Message-Oriented Middleware with QoS Awareness

Hao Yang, Minkyong Kim, Kyriakos Karenos, Fan Ye, and Hui Lei

IBM T. J. Watson Research Center
{haoyang,minkyong,kkarenos,fanye,hlei}@us.ibm.com

Abstract. Publish/subscribe messaging is a fundamental mechanism for interconnecting disparate services and systems in the service-oriented computing architecture. The quality of services (QoS) of the messaging substrate plays a critical role in the overall system performance as perceived by the end users. In this paper, we present the design and implementation of Harmony, an overlay-based messaging system that can manage the end-to-end QoS in wide-area publish/subscribe communications based on the application requirements. This is achieved through a holistic set of overlay route establishment and maintenance mechanisms, which actively exploit the diversity in the network paths and redirect the traffic over links with good quality, e.g., low latency and high availability. In order to cope with network dynamics and failures, Harmony continuously monitors the link quality and adapts the routes whenever their quality deteriorates below the application requirements. Harmony can operate on top of different data transport layers. When the transport layer has built-in message scheduling capability, Harmony takes advantage of it and utilizes a novel budget allocation scheme to control the scheduling behavior. We have fully implemented the Harmony messaging system, and our empirical experience has confirmed its effectiveness in providing end-to-end QoS in dynamic wide-area network environments.

1 Introduction

We are witnessing major transformations to the enterprise computing landscape. One of such transformations is the ever increasing awareness of the real-world events and conditions through massive sensing, analytics and control capabilities, leading to a proliferation of cyber–physical systems (CPS)[1]. Another major transformation is the growing interconnection and interoperation of enterprise systems over a geographically distributed wide area, as triggered by business practices like mergers and acquisitions, off-shoring, outsourcing, and the formation of virtual enterprises. The second transformation has been driving an emerging engineering discipline around the system of systems (SoS) [2]. Message-oriented middleware (MOM) is widely recognized as a promising approach to the integration of both CPS and SoS, because messaging is a simple and natural communication paradigm for connecting the loosely-coupled and distributed components in those systems. However, CPS and SoS have also introduced new non-functional requirements on MOM. Specifically, MOM must

L. Baresi, C.-H. Chi, and J. Suzuki (Eds.): ICSOC-ServiceWave 2009, LNCS 5900, pp. 331–345, 2009.

be aware of and satisfy the unique quality-of-service (QoS) needs of these new systems in order for it to be practically useful.

Consider cyber physical systems being developed for a wide variety of application domains ranging from the smart grid of electricity to environmental monitoring and to intelligent transportation. Voluminous sensor event data needs to be transported from field sensors to backend enterprise servers for complex event processing and integration with the business processes. Sensor data is often time-sensitive in that the correct data that comes too late may become the wrong data. Therefore sensor data must be transported in a very responsive and reliable manner. Similarly, control directives carried in the reverse direction of traffic may drive various mission-critical systems. The control directives may have stringent requirements on delivery performance and security in order to avoid catastrophic consequences. On the other hand, the communication infrastructure for sensor data and control directives presents a number of challenges. Sensors are often deployed in potentially hostile environments, which make the sensors more prone to malicious attacks and natural hazards. Further, sensors are connected through wireless links that are inherently weak. There may be a high degree of variability in wireless bandwidth due to moving obstructions, RF interference, and weather. There may also be periods of intermittent disconnections. Such characteristics make it very difficult for MOM to effectively address the QoS requirements of CPS.

In the realm of system of systems, the constituent systems may be distributed over a large geographic area, e.g., across a nation or even spanning multiple continents. Messages between the systems often have to travel a long communication path, incurring much larger delay than local-area messaging. It is also harder for a long-haul communication path to maintain high availability due to the increased number of nodes and links on the path. Further, the systems are likely to be deployed and operated by separate organizations, which result in different security properties and degrees of trustworthiness to be associated with these systems. Despite technical challenges arising out of the communication infrastructure, many SoS applications require messaging capabilities with certain assurance on a range of QoS metrics including latency, throughput, availability and security. One example of such an SoS assimilated multiple systems used by US federal agencies (FAA, DoD, DHS, etc.) to facilitate the distribution of real-time national air surveillance data among these agencies [3].

Existing MOMs fall into one of two categories: enterprise messaging systems and real-time messaging systems. Intended to address traditional business needs, enterprise messaging systems provide message delivery assurance and transactional guarantees. They usually implement the JMS standard [4] and can transport messages over a wide area across multiple domains. However, they do not proactively manage messaging performance. As such, applications cannot predict or depend on when messages will arrive at the destination. Real-time messaging systems, on the other hand, offer QoS assurance by allocating resources and scheduling messages based on application-specific QoS objectives. They often conform to the DDS standard [5]. Unfortunately these systems are limited to

QoS management within a local area or a single domain. They are not designed for wide-area messaging involving multiple separate domains. Neither enterprise messaging nor real-time messaging is adequate for the emerging CPS and SoS, which require QoS awareness and enablement for messaging in a large geographic area and through federated domains.

The Harmony messaging system developed at IBM T. J. Watson Research Center is designed to combine the best of enterprise messaging and real-time messaging to suit the needs of the emerging CPS and SoS paradigms. Specifically, Harmony facilitates the interconnection of disparate messaging domains over large geographic areas and heterogeneous network infrastructure, and provides compatibility and interoperability with de-facto messaging standards including both JMS and DDS. One salient feature of Harmony is the holistic provisioning of dependable and predictable QoS by effectively addressing system and network dynamics, heterogeneity and failure conditions. It allows the specification of required performance properties (i.e., latency, throughput), availability and reliability models, and security constraints separately for each message topic or connection session; it further transports messages across autonomously administered domains respecting the above requirements end-to-end.

In this paper, we focus on the provisioning of end-to-end latency QoS in Harmony in the context of MOM for wide-area federated domains. This is achieved through a holistic set of overlay route establishment and maintenance mechanisms for managing the end-to-end latency, including both network latency and processing latency. In particular, the overlay routing mechanisms actively exploit diversity in the network paths and redirect messages over those links with good quality, e.g., low latency and high availability. In order to cope with network dynamics and failures, Harmony continuously monitors the link quality and adapts the routes whenever their quality deteriorates below the application requirements. Harmony can operate on top of different data transport layers. When the transport layer has built-in message scheduling capability, Harmony also adopts a novel budget allocation scheme to control its scheduling behavior and adapt to short-term network dynamics. Our experience from a testbed deployment demonstrates that Harmony can effectively manage the end-to-end latency with respect to the application requirements, despite the dynamics commonly seen in the wide-area networks.

The rest of this paper is organized as follows. Section 2 reviews our network and system models, and Section 3 presents our design of Harmony, a QoS-aware messaging middleware over wide-area networks. Section 4 describes our implementation efforts, and Section 5 reports our empirical experience from a testbed deployment. Section 6 compares the Harmony system to the literature. Finally, Section 7 concludes the paper.

2 Network and System Models

Our work targets the emerging CPS and SoS paradigms which require message-oriented middlewares to interconnect massively distributed components, services

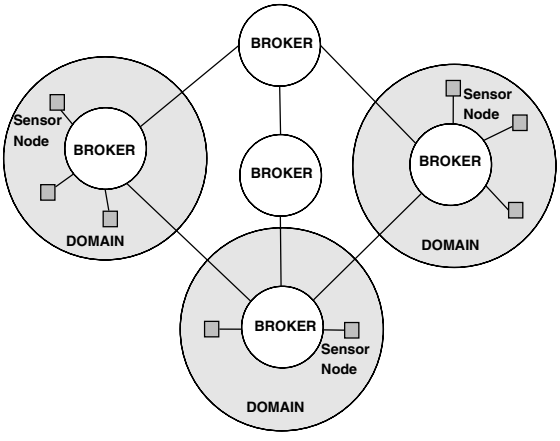


Fig. 1. Network Model

and systems over large geographic areas. Examples of such systems include Smart Grid for electricity distribution, smart city management and intelligent transportation. In all these applications, a large number of sensors and actuators are deployed in the field, and they must be interconnected with the event processing and analytics capabilities at the back end. A wide variety of event data and control directives are transported across different nodes in real time. This requires a messaging service that supports different communication paradigms, such as point-to-point, multicast and publish/subscribe. While the system we developed supports all these communication paradigms, we focus on the publish/subscribe aspect in this paper, because it provides the fundamental mechanism for asynchronous communication in distributed systems.

We assume that the endpoint nodes in the system are clustered into many local domains, and there is one broker node inside each domain. As shown in Figure 1, these brokers are inter-connected through an overlay network and collectively provide the publish/subscribe messaging service. Each endpoint node, such as a sensor, an actuator or a processing element, is attached to the local broker. There can be an arbitrary number of topics in the system, which can be defined either through administrative tools or dynamically using programming APIs. Each endpoint can publish and subscribe to one or multiple topics, while each broker can perform publish/subscribe matching, transport messages to local endpoints or neighboring brokers, and optionally perform message mediation (e.g., format transformation). Compared to the traditional approach using a single broker or a cluster of brokers, our overlay-based approach provides several architectural benefits as follow:

– *Scalability*: Each node only needs to know the local broker, while each broker only communicates with a small number of neighboring brokers. As such, we can avoid maintaining pair-wise connections, which is prohibitively expensive as the system scales up.

- *Federation*: The system is likely deployed and operated jointly by multiple organizations. In such a federated scenario, it is critical that each administrative domain can independently manage the access from/to its own nodes, which can be easily facilitated by the local brokers.
- *Heterogeneity*: The sensors are inevitably heterogeneous in a large-scale system. It is difficult, if possible, for any broker to understand all the protocols used by different nodes. With an overlay, the brokers can agree on a canonical protocol among themselves, and use a few adapters to communicate with the local sensor nodes.

Within each local domain, the sensor and actuator nodes can be connected to the broker through a variety of forms, e.g., wireless sensor networks. There have been numerous research in the sensor networking area, which is beyond our scope in this paper. Instead, we focus on providing Quality-of-Service (QoS) assurance within the broker overlay network. In the next subsection, we elaborate on the QoS model that we employ in this work.

2.1 Quality-of-Service Goals

Providing predictable QoS is an essential requirement for mission-critical applications. In particular, the messaging middleware should ensure timely and reliable delivery of critical messages, such as emergency alerts or real-time control commands. Formally stated, our goal is to provide QoS-aware publish/subscribe service in terms of *message latency and delivery rate between all matching pairs of publishers and subscribers*. Specifically, each topic is associated with a maximum delay that its messages can tolerate[1], and our system seeks to maximize the in-time message delivery rate, i.e., the percentage of messages that arrive before their respective deadline.

Note that the end-to-end delay for a given message consists of both processing delay at each intermediate broker and the communication delay between adjacent brokers. The former is affected by the load (i.e., message arrival process) of a broker, while the latter is affected by the characteristics of the network links. The broker processing delay also varies over time as each broker dispatches messages on multiple topics, and the messages may arrive in burst. Furthermore, since the sensors and actuators are deployed over a large geographic area, they will inevitably operate over wide-area networks, where the link quality fluctuates due to the dynamic traffic load. While some applications may employ dedicated networks, in general we do not assume the underlying network provides any QoS assurance. Such a relaxed network model allows our system to be applicable in different deployment scenarios, but it also poses challenges to our design as the messaging service must cope with such network and system dynamics, and ensure the end-to-end latency requirement is continuously satisfied.

[1] We consider per-topic latency requirement for ease of presentation. Our system can be easily extended to provide different QoS for individual publishers and subscribers.

3 Design

In this section, we present the design of *Harmony*, a message-oriented middleware with QoS awareness for wide-area publish/subscribe communication.

3.1 Overview

In order to meet the end-to-end latency requirements, our basic idea is to use overlay forwarding to bypass any congested network links or overloaded brokers, and to properly manage the network resources based on the message priorities. These techniques have been used in the literature for improving the QoS of point-to-point communication in the Internet [6][7][8]. However, there are a few non-trivial challenges in the context of publish/subscribe communication, where a topic may have many distributed publishers and subscribers. First, how can we establish QoS-aware overlay routes that interconnect all publishers and subscribers of a given topic, and adapt these routes in response to network dynamics such as link congestion and broker failures? Second, how can we coordinate the brokers along a route to collectively ensure the end-to-end latency performance?

Harmony addresses these challenges by a holistic set of overlay route establishment and maintenance mechanisms. Specifically, the brokers exchange control messages among themselves to discover remote subscriptions, and employ a distributed protocol to establish end-to-end overlay routes that satisfy the latency requirements. To handle network dynamics, each broker has a monitoring agent that keeps track of the latest processing latency and network latency to its neighboring brokers. These measurements are propagated among the brokers and used in the path computation to continuously find QoS-satisfied overlay routes. These overlay routing mechanisms can work with any data transport layer that supports publish/subscribe communication. Nevertheless, when the transport layer has additional message scheduling capability, Harmony allocates latency budgets for different topics at each hop, which are used to decide the scheduling priority of different messages at transmission time. This way, the system can handle short-term latency increase at one broker by increasing the latency budget at this broker, while reducing the budgets at other brokers. When the latency changes go beyond what can be handled by shifting budgets, however, new routing paths are computed to avoid congested links or overloaded brokers.

3.2 Overlay Routing

For simplicity, we assume that the set of brokers is known in advance, and the topology of the broker overlay is also decided a priori. Nevertheless, these brokers and links may fail and recover at any time. This assumption is reasonable in many application scenarios because the broker deployment only changes at very coarse timescales (e.g., once in a few weeks). In cases where brokers do frequently join and leave, a dynamic topology maintenance scheme is needed to adjust the overlay topology in runtime. We leave this issue for future study.

In general, there are two approaches for routing, namely link state (e.g., OSPF [9]) and distance vector (e.g., RIP [10]). While each approach has its own merits, our design follows the link state one which, as explained later, is more suitable for our specific context. We also employ several novel techniques to support QoS in distributed publish/subscribe communication.

Finding Subscribers. As discussed in Section 2, each endpoint can subscribe to any topic at any time. Such subscriptions are sent to the local broker which this endpoint is attached to. Each broker maintains a *local subscription table* to record which topics each local endpoint subscribes to. The brokers then propagate these topics to other brokers. As a result, each broker knows which topics any other broker needs; it maintains such information in a *remote subscription table*.

When an endpoint publishes a message on a topic, say T, the message is sent to the local broker. This broker first checks the local subscription table and transmits to all local subscribers of T. It also checks the remote subscription table to finds all remote brokers that subscribe to T, and sends the message to these brokers using the overlay routes. Upon receiving this message, these brokers further forward it to their respective local subscribers. As such, the message will eventually arrive at all subscribers of topic T in the system.

Monitoring and Link State Advertisement. Similar to OSPF [9], every broker periodically advertises its link states, including the measured processing latency for each topic and the network latency to each of its neighbors. Such link states are propagated to all other brokers through a simple neighbor forwarding mechanism [9]. Asa result, each broker has a local copy of the entire *network map*, i.e., the broker overlay topology with the latest latency measurements for all nodes and links.

Each broker employs a monitoring agent to measure processing and network latencies. It periodically pings neighboring brokers to obtain network latency. We use Exponentially Weighted Moving Averaging (EWMA) to avoid sudden spikes and drops in the measurements. On the other hand, if a neighbor fails to reply to three consecutive pings, it is considered to have failed and the link latency is marked as ∞. The monitoring agent also keeps track of the broker processing latency, including the time spent on publish/subscribe matching and the queueing delay. Both latency measurements are included in the link state advertisement so that each broker can build a complete network map.

QoS-aware Multipath Route Computation. For both resilient and in-time message delivery, Harmony employs *multipath* routing in which a message may be delivered to the subscribers via multiple parallel paths. Since every broker maintains the complete overlay topology from the link-state advertisements, it can compute the QoS-satisfied paths individually and use a source routing protocol, which will be described shortly, to establish these paths. In what follows, we consider *resiliency level* (or simply *resiliency*) as the probability of delivering a message end-to-end over one or more paths, which can be measured over long periods of time. We provide a path computation algorithm that takes into

account such failure probabilities towards choosing the most resilient combination of parallel paths. The failure probabilities of brokers and links are assumed to be known in advance, while our algorithm can accommodate various definitions of resiliency such as [11] or using historic information. For example, the percentage of time that a broker is *available* in a specific operational period of time can be extracted from traces such as the all-pairs-pings service.

Our algorithm takes as input the overlay network topology, the failure probability of each broker and each overlay link, the number of multipaths needed n, a delay constraint D and a maximum search depth k. The goal is to compute the n-multipath that provides the highest resiliency while satisfying the delay constraint. It first uses the k-shortest paths algorithm in [12] to find the k paths with the shortest delays between a source and a destination, in the order of increasing delays. It then excludes paths that exceed delay D. For the remaining k' paths we apply the provided failure probability of each broker to compute the resiliency of the remaining paths as follows: A path is considered available only when all brokers and all links along that path are also available. Thus, the resiliency of a path can be computed as $Pr(E) = \Pi_{i,j}(1 - p_i^n)(1 - p_j^l)$, where $Pr(E)$ is the resiliency of the path, and p_i^n and p_j^l are failure probabilities for brokers and links respectively. The algorithm then computes the resiliency of all the n-path combinations within the remaining k' paths, using *inclusion-exclusion* to compute $Pr(Q)$, i.e., the resiliency of the multi-path of n paths.

$$Pr(Q) = \sum_{j=1}^{n}(-1)^{j+1} \sum_{I \subseteq \{1 \ldots n\}, |I|=j} Pr(E_I)$$

where, I is a subset containing j of the n paths, $Pr(E_I)$ is the probability that all the j paths are operational, meaning their brokers and links are all on. The sum is done over all subsets of size j, and over all sizes of j (from 1 to n).

Observe that the selection step is of exponential complexity due to its combinatorial nature. Another observation is that when adding an additional path say, p_i to a multipath Q the resiliency of the new multipath $Q \cup p_i$ is *at least* equal to Q. This observation motivates the utilization of a *branch-and-cut-based* heuristic search. We construct a tree, the root of which is the complete set of paths. Each broker of the tree represents a multipath. For each broker of the tree, its children are associated to all its sub-paths. Clearly, when a broker does not satisfy a resiliency value, none of its children will; thus it can be safely eliminated along with its children.

QoS Route Establishment. In OSPF, each node independently runs Dijkstra's algorithm to determine the shortest path to every other node, and then populate its routing table accordingly. We do not directly apply this method in our broker overlay due to the need for controlling per-hop latency budget, as we shall describe in Section 3.3. Because each node on a route makes independent and possibly different decisions on how to reach the destination, the end-to-end routes change frequently; no single node can control the route. This makes it difficult to apply the budget allocation technique on a hop-by-hop basis.

Instead, we employ a novel source routing scheme, where a publisher broker locally computes the routes to all destinations (i.e., matching subscribers), and

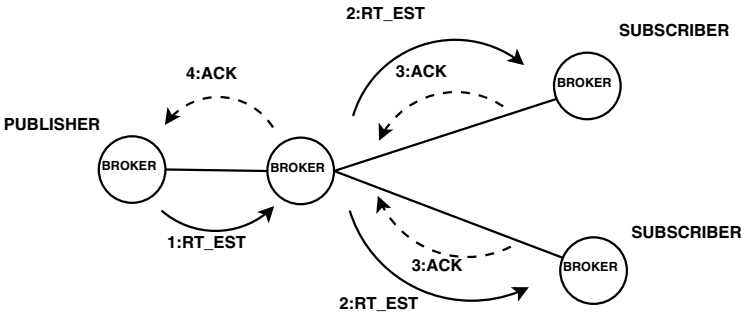


Fig. 2. Route establishment example. Numbers indicate the sequence of an operation.

uses a signaling protocol to set up these routes. As illustrated in Figure 2, the source node sends a route establishment (RT_EST) message to its next-hop neighbor on a route. The RT_EST message contains the topic name and all intermediate brokers on the route.

Upon receiving this message, a broker first checks whether it is the destination on the route. If so, it sends an acknowledgment to the upstream node from which it receives this message. Otherwise, it extracts its own next hops from the routes and forwards this RT_EST message to its next hop broker. When a node receives an acknowledgment from its downstream broker, it inserts the <*topic,next_hop*> pairs into its routing table, and then acknowledges to its own upstream node. Eventually, the source node receives the acknowledgment and the path is established. The process is repeated periodically to ensure the persistence of all QoS paths.

To briefly summarize, our scheme differs from OSPF in two fundamental aspects: 1) In OSPF, each node independently decides its next-hop nodes. In our scheme, the source node decides the entire routes. 2) In OSPF, a new link state advertisement may trigger an intermediate node to update its routing table, thus changing the end-to-end routes. In our scheme, once the routes are established, they remain fixed until the source node tears them down. To adapt to network dynamics, we employ a QoS-driven route maintenance mechanism.

Route Maintenance. Harmony updates the overlay routes only when they cannot meet the latency requirement. This could happen when the route is disrupted by broker failure or network outage, or when the route quality deteriorates as the brokers are overloaded or the network is congested. All these cases can be easily detected by a source node, because it receives link state advertisement from all other brokers[2]. Specifically, when a source node receives a link state update, it checks whether the reported latency affects any of its routes. If so, it updates the end-to-end latency of the current routes and compares it to the latency requirement. If the requirement is still satisfied, no action is taken.

[2] Assuming the overlay is not partitioned by the failures.

Otherwise, it re-computes a new set of routes and establishes them using the signaling protocol as described above.

When routes need to be updated, a task similar to the route establishment is performed, with the difference that routing tables are updated *incrementally*. In particular, the source compute the delta-path between the previous and current paths and sends out a route establishment (RT_EST) message the contains the list of new links as well as the list of obsolete links. Upon reception, a node will perform a similar operation as above, i.e. forward (RT_EST) to current and new downstream nodes but only wait for replies from its new downstream nodes. As soon as acknowledgments are received, the routing table is updated with the new downstream destinations and cleared of its removed links. This technique ensures that no flow will be interrupted while the update process is executed.

3.3 Latency Budget Allocation

The Harmony overlay routing mechanisms can work on top of many different data transport layers. We have integrated the system with TCP/IP transport, a JMS-based publish/subscribe transport, and a real-time transport [13] with built-in message schedulers. In this subsection, we discuss how we take advantage of the scheduling capability in [13], which implements a laxity-based scheduling algorithm [14]. While message scheduling provides an important QoS mechanism of proactive network resource management, it does not always lead to globally desirable performance. In particular, the multiple brokers that a message traverses make independently scheduling decisions, and the resulting end-to-end latency may not satisfy the QoS requirement. While one could use a centralized algorithm to find globally optimal decisions based on the queue behavior (e.g., arrival process, steady states) of all brokers, such information changes fast and is difficult to maintain in practice.

Instead, we apply a heuristics algorithm where the latency margin, the different between the delay requirement and the current end-to-end delay, is divided among all brokers. This way, each broker will have some "buffer" to absorb sudden latency increases, provided they are small enough compared to the margin.

Consider a broker B which is currently on the forwarding routes for a set of topics $T_1, T_2, \ldots, T_I$. Let D_i be the end-to-end latency requirement for topic T_i. The routes for topic T_i has K_i hops, and the measured latency at each hop is d_i^j, where $1 \leq j \leq K_i$.

Our intuition is to give higher priority to those topics whose end-to-end latency is approaching the bound. To do so, we calculate the *end-to-end latency margin* for each topic (say T_i) as:

$$L_i = D_i - \sum_{j=1}^{K_i} d_i^j \tag{1}$$

We equally split this end-to-end latency margin among the K_i hops in the route. Thus the *per-hop latency margin* for topic T_i is:

$$L_i^j = (D_i - \sum_{j=1}^{K_i} d_i^j)/K_i \tag{2}$$

Now the broker B can sort the topics in an increasing order of their per-hop latency margin. That is, the first topic has the smallest margin, thus should have the highest priority. Since laxity-based scheduling is used by the transmission queue, a high priority can be enforced by assigning a small latency budget for this topic. In general, for the n-th topic in the sorted list, we can assign a latency budget as (where δ is a step parameter):

$$LB_n = \min_{1 \le i \le I} T_i + n \times \delta \tag{3}$$

Note that equal splitting is one simplest method for allocating latency margin among the brokers. It allows coordinated scheduling across brokers such that messages close to their delay bound get preferential treatment. We leave other forms of budget allocation, such as differentiated splitting, as future work.

4 Implementation

We have implemented the Harmony system within *IBM Websphere Message Broker (WMB)*, an industry-leading messaging platform. WMB introduces the concept of *message flows*; a message flow comprises of one or more incoming connections, a message processing component and one or more outgoing connections. Incoming connections are used by local domain applications to access the Harmony messaging service. Our implementation allows the applications to access the messaging service via standard *Java Messaging Service (JMS)* APIs [4]. Thus, those legacy applications that are already JMS-compatible can readily switch to a Harmony-enabled system, while JMS adapters can be easily built in order for non-JMS-compatible applications to leverage Harmony. Finally, Incoming and outgoing connections are also established to interconnect brokers across the wide area network.

Harmony control sits between the incoming and the outgoing connections, handling the process of routing various messages to the appropriate outgoing connections. In this way, WMB acts as the integrating agent between the Harmony routing control layer and the data transport layer. Therefore, Harmony routing control layer remains decoupled from any specific transport.

4.1 Topic Structure and Data Forwarding

To facilitate message forwarding, Harmony defines a different topic name space and naming convention to make a clear distinction between (i) topics coming from and destined for the local domain applications, and (ii) topics coming from and destined for the wide-area broker overlay. Harmony will then handle the topic name transformation from local domains to wide-area overlay. More precisely, in

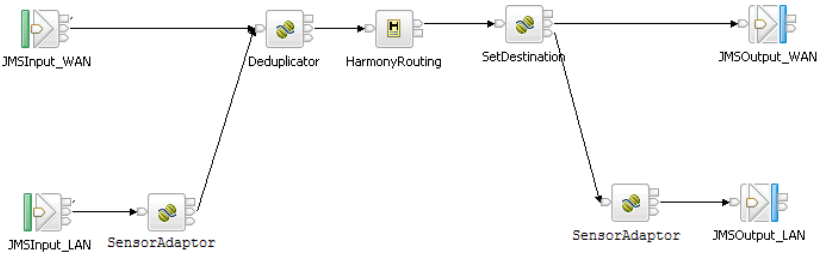


Fig. 3. WMB flow implementation of a Harmony overlay broker

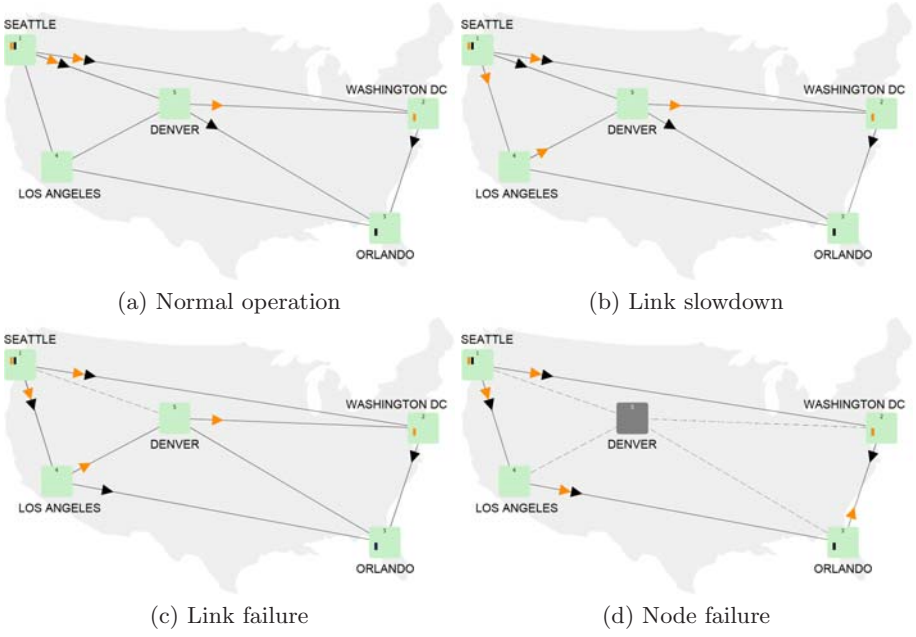
the local domain, a global topic name `T` is transformed into the form `/src/T` when forwarded to Harmony and `/dst/T` when sent out from Harmony. At the overlay, topic `T` will be transformed according to the destination as `/destID/T`. This novel forwarding approach significantly simplifies the routing process by directly leveraging the underlying publish/subscribe infrastructure, without requiring for a separate forwarding protocol. Moreover, it can be readily used among different publish/subscribe engines beyond the current JMS implementation.

The overall implementation is illustrated in Figure 3 where the actual Harmony WMB flow components are shown. Two JMS input components are seen, one subscribing to local domain topics application publications (JMSInput_LAN) and one for incoming messages from remote brokers (JMSInput_WAN). Messages topics from the LAN are transformed via the Sensor Adapter component to internal Harmony names. Then, these messages along with incoming wide area messages are forwarded to the routing component which maintains the per-topic routing destinations. A de-duplication component removes possible duplicate messages received at the local node which could occur in the case of multipath routing. Finally, similar to the incoming messages, JMS output components are used for publishing out local domain (JMSOutput_LAN) and wide area messages (JMSOutput_WAN) according to destinations provided by the Harmony routing component.

5 System in Action

We have deployed Harmony in several distributed testbeds across the nation. For illustration purpose, we present a simplified operational example in which five brokers are each deployed at a major communication hub, namely Los Angeles, Seattle, Denver, Washington D.C. and Orlando. The presentation of the scenarios is facilitated by *Harmonitor*, an administrative tool for real-time visualization of the Harmony system, such as node/link status and per-topic paths.

In the scenario illustrated, two topics are published by the Seattle broker (more precisely, application endpoints attached to the Seattle broker). The first topic is subscribed by the Washington D.C. broker, while the second by Orlando. The topic to D.C. is considered of higher priority as its required end-to-end



(a) Normal operation (b) Link slowdown

(c) Link failure (d) Node failure

Fig. 4. View of the deployed network from Harmonitor

latency is lower than that of the other topic. Figure 4(a) indicates the multipaths for each topic. Additional load is then introduced on the link between Seattle and Denver so as to slowdown that particular link, enough for the QoS of the first topic to be violated. As shown in Figure 4(b), Harmony provides differentiated service based on topic deadlines, and thus re-routes the higher priority topic away from the problematic link and through the Los Angeles broker. Note that while the second path is being reconfigured, data continue to flow within the QoS budget along the first path. In Figure 4(c), the previously slowed-down link is completely failed. The route for the topic that was flowing along the failed link, is immediately reconfigured to restore the multipath via the Los Angeles broker. Again observe that data delivery persists via the second path while the broken link is identified and the routes re-established. In the final Figure 4(d)), the Denver broker fails. The path that was routed via Denver is reset to forward traffic around the failed node from Los Angeles to Orlando and finally to Washington D.C.

6 Related Work

Message-oriented middleware has been widely used in today's enterprise IT infrastructure for integrating different applications and services in an SOA

environment. While these systems (e.g., IBM WebSphere MQ) provide essential features of reliability, security, transactionality and persistence, there is little consideration for real-time QoS such as end-to-end latency. Also, they are typically deployed within one or a few well-connected data centers. In contrast, Harmony is designed for a different set of application domains that need to integrate distributed sensors and actuators with back-end processing capabilities over wide-area networks, with an emphasis on QoS in the messaging service.

In recent years, overlay networks have been employed in an effort to provide QoS in the Internet. For example, overlay routing has been shown effective for providing resilient communication by recovering from Internet path failures [6], or increasing the available bandwidth between end-hosts by avoiding the bottleneck links [15]. Several strategies for selecting the alternative overlay paths are studied in [7]. The benefits of overlay routing are also established through rigorous analysis in [8]. Our work is inspired by these existing research efforts, but it studies a different problem of improving end-to-end latency for publish/subscribe communication through a broker overlay network. We also present an integrated routing and scheduling framework, with novel techniques in both layers.

The broker overlay in Harmony also resembles a Service Overlay Network (SON) [16,17] in that the overlay nodes are deployed at strategic locations to provide specific services. In our case, the services provided by the brokers are publish/subscribe matching and potentially message mediation. However, there is one fundamental difference between Harmony and SON: The brokers in Harmony collectively provide the publish/subscribe service, while each broker in SON independently provides a service. There are several proposals for assuring QoS in a SON [17,18]. In particular, QRON [18] is a QoS-aware routing protocol that seeks to find paths satisfying QoS requirements yet balance the traffic on different overlay link and nodes. However, it only considers overlay routes between a pair of nodes, while Harmony provide QoS-aware group communication between multiple publishers and subscribers on the same topic.

7 Conclusion

In this paper, we presented the design and implementation of Harmony, a QoS-aware messaging middleware for supporting wide-area publish/subscribe communication. Harmony constructs an overlay network on top of the physical topology and provides a novel fusion of routing, scheduling and delay budget allocation to maintain the end-to-end QoS requirements. It allows for path adaptation and reconfigurations when either network outages or excessive delays occur along a delivery path. We have implemented Harmony in an industry-leading messaging platform and verified its feasibility and advantages through real deployment.

We are currently extending the Harmony system in several aspects. We plan to support dynamic topology construction and adaptation as nodes join and leave the overlay. We are also developing new path computation algorithms to accommodate multiple end-to-end QoS requirements in parallel. Finally, we plan to integrate mediation functionality in Harmony to allow applications to perform various types of actions, such as transformation and filtering, on the messages.

Acknowledgments

We would like to thank Parijat Dube, William Jerome, Zhen Liu, Dimitrios Pendarakis and Cathy Xia for their past contribution to the Harmony project. We are grateful to Maria Ebling, Francis Parr and Paul Giangarra for their support and valuable feedback. We also thank the anonymous reviewers for their insightful comments.

References

1. Lee, E.A.: Cyber-physical systems - Are computing foundations adequate? In: NSF Workshop on Cyber-Physical Systems: Research Motivation, Techniques and Roadmap (2006)
2. SOS: System of systems, `http://www.sosece.org/`
3. Comitz, P., Pinto, A., Sweet, D.E., Mazurkiewicz, J.: The joint NEO Spiral 1 program: Lessons learned, operational concepts and technical framework. In: Proc. Integrated Communications, Navigation and Surveillance Conference, ICNS (2008)
4. JMS: Java messaging service, `http://java.sun.com/products/jms/`
5. DDS: Data distribution service for real-time systems, `http://www.omg.org/technology/documents/formal/data_distribution.htm`
6. Anderson, D., Balakrishnan, H., Kaashoek, M., Morris, R.: Resilient overlay networks. In: Proc. ACM Symposium on Operating Systems Principles, SOSP (2001)
7. Fei, T., Tao, S., Gao, L., Guerin, R.: How to select a good alternate path in large peer-to-peer systems? In: Proc. IEEE Conference on Computer Communications, INFOCOM (2006)
8. Opos, J.M., Ramabhadran, S., Terry, A., Pasquale, J., Snoeren, A.C., Vahdat, A.: A performance analysis of indirect routing. In: Proc. IEEE International Parallel and Distributed Processing Symposium, IPDPS (2007)
9. Moy, J.: OSPF version 2. RFC 2328 (1998)
10. Malkin, G.: RIP version 2. RFC 2453 (1998)
11. Gu, X., Wang, H.: Online anomaly prediction for robust cluster systems. In: Proc. IEEE International Conference on Data Engineering, ICDE (2009)
12. Martins, E., Pascoal, M.: A new implementation of Yen's ranking loopless paths algorithm. 4OR: A Quarterly Journal of Operations Research 1(2), 121–133 (2003)
13. Astley, M., Bhola, S., Ward, M., Shagin, K., Paz, H., Gershinsky, G.: Pulsar: A resource-control architecture for time-critical service-oriented applications. IBM Systems Journal 47(2), 265–280 (2008)
14. Ramamritham, K., Stankovic, J.: Dynamic task scheduling in hard real-time distributed systems. IEEE Software 1(3), 65–75 (1984)
15. Lee, S.J., Banerjee, S., Sharma, P., Yalagandula, P., Basu, S.: Bandwidth-aware routing in overlay networks. In: Proc. IEEE Conference on Computer Communications, INFOCOM (2008)
16. Duan, Z., Zhang, Z., Hou, Y.: Service overlay networks: SLAs, QoS, and bandwidth provisioning. IEEE/ACM Transactions on Networking 11(6), 870–883 (2003)
17. Gu, X., Nahrstedt, K., Chang, R., Ward, C.: QoS-assured service composition in managed service overlay networks. In: Proc. IEEE International Conference on Distributed Computing Systems, ICDCS (2003)
18. Li, Z., Mohapatra, P.: QRON: QoS-aware routing in overlay networks. IEEE Journal of Selected Areas in Communications 22(1), 29–40 (2004)