, M. Conarty, V. Bhanot, Network qos assurance in a multi-layer adaptive resource management scheme for mission-critical applications using the corba middleware framework, IEEE. RTAS.

[47] B. Dasarathy, S. Gadgil, R. Vaidyanathan, A. Neidhardt, B. Coan, K. Parmeswaran, A. McIntosh, F. Porter, Adaptive network qos in layer-3/layer-2 networks as a middleware service for mission-critical applications, JSS 80.

[48] A. I. T. Rowstron, A. M. Kermarrec, M. Castro, P. Druschel, Scribe: The design of a large-scale event notification infrastructure, in: Springer, COST, 2001, pp. 30–43.

[49] S. Michal, P. Pavel, F. Dario, C. Tommaso, C. Fabio, H. Zdenek, L. Giuseppe, Modular software architecture for flexible reservation mechanisms on heterogeneous resources, Journal of Systems Architecture 57 (2011) 366–382.

[50] G. L. Teodora, H. Schmidt, A. Schorr, F. J. Hauck, A. Kassler, A session initiation protocol based middleware for multi-application management, IEEE. ICC.

[51] C. Zhang, Sadjadi, S. Masoud, S. Weixiang, R. Raju, D. Yi;, A user-centric network communication broker for multimedia collaborative computing, IEEE, CollaborateCom.

[52] C. Antonio, F. Luca, A dds-compliant p2p infrastructure for reliable and qos-enabled data dissemination, in: IEEE-IPDPS, 2009, pp. 1–8.

[53] A. Corradi, L. Foschini, L. Nardelli, A dds-compliant infrastructure for fault-tolerant and scalable data dissemination, in: Proceedings of the The IEEE symposium on Computers and Communications, ISCC '10, 2010, pp. 489–495.

[54] S. R. E., L. J. P., R. Craig, S. D. C., K. Yamuna, I. Pyarali, Flexible and adaptive qos control for distributed real-time and embedded middleware, in: Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware, Middleware '03, 2003, pp. 374–393.

[55] S. Salsano, Cops usage for diffserv resource allocation (cops-dra), IETF Draft, draft-salsano-cops-dra-00.

[56] J. Manner, G. Karagiannis, A. McDonald, Nsis signaling layer protocol (nslp) for quality-of-service signaling, RFC 5974 (Experimental) (Oct. 2010).

[57] E. Rosen, Y. Rekhter, BGP/MPLS IP Virtual Private Networks (VPNs), RFC 4364 (Proposed Standard) (2006).

[58] R. Carlos, L. S. Rito, Álvarez Sabucedo Luis M., C. Paulo, An ontology for managing network services quality, Expert Syst. Appl. 39 (9) (2012) 7938–7946.

[59] J. L. Pastrana, E. Pimentel, M. Katrib, Qos-enabled and self-adaptive connectors for web services composition and coordination, Computer Languages, Systems & Structures 37 (1) (2011) 2 – 23.

[60] C. Bormann, A. P. Castellani, Z. Shelby, Coap: An application protocol for billions of tiny internet nodes., IEEE Internet Computing 16 (2012) 62–67.

[61] R. Campbell, R. Daley, B. Dasarathy, P. Lardieri, B. Orner, R. Schantz, R. Coleburn, L. R. Welch, P. Work, Toward an approach for specification of qos and resource information for dynamic resource management, in: Second RTAS Workshop on Model-Driven Embedded Systems (MoDES '04), 2004.

[62] B. Dasarathy, S. Gadgil, R. Vaidhyanathan, K. Parmeswaran, B. Coan, M. Conarty, V. Bhanot, Network QoS Assurance in a Multi-Layer Adaptive Resource Management Scheme for Mission-Critical Applications using the CORBA Middleware Framework, in: IEEE RTAS, 2005.