Mobile Applications with a Middleware System in Publish-Subscribe Paradigm

Eiko Yoneki

University of Cambridge Computer Laboratory,
William Gates Building, J J Thomson Avenue,
Cambridge CB3 0FD, UK

Eiko.Yoneki@cl.cam.ac.uk

Abstract

This paper presents applications using **Pronto**[12] and its corresponding functions. Pronto is a middleware system for mobile applications with messaging as a basis in both centralized and decentralized forms. It addresses design issues in mobile computing, including data optimization, resource constraints, and network characteristics. Pronto consists of a lightweight Message Oriented Middleware (MOM) client and an intelligent gateway as a message hub with store-and-forward messaging. The publish-subscribe paradigm is ideal for mobile applications, as mobile devices are commonly used for data collection under conditions of frequent disconnection and changing numbers of recipients. This paradigm provides greater flexibility due to decoupling of publisher and subscriber applications. Mobile applications benefit from advantages by deploying Pronto, including disconnected operation, Gateway caching, compression, data transformation by Gateway plug-in functions, and content-based subscription.

1 Introduction and Background

Computing devices are becoming increasingly mobile at the client end. In a mobile/wireless network environment, devices have a small ROM/RAM footprint, latency is high, bandwidth is low, connections are frequently interrupted, and many devices are not programmable. Middleware communications service is especially important for integrating such hybrid environments into coherent distributed systems.

Intercommunication is commonly achieved using directed links between tightly coupled senders and receivers; the message destination must be known at the time of sending, which is difficult with changing destinations or varying numbers of recipients. By contrast, Message Oriented Middleware (MOM) encourages loose coupling between message senders and receivers with a high degree of anonymity, with the advantage of removing static dependencies in a distributed environment. MOM's characteristics (intuitive programming model, latency hiding, guaranteed delivery, store-and-forward) are appealing for mobile

applications. Java Message Service (JMS) [8] is a recent Java technology, providing an API for inter-client communication among distributed applications. It is a service-oriented API specification, providing an architecture for MOM and prescribing messaging functionality in terms of interfaces and associated semantics. It offers a publish-subscribe paradigm and expands previous messaging capabilities. Most JMS products implement a centralized server model. To provide rich JMS functionality, especially persistent message delivery, servers require databases (for storing messages), yet none of the commercial products has successfully implemented JMS in a decentralized model. There have been efforts to construct messaging systems over peer-to-peer networks, but none provide enterprise level messaging functionality thus far. Given the characteristics of mobile devices and wireless networks, more work is required for good performance. Some important design issues are specified below:

- Wireless networks become increasingly packet-oriented. With a packet-oriented bearer such as GPRS (General Packet Radio Service) or UMTS (Universal Mobile Telecommunications System), users typically pay only for the data they communicate. Reducing data size for transmission is crucial.
- Because of low bandwidth, high latency, and frequent disconnections, a middleware should provide an interface to applications that allows for communication to be maintained during disconnected operation. Dependable caching is essential.
- A data source can be interpreted in different formats and semantics depending on the specifications of mobile devices and wireless networks. Semantic transcoding technology [6] should allow for more efficient data flow.
- There are various bearers such as 2G, 2.5G, 3G, Bluetooth, and IEEE 802.11 and many devices are non-programmable. A middleware needs to offer an interface that provides a communication abstraction.

We have developed **Pronto** [12], a middleware system for mobile applications. The basis of Pronto is a MOM, based on JMS in both centralized and decentralized forms. Pronto also introduces an intelligent gateway [11] for reliable and efficient transmission between mobile applications and servers, taking advantage of plug-in components for caching, device-specific transport, compression, and semantic transcoding as well as supporting disconnected operation. Pronto provides a useful MOM layout from a server to mobile devices over a wireless network, and its performance is optimized by applying and comparing different techniques.

To demonstrate the potential of Pronto, we wrote several applications. Pronto can be used for pervasive computing, thus providing instantaneous access to information (e.g. stock quotes, email, calendar-alerts) in a reliable and efficient manner. Over a temporal network environment, Pronto can be used to establish group communication without a central server. This paper presents applications constructed with Pronto and the corresponding functions. Section 2 describes the component overview of Pronto and deployment possibilities. Section 3 presents experimental applications and deployment examples. More details of Pronto functions follow in Sections 4-7. Section 8 shows benchmark testing for performance improvement. This paper finishes with an overview of related work (Section 9) and a conclusion and future work (Section 10).

2 System Overview

Pronto is designed as a middleware forming a collection of distributed services. Three main functions form the backbone:

MobileJMS Client: MobileJMS Client is designed to adapt the JMS client to the mobile environment. The challenge is to accommodate JMS with constrained mobile devices. The specifications and interfaces of JMS are complex, but not all functions are mandatory for a mobile environment. For example, Pronto does implement neither MapMessage nor message priority. MobileJMS Client is described in more detail in Section 4. Furthermore, a simple JMS server was implemented to support the MobileJMS Client, which is out of scope of this paper.

Serverless JMS: Serverless JMS is a novel serverless version of MobileJMS Client. The aim is to put the JMS scheme in a decentralized model, using IP multicast as transport mechanism. In a mobile environment, the nature of data distribution may often fit better into a multicast/broadcast model. Multicast transport mechanisms allow the efficient transmission of data from one to many without redundant network traffic. Serverless JMS will perform best over an ad-hoc network and for high-speed transmission of a large number of messages distributing the workload of one to several servers. The ad-hoc network, another feature of a mobile/wireless network, is a dynamically re-configurable network without fixed infrastructure and without a requirement for the intervention of a centralized access point. The message domain publish-subscribe can reside well on an ad-hoc network.

Gateway and SmartCaching: An intelligent Gateway [11] is introduced as a message hub with store-and-forward messaging, taking advantage of plug-in components for caching, device specific transport, and message transformation. SmartCaching is designed to provide generic caching; it is embedded as a central function for message storage in Gateway. Gateway and SmartCaching are key technologies for improving messaging among mixed mobile-tier environments in dynamic connectivity scenarios (see Sections 6-7 for details). Plug-in components are not discussed in this paper.

Pronto can be deployed in a centralized model with MobileJMS Client and Gateway functions and in a decentralized model with Serverless JMS. Combination of Gateway and Serverless JMS optimizes message flow in distributed systems (see Section 3.5 Gateway Cascade).

2.1 Distributed Systems with Pronto in a Centralized Model

Fig. 1 shows an overview of a distributed system with Pronto in a centralized model. Different deployment possibilities are illustrated:

- **Application with MobileJMS Client**: An application in a mobile device uses a MobileJMS Client API; it communicates directly with the JMS server.
- **Application with LocalGateway**: An application in a mobile device uses a MobileJMS Client API or a Gateway API. LocalGateway is a mode of Gateway and can run as a separated thread from the application or within the application and performs caching and transcoding through plugged-in components.

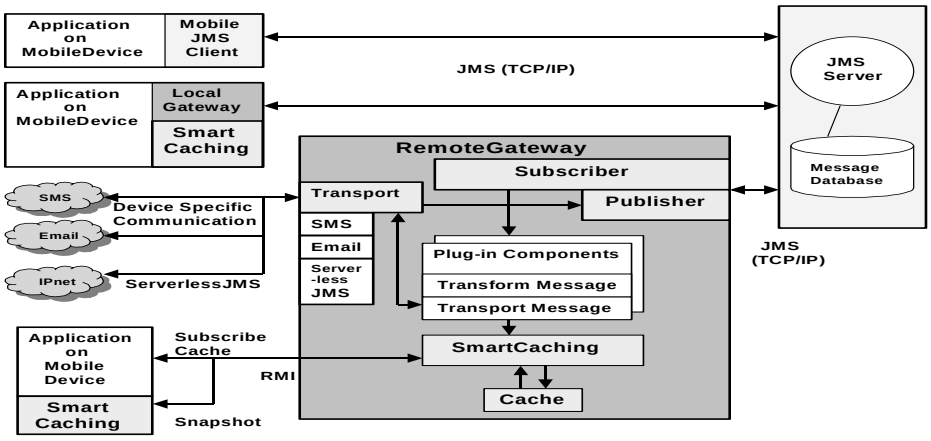


Figure 1: System Overview of Distributed System with **Pronto**

- **Application with RemoteGateway**: An application in a mobile device uses a MobileJMS Client API or a Gateway API. RemoteGateway is another mode of Gateway and runs as a separate process. Currently RMI-based transport between a RemoteGateway and MobileJMS Client is implemented.
- **Non-Programmable Devices with RemoteGateway**: Non-programmable devices require RemoteGateway to perform proper transportation and message transformation for the target device. RemoteGateway represents every subscriber and publisher for the non-programmable device.

2.2 Pronto in a Decentralized Model

Serverless JMS supports a decentralized model. A publisher acts as a temporary server and keeps a subscription list. Fig. 2 shows the data flow in Serverless JMS.

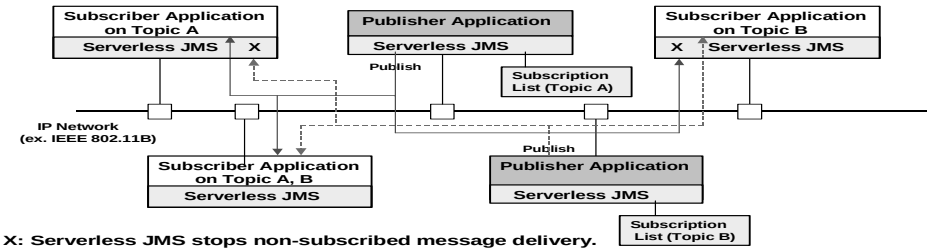


Figure 2: Serverless JMS over IEEE 802.11 Network

2.3 Disconnected Operations

The following approaches are designed for disconnected operation in Pronto:

- **Durable subscription via MobileJMS Client:** Durable subscription is part of the JMS API. Non-durable subscriptions last for the lifetime of the subscriber object. A subscriber can, optionally, be durable by registering a durable subscription with a unique identity.
- **Gateway Cache:** Gateway maintains its cache even if applications are inactive. Applications can use the Gateway cache after regaining connection; they can use the pull, subscribe, and snapshot operations of SmartCaching as appropriate.

3 Applications and Deployments

Mobile applications benefit from various advantages by deploying Pronto. Applications and deployment examples are shown below.

Mixed Media Chat: Chat among an applet, an application, Palm pilot, iPAQ, and SMS phone. All clients subscribe to the topic 'Chat'. A Gateway residing in the mobile laptop contains two plug-in components, Voice Synthesizer and SMS. Two iPAQ devices are monitoring the chat conversation through the Gateway cache. Additional devices can join the chat anytime without change of the other components(see Fig.3).

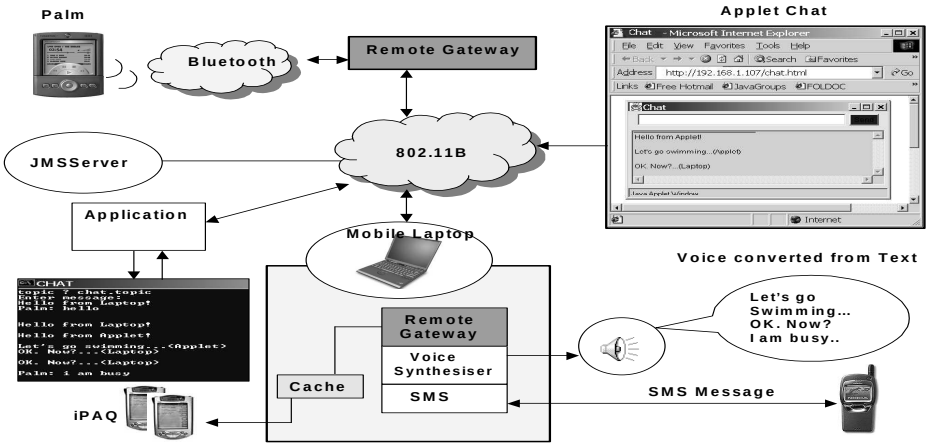


Figure 3: **Application:** Mixed Media Chat

Chat with Serverless JMS: This is a chat system using ServerlessJMS over a temporal network. Jurg subscribes 'Football news' via JMS. When he obtains an important news, his PDA starts up a publisher application with the topic 'News'. The application uses the auto-discovery function to find whether any subscribers are listening to the topic. After finding subscribers, it publishes a messages. Hedi and Willi respond to this message and join the chat. The publisher application in Jurg's device removes the subscription by Hedi when she disappears from the network. The whole process does not require any central access point. Once Jurg's publisher application disappears from the network the chat ends. See Section 5 for the details of ServerlessJMS(see Fig.4).

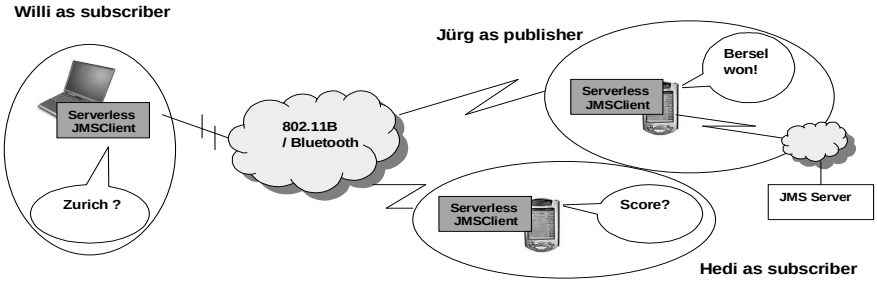


Figure 4: **Application:** Chat over an temporary network

Time Series of Video Data Publishing over 802.11B Network: A video camera takes 15 seconds of video every 30 seconds, and data are published under the topic 'Video'. All subscribers receive the published video data. A Laptop/iPAQ user is moving, leading to occasional disconnections. Gateway running on Laptop/iPAQ is set to durable subscription, and all published data are cached within the Gateway. The application subscribes the cache from Gateway and shows the video data via the media player. Gateway's caching provides the entire process without significant delay. At the same time, Gateway plug-in functions, Email and SMS, send out the data. Data are transcoded to the appropriate formats of the target devices(see Fig.5).

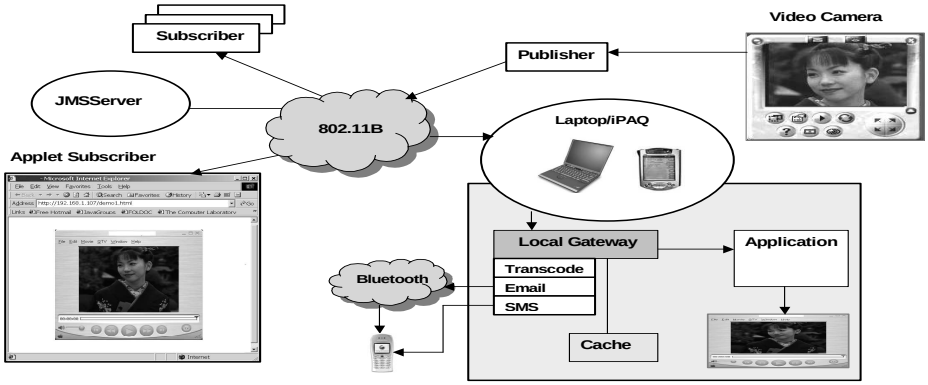


Figure 5: **Application:** Video Data Publishing over 802.11B Network

Gateway Cache and Message Selector: Fig.6 shows an application with Gateway and SmartCaching. Subscriber applications register topics and Message Selectors (see Section 4 for the detail of Message Selector). A publisher publishes messages containing income information for the people within a group. One application is of interest only to people who earn over 100,000 Euro, whereas the other application is of interest to all people regardless of salary. Gateway keeps the cache, and applications obtain cache in the following manner:

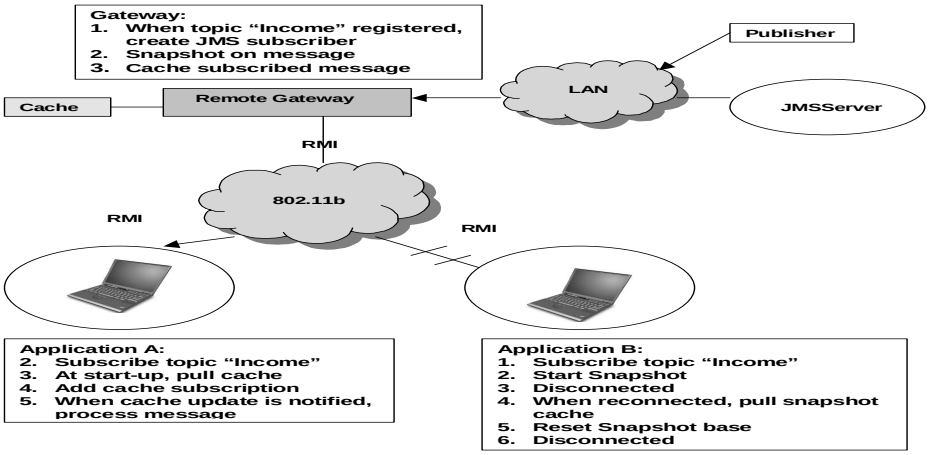


Figure 6: **Application:** Gateway Cache and Message Selector

- Application A subscribes cache update and gets notified when messages with the topic 'Income' are added. It is interested only in messages containing information of people earning over 100,000 Euro. Thus, the message selector is used, and A only gets notification when the message matches the selection.
- Application B subscribes to the topic 'Income'. This device is disconnected most of the times. B sets the Snapshot baseline (see Section 7 for details of Snapshot), then disconnects from Gateway. While the device is disconnected, the Gateway keeps receiving messages and caches them. When B is re-connected, it pulls the Snapshot cache and resets the Snapshot base line.

Gateway Cascade: Multiple Gateways can be used to distribute messages to target Gateways where they are sent to the devices. JMS bus is a Serverless JMS over a high-speed bus. A high-speed bus can be LAN-based or WAN-based as far as the routers allow IP multicast. Combination of Gateway and JMS bus offers powerful message flow control and distributed filtering to minimize network traffic as appropriate for the network characteristics. With increasing distance from the source, data are expected to become more localized by deploying Gateways(see Fig.7).

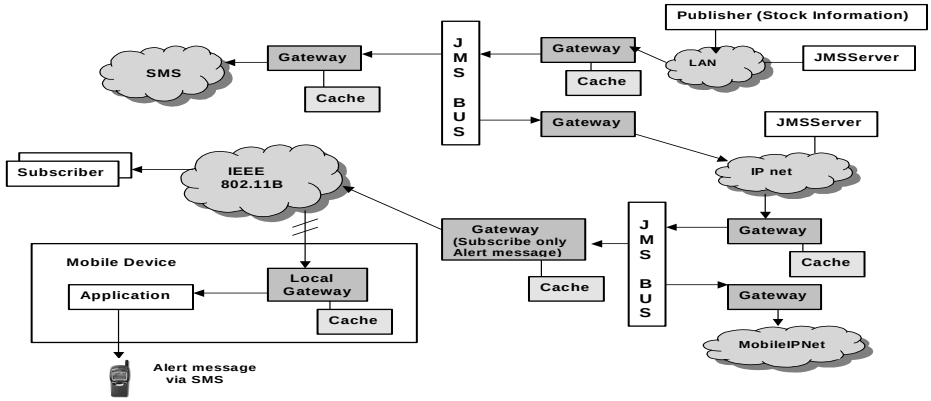


Figure 7: **Deployment:** Gateway Cascade

4 MobileJMS Client

MobileJMS Client is designed to follow the JMS API model. In Pronto, a message is a lightweight object consisting of a header and body. The header contains identification and routing information and is optimized for minimal size. The body contains application data. Essential message types such as TextMessage, BytesMessage, and ObjectMessage from the five message types defined in JMS are implemented. Messaging services such as persistent delivery, durable subscription, and the time-to-live option on a message show the range of delivery methods. Durable subscription is essential to support disconnected operation. Pronto implements the publish-subscribe paradigm for asynchronous messaging, but not the point-to-point paradigm. A connection represents an open connection to the JMS server. JMS does not define any specific transport mechanism and HTTP via TCP/IP is implemented in Pronto.

Message Selector (Content Based Subscription): Pronto currently provides Message Selector for content-based subscription as an extension of JMS API. This gives greater flexibility for applications as there is less coupling between producers and consumers.Message Selector is a filter for a topic defined by the consumer. In Pronto, this filter is implemented with an XML based TextMessage. A message selector is a string, whose syntax is based on a subset of SQL92 conditional expression syntax. In the example shown in Fig.8, only the second message published will be delivered to the subscriber.

```
Publisher:
    . . . . .
    TextMessage msg1 = session.createTextMessage
        (‘<?xml version=\’1.0\’ encoding=\’UTF-8\’ ?><List>
              <Millionaire Name=\’Bahler Income=500\’/></List>‘);
    TextMessage msg2 = session.createTextMessage
        (‘<?xml version=\’1.0\’ encoding=\’UTF-8\’ ?><List>
              <Millionaire Name=\’Gates Income=10000\’/></List>‘);
    publisher.publish(msg1, Message.PERSISTENT);
    publisher.publish(msg2, Message.PERSISTENT);
Subscriber:
    . . . . .
    Connection.start();
    subscriber = session.createXMLSubscriber(topic,’Millionaire.Income >= 5000 ‘);
    subscriber.setMessageListener(new Listener(subscriber));
```

Figure 8: Message Selector Use

5 Serverless JMS

Many underlying transmission media such as Ethernet provide support for multicast at the hardware level. Applications with Serverless JMS over such a network leads to a significant performance improvement (see Section 8). Serverless JMS currently implements basic functions, while some JMS features such as persistent delivery and durable subscription were omitted, given the nature of the network model and IP multicast protocol. Serverless JMS does not support ad-hoc networks that have no IP multicast capabilities. Key features are shown below:

Multicast Group: Each IP multicast address corresponds to a channel to identify groups of hosts interested in receiving the same content. Two channels are used: the ManagementChannel serves administration purposes, while the MessageChannel serves message transmission. As an option, MessageChannel can be defined on each topic.

Reliable Protocol: The basic service provided by IP multicast is an unreliable datagram multicast service, and there is no guarantee that a given packet has reached all intended recipients. Serverless JMS implements both reliable and unreliable multicast transports. A novel protocol to support reliable multicast is designed using a negative acknowledgment. A protocol itself is out of scope of this paper. The transparent fragmentation and re-assembly of messages that exceed a UDP datagram size is also implemented.

Flow Control: The speed of modern LAN transmission is high, and packet loss will be rare with good network quality. However, due to the high speed, the network buffer may be overwritten and messages discarded if it is not large enough and the subscriber cannot keep up with the speed of incoming data. This

corresponds to packets being lost during the transmission. Thus, window-based flow control between publishers and subscribers is implemented to prevent data retransmission and to improve throughput.
Subscription Registration: Two subscription modes are defined: the administrated and non-administrated modes. In the non-administrated mode, publishers publish messages independently of the existence of subscribers. Administration mode maintains a subscription list and prevents publishing messages without subscribers.
Auto Discovery: An auto discovery function is designed. A publisher runs an independent thread for auto discovery, which sends management data that require an echo from subscribers via ManagementChannel.

6 Gateway

Gateway distributes messages to other Gateways and applications. Multiple gateways can be used, as appropriate for the network environment and client characteristics. This allows the construction of a distributed messaging system over JMS servers and offers load sharing and load reduction for good performance. Gateway is designed as a framework to perform plug-in functions for which two interfaces are defined:

- **Transport**: an interface for mobile device transport
- **Transform**: an interface for message transformation

The plug-in functions should follow these interface definitions. Gateway initially creates *Transport* and *Transform* objects, according to XML-based configuration data. The configuration contains the class names that implement the *transport* and *transform* interface, and the target topic names indicate the message groups to be transformed. The Encode-Decode component carries out the message transformation as defined in the configuration. Gateway is also a MobileJMS client sharing its configuration information with client applications. The configuration information has to be managed to be accessible by both parties. Specific configuration utility is not implemented in Pronto.
Local and Remote Gateway: Gateway itself is defined as an interface, with two implementations, *LocalGateway* and *RemoteGateway*. LocalGateway can run as a separate thread or within the application. RemoteGateway is currently implemented as a Java Remote Method Invocation (RMI) *Object* and can run as a separate process. Mobile devices can take advantage of both Gateways depending on the application.
Plug-In Components: For the *Transform* interface, caching, compression, and semantic transcoding are good candidates to reduce data size and network traffic. Security (encrypting/decrypting data) functions can also be plugged in. Semantic transcoding offers more than simple data reduction. The information itself is made more abstract (to provide compaction), and the data should be evaluated whenever necessary. In a mobile environment, a reduction of data size on the network dramatically increases performance, and the concept of semantic transcoding is important. The data are linked to an annotation [6], which can be text corresponding to a video clip, a document summary, or a linguistic description

of the content for voice synthesis or grayscale/downsized/low-resolution image data. For *Transport* interface, device-specific transport for non-programmable devices such as SMS(Short Message Service) function is a good candidate. The registration of a *Transport* interface to Gateway activates a subscription to a JMS server on the specified topic.

7 SmartCaching

Caching is essential for performance improvement by reducing network traffic and improving latency. SmartCaching supports multi-tiered applications across platforms and devices. It currently implements basic functions, while persistent caching, cache validation, synchronization, and coherency management are beyond the scope of this study. In SmartCaching, cached data are decoupled from the data source, and cached data can be made active or up-to-date by *CacheHandler*, which is responsible for updating the cache. For example, Gateway is a *CacheHandler*, and it uses SmartCaching to store subscribed messages. Key functions to clients are the **Pull, Subscribe**, and **Snapshot** services. The Subscribe service provides asynchronous notification of cached data to client applications, and applications do not need to request to pull data that have already been requested. Snapshot provides a specified period that can be used by the mobile application to obtain the last cache image after disconnection. *CacheManager* is the main component in SmartCaching. It creates objects and manages requests and responses to requesters. Cache is an object that contains a key and the actual caching object. The *Cache* object contains the expiration date, and the CacheManager will remove expired objects. Alternatively, the Cache object can be removed after delivery.

Snapshot: If the data source sends messages via minimal delta information, caching updates existing data, applying only the delta information. Snapshot needs to know when the baseline starts. Each time a new message is received, the Snapshot rule is applied and the data persist in the cache. If a client requests Snapshot, it will receive the latest data only. During disconnection, the client can continue to operate using its own local cache. After restoring communication, only the last image of the cache needs to be updated. This reduces the need for reconnection by skipping all intermediate data. The Subscribe service can then inform applications of later changes in the underlying cached data. When Snapshot is on, cache update notification is done only when the last image changes. The data flow of Snapshot is shown in Fig. 9.

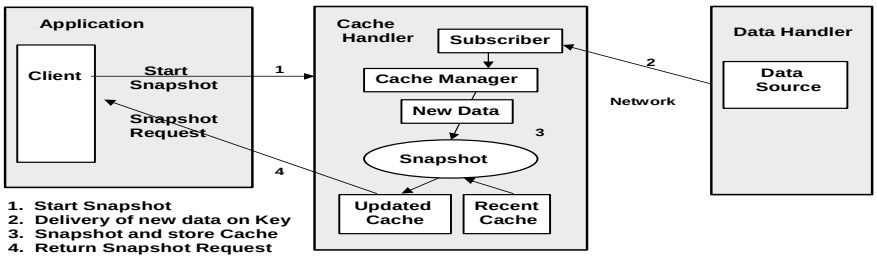
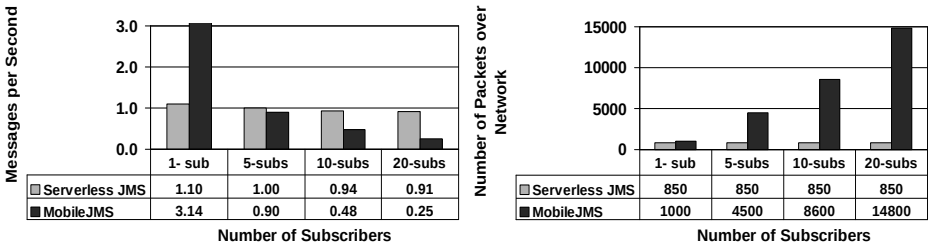


Figure 9: SmartCaching: Snapshot Data Flow

8 Benchmark Test over 802.11B Network

A sample from the benchmark test is shown in Fig. 10 (for more benchmark test results, see [11]). This test measures the capability of Serverless JMS. 20 of 250KB BytesMessages are to be published using Reliable option. No message retransmission occurred in this test. The results show that:

- The number of subscribers does not have an impact on the performance in Serverless JMS, whereas regular JMS delivery shows an impact proportional to the number of subscribers.
- The number of packets over the network stays the same with increasing numbers of subscribers.



(a) Throughput **(b) Network Traffic**

Figure 10: Performance comparison between MobileJMS and Serverless JMS (The PCs used for testing were X86(Pentium III) 256-392MB RAM 600-800MHz.)

9 Related Work

Since the initial JMS specification was released in 1998 [8], the existing MOM software have been rapidly integrated under the JMS API. Examples are TIBCO's TIB/Rendezvous [10] and Softwired's iBus [7]. However, Softwired's iBus/Mobile is essentially the only one to extend JMS to mobile-tier. iBus/Mobile is designed as an extension of J2EE application servers. In contrast, Gateway in Pronto is a message hub that can reside in the device or anywhere in between. Pronto provides a flexible N-tier layout, deploying multiple gateways instead of a tight linkage with a server. Gateway in Pronto offers more than a transport protocol as described above. Currently, several JMS products support multicast transport such as TIB/Rendezvous, but JMS has not been tried on mobile ad-hoc networks. Much research currently focuses on general datagram routing in both unicast and multicast routing [4], but no definite solution to provide JMS semantics using these protocols has been provided. For reliable transport over IP multicast, various protocols such as SRM [1], and RMTP [5] have been proposed and implemented. Pragmatic General Multicast (PGM) [2] is a reliable multicast support protocol for applications that require ordered or unordered duplicate-free multicast data delivery, providing a promising approach. However, the PGM header is not yet supported by any Java package. A reliable protocol based on negative acknowledgment is currently implemented in Pronto. Optimizing data over a wireless environment has been successful, although most technologies are

tightly coupled with the applications based on a client-server model. For example, IBM's WebExpress [3] provides a web browser proxy between mobile clients and a web server to optimize HTTP data. Caching is also tied to applications in most cases. Java Temporary Cache (JCache) [9] has been proposed (but not yet implemented practically) by Oracle and provides a standard set of APIs and semantics including N-tier support.

10 Conclusion and Future Work

This paper presents various applications and deployments with Pronto to show how the messaging can be extended over a wireless network and solve the problems arising. Deploying different plug-in functions with Gateway demonstrates the construction of an efficient message flow over a publish-subscribe system. Using disconnected operation and SmartCaching gives better flexibility for the design of mobile applications. JMS is more complex than discussed here. Pronto would need support for administration, security, error handling and recovery, and distributed transactions. Most importantly, it is critical to establish a standard API for publishing, managing, and accessing public references to distribute functionality over mobile environments, including security aspects such as encryption, authentication, and access control on distributed objects.

Acknowledgments. I would like to thank Jean Bacon and Jon Crowcroft (University of Cambridge) for critical reading and constructive comments.

References

[1] S. Floyd et al. A Reliable Multicast Framework for Light-weight Session and Application Framing. *ACM SIGGOMM Communications Review*, 1995.
[2] J. Gemmell et al. The PGM Reliable Multicast Protocol. *IEEE Network special issue on Multicast:An Enablling Technology*,2003.
[3] B. Housel and D. Lindquist. WebExpress: A System for Optimizing Web Browsing in a Wireless Environment. *Proc. of Int. Conf. on MobiCom*, 1996.
[4] S. Lee et al. On-Demand Multicast Routing Protocol. In *Proc. of IEEE WCNC '99*, 1999.
[5] J. Lin and S. Paul. Reliable Multicast Transport Protocol (RMTP). *Proc. of IEEE INFOCOM '96*, 1996.
[6] K. Nagao. Semantic Transcoding: Making the World Wide Web More Understandable and Usable with External Annotations. *Proc. of Int. Conf. on Advanced in Infrastructure for Electronic Business, and Education on the Internet*, 2000.
[7] Softwired. iBus Messaging. http://www.softwired-inc.com/.
[8] Sun Microsystems. *Java Message Service (JMS) API Specification.* http://java.sun.com/products/jms/.
[9] Sun Microsystems. *JCache: Java Temporary Caching API.* http://www.jcl.org/jsr/detail/107.prt.
[10] TIBCO. TIB/Rendezvous Concepts. http://www.rv.tibco.com.
[11] E. Yoneki and J. Bacon. Gateway: a message hub with store-and-forward messaging in mobile networks. *Proc. of the First Int. ICDCS Workshop on Mobile Computing Middleware(MCM03)*, May 2003.
[12] E. Yoneki and J. Bacon. Pronto: Messaging Middleware over Wireless Networks. to appear in *ACM/IFIP/USENIX International Middleware Conference (Work in Progress)*, June 2003.