Efficacy of techniques for responsiveness in a wide-area publish/subscribe system

Minkyong Kim
IBM Watson Research
minkyong@us.ibm.com

Kyriakos Karenos*
Microsoft U.K.
kyriak@microsoft.com

Fan Ye
IBM Watson Research
fanye@us.ibm.com

Johnathan Reason
IBM Watson Research
reason@us.ibm.com

Hui Lei
IBM Watson Research
hlei@us.ibm.com

Konstantin Shagin
IBM Haifa Research
konst@il.ibm.com

ABSTRACT

As the multiplicity of organizational domains often span across nations, or even continents, the need for federated communications across domains becomes paramount. Consequently, messaging middleware has become critical towards enabling cross-domain, wide-area federations. Cross-domain federation has placed increased emphasis on the need for the messaging system to provide Quality of Service (QoS), particularly with respect to responsive delivery of messages. Responsiveness, or timely delivery of messages, is critical in real-world services, such as a smart utility grid system. This study explores the efficacy of providing responsiveness in wide-area publish/subscribe messaging by evaluating several key techniques for managing latency. Specifically, this paper evaluates the following techniques: proactive best-path routing, reactive QoS-aware routing, and multipath routing. We present Harmony, a QoS-aware publish/subscribe middleware system, that adapts these techniques in order to provide responsive and high availability messaging. This study seeks to provide an in-depth understanding of how different techniques to manage responsiveness affect the end-to-end performance under various network conditions.

1. INTRODUCTION

As the organizational domains often span across nations, the need for federated communication across domains becomes extremely important, and Message-Oriented Middleware (MOM) has become critical towards enabling cross-domain, wide-area federation. With the advent of real world-aware applications, there has been increasing emphasis on *responsiveness* declared by a level of QoS provided by MOM. Two of the most common MOM paradigms are message queuing [4] and publish/subscribe (pub/sub) messaging [13, 7]. Message queuing provides buffering between pairs of hosts over point-to-point paths, an approach that is most suitable for applications requiring persistence, but less amenable to applications requiring responsiveness. In pub/sub messaging, subscribers specify their interest in messages typically by requesting a type of messages or by requesting messages that have certain attributes based on the message contents. When a publisher sends a message, subscribers who have registered their interest in the message receive it asynchronously. Pub/sub model follows a many-to-many communication pattern, allowing for a decoupling between publishers and subscribers, while the queuing model follows a one-to-one model. It is this decoupling nature of pub/sub that makes it more suitable for applications requiring responsiveness, because loose coupling allows for rapid path re-configuration when adverse network conditions are detected.

In this paper we investigate and report on the efficacy of using different techniques to provide responsive, wide-area messaging by presenting an in-depth study of a QoS-aware messaging system, Harmony [21]. Harmony is a wide-area, pub/sub system for facilitating the interconnection of disparate messaging domains over large geographic areas through an overlay network of brokers. Harmony is targeted for wide-area networks (WANs), where the path characteristics change dynamically and failures are non-negligible. A distinguishing feature of Harmony is the holistic provisioning of predictable QoS by effectively addressing network dynamics and heterogeneity. Separately, for each message topic, Harmony allows to specify performance requirements (i.e., latency, throughput), availability and reliability models, and security constraints. Harmony delivers messages across autonomously administered domains, while respecting the above requirements end-to-end.

To provide QoS-aware messaging, Harmony employs three distinct path selection techniques: 1) periodically search for the lowest latency path (proactive best-path), 2) upon detecting a violation of the latency constraint, search for a better path (reactive QoS-aware), and 3) send topics across multiple paths simultaneously (multipath). The main contribution of this paper is to present empirical results that quantify the efficacy of using these different approaches to path selection for QoS awareness in a wide-area, pub/sub system. To the authors' knowledge, this is the first study that seeks to categorically identify under what network conditions these different techniques are applicable and for each technique, quantify the benefits (with respect to latency and loss) and cost (with respect to traffic). This paper provides a quantitative assessment of Harmony's techniques that provide responsive messaging, even under adverse network dynamics, including path performance degradation and path failures. In summary, these results demonstrate that robust QoS awareness can be achieved by employing the appropriate technique based on the network conditions.

Harmony has been explored for various real-world applications, including the wide-area distribution of air surveillance data and

*This work was done while this author was at IBM.

.

demand-response management in smart electric grids. Air surveillance data, as measured by radar, needs to be delivered to various US federal agencies such as FAA, DoD, and DHS in a timely manner. These agencies typically belong to different network domains and are located in multiple cities. Thus, it is critical to provide a messaging middleware that delivers messages seamlessly over multiple network domains and is responsive to WAN dynamics, while being cognizant of delay performance. We are also engaged in a project to provide responsive MOM for smart electric grid communication. In this application, a regional electricity generation provider together with several regional utilities seek to improve generation efficiency through better load forecasting and by offering price incentives to customers who are willing to allow intelligent management of their load. In a macro sense, pricing signals flow from the generation provider to the utilities to the loads, and demand signals flow in the reverse direction. Control algorithms at each junction of the signal flow expect Harmony to manage the delivery performance of these signals, per some QoS requirements.

2. SYSTEM DESIGN

Harmony uses *overlay networks* to construct a network on top of the physical IP topology and divert the flow of data traffic through one or more broker nodes. The application endpoint nodes, such as sensors, cluster themselves and form a domain. In each domain, there is at least one broker node and the endpoint nodes may use the pub/sub messaging by connecting to the local broker node. The use of overlay middleware to extend application-layer control over the available physical topology has, in fact, been utilized in the design of many existing systems [1, 2, 11, 14, 17]. Our approach adopts the existing concept of the overlay, but further incorporates various techniques to provide a QoS-aware messaging service over the wide-area network. The goal of this paper is to study the benefits gained by enabling these different techniques. We aim to provide insights into how different techniques can be combined to meet the needs of applications. More specifically, we have incorporated three major path selection techniques in Harmony for enhancing message delivery using multi-hop overlay paths. First, to address dynamic changes in the underlying network, we periodically check the performance of the possible overlay paths and switch the existing path to the best-path. Second, to accommodate the preservation of the applications' QoS, we monitor the individual hops to identify QoS violations and react if the current path violates the required QoS. Finally, we explore multiple parallel paths *(multipath)* to improve reliability and performance of message delivery.

2.1 Proactive best-path routing

The performance of Internet paths can change dynamically. Delays between Internet endpoints can vary significantly as a result of congestion, Internet path changes, disconnections and processing delays. To adapt to this dynamic behavior, Harmony brokers monitor the delay incurred over individual overlay links by exchanging ping-style probes. Each broker then computes the average value using an Exponentially Weighted Moving Average (EWMA), with the weight of 0.25 on the new measurements, and broadcasts the information to neighboring brokers. A publishing broker then utilizes this information to compute the shortest path to each destination. Harmony constructs and maintains the end-to-end paths over multiple overlay hops using the source routing approach based on shortest-path [10].

Harmony proactively recomputes overlay paths based on the performance (e.g., delay or loss rate) of the hops that are currently available. For a particular publisher-subscriber pair, Harmony periodically compares the performance of the currently used path and the best possible path. If the delay performance of the candidate path shows improvement by a predefined threshold, then the path switches to the better performing path. Note that the use of a threshold, instead of always switching the path, serves the purpose of avoiding path flapping, which is continuous switching between paths when the delay changes frequently. Another parameter besides the threshold that affects the agility of the system is how frequently the path search is performed. To make the system respond quickly to changes, the period should be adequately short.

2.2 Reactive QoS-aware routing

Harmony allows applications to specify QoS such as an end-to-end delay requirement or deadline. While proactive best-path selection improves the performance of paths periodically, it does not consider whether the current path *maintains* the QoS requirement. We extend proactive best-path with reactive QoS-aware routing that monitors the performance of existing paths and reacts quickly when QoS violation occurs. To achieve this, Harmony uses the QoS requirements specified by the application to implement a *delay budgeting* mechanism. Since this mechanism has been included in our earlier paper [21], we only briefly describe the mechanism. We apply a heuristics in which *latency margin* (i.e. the difference between the delay requirement and the current end-to-end delay) is divided among all links along a given path. Consider a topic with the end-to-end delay requirement D, and the latency margin $D-d$, where d the current measured delay along the path. Assuming the path comprises of k links, we split the latency margin equally on each link and thus the budget at link i becomes $b_i = (D-d)/k$. We perform this process for each topic. Observe that the budget computed at each link for each topic provides an indication of the topic's urgency. Thus, we schedule the topics according to their urgency. When Harmony identifies a budget violation at a *link*, it immediately recomputes the paths and reroutes the traffic to maintain the QoS requirement.

The main difference between the proactive routing and the reactive QoS-aware routing is that in the latter, rerouting is based on the performance requirements of the application. The reactive QoS-aware routing attempts to *maintain* the particular performance that is required by the application. If the proactive routing is made extremely aggressive (i.e. frequent periodic checking and small threshold), the benefits of QoS awareness may be overshadowed.

2.3 Multipath

As we discussed above, IP paths can be interrupted, disconnected or become extremely congested, effectively becoming unusable. Unless messages are persisted, upon a link disconnection, all data can be lost. Furthermore, if a disconnection happens, even with persistence, message delivery can be delayed, causing QoS violations. Using multiple overlay paths to deliver messages can address these issues effectively.

In Harmony, we provide two basic methodologies for constructing a multipath. The first approach is that an application defines the level of "availability" for any path between publishers and subscribers. The availability is defined as the long term probability that at a given time, at least one path exists between the pair of communicating end points. This requires knowledge of the availabilities of the various links and nodes and can be extracted either by historical data or by real-time monitoring. This approach may require an arbitrary number of paths to ensure the level of availability. The second approach, which we employ herein, is based on selecting two parallel paths at any given time. We rank the paths based on their delay (using the k-shortest paths algorithm [10]) and then select the two shortest paths that maximize the availability by assuming a normal

distribution of failures across all nodes and links. If QoS is enabled, we also restrict the shortest paths to those that meet the QoS. Note that this technique will result in selecting disjoint paths, unless no disjoint path combination satisfies the QoS. By sending redundant messages through multiple paths, Harmony aims to achieve uninterrupted operation in case of failures.

2.4 Data transport layer

Harmony provides end-to-end QoS-aware messaging on top of the data transport layer. For this paper, we use a home-grown transport prototype called Tempore. We describe the key features of Tempore in this section. Tempore provides pub/sub and point-to-point messaging services over one-hop (direct link). Tempore accepts messages from an upper layer and processes them for fast and reliable delivery to a receiver or a set of receivers, either on unicast or multicast transport. Tempore utilizes the available real-time support in underlying platforms, such as RT OS or real-time Java. In the most common setting, Tempore employs the services of an Internet Protocol (IP) network. Thus, two IP-interconnected nodes can exchange messages. In such an environment, Tempore does not need to be present on the network routing components, but it is sufficient that the endpoint nodes host a Tempore instance. An automatic discovery mechanism allows publisher and subscriber endpoints to discover each other. This enables the matching endpoints to connect. The Tempore discovery mechanism makes use of UDP multicast capabilities if available. Otherwise, discovery is performed through unicast, by transmitting discovery data to a predefined list of network addresses.

Tempore provides QoS-aware messaging over a single overlay hop. Applications can specify QoS properties for each topic. Available properties include timeliness, reliability, failover, resource management, transport protocol and wire format. For example, Tempore timing guarantees cover the direct host-to-host communication. Within a process, the timing service starts when a message is submitted to the transmitter in Tempore and ends when a message is passed to the receiver in the layer above Tempore. If a timeliness requirement is violated for any reason, the receiver-side application is immediately notified.

Tempore also provides standard pub/sub messaging interfaces, including JMS [16]. Using JMS allows Harmony clients (i.e., publishers and subscribers) and Harmony brokers to communicate via a standard interface, thereby facilitating interoperability.

3. EVALUATION

We evaluate the performance of Harmony in a private computing cloud infrastructure using 5 large virtual machines (VMs), each of which has 4 cores, 6 GB memory and 70 GB hard drive. We deploy a Harmony Broker on each VM and set the path delay to emulate a wide-area network environment. In a real deployment, each Harmony broker would handle a domain, consisting of multiple publishers and subscribers. For our testbed, we put our publisher and subscriber applications on the same machine as brokers for simplicity. Publisher 1, 2, 3 are located on Node 2, 3, 4, respectively; publisher ID denotes the topic ID. Each publisher sends out messages at the rate of 100 messages/sec. All nodes, except Node 2, host one or two subscribers. Figure 1 shows the overlay path delay and the publishers and subscribers on each node.

To measure the performance of Harmony, we use two metrics: end-to-end delay and message loss. Both are defined on a per subscriber-topic pair basis; $s_i.t_j$ denotes the topic j that is received by the subscriber located at node i. The average end-to-end delay is the average latency a message experiences from when it is sent by its publisher, to when it is received by a subscriber. It includes

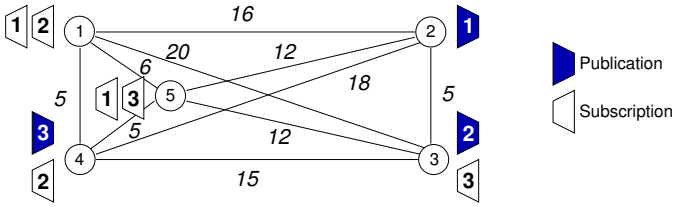


Figure 1: Testbed setup, consisting of five nodes, shown as circles. Trapezoids show the topics that each node publishes and subscribes to; the number inside of trapezoids denotes the topic ID. The path delays are in milliseconds.

the network delays and processing delays over the messaging path. The message loss rate is the percentage of messages that are published but not received by the subscriber.

We emulate different network conditions, including dynamic path delay and path failures. We use the delay shown in Figure 1 as the default values. To emulate dynamic changes in path delay, which happen when paths are congested, we increase the path delay of some paths to a large value (e.g, 100 ms). To emulate path failures, we set the packet loss rates on some paths to be 100%. We use the Linux command `/sbin/tc` to set the path delay and packet loss on a per destination IP basis.

Harmony employs multiple techniques and different combinations of techniques can be enabled to address application requirements. We evaluate four different versions of Harmony through combining techniques including proactive best-path routing (BP), reactive QoS-aware routing (QoS), and multipath (MP). First, the direct path (DP) serves as our baseline. It does not use the overlay paths at all. Instead, messages are delivered by Tempore through the direct path. Note, however, that in cases of path failure, the sender will buffer messages and wait until the path resumes and send them. Second, the best-path (BP) uses overlay paths in which messages can flow through multiple overlay nodes towards the subscriber. Periodically, the Harmony middleware tries to identify the path with the minimum delay. (The period is set to 30 seconds.) To avoid path flapping, the technique switches to a better path only if the delay can be reduced by more than a threshold (set to 20%). Third, the reactive QoS-aware routing on top of BP (BP+QoS) tries to find the best path while at the same time maintain the QoS requirements in respond to delay variations. In particular, budget assignment is performed end-to-end and whenever a QoS violation is identified, it will attempt to find another path that meets the requirement. Finally, the multipath on top of proactive best-path and reactive QoS-aware routing (BP+QoS+MP) encompasses the above feature but further employs multipath to deliver messages with minimal interruption in the flow. Our overlay-based techniques rely on the status information collected through ping-style probes, exchanged between neighbors every 5 seconds. These values are averaged and broadcasted to the neighbors every 10 seconds.

3.1 Dynamic path delay

This section provides the performance when the delay changes dynamically. We emulate the effect of cross traffic by increasing the path delay temporarily. During each run, we change the path delay 10 times, keeping the same delay values for three minutes. At each change for each path, we increase the delay value to 100 ms with probability of 0.12.* The QoS deadline for each topic is set to 50 ms. We repeat the experiment five times and report the average.

*The probability value was chosen based on the findings presented in [1], in which about this percentage of packets showed substantial improvements using overlay compared to using the direct path.

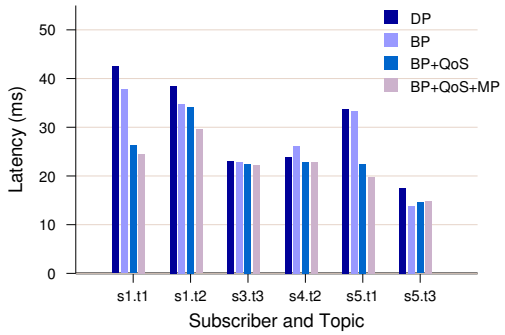


Figure 2: Latency under dynamic path delay

Sub.Topic	Direct path between	Degradation
s1.t1	node 1 - node 2	10
s1.t2	1 - 3	9
s3.t3	3 - 4	3
s4.t2	3 - 4	3
s5.t1	2 - 5	8
s5.t3	4 - 5	4

Figure 3: Number of path degradation during five runs. For each subscriber-topic pair, the table shows how many times the direct path between the publisher and subscriber for that topic experiences degradation (either in terms of delay or loss).

Figure 2 shows the latency of different versions when the path delay changes. DP obviously has the largest latency because it only uses the direct overlay path; when the direct overlay path experiences an increased delay, the latency is increased. BP approach could not always handle increased delays effectively because it checks for a better path only periodically. BP+QoS version shows smaller delay than BP. This is because an increase in delay immediately causes a QoS violation; the path delay is increased to 100 ms, which is much larger than the deadline requirement of 50 ms. Thus, Harmony reroutes the affected flows immediately when the delay is increased. BP+QoS+MP version has a slightly shorter latency than BP+QoS version; the improvement comes from maintaining two paths simultaneously and always delivering first the messages that traversed the fastest of the two paths.

There are different magnitudes of performance gains achieved by BP+QoS+MP over DP across different subscriber-topic pairs. This is because the number of path degradations between different publisher and subscriber pairs are different. Figure 3 shows the number of degradations that the direct path between each publisher and subscriber experiences during five runs. The performance gain is larger for s1.t1, s1.t2 and s5.t1 because those subscriber-topic pairs experience 8-10 degradations while the other three only experience 3-4 degradations.

To provide a deeper insight into the behavior of different versions, we chose one subscriber-topic pair, s5.t1, and looked into the CDF of latency across messages. We include the latency measurements from five runs. During five runs, this pair experienced the delay increase 8 times out of 50 three-minute slots. Figure 4 shows CDF of latency across around 980,000 messages. Both BP+QoS and BP+QoS+MP versions were mostly not affected by the increased delay, while roughly 22% of messages in DP version and 16% of messages in BP version were severly affected (having latency larger than 100 ms). The short flat area in the beginning of all curves is anomaly observed during the issue of `tc` command;

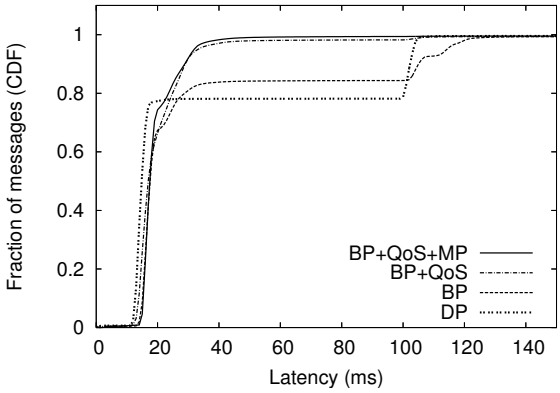


Figure 4: CDF of latency across around 980,000 messages of s5.t1, during five runs.

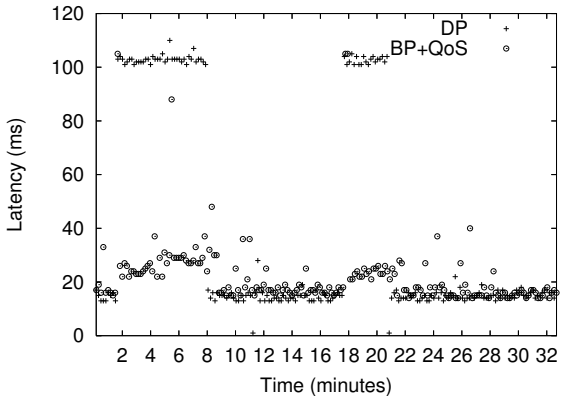


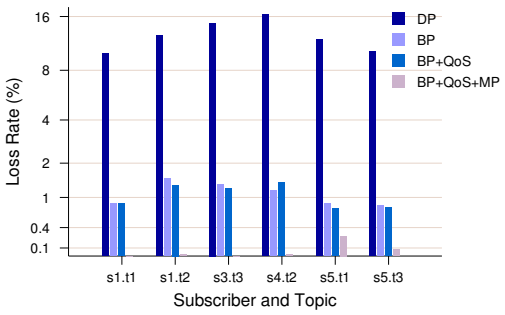
Figure 5: Latency across time of s5.t1. The path delay is increased to 100 ms three times during this one run.

less than 1% of messages have the latency less than the path delay (12 ms).

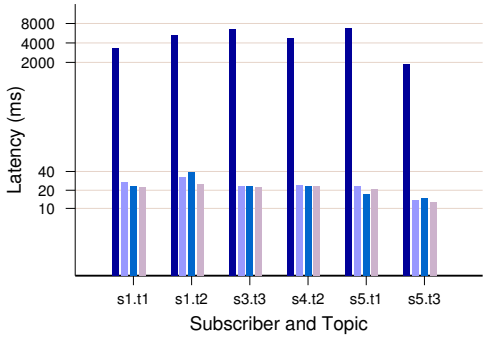
Figure 5 shows the latency over time for two representative versions, DP and BP+QoS, during one 33-minute run. To make the graph readable, only every 1000th messages are included in this graph. During this run, the delay increased to 100 ms three times. During degradations, latency of DP increased to 100 ms, while BP+QoS version produced much shorter latency by finding an alternate path, which was not affected by path degradation.

3.2 Path failures

In this section, we consider how Harmony copes with path failures. Every three minutes, we fail a path with the probability of 0.12 and repeat this process 10 times during each run. Figure 6(a) shows the average loss rate. Note that y-axis is drawn in log scale. Not surprisingly, DP shows huge loss rate (10.0%-16.4%) because when the direct path fails, all the messages are lost. BP and BP+QoS both lose some messages (0.7%-1.5%) while they are discovering the failure and finding alternative multi-hop paths. BP+QoS+MP has the lowest loss rate ($< 0.2\%$) because messages are sent through two paths in parallel; even when one of the paths fails, the other path successfully delivers messages almost always. With multipath, the loss rate is mostly zero, but it is nonzero for a few subscriber-topic pairs who experienced path failures on both paths simultaneously. Although multipath cannot guarantee no loss of messages,



(a) Loss rate



(b) Latency

Figure 6: Loss rate and latency under path failures

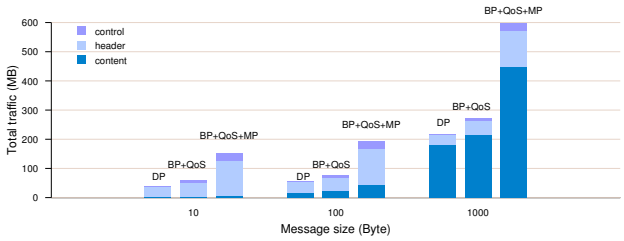


Figure 7: Overhead. Comparison of the total traffic generated by the two versions of Harmony under static delay.

it assures almost uninterrupted operation even under severe failure scenarios.

One way to avoid message loss is to use a *persistent* data transport layer. Most messaging transport services including Tempore provide both persistent and non-persistent modes of operation. When the persistent mode is chosen, all messages are written to disk. However, this introduces extra delay for all messages because disk access is costly. Thus, we use the non-persistent mode of Tempore in all our experiments. However, to show the effect of using the persistent mode of Tempore, we also run the DP version with persistent Tempore. In this case, DP has loss rate of 0%, but the delivery delay is unacceptably large, especially with respect to responsive applications. Figure 6(b) shows the latency when the paths experience failures. The y-axis is again drawn in log scale. In case of DP with persistent Tempore, all messages are persisted during the path failures, and messages are sent to the subscribers when a path recovers from a failure. The delay of the other three versions, with non-persietent Tempore, are much smaller than that of DP.

3.3 Overhead

In this section, we consider the overhead in terms of traffic. We present the results of DP, BP+QoS, and BP+QoS+MP; we omit the result of BP version because it is very close to that of BP+QoS. We perform the experiment for three different message sizes (i.e., payload) and show the overhead of control, data payload and header. The control overhead includes messages used for connection establishment and maintenance between publishers and subscribers in the data transport layer. In case of BP+QoS and BP+QoS+MP versions, it also includes path monitoring, overlay path construction and maintenance. The experiment runs for 300 sec with a rate of 100 messages/sec for each of the publishers-subscriber pairs. The path delay is static, as shown in Figure 1.

Figure 7 shows the total traffic generated across different payload sizes. The first observation is that the message size does not affect the control or message header overhead. Across different message sizes, only the message payload (content) is increased, while the size of the header remains the same; the control messages are also independent of message payload. Thus, because the total number of messages sent is the same across different message sizes, the control and message header overhead remain the same. Second, we observe that in comparing DP and BP+QoS, the control overhead traffic to maintain multi-hop paths is small. We observe that the data traffic for BP+QoS is slightly higher than DP. In general, this difference depends on how often indirect paths are chosen over direct paths. Third, the overall control overhead for the BP+QoS and BP+QoS+MP versions is reasonably small. The total control traffic for all message sizes is about 7.8 KB/sec for BP+QoS and 26.6 KB/sec for BP+QoS+MP. Since we have 6 subscribers, the control traffic is about 1.3 KB/sec and 4.4 KB/sec per subscriber for BP+QoS and BP+QoS+MP, respectively. Fourth, the BP+QoS+MP version has higher control and header overhead and also higher data traffic. For this version, the control overhead increases by 3-4 times due to the messages for maintaining the additional path. A secondary parallel path is generally longer in hops thus requiring more maintenance overhead than the primary path (i.e. single-hop, direct path versus a multi-hop indirect overlay path). Similarly, the data overhead of BP+QoS+MP is about 2-3 times that of BP+QoS version (which uses single path) because multiple copies of the same message are sent over parallel paths.

4. RELATED WORK

Approaches to latency-aware delivery for pub/sub overlay networks are studied in [9, 6]. The authors in [9] propose a technique to dynamically update the broker overlay topology to improve event dissemination latency for applications involving mobile clients. Using the pub/sub middleware JECho [22], they compare their opportunistic overlay topology to a static topology and show that end-to-end latency can be substantially improved when the broker overlay is changed to minimize the path lengths between mobile publishers and subscribers. However, their approach does not provide a mechanism to specify a latency requirement, nor a means to monitor and control latency performance relative to the requirement. IndiQoS [6] uses a resource reservation approach to provide latency awareness. As its underlying pub/sub middleware it uses Bamboo, a Distributed Hash Table, to perform broker overlay routing [20]. While having the benefits of low signaling overhead, IndiQoS assumes that the underlay supports a mechanism to reserve resources for latency and bandwidth in the paths of the overlay—an assumption that is not usually valid for an Internet-scale overlay. In addition, IndiQoS assumes a *stable* network; that is, once resources are reserved, it is assumed that latency and bandwidth will remain satisfied for all active flows. IndiQoS does not consider latency awareness during periods of delay performance

degradation, path failure, and broker failure.

Availability has been studied well [19, 8, 5, 18, 15, 12]. Similar to Harmony, these approaches seek to provide availability through fault detection, where the underlying physical network's performance is measured using periodic heartbeats. When a failed path or broker is detected, an alternative path is selected that avoids the failed broker or path. Harmony also uses the pairwise heartbeats to maintain path state for the measured delay, which Harmony employs to satisfy latency constraints when used in conjunction with priority scheduling. Thus, Harmony exploits underlay-aware techniques to provide more comprehensive control over QoS than these prior work.

Reliable message delivery using pub/sub overlays has been considered in [3, 17]. These papers propose an exactly-once, in-order, guaranteed delivery service using a content-based pub/sub. These approaches are orthogonal to Harmony because they do not consider latency-aware delivery and they are primarily suited for reasonably stable environments, such as data centers. In contrast, the QoS-aware techniques used in Harmony are designed to address WAN dynamics.

5. CONCLUSION

This paper presented empirical results to demonstrate the efficacy of using different latency-aware techniques to provide responsive messaging in a wide-area network, across autonomously administered domains. These techniques—proactive best-path, reactive QoS-aware, and multipath—were implemented in a messaging middleware, Harmony, and evaluated under various network conditions. The results demonstrated that each technique yields its strongest benefit under specific network conditions and by using all techniques in a coordinated fashion, Harmony provides a robust responsive messaging solution. When there is significant variability in the link delays, reactive QoS-aware path selection on top of best-path has superior performance. When packet loss is prevalent, the reactive approach is not always effective and multipath selection is the preferred choice to satisfy QoS requirements.

Acknowledgment

We would like to thank the following people who have contributed to or helped with the Harmony project: Donald Fox, Gidon Gershinsky, Paul Giangarra, John Hawkins, Francis Parr, Wim De Pauw, Dave Renshaw, Chris Sharp, and Hao Yang.

6. REFERENCES

[1] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of the 18th SOSP*, pages 131–145, Banff, Canada, October 2001. ACM.

[2] D. G. Andersen, A. C. Snoeren, and H. Balakrishnan. Best-path vs. multi-path overlay routing. In *Proceedings of the Internet Measurement Conference*, pages 91–100, Miami, FL, October 2003. ACM SIGCOMM.

[3] S. Bhola, R. Strom, S. Bagchi, Y. Zhao, and J. Auerbach. Exactly-once delivery in a content-based publish-subscribe system. In *Proceedings of DSN*, pages 7–6, Bethesda, MD, June 2002. IEEE/IFIP.

[4] B. Blakeley, H. Harris, and R. Lewis. *Messaging and queueing using the MQI*. McGraw-Hill, Inc., New York, NY, 1995.

[5] F. Cao and J. P. Singh. MEDYM: match-early with dynamic multicast for content-based publish-subscribe networks. In *Proceedings of Middleware*, pages 292–313, Grenoble, France, November 2005. Springer-Verlag New York, Inc.

[6] N. Carvalho, F. Araujo, and L. Rodrigues. Scalable QoS-based event routing in publish-subscribe systems. In *Fourth IEEE International Symposium on NCA*, pages 101–108, Cambridge, MA, July 2005. IEEE.

[7] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.*, 19(3):332–383, 2001.

[8] R. Chand and P. Felber. XNET: a reliable content-based publish/subscribe system. In *Proceedings of the 23rd IEEE International Symposium on RDS*, pages 264 – 273, Florianpolis, Brazil, October 2004.

[9] Y. Chen and K. Schwan. Opportunistic overlays: efficient content delivery in mobile ad hoc networks. In *Proceedings of Middleware*, pages 354–374, Grenoble, France, November 2005. Springer-Verlag New York, Inc.

[10] E. de Queirós Vieira Martins and M. M. B. Pascoal. A new implementation of Yen's ranking loopless paths algorithm. *4OR*, 1(2):121–133, 2003.

[11] E. Di Nitto, D. J. Dubois, and R. Mirandola. Overlay self-organization for traffic reduction in multi-broker publish-subscribe systems. In *Proceedings of the 6th ICAC*, pages 61–62, Barcelona, Spain, June 2009. ACM.

[12] C. Esposito, D. Cotroneo, and A. Gokhale. Reliable publish/subscribe middleware for time-sensitive internet-scale applications. In *Proceedings of the Third ACM International Conference on DEBS*, pages 1–12, Nashville, TN, July 2009. ACM.

[13] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, June 2003.

[14] T. Fei, S. Tao, L. Gao, and R. Guerin. How to select a good alternate path in large peer-to-peer systems? In *Proceedings of the 25th IEEE INFOCOM*, pages 1 –13, Barcelona, Spain, April 2006.

[15] H. Jafarpour, S. Mehrotra, and N. Venkatasubramanian. A fast and robust content-based publish/subscribe architecture. In *Seventh IEEE International Symposium on NCA*, pages 52 –59, Cambridge, MA, July 2008.

[16] JMS. Java messaging service. http://java.sun.com/products/jms/.

[17] R. Kazemzadeh and H.-A. Jacobsen. Reliable and highly available distributed publish/subscribe service. In *28th IEEE International Symposium on Reliable Distributed Systems*, pages 41 –50, Niagara Falls, NY, September 2009.

[18] M. Kumar S.D and U. Bellur. An underlay aware, adaptive overlay for event broker networks. In *Proceedings of the 5th Workshop on Adaptive and Reflective Middleware*, page 4, Melbourne, Australia, November 2006. ACM.

[19] P. R. Pietzuch and J. M. Bacon. Hermes: A distributed event-based middleware architecture. In *Proceedings of the 1st international workshop on Distributed Event-Based Systems*, pages 611–618, Vienna, Austria, 2002. ACM.

[20] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *Proceedings of the USENIX Annual Technical Conference*, Boston, MA, June 2004.

[21] H. Yang, M. Kim, K. Karenos, F. Ye, and H. Lei. Message-oriented middleware with QoS awareness. In *Proceedings of the 7th International Joint Conference on Service-Oriented Computing*, pages 331–345, Stockholm, Sweden, November 2009. Springer-Verlag.

[22] D. Zhou, K. Schwan, G. Eisenhauer, and Y. Chen. JECho-interactive high performance computing with Java event channels. In *Proceedings of 15th IEEE IPDPS*, San Francisco, CA, April 2001. IEEE Computer Society.