Carleton University, Systems and Computer Engineering, Technical Report SCE-05-14, August 2005

Evaluating the Impact of Application Design Factors on Performance in Publish/Subscribe Systems over Wireline and Wireless Networks

Abdulbaset Gaddah and Thomas Kunz

Department of Systems and Computer Engineering
Carleton University, 1125 Colonel by Drive
Ottawa, Ontario, Canada K1S 5B6

{agaddah, tkunz}@sce.carleton.ca

Abstract

The publish/subscribe interaction paradigm has recently received great attention due to its flexibility and scalability in distributed applications. The decoupling of publishers and subscribers in time and space along with the inherently asynchronous communication pattern make the publish/subscribe paradigm well-suited for mobile wireless environments. A careful design of the publish/subscribe applications is required however for achieving high performance. This paper evaluates the performance of a publish/subscribe system in wireline and wireless network domains. We first identify the factors that affect the performance of the publish/subscribe system and study their behavior in both network environments. In our analysis, we have used different test cases as a suitable means to cover a broad range of such factors and to motivate their selections. Based on our evaluation study, we observe that the performance of publish/subscribe system can be greatly affected by several factors. The results also show that wireless characterizations influence the system performance. We believe that our measurements provide valuable insights into system behavior and performance.

1 Introduction

The advances in wireless technologies and portable handheld devices like pocket PCs and cell phones have created a new paradigm of computation that allows mobile users to access different services and information while they are roaming. However, such a dynamic paradigm introduces many challenges for the developers and users of information dissemination applications. Intermittent connectivity of wireless

environments and user's mobility are good examples of these challenges. Users may get disconnected from the network due to poor network connectivity or when commuting between locations. They expect that data disseminated while they are disconnected can still be delivered upon their reconnection. Under these circumstances, the notion of middleware is crucial for supporting disconnected operations and facilitating the development of mobile information dissemination applications.

It has become apparent that traditional client/server middleware such as CORBA, RMI, and DCOM are not adequate to provide seamless support to mobile wireless computing systems. They mainly impose a tight coupling between the sender and receiver parties and rely on the permanent availability of the connection. In mobile scenarios, such type of communication is impractical as the users often move from one access point to another and inter-communicate through extremely variable connectivity. Traditional and next generation middleware solutions are discussed in more detail in [1] in terms of their suitability in mobile wireless computing domains. This research however focuses on publish/subscribe middleware that is currently considered as one of the most promising candidate paradigms for supporting mobile information dissemination applications.

The interaction style of publish/subscribe systems has been effectively used to model information dissemination applications [2], where publishers are information sources, subscribers are information destinations, and a broker entity is a router mechanism. The publish/subscribe systems naturally support a number of desirable features for mobile applications. They are characterized by decoupling the interacting parties, both in time and space, allowing them to communicate without being connected simultaneously or being aware of each other. They also employ an asynchronous communication style that allows mobile clients to issue requests for services, disconnect from the network, and collect their results later. Publish/subscribe systems moreover can efficiently filter and disseminate a significant amount of data to a large number of clients. These characteristics thus make the publish/subscribe paradigm a very good candidate for supporting mobile applications.

Although extensive research on publish/subscribe systems has been conducted [3][4][5][6], both in industry and in academia, for the fixed environments, comparatively few research has studied the behavior of these systems in the mobile wireless environments [7][8]. In this paper, we study the behavior of a distributed publish/subscribe system running on wireline and wireless environments. The main goals of our evaluation study are as follows:

- To assess the performance of publish/subscribe systems under various combinations of workload parameters.
- To explore the application design factors that impact performance and motivate their selections.
- To present the evaluation of publish/subscribe system over two types of network infrastructures and compare their performance.
- To gain insights into system behavior and understand the performance challenges.

This paper is organized as follows. Section 2 provides background knowledge on the publish/subscribe paradigm. Section 3 describes the semantics of Java Message Service

(JMS) [9], one of the most widely accepted messaging system standards. Section 4 presents the experimental environment and Section 5 discusses the results of the experiments. Section 6 describes insights gained into system behavior and performance. Section 7 draws our conclusions.

2 Publish/Subscribe Background

A publish/subscribe system is a collection of autonomous components, which interact by delivering *events* (or *messages*) from sources to interested destinations. Components that generate messages are known as *publishers*, whereas components that consume messages are known as *subscribers*. The interactions among publisher and subscriber components are coordinated by a mediated entity called *dispatcher* (or *event broker*). Publishers are the information providers that notify the outside world about the occurrence of certain events. When subscribers want to receive particular classes of events they express their interest by means of *subscriptions*. Upon the publication of a new event to the system, the event broker matches the event against all the subscriptions and then forwards it to all interested subscribers. Hence, the architecture of a publish/subscribe system relies on the mediated entity that handles the collection of subscriptions as well as the distribution of events and acknowledgements.

Publish/subscribe systems are based on two different types of event subscriptions known as *topic-based* and *content-based* subscriptions. In topic-based systems, subscribers may register to one or more topics and hence receive all the events delivered to those topics. Subscribers that share the same topic will receive a copy of each event within that topic. Content-based systems on the other hand allow subscribers to assign certain queries on the event content as part of their subscriptions. Thus, subscribers are able to receive a specific set of events within a topic. It should be noted that events do not rely on an explicit destination address set by the publishers. They are instead routed to the end destination based on their content.

The architecture of the event broker can be either *centralized* or *distributed*. A centralized architecture consists of a single broker entity that connects several publisher and subscriber components. This central entity is potentially a performance bottleneck and a single point of failure. This affects system scalability and limits the use of centralized architectures to small scale deployments. In a distributed architecture, a number of interconnected brokers collaborate in collecting subscriptions and forwarding events to the interested subscribers. Publishers and subscribers are not attached to a single broker entity; instead, they are distributed over several interconnected brokers. This can potentially reduce the network load and alleviate system scalability. The interconnected brokers can be represented in several topologies that differ in terms of their strategies in routing subscriptions and events. Two different broker topologies are presented in Figures 1 and 2.

In a hierarchical (or *multicast*) topology, the event brokers are organized in a forwarding tree that has a *root* broker and several *downward* brokers. Excluding the root broker, each broker is considered as a client to the broker at the upward level of the hierarchy. Subscribers may connect to any broker regardless of the location of the corresponding publishers in the hierarchy. Whenever a new subscription is received, the broker

propagates it upward to the root broker. Each broker on the way from the subscriber to the root broker stores a copy of the subscription. When an event is received by a broker, it is forwarded to the broker's parent. The event is also matched against all the stored subscriptions. This includes any subscriptions from downstream brokers. The broker propagates the event to any interested children (subscriber/broker) only if the matching result is true. Thus, events are always forwarded upward to the root broker, and downward towards any interested subscribers. In this topology, each broker node is a critical point of failure. Also, parent brokers are potentially overloaded as they perform extra work for their children.

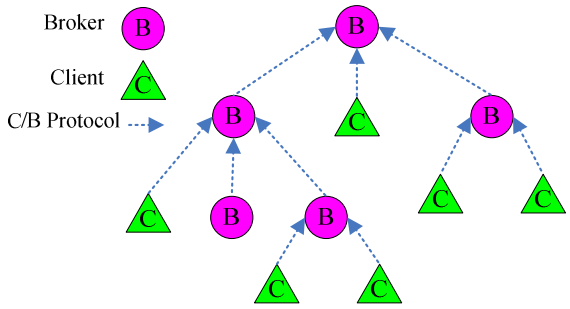


Figure 1: Hierarchical Client/Broker Topology

A peer-to-peer (or broadcast) topology consists of a set of brokers that are connected in the form of symmetrical peers. Their communication protocol supports a bi-directional flow of subscriptions and events. Each broker is responsible for a partial number of subscriptions. A publisher delivers an event to any broker that it is connected to. That broker than becomes responsible for broadcasting the event to all other brokers in the topology. When a new event enters the system, each broker checks the event against its own subscriptions and forwards it as necessary. It is apparent that the matching and forwarding overhead is reduced in comparison with the previous topology. This is because each broker needs to match events against a portion of subscriptions. In this topology, the network will be flooded by the generated events since they travel to all brokers.

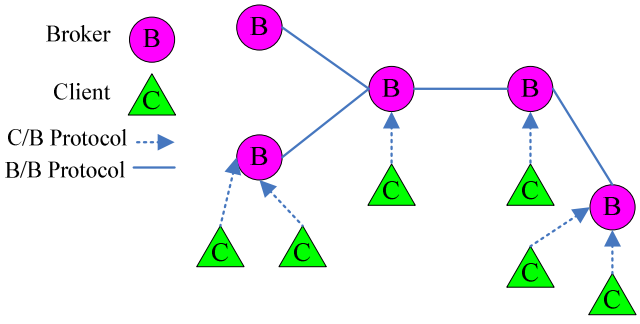


Figure 2: Peer-to-Peer Broker Topology

3 The Java Message Service Semantics

This research is based on one of the most popular messaging system standards know as Java Message Service [9] (JMS). The motivations of selecting JMS as our standard platform in this research come from surveying a set of representative publish/subscribe systems in our literature review. JMS is a Java API, developed by Sun Microsystems and partners. It allows applications to create, send, receive, and read messages. It supports two domains of messaging styles (*point-to-point* and *publish/subscribe*) and messaging consumptions (*asynchronous* and *synchronous*). A high level description of the JMS interaction architecture is presented in Figure 3.

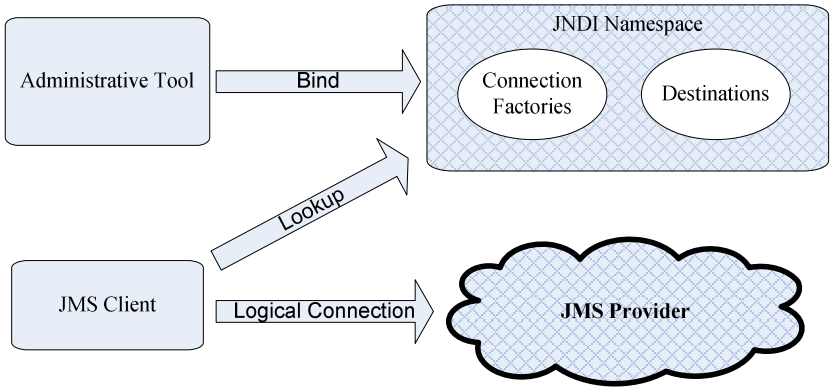


Figure 3: JMS Architecture [10]

The JMS architecture [10] consists of a number of components that are an essential part of any JMS application. The following is a brief description of these components.

JMS Provider: this is a messaging system that implements JMS interfaces in addition to the other administrative and control functionalities.

JMS Clients: these are Java programs that produce and consume messages.

Messages: these are the entities that are used to exchange information between JMS clients.

Administrated Objects: these are predefined JMS objects that are administratively created and customized and later used by JMS clients.

Native Clients: these are client applications that implement a message system's native API client instead of JMS. These refer to the applications that were developed before the availability of JMS.

The connection factories and destinations are two types of administrated objects. It is desirable that these objects are created administratively rather than programmatically since their underlying technology might vary from one JMS implementation to another. JMS administered objects are created and placed into the Java Naming and Directory Interface (JNDI) namespace by using administrative tools. A JMS client first looks up the JMS objects in the namespace and then connects to these objects through the JMS provider.

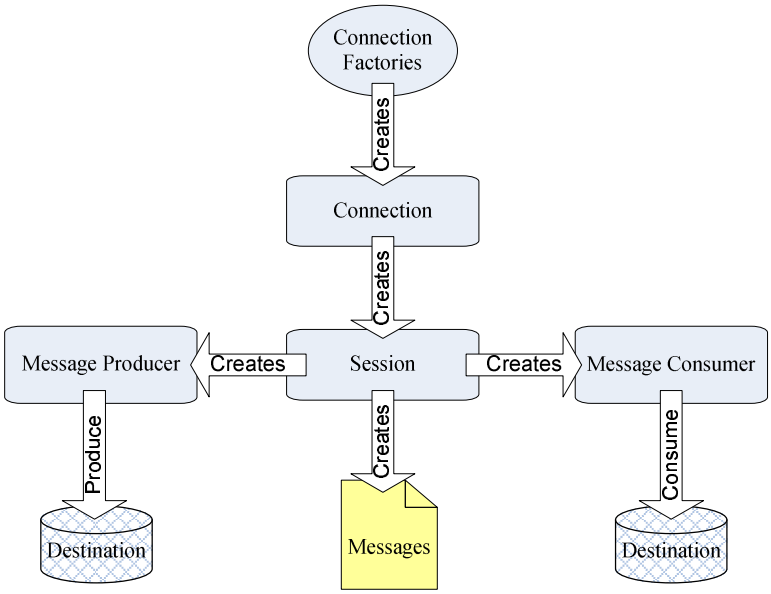


Figure 4: The JMS Programming Model [10]

Figure 4 shows the relationship between the basic building blocks of the JMS programming model [10]. A *connection factory* object encapsulates a collection of connection parameters that are ordinarily configured by an administrator. A JMS client uses this object to establish a connection with a JMS provider. A *destination* object is used by a JMS client to define the ultimate address of all messages it generates and the source poll of messages it consumes. The object contains provider-specific addresses and other configurable information. A *connection* object represents a client's active connection with a JMS provider. A connection may represent an open TCP/IP socket between a client and a provider service daemon. The connection object is used to create one or more session objects. A *session* object is a single-threaded context for generating and consuming messages. It is responsible for creating message producers, message consumers, and messages. A *message producer* is an object that is used for delivering messages to a destination while a *message consumer* object is used to drain messages from a destination. The JMS provider is responsible for forwarding messages from a particular destination to the interested consumers. A *message* is an object that carries information from the producers to the consumers. JMS messages compose of three parts: header, properties, and body.

- A *message header* has a set of predefined fields that contain values used by both clients and providers to identify and route messages.
- *Message properties* are extra fields to those provided by the message header fields that can be used to provide compatibility with other messaging systems or to create message selectors.
- A *message body* represents the part that holds the actual information. JMS supports five types of message body formats: *BytesMessage*, *TextMessage*, *StreamMessage*, *ObjectMessage*, and *MapMessage*, which provide compatibility with existing messaging styles currently in use.

JMS supports two types of subscriptions, *nondurable* and *durable*, that can be used by a subscriber to register with a JMS provider. Upon receiving a message, the provider matches the message against all the subscriptions and forwards the message to a subscriber whose subscription matches the message.

Nondurable subscriptions allow subscribers to receive messages published on their topic destinations as long as they are active. If a nondurable subscriber disconnects from the network, it will then miss all the published messages during the period of its inactivation. This scenario is presented in Figure 5. Nondurable subscriptions maintain low levels of reliability as the JMS provider does not keep the records of inactive subscribers. On the other hand, they achieve high levels of throughput since the published messages are not stored persistently for inactive subscribers, thereby reducing the overhead costs.

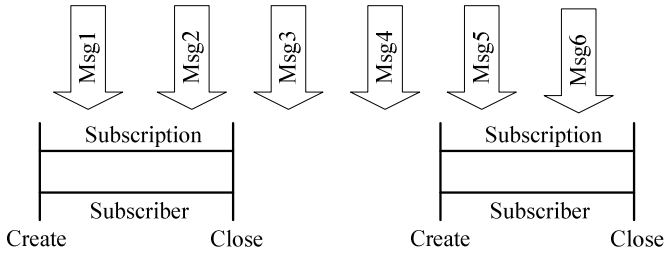


Figure 5: Nondurable Subscribers and Subscriptions

Durable subscriptions provide high levels of reliability at the cost of higher overhead. If a durable subscriber becomes inactive for a certain period of time, the JMS provider retains all the messages for the subscriber until it reactivates and consumes them. Figure 6 illustrates the concept of durable subscriptions. Thus, durable subscriptions naturally support disconnected operations in mobile environments. However, they increase the system overheads due to maintaining inactive subscriber lists, storing messages persistently, and forwarding the messages as the subscribers become active again. Thus, the throughput values of durable subscriptions are usually diminished as the number of inactive subscribers is increased.

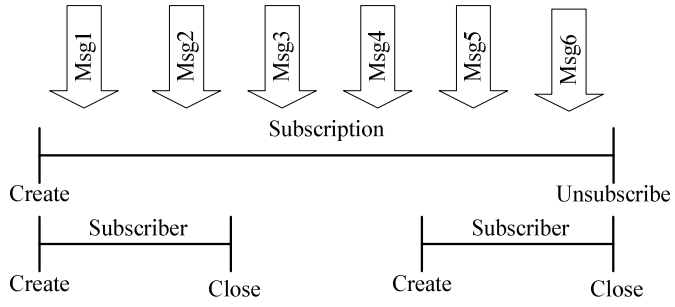


Figure 6: A Durable Subscriber and Subscription

Message consumption in publish/subscribe systems is inherently asynchronous in which there is no time dependency between message producers and message consumers. JMS supports two consumption models that can be used for consuming messages. The *Synchronous* model allows a subscriber to consume messages by explicitly invoking the

receive() method. Once the method is invoked the subscriber gets blocked until the message is received or the method timeout is reached. The *Asynchronous* model is achieved by registering an event listener object with a subscriber. This object acts as an asynchronous event handler for messages and encapsulates only one method called *onMessage()*. This method contains the necessary action to be taken when the subscriber receives the messages. Whenever a message enters to the destination, the JMS provider sends the message to the subscriber by invoking the listener's *onMessage()* method.

Subscribers sometimes would like to receive a particular type of messages rather than receiving all the messages with a destination. This can be done with the help of message selectors. A message selector is an object of type String that is used to hold conditional expressions. JMS allows subscribers to specify their message selectors as an argument when they create their subscriptions. Thus, a subscriber receives only the messages whose header and property fields match the selection syntax. The matching work is assigned to the JMS provider. As a result, the matching overhead can affect the provider as it increases linearly with the number of submitted selectors to the system.

JMS provides two modes of message delivery, *non-persistent* and *persistent*. The non-persistent mode has lower overhead since it does not require the JMS provider to log the messages in a stable storage. Accordingly, messages can be lost if the provider fails. By contrast, persistent mode introduces extra overheads as it instructs the JMS provider to ensure that messages will still be delivered even in the case of provider failure. Thus, messages are logged in an external storage until it is confirmed that they are consumed successfully.

The delivery of messages can be acknowledged by one of the following three options:

DUPS_OK_ACKNOWLEDGE: this option minimizes the overhead on the JMS provider since it is not required to prevent message duplication. The acknowledgement of message delivery is performed in a lazy manner that most likely leads to the delivery of some duplicated messages. It is recommended to use this option with consumers that tolerate duplicated messages.

AUTO_ACKNOWLEDGE: with this option, some extra overhead is introduced as the JMS provider has to ensure the delivery of messages once-and-only-once. The system automatically acknowledges the receipt of a message as soon as it has been consumed by a subscriber.

CLIENT_ACKNOWLEDGE: here, a subscriber acknowledges the message delivery by explicitly invoking the *acknowledge()* method. This acknowledgement option is similar to the previous one except it has to be done manually. If a particular message is acknowledged, this will automatically acknowledge the receipt of all messages that have been consumed by its session. For example, if a subscriber has consumed five messages and then acknowledges the receipt of the second message; all five messages will be acknowledged.

4 Experimental Setup

For our experimental study, we have reviewed several JMS implementations that are available in the public and commercial domains. The purpose of this review is to motivate our selection of the target platform for our research activities. The platform availability for public use and its true support for distributed implementation are the major criteria in our choice. Among the platforms that have met our selection criteria are OpenJMS [11], Joram [12], FiornaMQ [13], JBossMQ [14], and JavaSMQ [15]. We have selected JavaSMQ as a base platform for our research work. JavaSMQ is considered a robust, reliable, and scalable JMS implementation [10].

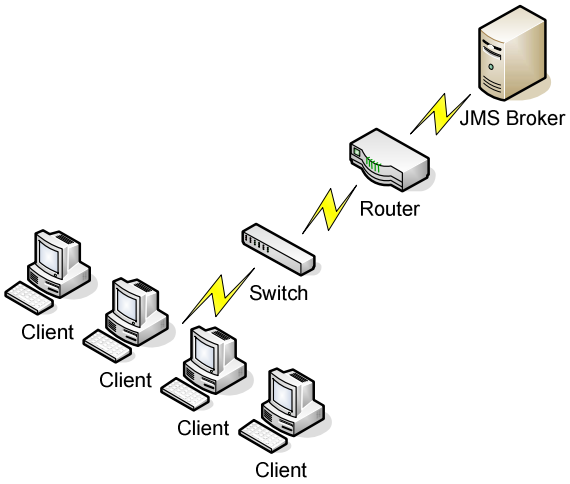


Figure 7: General View of Experimental Environment

4.1 Experimental Environment

Figure 7 illustrates a general view of our experimental environment. Experiments were performed by executing the publish/subscribe system on an overlay network of six Intel based Pentium 4 nodes running RedHat Linux operating system version 9 and inter-connected by a 100 Mbps switch. One node was used for running JMS broker with its default configuration values. A router node was used for running a wireless network emulator. The sender and receiver clients were equally distributed and run on the remaining nodes. To avoid the difficulty of clock synchronization for measuring message latency, we run the sender and receiver clients on the same machine. Clients that share the same machine run in separate threads and connections, but use the same Java Virtual Machine and JMS client's library. The JVM used for running the JMS broker and the clients is Sun SDK 1.4.2 (build 1.4.2_05-b04), started with the options –Xms64m and –Xmx256m as a minimum and maximum heap size. Although this is a limited configuration for our evaluation environment, it is sufficient for investigating the system performance which is the focus of this paper.

We mapped this configuration onto two types of network environments. The first one represents the scenario where the sender and receiver clients are connected to the publish/subscribe system through wireline links, while in the second one the clients are connected to the publish/subscribe system via an IP network tunneled through a wireless network emulator called NistNet [16].

NistNet is well-known software that is implemented as a kernel module extension to the Linux operating system. It can be used to emulate various network environments. We used NistNet to model the characteristics of IEEE 802.11 wireless LAN network based on a set of configuration parameters such as packet delay, packet loss, bandwidth, and packet duplication. All these parameters were set to the following values: 0ms packet delay, 3% packet loss, 1Mbps bandwidth, and 0% packet duplication. In our experiment, NistNet works by emulating a point-to-point communication channel extending over an IEEE 802.11 wireless LAN network. One end corresponds to the mobile host, running a client application and the other end represents the fixed host, running a publish/subscribe access point. Using IP addresses of both ends, we configured NistNet in such a way that all IP traffic between the two ends was routed through a single adapter with a specific set of configuration parameters. This represents the realistic scenario where a wireless channel is shared by many users for their inflow and outflow traffic, thereby; they share the same resources available on that channel.

There are a number of well-known methods (i.e. simulation, analytic modeling, and measurement) used in system performance evaluation. We have selected the simulation approach to capture the effect of real system behavior and overheads that is difficult to achieve with other methods. Thus, we have implemented a Java based program to simulate all test scenarios in the two experiment sets. Our program uses JMS APIs to perform several operations such as creating a TCP connection, producing messages and asynchronously consuming messages.

We ran a number of experiments for evaluating the performance of the publish/subscribe system under different workload parameters. During the course of our experiments, seven factors were used to control the workload of the experimental system. Before presenting the results of the experiments, we briefly describe these factors and the performance measures of our interest in the following two subsections.

4.2 Workload Parameters

4.2.1 Connection Load

Connection load refers to the number of concurrent connections utilized by publishers or subscribers. Since each publisher/subscriber client uses a separate connection to send/receive messages, the overhead to handle each connection is expected to increase with the number of clients.

4.2.2 Delivery Mode

Persistent and non-persistent messages are the two types of delivery mode provided by JMS implementations. Persistent messages are stored in a stable storage (i.e. hard drive) by the broker until all interested consumers acknowledge the receipt of the messages. This mode can guarantee the delivery of messages in the event of a message server failure. However, it introduces an extra load on the system.

4.2.3 Acknowledgement Mode

A client can use one of three acknowledgement modes: AUTO_ACKNOWLEDGE, DUPS_OK_ACKNOWLEDGE, or CLIENT_ACKNOWLEDGE. In this paper, we focus on the first two modes since the last mode is similar to the first one except it has to be

done manually. The AUTO mode ensures that the system automatically acknowledges a message once the client has received it. In the DUPS_OK mode, the system is instructed to acknowledge messages in a lazy manner. In the case of system failure, the AUTO mode guarantees that messages can be redelivered only once, whereas the DUPS_OK mode allows a message server to send duplicated messages to speed up the acknowledgement processing.

4.2.4 Subscription Type

Subscribes can use two different types of subscriptions, non-durable and durable, when they register with a broker. Subscribes with non-durable subscription can only receive messages produced during their period of activation. By contrast, subscribes with durable subscription can still see their messages after they become active again. Durable subscription provides a higher reliability, but increases the cost.

4.2.5 Message Filtration

Message selectors allow consumers to receive messages of their interest. A selector is a string of a logical condition that is used to match the property values of the produced messages. For example, the selector $age > 20$ allows only the messages with an *age* property value greater than 20 to be delivered to a consumer. When selectors are applied, each message property needs to be retrieved and parsed against the selector as each message is routed. This can influence the overall performance of the messaging system.

4.2.6 Message Size

In JMS, a message consists of three parts: message header, message properties, and message body. Thus, the total length of a message is the sum of the lengths of the three parts. Since the size of the header and properties is almost the same with all the messages, we only varied the length of the message body. We evaluated the system performance against three different values of body length: 100 bytes, 2 Kbytes, and 5 Kbytes.

4.2.7 Message Body Type

JMS provides five different types of message bodies, with simple and complex formats as listed below roughly in ascending order of complexity:

- BytesMessage
- TextMessage
- StreamMessage
- MapMessage
- ObjectMessage

In general, complicated types such as MapMessage and ObjectMessage can increase the overhead cost due to data serialization and deserialization. Thus, the system performance depends on how simple or complex the data is. In our experiments, we used two types of message bodies, *BytesMessage* and *ObjectMessage* to evaluate the impact of simple and complex data formats on performance.

In the two experimental sets, the above parameters were used as the variable factors. By using various combinations of these parameters it is possible to explore different types of computation and communication bound systems. In a computation bound system, the message providers are heavily loaded whereas the message body sizes are small. A

communication bound system is characterized by large messages and/or long end-to-end latency but small service times for client requests. Different combinations of workload parameters were used in our experiments to cover a broad range of such systems.

4.3 Performance Measures

Publisher throughput: the average total number of messages per second that can be pumped to a messaging system by all publishers. It is obtained by the summation of the mean throughputs of all the publishers. This metric is a measure of system capacity for delivering messages to the topics.

Subscriber throughput: the average total number of messages received per second by all subscribers in the system. It is also obtained by the summation of the mean throughputs of all the subscribers. This metric is a measure of system capacity for delivering messages to subscribers.

Message latency: the average total time, in milliseconds, it takes messages to be delivered from a publisher to a subscriber. It is obtained by calculating the mean latency of all received messages. This is the sum of the total messages' latencies divided by the total number of received messages.

To measure the actual performance of the server, it is necessary to take multiple measurements over the duration of the experiment. Durations long enough can often discover situations of server stress and performance issues. We run all our experiments for a sufficiently long period of time which this paper refers to it as a *measurement time window*. The throughput of a publisher/subscriber is measured by using a counter in their threads to capture the number of messages produced/consumed. At the end of each test cycle, the mean throughput of all publishers/subscribers is calculated. When the measurement time window expires, the throughput of publisher/subscriber is calculated by adding up the mean throughput of all publishers/subscribers.

The results of the performance indices were captured from the measurement data obtained by applying a number of testing conditions. In each experiment, the number of test cycles was large enough to generate an interval that is less than ±5% of the mean at a confidence level of 95% for most of these performance metrics. It was ensured that client and broker machines were not the bottleneck region in these performance tests. As a result, neither clients' machines nor broker's machine exceeded 60% of CPU or memory utilization. Before running each experiment, topic destinations and message stores were purged and reinitiated to start each test with a clean slate. Publishers were producing messages as fast as possible. There was no thinking time introduced among produced messages. At the other side, subscribers were consuming messages in asynchronous manner. Each client was using a separate connection to produce or consume messages. In addition, network latencies for establishing client connections were not included in our results. We next discuss the achieved results of experiment set 1 and 2 under the above test conditions.

5 Experimental Results

We conducted two sets of experiments to cover a variety of different combinations for workload parameters. The results of these sets presented in total 50 different test cases. All the test cases were evaluated over wireline and emulated wireless environments. The overall results provide valuable insights into system behavior and performance that are important to the designers of publish/subscribe applications in general and middleware systems in particular. The following two subsections provide a detailed description and discussion of the results in both experimental sets.

Table 1: Publish/Subscribe Results of Experimental Set 1

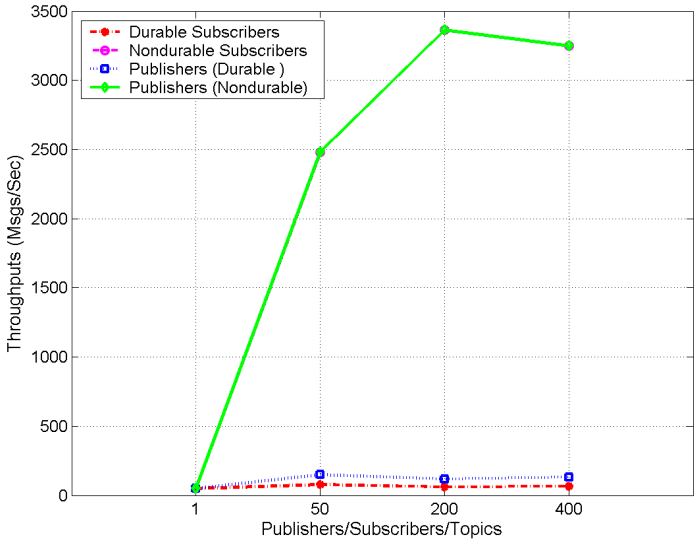
Test#	Number of Pubs/Subs/Topics	Message Size (Byte)	Subscription Type & Delivery Mode	Publisher Throughput		Subscriber Throughput		Latency	
				Wireline	Wireless	Wireline	Wireless	Wireline	Wireless
Test 01	1 / 1 / 1	100	DURABLE / PERSISTENT	49.4	27.9	49.4	27.9	3.4	143.1
Test 02	1 / 1 / 1	5120	DURABLE / PERSISTENT	49.3	7.6	49.3	7.6	5.3	365.3
Test 03	1 / 1 / 1	100	NON_DUR / NON_PERS	49.9	27.7	49.9	27.6	0.5	111.2
Test 04	1 / 1 / 1	5120	NON_DUR / NON_PERS	49.8	7.6	49.8	7.6	1.7	318.6
Test 05	50 / 50 / 50	100	DURABLE / PERSISTENT	148.2	96.8	78.4	55.2	226388.0	137402.2
Test 06	50 / 50 / 50	5120	DURABLE / PERSISTENT	52.7	11.0	28.0	10.9	165571.8	14848.1
Test 07	50 / 50 / 50	100	NON_DUR / NON_PERS	2481.4	110.7	2480.1	107.7	3.9	10640.7
Test 08	50 / 50 / 50	5120	NON_DUR / NON_PERS	1879.8	11.3	1879.2	11.0	13.5	13685.5
Test 09	200 / 200 / 200	100	DURABLE / PERSISTENT	119.1	89.1	59.8	45.7	209588.9	218178.4
Test 10	200 / 200 / 200	5120	DURABLE / PERSISTENT	56.5	13.4	29.5	12.4	218677.0	55460.3
Test 11	200 / 200 / 200	100	NON_DUR / NON_PERS	3363.2	114.2	3362.0	107.5	183.0	23648.6
Test 12	200 / 200 / 200	5120	NON_DUR / NON_PERS	1729.0	12.8	1725.4	12.8	213.8	52012.6
Test 13	400 / 400 / 400	100	DURABLE / PERSISTENT	131.4	83.6	65.7	44.6	260629.8	268192.3
Test 14	400 / 400 / 400	5120	DURABLE / PERSISTENT	58.9	15.0	30.6	14.1	251992.8	106196.6
Test 15	400 / 400 / 400	100	NON_DUR / NON_PERS	3249.2	130.7	3248.0	126.4	509.7	32827.3
Test 16	400 / 400 / 400	5120	NON_DUR / NON_PERS	1586.7	16.0	1586.7	15.1	469.3	102529.8
Test 17	1 / 50 / 1	100	DURABLE / PERSISTENT	7.7	3.5	382.9	176.7	2240.6	1834.1
Test 18	1 / 50 / 1	5120	DURABLE / PERSISTENT	7.3	0.4	363.4	19.6	3086.6	7074.3
Test 19	1 / 50 / 1	100	NON_DUR / NON_PERS	49.5	3.4	2472.8	171.8	3.3	622.9
Test 20	1 / 50 / 1	5120	NON_DUR / NON_PERS	25.4	0.4	1270.0	19.3	33.9	7233.0
Test 21	10 / 100 / 10	100	DURABLE / PERSISTENT	27.7	16.1	265.1	154.0	12668.3	15421.1
Test 22	10 / 100 / 10	5120	DURABLE / PERSISTENT	22.4	2.0	213.4	19.7	12871.3	18931.0
Test 23	10 / 100 / 10	100	NON_DUR / NON_PERS	491.3	17.3	4913.4	168.4	4.3	10729.6
Test 24	10 / 100 / 10	5120	NON_DUR / NON_PERS	207.8	2.0	2078.1	19.7	45.2	13344.4
Test 25	50 / 1 / 1	100	DURABLE / PERSISTENT	248.1	228.6	3.5	4.6	499811.9	411579.2
Test 26	50 / 1 / 1	5120	DURABLE / PERSISTENT	102.6	21.2	1.2	0.2	556062.0	564913.2
Test 27	50 / 1 / 1	100	NON_DUR / NON_PERS	2294.4	211.6	2214.8	4.5	12358.1	246694.9
Test 28	50 / 1 / 1	5120	NON_DUR / NON_PERS	77.5	20.7	22.6	0.2	489474.6	545201.9
Test 29	100 / 10 / 10	100	DURABLE / PERSISTENT	252.0	200.0	13.2	19.1	329637.3	361336.7
Test 30	100 / 10 / 10	5120	DURABLE / PERSISTENT	103.8	18.2	5.4	0.9	476159.3	396149.8
Test 31	100 / 10 / 10	100	NON_DUR / NON_PERS	2290.3	227.9	2107.6	22.6	36044.0	289654.5
Test 32	100 / 10 / 10	5120	NON_DUR / NON_PERS	204.5	18.3	152.3	1.0	152140.6	402138.6

5.1 Experiment Set 1: *Effect of Subscription Types, Connection Loads, and Message Sizes*.

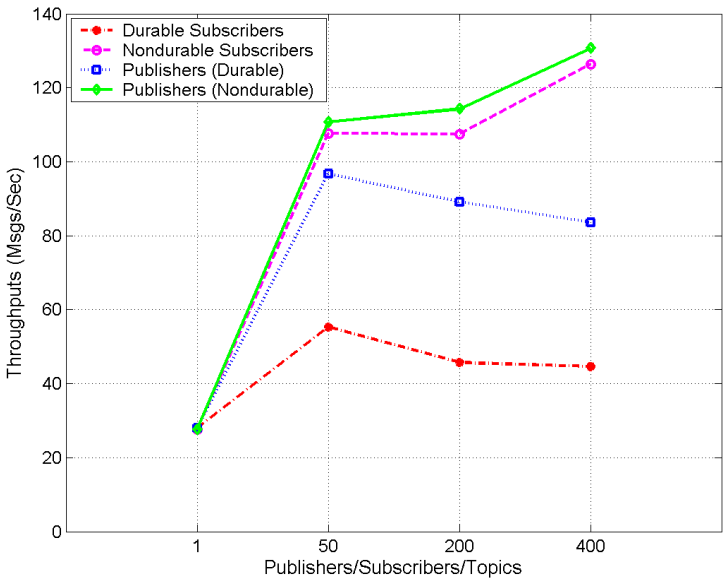
Experiment set 1 consisted of 32 testing scenarios as shown in Table 1. It presents and discusses the impact of three factors (durable and non-durable subscriptions, number of connections, and message body sizes) on the performance. The test scenarios of this set were varied in the number of publishers, subscribers, and topics. They also included two types of subscriptions, durable and non-durable, which were configured to be used with persistent and non-persistent messages respectively. In all figures of this set, we refer to the publisher that sends messages to the durable and non-durable subscribers as *publisher (Durable)* and *publisher (Nondurable)* respectively. For all messages, the message body type and acknowledgement mode were held at fixed values: BytesMessage and AUTO, whereas a message body size of 100bytes and 5Kbytes were used. To explore the impact of different connection models, we have divided this set of experiments into three different models (one-to-one, few-to-many, and many-to-few). In each model, a two factor-at-a-time method is adopted to minimize the number of experiments and to compare the effect of two parameters at a time on system performance. All of the models were executed over wireline and emulated wireless environments. The following subsections present and discuss the results achieved for each connection model.

5.1.1 One-to-One Model

In the one-to-one model, each publisher and subscriber inter-communicates via a single pre-defined topic destination. This model helps us to achieve a direct comparison between the publisher and subscriber throughputs. The first sixteen test scenarios shown in Table 1 represent the one-to-one publish/subscribe model. Figure 8 shows the average message throughputs as experienced by both subscriber and publisher clients at increasingly higher number of connections in the system. Both graphs illustrate the results obtained from test scenario 1, 3, 5, 7, 9, 11, 13, and 15 presented in Table 1. Messages with a fixed size of 100bytes each were created and sent to all receivers. Since each publisher and subscriber uses a separate connection, the number of connections in any test scenario is twice the value of the x-axis. Therefore, our test scenarios included results of 2 to 800 connections.



(a) Scenarios of Fixed Wireline Network



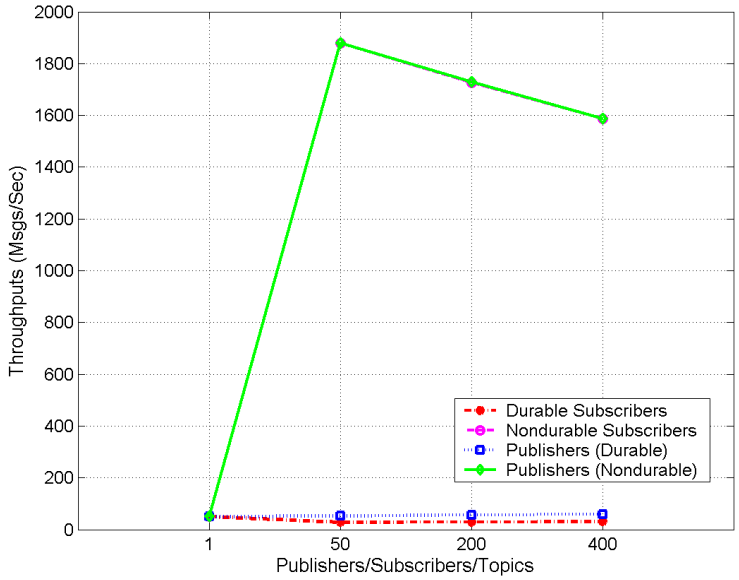
(b) Scenarios of Emulated Wireless Network

Figure 8: Publish/Subscribe One-to-One Model: 100 Byte Messages

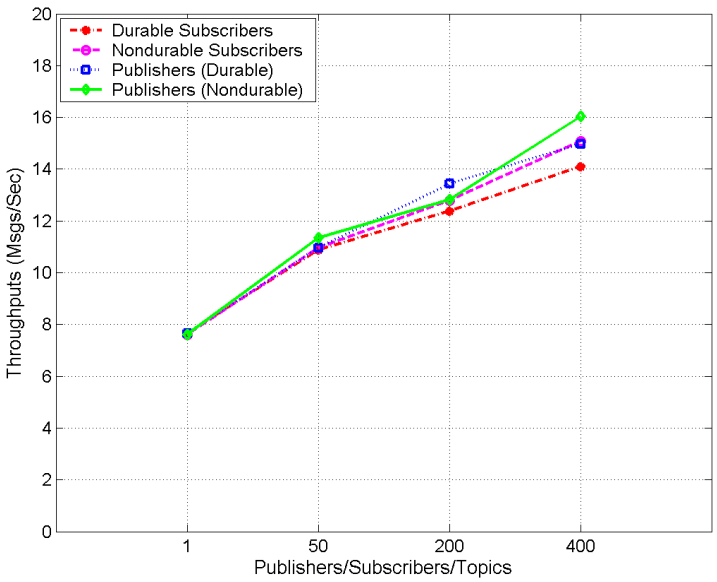
We can see in Figure 8(a) that non-durable subscription is far outperforming durable subscription. With a heavy load of connections (400×2), the average throughputs of non-durable subscriptions was 50 times the one achieved by durable subscriptions. On the other hand, at a relatively light load (1×2) the two subscription types achieved almost the same throughputs. The non-durable and durable subscriptions reached their peak values at 3361 and 78 msgs/sec respectively. The reason that durable subscription achieved lower throughput can be attributed to the overhead of processing persistent messages. Each received message gives rise to an overhead resulting in a significant drop in throughput. By comparing the throughput of non-durable subscriptions with their publishers, it is apparent that they achieved approximately similar results. This implies that almost all of the produced messages were received by all consumers within the measurement time window. By contrast, except for the test case 1, the throughput achieved by publishers is over double the subscriber throughput for durable subscriptions. As a result, more than 50% of the produced messages were received by subscribers after the measurement time expiration. As the connection load rises, the throughput results hits a knee point for both subscription types. The knee point of durable subscription occurs when the number of concurrent connections exceeds (50×2). As for non-durable subscription, the drop point appears after opening (200×2) connections. This gives some idea about the limitation of connection load with each subscription type.

The results shown in Figure 8(b) follow a similar trend of the results in Figure 8(a). However, the overall throughput has decreased in both subscription types, in particularly by an order of magnitude for non-durable subscription. This is due to the wireless channel characterizations, low bandwidth, high latency, and high error rate. Even with the wireless scenario, non-durable subscription is still outperforming in most test cases. Except for the test case 1 and 3, the throughput of non-durable subscriptions is twice that of durable subscriptions. This difference is much less than what we saw in the wireline

scenarios. The reason is that the network link becomes the bottleneck region, thereby reducing the load on the message provider. Therefore, the results of both subscriptions become much closer. Interestingly, we have not seen a knee point in the scaling curve shown in Figure 8(b) for non-durable subscription as the number of connections tops (400×2). By contrast, the peak value for durable subscription reached 55 msgs/sec with (50×2) connections, the throughput results gradually degraded beyond this number of connections. It can be noted that there is a small difference between the throughput results of non-durable subscribers and their publishers, whereas the throughput results of publishers exceeded the corresponded durable subscriptions by at least a factor of 1.7. This indicates that a large percentage of messages arrived at the subscribers after the measurement time expiration.



(a) Scenarios of Fixed Wireline Network



(b) Scenarios of Emulated Wireless Network

Figure 9: Publish/Subscribe One-to-one Model: 5KByte Messages

The results in Figure 9 correspond to the test cases 2, 4, 6, 8, 10, 12, 14, and 16 presented in Table 1. In these cases, we have increased the message body size to 5Kbytes in order to study the impact of message size on performance. Compared to 100bytes scenarios depicted in Figure 8, the throughput results of both non-durable and durable subscriptions tend to decrease with increasing message size. This is due to the fact that large messages require a considerable amount of time for processing and delivery during which the publisher remains idle until the delivered messages are acknowledged. As a result, fewer messages can be produced with a substantial end-to-end delivery time. Test results shown in Figure 9(a) indicate that non-durable subscription consistently achieves higher throughput than durable subscription, yielding better performance results. At the most pronounced case, non-durable subscription peaks at 1879 msgs/sec which is over 35 times faster than durable subscription. Almost identical throughput results were achieved by non-durable subscriptions and their publishers. This indicates that the service time of the message provider is relatively small due to the lower publication rate.

On the other hand, the throughput results of publishers in the durable scenarios exceeded that of durable subscriptions by at least a factor of 1.8. This is more likely because of the overhead introduced by processing persistent messages. In comparison with the results in Figure 8(a), we notice a more pronounced knee in the scaling curve shown in Figure 9(a) as the number of connections increases. The non-durable subscription hit its knee point at a value of (50×2) connections, whereas the knee point of durable subscription occurred just after a value of (1×2) connections.

From Figure 9(b), we clearly see the impact of message size and network characterization on performance in comparison with the results in Figure 8(b) and 9(a). The throughput results were severely diminished. The durable and non-durable subscriptions as well as their publishers approximately achieved similar throughput results. This might be a good

example for communication-bound system, where the message size and/or latency are large but the service time is small. The scaling curve in Figure 9(b) shows a linear relationship between the number of connections and the achieved throughputs. The peak values of durable and non-durable subscriptions are 14 and 15 msgs/sec respectively. Almost all the produced messages reached the consumers side. There was little affect on performance by opening up to (400×2) concurrent connections.

In Table 1, we listed the end-to-end latency in elapsed milliseconds for the first sixteen scenarios. We have defined message latency as the difference between the time a publisher sends a message and the time a subscriber receives it. The latency results of the scenarios presented in Figure 8 and 9(a) show that non-durable subscription experienced much lower latency than durable subscription. This is because of the extra overhead involved with durable subscription to provide a higher reliability. On the other hand, both durable and non-durable subscriptions experienced roughly similar latency results in the scenarios illustrated in Figure 9(b). This is due to the effect of message size and wireless network characterization; thereby both subscriptions achieved similar throughput results. It also should be noted that opening more connections causes a substantial increase in latency for all one-to-one test scenarios. This behavior is an outcome of the overhead introduced at higher number of connections. Large messages may increase the latency by an order of magnitude since they incur a longer time to process them. This is explained further with the latency results reflected by the scenarios in Figure 9. These results show a noticeable rise in the latency with a message size of 5Kbytes. We note that the latency results of non-durable subscription in wireless scenarios were significantly increased in comparison to ones in fixed wireline scenarios. A large portion of message latency was experienced during message transmission due to the low bandwidth offered by the wireless environment. This moves the bottleneck region to the wireless link and hence small service times are provided by message server. To some extent, this can benefit durable subscription where an extra overhead is introduced to process persistent messages. On the other hand, it affects the performance of non-durable subscription since the transmission delay is increased but the same service times are required to process messages.

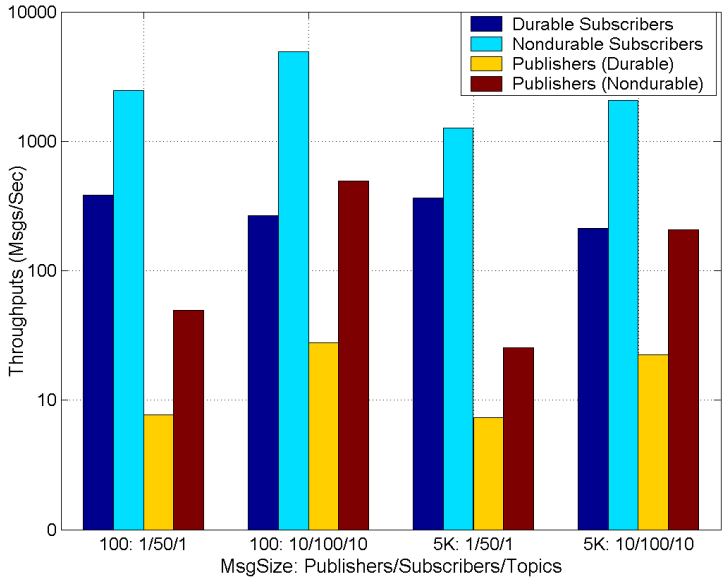
5.1.2 Few-to-Many Model

The few-to-many model presents the scenarios where there are fewer senders and many receivers attached to a particular topic destination. All messages delivered to that topic will be distributed to all receivers registered to that topic. The few-to-many model corresponds to the test cases 17 to 24 shown in Table 1. Figure 10 presents the achieved results in these cases as experienced by both subscription types and their publisher clients with small and large message size. The results of durable subscriptions, nondurable subscriptions, and their publishers are illustrated from left to right in each category of the bar chart. The first two sets of results are for 100bytes messages whereas the last two sets are for 5Kbytes messages.

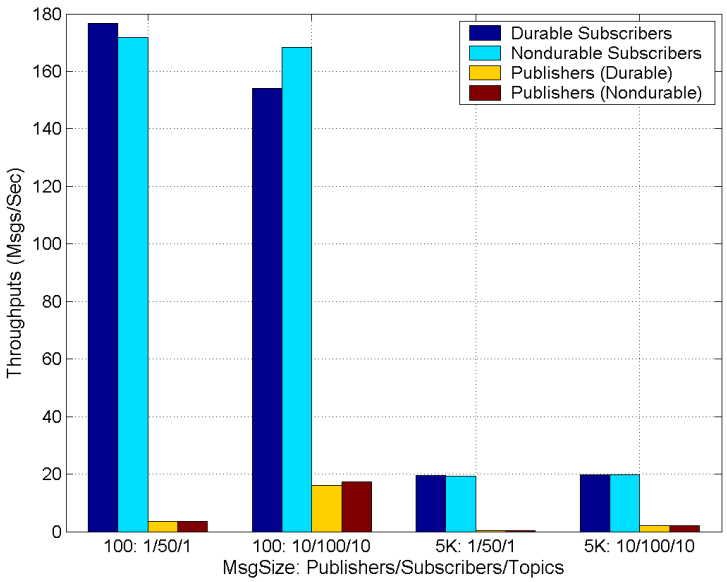
From Figure 10(a), we can see that non-durable subscription outperforms durable subscription by 6 to 18 times in the case of 100bytes message size. By doubling the number of subscribers and increasing the number of publishers and topics by 10 times, the throughput result of non-durable subscription increased by almost a factor of 2. By contrast, an inferior throughput was achieved in the durable subscription scenarios when

the number of subscriber/publisher clients and topics were increased. This is most likely because the overhead of processing persistent messages has increased. Each topic needs to store its messages persistently and updates its information table at the time of receiving a subscriber's acknowledgement. Similar results are held for both subscription types with a large message size of 5Kbytes. However, the overall throughput is decreased by almost 50% compared to the case of 100bytes message size. We can note that for either message length (100bytes or 5Kbytes) non-durable subscription achieved higher throughput results.

In Figure 10(b), we note that both subscription types achieved approximately similar throughput results. These results are much less than what was demonstrated in Figure 10(a). This is again due to the wireless channel characterization. It is apparent that increasing message length to 5Kbytes impacted the system performance. The overall throughput was decreased by more than 50% for both subscription types. On the other hand, increasing the number of subscriber/publisher clients and topics had little impact on the throughput results for both subscription types in the two cases of message sizes.



(a) Scenarios of Fixed Wireline Network



(b) Scenarios of Emulated Wireless Network

Figure 10: Publish/subscribe Few-to-Many Model

The end-to-end latency for the few-to-many model (test scenarios 17 to 24) is presented in Table 1. For the scenarios presented in Figure 10(a), durable subscription experienced much higher latency in comparison with nondurable subscription. This is mainly due to the extra overhead for processing persistent messages. Interestingly, message latency forms a linear relationship with the number of opening connections in both message size cases: 100bytes and 5Kbytes. Compared to durable subscription, there was a less noticeable increase in latency for non-durable subscription when the number of connections increased. With a large message size of 5Kbytes, the latency increased by a factor of 1.2 to 10. This is because large messages require longer processing and transmission time.

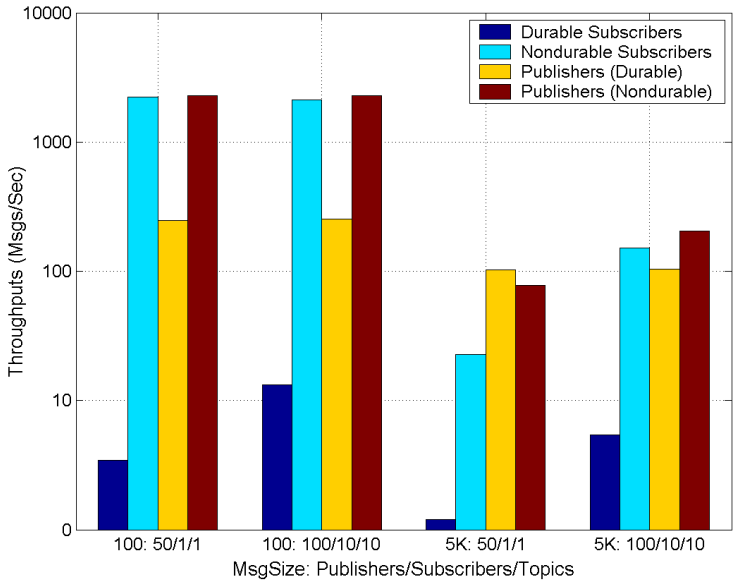
The level of latency for the wireless scenarios illustrated in Figure 10(b) was higher than the one in the fixed network scenarios presented in Figure 10(a). This is due to wireless network limitations. We note again that the latency results of non-durable subscription were significantly increased due to the same reasons explained previously. With a larger message size of 5Kbytes, the message latency increased by a factor of 1.2 to 11.6. This confirms that message size can greatly affect the system performance.

5.1.3 Many-to-Few Model

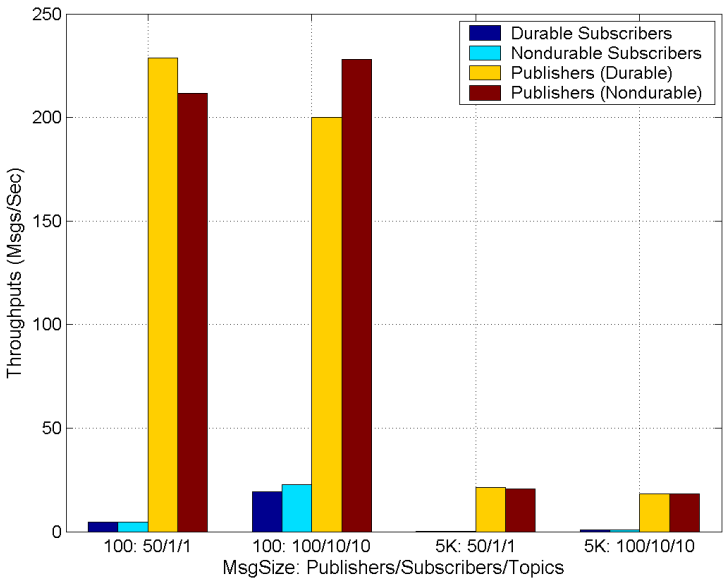
The many-to-few model presents an opposite interaction approach from the one presented in the previous subsection. Here, the number of receivers is fewer than the number of senders attached to the same topic destination. The test scenarios of this model allow us to investigate the flow-control behavior of the message server. This refers to the broker's capability for handling messages when its topics are filling or filled to capacity. The scenarios of the many-to-few model are presented in the test cases 25 to 32 of Table 1. Figure 11 demonstrates the throughput results for both subscription types as well as their publisher clients with the two cases of message size: 100bytes and 5Kbytes.

From Figure 11(a), we can see that durable subscription achieved lower throughput results in all cases compared to non-durable subscription. By contrast, under the flow-control conditions, non-durable subscription as well as their publisher clients reached impressive throughput results with 100bytes and 5Kbytes message sizes. It is apparent that a large message size severely reduced message throughputs. The throughput results of non-durable subscription and their publisher clients are relatively close. This indicates that non-durable subscription was slightly affected by the flow-control behavior. However, there is a dramatic difference between the throughput results achieved by durable subscriptions and their publishers. It can be noted that there was a slight gain in the throughput as the number of publishers increased by a factor of 2. We suspect that this behavior is a result of the flow-control mechanism used by the message server.

The graph shown in Figure 11(b) indicates that both subscription types approximately achieved similar throughput results in our emulated wireless scenarios. This also applies to the publisher clients. Compared to the scenarios in Figure 11(a), the publishers' throughputs are higher than the subscribers' throughputs in all cases. This implies that consumers are slow in draining messages out of the topic. This is due to the flow-control behavior overhead. By default, the JMS broker tends to return a JMSException to a publisher when an attempt to send a message to a full topic destination is made. This requires the publisher to catch the exception and attempt to resend the message. Hence, the publisher keeps trying until a successful attempt occurred. This introduces an extra overhead to the broker since it has to deal with all unsuccessful attempts. There is a possibility to decrease this overhead by configuring the broker to block the publisher until becomes ready to process the messages. However, this is not the default behavior of the broker. With the scenarios of 5Kbytes message-sized, the throughput results were degraded by more than 50%. We can see that there was much gain in the throughput as the number of subscribers, topics, and publishers increased by a factor of 10, 10, and 2 respectively. This is because we have more topics to fill, thereby reducing the overhead of flow-control.



(a) Scenarios of Fixed Wireline Network



(b) Scenarios of Emulated Wireless Network

Figure 11: Publish/Subscribe Many-to-Few Model

Table 1 presents the results of message latency of the many-to-few model for both durable and non-durable subscription types. In both network scenarios, durable subscription experienced a large amount of latency in comparison with non-durable subscription. Compared to the scenarios of the few-to-many model, we observed a much more pronounced increase in latency for both subscription types. This is mainly because of the flow-control behavior. We note that 5Kbytes of message length increased the latency by a factor of 1.1 to 39.5 in the scenarios of fixed wireline network. Similarly in the wireless network scenarios, the message size of 5Kbytes relatively increased the latency by a factor of 1.1 to 2.2. In most cases, adding more publishers, topics, and subscribers results in reducing the message latency. This is a result of using more topics to minimize the behavior of flow-control. We note that the latency of all non-durable subscriptions running over a wireless environment was increased by a large order of magnitude. On the other hand, a relative increase in the latency was experienced by all durable subscriptions as they run over wireless environment. This is because of the same reason as explained in Subsection 5.1.1.

5.2 Experiment Set 2: *Effect of Acknowledgement Modes, Delivery Modes, Message Filtration, and Message Body Types.*

Experiment set 2 demonstrates and discusses the affect of four factors on performance. These factors are as follows: acknowledgement modes, delivery modes, message selectors, and message body types. The experimental set consisted of 18 testing scenarios as presented in Table 2. In each test case, we used a single publisher, subscriber, and topic destination. We consistently varied the acknowledgement (AUTO and DUPS_OK) and delivery (PERSISTENT and NON-PERSISTENT) modes as well as the subscription types, durable and non-durable. We also introduced the notion of message selectors to receive a specific set of messages. For simplicity, we used only two types of message bodies (BytesMessage and ObjectMessage) to explore the impact of simple and complex

data formats on performance. In all the test scenarios, we fixed the message size to 2Kbytes. We also run all the scenarios over fixed wireline and emulated wireless environments. Next we present and discuss the achieved results of some tested scenarios due to the space limitation. For a complete list of results, readers are referred to Table 2.

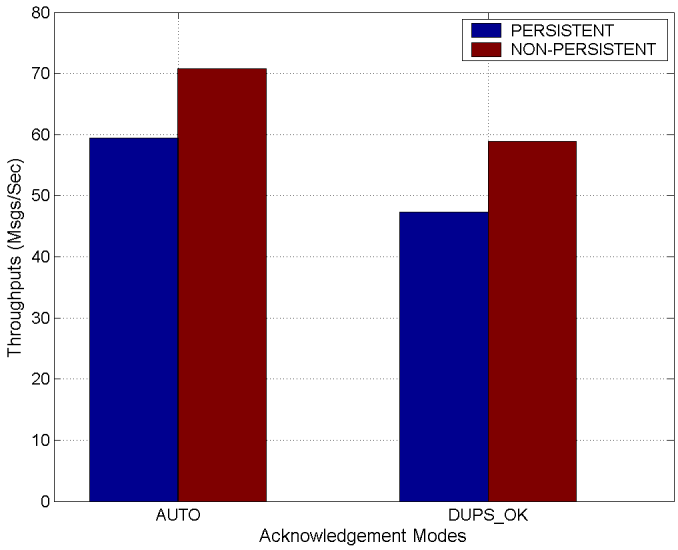
Table 2: Publish/Subscribe Results of Experimental Set 2

Test#	Subscription Type & Delivery Mode	ACK Mode	Selector	Publishers Throughput		Subscriber Throughput		Latency	
				Wireline	Wireless	Wireline	Wireless	Wireline	Wireless
BytesMessage Body Type									
Test 01	DURABLE / PERSISTENT	DUPS_OK	NO	102.29	14.28	47.31	14.28	116653.46	136.29
Test 02	DURABLE / PERSISTENT	AUTO	NO	113.03	13.99	59.44	13.99	141056.05	542.48
Test 03	DURABLE / NON_ PERS	DUPS_OK	NO	122.06	13.95	58.86	13.94	162888.40	144.71
Test 04	DURABLE / NON_ PERS	AUTO	NO	122.78	14.15	70.81	14.15	140999.99	512.43
Test 05	NON_DUR / PERSISTENT	DUPS_OK	NO	799.65	14.49	799.63	14.48	2.13	124.26
Test 06	NON_DUR / PERSISTENT	AUTO	NO	777.23	14.29	777.21	14.26	1.33	455.19
Test 07	NON_DUR / NON_ PERS	DUPS_OK	NO	802.07	14.26	802.06	14.26	1.73	119.53
Test 08	NON_DUR / NON_ PERS	AUTO	NO	786.69	14.43	786.68	14.42	1.37	473.25
Test 09	DURABLE / PERSISTENT	DUPS_OK	YES	871.07	18.67	86.56	1.91	6.38	107.21
Test 10	DURABLE / PERSISTENT	AUTO	YES	873.54	18.50	87.87	1.76	6.55	109.08
Test 11	DURABLE / NON_ PERS	DUPS_OK	YES	871.99	18.87	87.32	1.90	6.34	107.11
Test 12	DURABLE / NON_ PERS	AUTO	YES	880.04	18.90	87.81	1.86	6.29	124.76
Test 13	NON_DUR / PERSISTENT	DUPS_OK	YES	1228.76	18.65	122.35	1.80	1.25	97.12
Test 14	NON_DUR / PERSISTENT	AUTO	YES	1227.51	18.66	123.35	1.85	1.25	101.54
Test 15	NON_DUR / NON_ PERS	DUPS_OK	YES	1227.28	18.47	123.78	1.87	1.25	95.57
Test 16	NON_DUR / NON_ PERS	AUTO	YES	1226.11	18.55	123.32	1.93	1.25	98.15
ObjectMessage Body Type									
Test 17	DURABLE / PERSISTENT	AUTO	NO	467.33	14.27	46731	14.27	1.75	444.90
Test 18	NON_DUR / PERSISTENT	AUTO	NO	105.78	13.77	57.19	13.76	142271.55	537.01

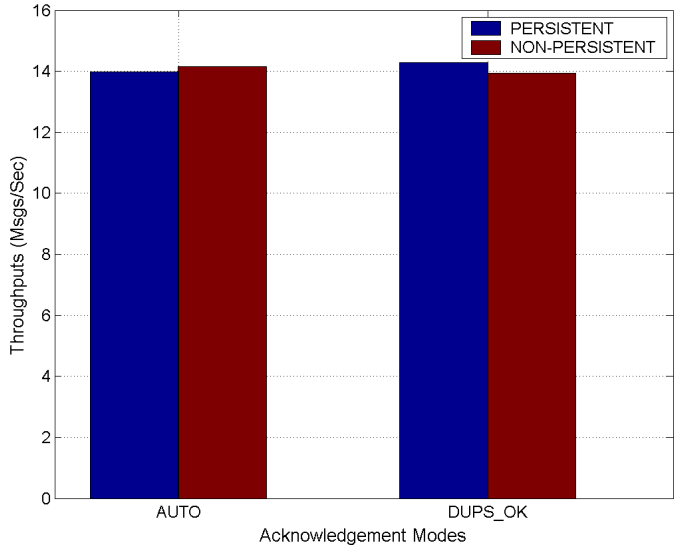
Figure 12 illustrates the throughput results of sending 2Kbytes persistent and non-persistent messages with no selectors to a topic destination. In both graphs, the left and right bars in each category present the results of the two delivery modes. Meanwhile, the bars on the left and right categories respectively illustrate the results of AUTO and DUPS-OK acknowledgment modes. These results correspond to the test cases 1 to 4 presented in Table 2.

From Figure 12(a) we note that non-persistent messages relatively achieved higher throughput results than persistent messages with both types of acknowledgement mode. This is due to the overhead required for storing messages persistently. It was noted that AUTO acknowledgement in both delivery modes achieved slightly better results than the ones achieved by DUPS-OK acknowledgement. We believe that the reason behind this is the behavior of DUPS-OK mode. This mode instructs the system to acknowledge messages in a lazy manner that most likely leads to the delivery of some duplicated messages. It can be noted that more than 40% of the produced messages arrived after the test time expiration. This is due to the overhead cost introduced by durable subscription.

The throughput results in Figure 12(b) are much lower than the results presented in Figure 12(a). This is due the impact of the wireless network characterization. Figure 12(b) shows that approximately both delivery and acknowledgement modes achieved similar results. This is most likely because the bandwidth becomes the bottleneck point in the system. We observed that less message latency was experienced compared to the scenarios presented in Figure 12(a). This is again because the overhead on the message server is much less than what it was in the scenarios shown in Figure 12(a). Thus, most of the produced messages were consumed before the measurement time expiration.



(a) Scenarios of Fixed Wireline Network



(b) Scenarios of Emulated Wireless Network

Figure 12: Durable Publish/Subscriber One-to-One Model: 2KByte Messages

Test cases 9 to 16 presented in Table 2 illustrate the throughput results when the message selector is introduced. For simplicity, we have assigned a single selector value ranging

from 0 to 99 to each produced message. The selector values were randomly generated with uniform distribution by the publisher. The subscriber registers to receive messages within a specific selector range. The range values were also randomly chosen to be 1/5th of the total range. Even though this is a simple way of using selectors, it is sufficient to illustrate the impact on performance. Figure 13 demonstrates a sample of the achieved throughput results that correspond to the test case 6 and 14 shown in Table 2. In these test cases, the values of message size, delivery mode, and acknowledgement mode were fixed to 2Kbytes, PERSISTENT, and AUTO respectively. Nondurable subscription with a particular selector value was used to receive messages that match its selector value. From the figure, it is obvious that using message selector has degraded the throughput of the nondurable subscription running on both network environments. As expected, the overhead of processing message filtration adds extra load on the broker, thereby decreasing performance.

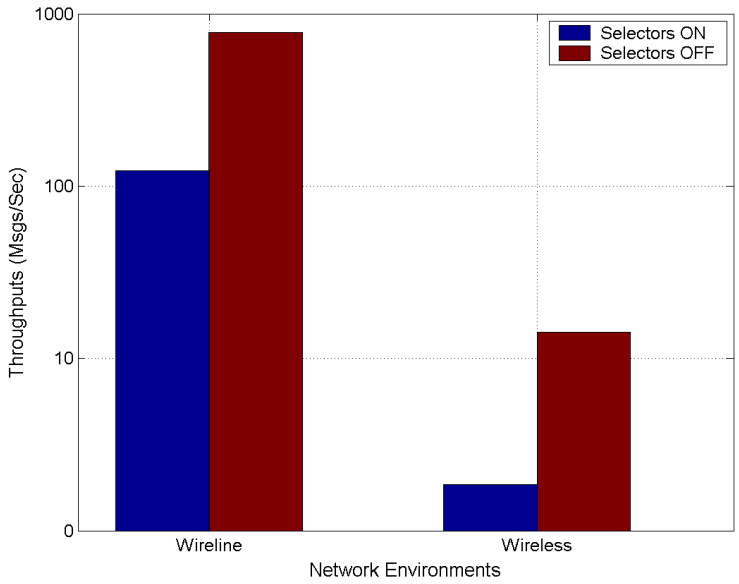


Figure 13: The Impact of Message Selectors on Performance: Nondurable Subscriber, Persistent Messages, and Auto Acknowledgement Mode

The last two test cases shown in Table 2 demonstrate the throughput results of using ObjectMessage as a message body type. In these cases, we fixed the values of message size to 2Kbytes, delivery mode to PERSISTENT, and acknowledgement mode to AUTO. Figure 14 illustrates the throughput results of BytesMessage body type (test case 6) and ObjectMessage body type (test case 17). A nondurable subscriber was used to consume these types of messages. It can be noted from the figure that ObjectMessage has achieved lower throughput than BytesMessage in the wireline environment. This is due to the overhead cost of ObjectMessage serialization and deserialization. As expected, both message body types approximately achieved similar throughput results in the wireless environment. The reason is that the network link becomes the bottleneck region, thereby reducing the load on the broker.

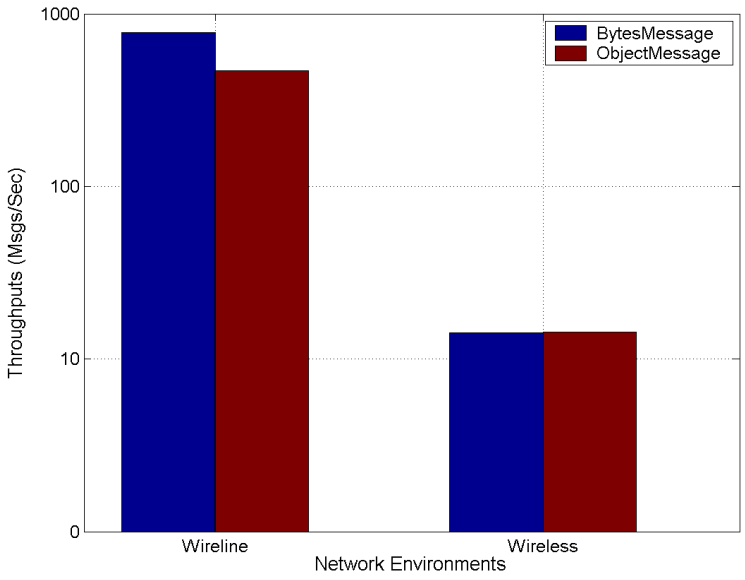


Figure 14: The Impact of Message Body Types on Performance: Nondurable Subscriber, Persistent Messages, and Auto Acknowledgement Mode

6 Discussion and Insights into System Behaviour

The experimental results shed some light on a number of performance scenarios of a publish/subscribe middleware system that is based on JMS technology. These results provide insights into the system behavior and performance that can be valuable to the system designers and users. We briefly discuss some of these observations in this section.

The nondurable subscription is observed to achieve a better performance results in most situations. However, in wireless environments where the available bandwidth is low and the transmission delay is large the performance of nondurable subscription seems to be severely affected: for different experimental tests the nondurable subscriptions were observed to perform comparably with the durable subscriptions. The publication rate relatively depends on the transmission delay among nodes running the publisher and the broker. This occurs because a publisher sends a message to a broker and then gets blocked until it receives an acknowledgement for the message delivery. Thus a large delay reduces the number of messages that can be generated by the publisher and decreases the load on the broker. This may benefit the durable subscription where the overhead of message storing is involved. By contrast, the performance results show that the nondurable subscription is greatly affected in low bandwidth environments.

It was observed that a small number of messages arrived after the test time expiration in most experiments of nondurable subscriptions. This mainly occurs because nondurable subscription imposes a low overhead on the broker to forward the published messages even with a large number of subscribers. The situation of durable subscriptions is different as the overhead cost of message processing is high. Due to this a significant number of messages were received after the test time expiration. It was interesting to notice that large messages delivered over the wireless channel arrived within the test time

duration. This most likely happened because the wireless link becomes the bottleneck region, thereby reducing the load on the broker.

It was noticed that large message sizes have a great impact on the performance of durable and nondurable subscriptions in the experiments we have conducted. Although nondurable subscriptions demonstrate better performance results with small and large messages, they tend to be more sensitive to the large messages than durable subscriptions. The throughput ratio between the small and large messages for both subscription types shows that nondurable subscriptions achieved a higher ratio in most test scenarios. Large message sizes require a long time to be transmitted and hence decrease the publication rate as stated previously. This will reduce the load on the message provider and improve its service time. As a result, this may benefit durable subscriptions as they incur a large service time compared to nondurable subscriptions. On the other hand, nondurable subscriptions do not gain a significant performance benefit as the condition and the characteristic of network link seem to considerably influence their behavior. This can be clearly seen in the wireless scenarios with large messages where both subscription types achieved approximately similar throughput results.

By varying the number of interacted subscribers and publishers, we noted that durable subscriptions performed poorly as the number of durable subscribers and/or publishers was increased. The throughput results of the one-to-one model for example show that durable subscriptions reached the breakpoint at a low number of clients. Similarly, in the few-to-many model, the performance of durable subscriptions was affected as the number of subscribers doubled. Moreover, durable subscriptions achieved very poor results as the number of publishers in many-to-few model was increased. This expected behavior is an outcome of the excessive load on the broker incurred at higher number of durable subscribers and/or publishers. This load is a function of the total number of clients associated with the broker. The load includes storing and forwarding messages as well as receiving acknowledgements and removing messages from a local storage. As a result, durable subscriptions perform considerably poor as the message provider becomes the bottleneck region.

Message selector is one of the JMS features that allow subscribers to receive a certain set of messages. As expected, applying this feature can greatly affect the performance of durable and nondurable subscriptions. This is because message selectors add extra overhead on the broker as it has to match all messages against the provided selectors. By increasing the numbers of subscriber we can expect system performance to be severely diminished. Selectors may benefit durable subscriptions since the broker stores only the messages that match the selectors, thereby decreasing the storing load. On the other hand, they may influence the performance of nondurable subscriptions as selectors add an additional load without any compromising. This can be seen from the results shown in Table 2.

This paper has focused on the evaluation of different application design factors. The appropriate choice of the investigated factors is beyond the scope of this paper. These factors should be chosen in such a way that it suits applications needs. There is always a tradeoff between reliability and system performance. Parameters that provide high reliability tend to negatively affect system performance.

7 Conclusions

The popularity of publish/subscribe systems that combine the advantages of Java Message Services (JMS) technology with the support of distributed applications is rising. The publish/subscribe systems support many features that facilitate their deployments over wireless environments. However, much work still needs to be done to make these systems achieve high reliability and performance in such environments. In this paper we have described our performance evaluations of a distributed publish/subscribe system deployed over wireline and wireless networks. For now, we have narrowed our evaluation to multiple clients and a single message provider without any mobility. The main goal of our analysis is to investigate the impact of different factors on performance. We achieved this goal by using two different experimental sets that investigated various combinations of parameters.

The first set of experiments focused on measuring the impact of three factors: subscription types, connection loads, and message sizes. A total of 32 test scenarios were conducted to study the behavior of these factors under wireline and wireless environments. Results of the experiments show that nondurable subscriptions outperformed durable subscriptions in most situations. This occurs because nondurable subscriptions incur a small overhead on the broker since they provide low reliability. By increasing the number of connections the performance of both subscription types was severely affected due to the higher overhead cost involved. Similarly, increasing message sizes impact the system performance as they require longer time to be processed.

The second set of the experiments looked at the impact of the following factors: delivery and acknowledgement mode as well as message selectors and message body types. A total of 18 test scenarios were performed to investigate the behavior of these factors over wireline and wireless environments. Our experiments reveal that persistent messages add extra overhead on the broker as they need to be stored and removed from a local storage. Therefore, persistent messages affect system performance but provide higher reliability. We were not able to observe a considerable impact on performance from our study for two different acknowledgement modes: AUTO and DUPS_OK. However, we expect that AUTO mode has some influence on performance since it provides higher reliability. It was observed that using message selectors introduces extra overhead cost that can affect performance. This overhead is a result of matching each message against all selectors. There are five types of message body that can be in simple or complex formats. Simple message bodies introduce lower overhead than the complex ones. Hence, they achieve higher performance results as it is shown in Table 2. Therefore, message bodies should be selected carefully as they may impact the performance.

This paper illustrates our evaluation study of a publish/subscribe middleware system and provides a number of insights into the relationship between various workload parameters and performance. This work is part of our study of publish/subscribe middleware and a primary step in considering the suitability of these systems in mobile wireless domains. Further performance investigations are still need to be done when user mobility is introduced. In continuing work, we wish to develop a service that supports user mobility in a seamless fashion and evaluate its performance.

8 References

[1] Abdulbaset Gaddah and Thomas Kunz, 'Does Modern Middleware Address Mobile Computing Requirements?', in Proceedings of the 8th World Multi-Conference on Systemics, Cybernetics and Informatics, Orlando, USA, Vol. 5, pp. 493-499, July 2004.

[2] A. Carzaniga and A. L. Wolf, 'Content-based networking: A New Communication Infrastructure', In NSF Workshop on an Infrastructure for Mobile and Wireless Systems, Number 2536 in Lecture Notes in Computer Science, pp. 59-68, Scottsdale, Arizona, October 2001. Springer-Verlag.

[3] M.K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley and T. D. Chandra, 'Matching Events in a Content-Based Subscription System', in Proceedings of the 18 th ACM Symposium on Principles of Distributed Computing (PODC'99), Atlanta, GA, May 1999, pp. 53-61.

[4] G. Banavar, T. D. Chandra, B. Mukherjee, J. Nagarajarao, R.E. Strom and D. C. Sturman, 'An Efficient Multicast Protocol for Content-based Publish/Subscribe Systems', in Proceedings of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS'99), Austin, TX, May 1999, pp. 262-272.

[5] L. Opyrchal, M. Astley, J. S. Auerbach, G. Banavar, R. E. Strom and D. C. Sturman, 'Exploiting IP Multicast in Content-Based Publish/Subscribe Systems', in Proceedings of 1FIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware 2000), New York, NY, April 2000, pp. 185-207.

[6] Y. Zhao and R. Strom, 'Exploiting Event Stream Interpretation in Publish/Subscribe Systems', in Proceedings of the 20th ACM Symposium on Principles of Distributed Computing (PODC 2001), Newport, RI, August 2001, pp. 219-228.

[7] B. Segall and D. Arnold, 'Elvin has Left the Building: A Publish/Subscribe Notification Service with Quenching', in Proceedings of AUUG97 Conference, Brisbane, Australia, September 1997, pp. 243-255.

[8] G. Cugola, E. Di Nitto and A. Fuggetta, 'The JEDI Event-Based Infrastructure and its Application to the Development of the OPSS WFMS', IEEE Transactions on Software Engineering, Vol. 27, No. 9, September 2001, pp. 827–850.

[9] Sun Microsystems, 'Java Message Service (JMS) API Specification', http://java.sun.com/products/jms

[10] Sun Microsystems, 'Java Message Service (JMS) Tutorial', Tutorial. Available from http://java.sun.com/products/jms/tutorial/index.html

[11] Exolab Group, OpenJMS, http://openjms.sourceforge.net/license.html

[12] Object Web, Joram, http://www.objectweb.org/joram

[13] Fiorano Software Inc., FioranoMQ, http://www.fiorano.com/products/fmq

[14] JBoss Group, JBoss, http://www.jboss.org.

[15] Sun Microsystems, Java System Message Queue (JavaSMQ), http://www.sun.com/software/products/message_queue

[16] National Institute of Standards and Technology, NIST Network Emulation Tool, http://snad.ncsl.nist.gov/itg/nistnet/index.html