Many Aspects of Reliabilities in a Distributed Mobile Messaging Middleware over JMS

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. Pronto[19] is a middleware system for mobile applications with JMS messaging as a basis in both centralized and decentralized forms. Asynchronous messaging is essential to support reliable communication with mobile devices. An intelligent gateway and smart caching in Pronto support the construction of distributed messaging systems over JMS servers. The main aim of Pronto is to support a reliable and efficient message flow from data source to consumers by applying and comparing different techniques. Pronto provides a solution for mobile application specific problems such as resource constraints, network characteristics, and data optimization, which achieves high reliability in end-to-end communication.

1 Introduction and Background

Computing devices are becoming increasingly mobile. Mobile computing need to deal with more dynamic environments and more resource constrains than traditional desktop computing. In a mobile/wireless network environment, devices have a small ROM/RAM footprint, latency is high, bandwidth is low, connections are frequently interrupted, location of devices changes at any time, and many devices are not programmable. This diversity of clients creates complex environments in distributed systems. It is challenging to provide reliability under such circumstances, especially as the information delivered to mobile devices may be mission critical messages. Thus, middleware communication service is important for integrating hybrid environments with high reliability.

Several communication mechanisms such as Remote Procedure Call (RPC) and Remote Method Invocation (RMI) have been used for sharing the workload in distributed systems. Intercommunication is commonly achieved using directed links between tightly coupled senders and receivers, where the message destination must be known at the time of sending; this 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, the advantage of taking away static dependencies in a distributed environment. MOM's characteristics (intuitive programming model, latency hiding, guaranteed delivery, store-and-forward) are highly appealing for mobile applications to provide reliable message delivery.

R. Meersman and Z. Tari (Eds.): OTM Workshops 2003, LNCS 2889, pp. 934–949, 2003.

Java Message Serice (JMS)[15] is a recent Java technology, providing an API for inter-client communication among distributed applications. The communication mechanism of JMS differs from others such as EJB (Enterprise Java Beans), and CORBA (Common Object Request Broker Architecture), which define a higher level of logic for applications. JMS is a service-oriented API specification, providing an architecture for MOM and prescribing messaging functionality in terms of interfaces and associated semantics. It offers publish-subscribe and point-to-point paradigms and expands previous messaging capabilities. JMS works well in an environment where network connections can break, and where the available bandwidth can vary within a short time. 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 in distributed format, but none provide enterprise level messaging functionality thus far. Given the characteristics of mobile devices and wireless networks, more work is required for high performance and reliability. 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 [11] 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.

This paper presents **Pronto**, a middleware system for mobile applications, that is briefly introduced in [19]. The basis of Pronto is a MOM based on JMS in both centralized and decentralized forms. The decentralized form is called **Serverless JMS**. Pronto also introduces an intelligent **Gateway** with store-and-forward messaging [18] for reliable and efficient transmission between mobile applications and servers, taking advantage of plug-in components for caching, device-specific transport, compression, semantic transcoding, and message transformation, as well as supporting disconnected operation. **SmartCaching** component provides generic caching in an N-tier architecture, an essential function of Gateway. Caching reduces the communication between mobile devices and data source, leading to gain high reliability. Constructing a distributed system by deploying multiple gateways over JMS servers controls the

flow of messages, which limits unnecessary incoming network traffic to mobile environments. The original data source should be made abstract to provide compaction, then data should be evaluated whenever necessary. By going through gateways, messages need to be more localized. This gives optimized reliable and efficient message flow.

1.1 Delivery Semantics and Reliability

In messaging middleware systems, in particular when considering communication models and protocols, precise specification of the semantics of a delivery is an important issue. Delivery guarantees are often limited by the behavior of lower communication layers, down to the properties of the network itself, limiting the choice of feasible semantics. Reliable point-to-point communication is usually achieved by applying TCP. In messaging middleware, messages may be sent without generating a reply, and they may be processed hours after having been sent. The communicating parties do not handle how messages are transmitted and when they are processed. Thus, the messaging system must provide guarantees both in terms of reliability and durability of the messages. It is not sufficient to know that a message has reached the messaging system that sits between the producers and consumers. It is necessary to guarantee that the messages will not be lost due to failure of that messaging system. Messaging systems need to provide various guarantees regarding reliability and fault tolerance ranging from best-effort to guaranteed and timely.

Persistence is generally presented in messaging systems in a centralized model and stores messages until consumers process them. It is based on store-and-forward messaging, i.e. the provision of reliability over a network where connections may be dropped and restored. It is a trade-off because more processing time is required for store-and-forward. Therefore a compromise to accomplish the balance is important. On the other hand, distributed messaging middleware in a decentralized model generally do not offer persistence since messages are typically replicated upon dissemination. By sending one copy to each subscriber, the system provides some degree of fault tolerance. However, a subscriber that has unstable communication condition may not be able to obtain missed messages when recovering. In distributed messaging systems such as Scribe/Tapestry can repair the multicast tree by periodically sending heartbeat messages. Other systems use a mesh network such as Buyeux. The redundancy of a mesh network can guarantee reliable delivery when a path is broken due to a faulty broker. Gryphon achieve reliability in the similar manner. While sometimes a high throughput is important and a low degree of reliability is acceptable, some applications set higher priority over throughput. Messaging systems therefore should provide different qualities of services to meet the demands of different applications. The delivery semantics for notifications offered by existing systems can be roughly divided into two groups, see also Section 3 for JMS Message Flow:

- **Unreliable delivery**: Protocols for unreliable delivery give few guarantees. These semantics are often used for applications where the throughput is the most important issue, but the loss of certain messages is not fatal for the application.
- **Reliable delivery**: Reliable delivery indicates that a message will be delivered to every subscriber despite the failure.

1.2 Many Aspects of Reliability

As distributed applications grow both in size and complexity, there is an increasing demand to improve reliability. Different clients of messaging middleware will have different degree of requirements regarding to reliability and quality of service (QoS) guarantees provided by the middleware. Fault-tolerance mechanisms have to be considered within the messaging middleware design, thus isolated network or component failures do not affect the entire systems. Techniques such as persistent messages and replication help to achieve a more robust messaging middleware implementation. The servers can also be several times clustered to provide fault tolerance. Publish-subscribe paradigm in messaging middleware is a powerful and scalable abstraction, but an appropriate distributed infrastructure is the key to a scalable implementation of a messaging middleware system. However, scalability might contradict with other required properties. Increasing reliability guarantees the involvement of overheads such as logging or detecting retransmitting and missing messages. The amount of traffic resulting from the acknowledgments generated by such protocols is considerable and limits their scalability. Among distributed messaging middleware systems, probabilistic protocols have recently received increasing attention since they fit the decoupled and peer-to-peer natures. Instead of providing complete reliability by deterministic approaches, probabilistic protocols ensure that a given notification will reach a given subscriber with a high and quantifiable probability. Appropriate trade-off has to be defined to cope with scalability and expressiveness in messaging middleware systems. In mobile environments, messaging middleware systems should maintain communication during periods of disconnection. A disconnected period might be caused intentionally by the consumers, or by device movements. Mobility support requires recovering messages during the disconnected period, and supporting mobility and disconnected operation should be considered as a part of the reliability support. An intelligent caching helps to reduce communication over unreliable links and indirectly contributes to obtain high reliability. Content-based subscription gives flexibility through a combination of topic-based subscription to construct reliable and efficient message flow in distributed messaging systems. It limits the network traffic to unnecessary consumers. Pronto aims to accomplish high reliability for mobile applications by applying and comparing different techniques. Section 2 gives an overview of Pronto and deployments. More details on Pronto functions follow in Sections 3-6. This paper finishes with an overview of related work (Section 7) and a conclusion and future work (Section 8). See example applications and benchmarks for performance improvement in [18], [20].

2 Pronto 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 mobile environments. 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 mobile environments. For example, Pronto does implement neither map-message type nor message priority. On the other hand, as an extension of the JMS API, Message Selector is added to support content-based subscription. It is important to limit incoming network traffic to mobile environment by content-based subscription, that provides more precise message filtering function. One of the aims is to create a mobile-specific JMS client API to optimize slim client library to obtain high reliability. MobileJMS Client is described in more detail in Section 3. 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 mobile environments, 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. 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 of a multicast group. Serverless JMS implements both reliable and unreliable multicast transports. A novel protocol to support reliable multicast is designed using a negative acknowledgment. Serverless JMS will perform best over an ad-hoc network. The ad-hoc 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 in JMS can reside on an ad-hoc network, but not implementations based on a centralized server model. Here, Serverless JMS will play an important role. Serverless JMS can be deployed over a high-speed bus for transmitting a large number of messages distributing the workload of one to several servers.

Gateway and SmartCaching: An intelligent Gateway [18] 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. Multiple gateways can be deployed to construct a message broker network over JMS servers, which should produce a reliable and efficient message flow from the source to the consumer. Store-and-forward messaging gives distributed cache and it provides reliability option even it causes messaging process redundant. SmartCaching is designed to provide generic caching for N-tier architecture; 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 5-6 for details). Plug-in components are not discussed in this paper.

2.1 Distributed Systems with Pronto in a Centralized Model

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

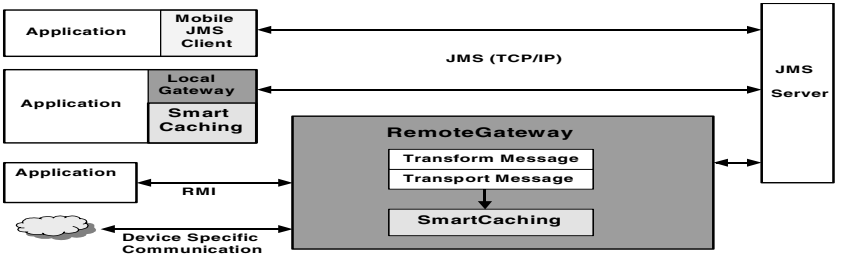


Fig. 1. System Overview of Pronto in a Centralized Model

- **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 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.
- **Application with RemoteGateway**: An application in a mobile device uses 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 devices. RemoteGateway represents every subscriber and publisher for the non-programmable device.

2.2 Pronto in a Decentralized Model

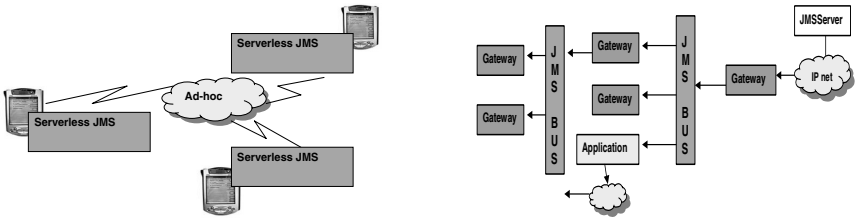
Serverless JMS supports a decentralized model. A publisher acts as a temporary server and keeps a subscription list. Serverless JMS performs the best over adhoc network environments (Figure 2a), and over high speed bus (Figure 2b). In Figure 2b, JMS BUS is a Severless JMS over a high speed bus, which supports to distribute a server's work load to several servers, or replicate a large amount of data to backup machines with classifying to the topics. Serverless JMS can utilize auto-discovery function to maintain subscriptions. Current serverless JMS requires IP Multicast capable routers.

2.3 Distributed Gateways over JMS Servers

Multiple Gateways can be used to distribute JMS messages to the target Gateways where messages are sent to the devices. Gateways are like message brokers, and they can form arbitrary topologies among themselves. Serverless JMS can be also embedded in Gateway as a plug-in component and the cascaded gateways act as multi-message hubs for distributing messages. 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 for optimization and reliability support, as appropriate for the network characteristics. A deployment example is shown in Figure 3. When the publisher publishes video message, the first gateway persists it in cache. The message is transformed to audio data by extracting the audio portion from video data then sent to the gateway subscribers. In this scenario, all the gateway subscribers are supposed to distribute messages either audio or short text. The gateway can be residing in the mobile device, and it can move from one network to another network as far as the client identifier is set uniquely within the name space. Gateways can act like event brokers grid to distribute messages in reliable and efficient manner. The principle is with increasing distance from the source, data are expected to become more localized by deploying Gateways.

3 Mobile JMS Client

MobileJMS Client is designed to follow the JMS API model. The common building block of a messaging service is the message. Messages consist of events, requests and replies that are created by and delivered to clients. Messaging services such as persistent delivery, durable subscription, the time-to-live option on a message, and transactions show the range of delivery methods. Durable subscription is essential to support disconnected operation in Pronto. Asynchronous messaging is a key integration point in mobile environments. JMS defines two messaging paradigms, publish-subscribe and point-to-point, the latter being less suited for mobile environments. Pronto implements



(a) Over Ad-hoc Network (b) Over High Speed BUS

Fig. 2. Pronto in a Decentralized Model

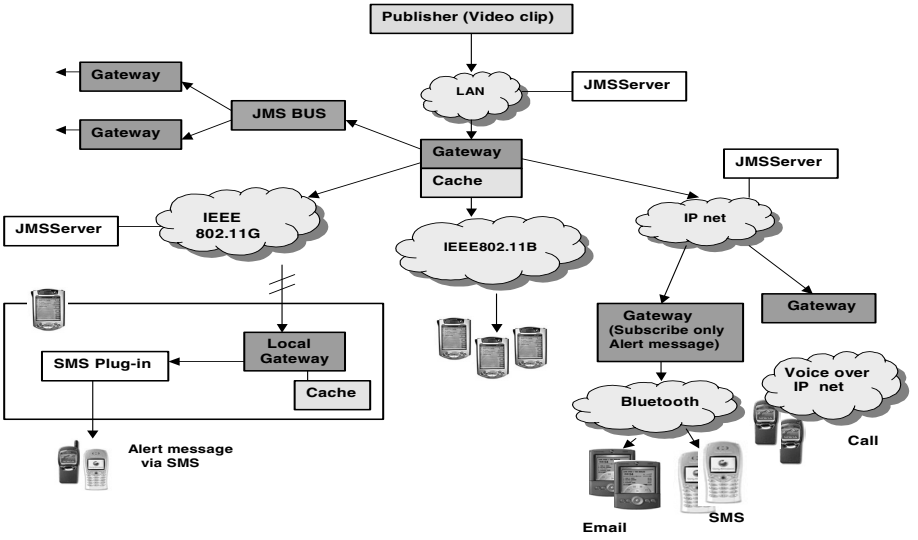


Fig. 3. Multiple Gateways Deployment

publish-subscriber paradigm. JMS does not define standard address syntax, using instead the *Destination* object that encapsulates the address [4]. In the publish-subscribe paradigm, the *Destination* is called Topic. Messages are sent to the *Destination* rather than specific processors or ports. Communication is typically one-to-many and asynchronous. The publish-subscribe paradigm supports the development of location-independent applications that can be moved from one machine to another without affecting their peer applications. In order to specialize JMS to a mobile environment the points below are considered:

Connection: A connection represents an open connection to the JMS server. JMS does not define any specific transport mechanism. In Pronto, HTTP via TCP/IP is implemented, which allows the applets using MobileJMS Client to connect through firewalls to the JMS server. SSL pipe can be used to provide secure connection.

Session: Connections create Sessions. A session is a single-threaded context that handles message-passing operations. A JMS server has a session pool and can execute separate messages concurrently, thus improving performance. If client code needs to receive asynchronous messages concurrently, the client can use multiple sessions to partition client operations, i.e. one client thread can drive one session while another client thread can drive another. Each session object serializes execution of message listeners. Thus, message listeners can share session-related resources. In order to avoid the complex threaded model, connections and sessions share one thread to receive a message in Pronto.

Message Flow: Figure 4 and 5 show the differences between two delivery modes and figures explain the message flow including JMS server behavior. When using the durable delivery mode, each message is stored by the JMS server persistently before delivery to the consumer and is removed after message delivery. This has a huge impact on performance but gaining reliability.

Figure 4 shows the durable subscription/persistent mode of the message flow. The message has an expiration time, which is from the time-to-live beyond the time of publication. It can be set to 'forever' but will be discarded as soon as delivery to all current subscribers and all durable subscribers is complete, as recognized by the acknowledgments of the subscribers. If message delivery 'persistence' is specified, the JMS server holds the message in persistent storage. Figure 5 shows the non-durable/non-persistent mode of message flow. For non-durable messages, the time that messages take to deliver to the Destination depends on their numbers and Destination sizes.

A large number of messages collected in a Destination will take more time to deliver. Redelivery delay time defines when to redeliver a message if a failure occurs. With shorter times, the frequency of redelivery is high, thus increasing network traffic. High Redelivery delay times give therefore better performance. Redelivery Limit defines the number of times a message should be redelivered. Although the probability of guaranteed messaging is lower with lower Redelivery limit, it increases performance due to reduced memory overhead for non-durable messages and persistent overhead for durable messages. It is therefore important to set a sensible Redelivery limit to balance between high throughput and reliability.

Durable Subscription: A durable subscriber registers a durable subscription with a unique identity. A subsequent subscriber with the same identity resumes the subscription in the state left by the previous subscriber. If there is no active subscriber for a durable subscription, JMS retains the subscription's messages until they are received by the subscription or until they expire. This supports not only disconnected operation but also location independence automatically.

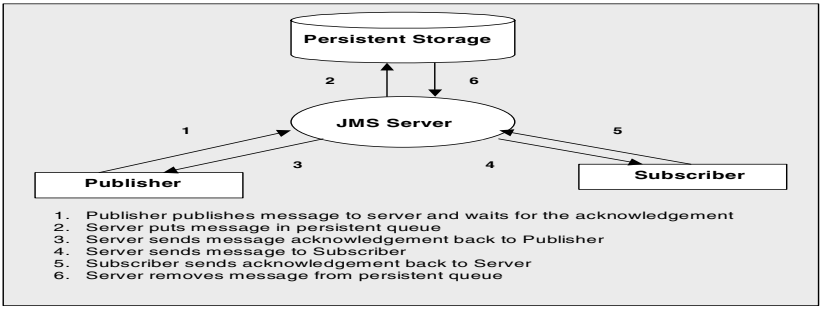


Fig. 4. Persistent Message Delivery

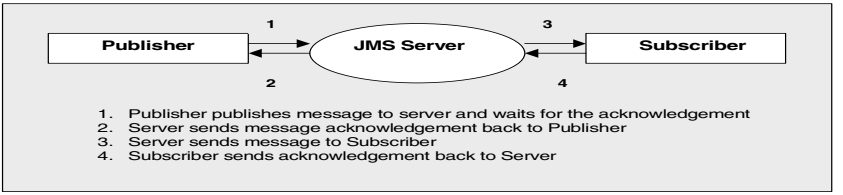


Fig. 5. Non-Persistent Message Delivery

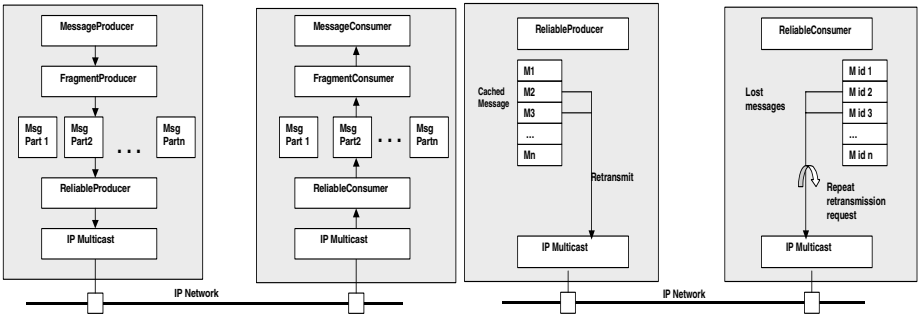
Message Selector (Content-based Subscription): Topics can be structured into hierarchies, and subscriptions can be a part of the hierarchy. This gives content-based messaging and greater flexibility for applications as there is less coupling between producers and consumers. Content-based addressing is more consumer-oriented, whereas subject-based addressing is producer-oriented. Content-based subscription is not part of the JMS specification, but it can be effective in mobile environments to control traffic to gain the relibility. Pronto takes an approach to provide Message Selector for content-based subscription as an extension of JMS API. 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.

4 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 lead to a significant performance improvement. 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. Key features are shown below:

Multicast Group: Groups of machines representing a multicast group are identified by an IP multicast address. Each address can be considered as 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. With such a service, there is no guarantee that a given packet has reached all intended recipients of a multicast group. Serverless JMS implements both reliable and unreliable multicast transport. The reliable version uses a negative acknowledgment. The transparent fragmentation and re-assembly of messages that exceed a UDP datagram size is implemented.



(a) Protocol Stack in Serverless JMS (b) Reliable Message Queue

Fig. 6. Reliable Protocol in Serverless JMS

This provides the highest possible delivery guarantee in a multicast environment. Figures 6a and 6b show the components to support the reliability and the fragmentation in Serverless JMS. On the producer side, the *FragmentProducer* splits the message to fit into the packet if the message to be published is larger than the defined network packet size. The *ReliableProducer* assigns a unique order number to each message to allow identification by the consumer. At the consumer side, the *ReliableConsumer* keeps an array that contains the unique IDs of the messages. Once the *ReliableConsumer* detects lost messages, it sends a retransmission request with the missing message IDs. This retransmission request repeats until the messages are retransmitted or it runs into a timeout. The retransmission is performed via the MessageChannel. Once the ordered message comes in, the message will be pushed up to the *FragmentConsumer*. The *FragmentConsumer* reassembles the data and pushes them up to the *MessageConsumer*. The *ReliableProducer* retransmits the requested messages on demand from the *ReliableConsumer*. The *ReliableProducer* keeps the message cache array. This package provides reliability on message delivery. However, this compromises throughput, because it slows down the procedure when the producer interferes with the retransmission process.

Pronto uses deterministic protocols to provide reliability, which retransmits the messages until all recipients acknowledge their receptions or are declared failed. When the network is congested or a receive is perturbed, messages accumulate in buffers until fully acknowledged. Since buffers are bounded, once they fill up the producer is not allowed to send further messages.

Flow Control: The speed of the modern LAN transmission is high, and packet loss will be rare due to good network quality. However, due to the high speed, if the network buffer is not large enough and the subscriber cannot keep up with the speed of incoming data, the buffer will be overwritten and messages will be discarded. It is important to set the network buffer size large enough, but the recent high-speed router fills up the buffer quicker than the subscriber can process. That creates the same symptoms as when packets are lost during transmission. The most effective way to avoid overwriting of the buffer is slowing

down the publishing messages and giving time to the subscriber to keep up with the speed of incoming messages. In *ReliableProducer*, a mechanism slows down or speed up message sending, depending on the detected transmission speed. The *ReliableProducer* keeps a table containing the delay values. Between the sending of two messages, the *ReliableProducer* does wait for the time of these timeout values. The message cache array on the producer and the message ID array on the consumer have to be coordinated with the speed of message transmission. If message loss is higher than the amount of cached messages, the consumer has no way to receive the retransmitting messages. In general, flow control is difficult and expensive. There is no perfect logic to control this. One has frequently to compromise between throughput and reliability.

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 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 and maintains the subscription list. Auto discovery repeats this at defined intervals.

5 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, leading to achieve an efficient and reliable message flow. Gateway uses store-and-forwarding messaging paradigm, which is an overhead process and redundant. However, to achieve high reliability, a redundant process may be useful. Gateway is designed as a framework to perform plug-in functions for which two interfaces are defined: **Transport** for mobile device transport and **Transform** 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 Encode-Decode component carries out the message transformation as defined in the configuration. Specific configuration utility is not implemented in Pronto.

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 [11], which can be text corresponding to a video clip, a document summary, or a linguistic description of the content for voice synthesis or image data in grayscale/downsized/low-resolution. For *Transport* interface, device-specific transport for non-programmable devices such as Short Message Service (SMS) or email functions are candidates.

6 SmartCaching

Caching is essential for performance improvement by reducing network traffic and improving latency. The cached data can be raw or processed and stored for reuse, thus avoiding revisiting the source and passing the data through the chain of reformatting and representation. SmartCaching, an intelligent cache function, 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. Using the Subscribe service client applications may be event-driven and active. This simple change has a major impact on performance and on the design of the applications. 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, kept as a linked list. The *Cache* object contains the expiration date, and the CacheManager will remove expired objects. Alternatively, the Cache object can be removed once it is delivered to the subscriber. The three main functions above operate in response to requests from *CacheManager*.

Pull: An application requests a cache synchronously.

Subscribe: An application requests a cache update notification to a cache handler, which notifies the application after the cache is updated.

Snapshot: When data are delivered piecemeal to applications in a time sequence, clients should be able to reconstruct the latest view of the information. This can be achieved by obtaining all data from the data source or by retaining

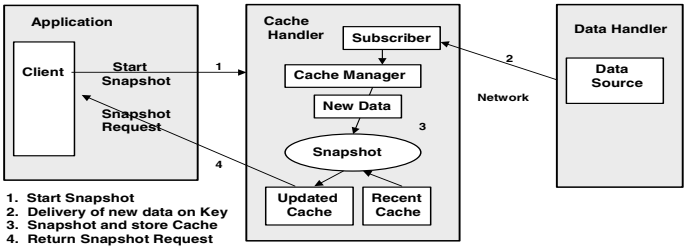


Fig. 7. SmartCaching: Snapshot

the last image in a shared cache. The second option corresponds to the Snapshot service. 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. The Snapshot rules can be provided by applications. If a client requests Snapshot, it will receive the latest data only. It is the responsibility of the client application that made the Snapshot request to retain all data, and, after the snapshot's arrival, to apply the data to bring that snapshot up-to-date. During disconnection, the client is able to continue to operate using its own local cache to satisfy requests as far as possible. After restoring communication, only the last image of the cache needs to be updated. This can reduce the need for reconnection by skipping all intermediate data. The event notification mechanism 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 Figure 7.

7 Related Work

Reliability specific related work in messaging systems is discussed in Section 1. In this section, related work to Pronto is described. Since the initial JMS specification was released [15], the existing MOM software has been rapidly integrated under the JMS API. Examples are IBM's MQSeries [8], TIBCO's TIB/Rendezvous [17], Softwired's iBus [14], and BEA's WebLogic [1]. However, Softwired's iBus Mobile is essentially the only one to extend JMS to mobile-tier, and designed as an extension of J2EE application servers. It includes a messaging middleware client library compatible with the JMS standard as well as a middleware gateway used to connect mobile applications to J2EE application servers. It supports mobile communication-specific protocols such as GPRS and UMTS. 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 offers more than a transport protocol as described above. IBM's MQSeries Everyplace

belongs to the MQSeries family of business quality messaging products. It is designed to satisfy the messaging needs of lightweight devices with own API. There is no standard messaging API for the mobile environment.

The original iBus before JMS heavily used multicast. Currently, several JMS products support multicast transport such as TIB/Rendezvous. However, JMS has not been tried on mobile ad-hoc networks. Much research currently focuses on general datagram routing in both unicast and multicast routing [13,9], but no definite solution to provide JMS semantics using these protocols has been provided. An example of a drawback of using multicast is the drastic performance reduction with redundant rebroadcasts [12]. For reliable transport over IP multicast, various protocols such as SRM [5], RMTP [10], and TRAM [3] are implemented. Pragmatic General Multicast (PGM) [6] is a reliable multicast support protocol for applications that require ordered or unordered duplicate-free multicast data delivery from multicast sources to multiple receivers. For publish-subscribe messaging systems, PGM provides a building block for the messaging system itself, allowing higher performance and scalability for messages that need to go to many destinations. This is a promising approach, but the PGM header is not yet supported by any Java package. For now a reliable protocol based on negative acknowledgment is designed and implemented in Pronto.

Optimizing data over a wireless environment has been successful, although most technologies are tightly coupled with the applications or the servers, based on a client-server model. Techniques for optimization include caching, protocol reduction, header reduction, and adding an asynchronous model. For example, IBM's WebExpress [7] 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) [16] has been proposed (but not yet implemented practically) by Oracle and provides a standard set of APIs and semantics that are the basis for most caching behavior including N-tier support.

8 Conclusion and Future Work

Pronto attempts to integrate technologies into a compact semantics-based middleware in support of issues specific to mobile environments. A mobile computing environment is resource constrained and support of reliability requires precise functionalities in both explicit and implicit ways. Deploying different plug-in functions with Gateway demonstrates the construction of a reliable and efficient message flow over a publish-subscribe system. Using disconnected operation and SmartCaching improves flexibility for the design of mobile applications.
JMS is more complex than discussed here. Although JMS lacks security definition, Pronto would need to support other reliability functions such as administration, security, error handling and recovery as well as 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; this includes security aspects such as encryption, authentication, and access control on distributed objects.

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

References

1. BEA. WebLogic 6.0 JMS. http://www.bea.com.
2. K. H. Britton et al. Transcoding: Extending e-business to new environments. *IBM System Journal*, 40(1), 2001.
3. D. Chiu, S.Hurst, M. Kadansky, and J. Wesley. TRAM: Tree-based Reliable Multicast Protocol. *Sun Microsystems Technical Report TR-98-66*, 1998.
4. P. Eugster et al. The Many Faces of Publish/Subscribe. *Technical Report TR-DSC-2001-04, EPFL*, 2001.
5. S. Floyd et al. A Reliable Multicast Framework for Light-weight Session and Application Framing. *ACM SIGGOMM Communications Review*, 1995.
6. J. Gemmell et al. The PGM Reliable Multicast Protocol. *IEEE Network special issue on Multicast:An Enablling Technology*,2003.
7. B. Housel and D. Lindquist. WebExpress: A System for Optimizing Web Browsing in a Wireless Environment. *Proc. of Int. Conf. on MobiCom*, 1996.
8. IBM. MQ Series. http://www.ibm.com/software/ts/mqseries/.
9. S. Lee et al. On-Demand Multicast Routing Protocol. In *Proc. of WCNC*, 1999.
10. J. Lin and S. Paul. Reliable Multicast Transport Protocol (RMTP). *Proc. of IEEE INFOCOM '96*, 1996.
11. 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.
12. S. Ni et al. The broadcast problem in a mobile ad-hoc network. In *Proc. of ACM/IEEE MobiCom*, 1999.
13. E. Royer and C. Perkins. *Multicast Ad-Hoc On-Demand Distance Vector Routing*, 2000. draft-ietf-manet-maodv-00.txt.
14. Softwired. iBus Messaging . http://www.softwired-inc.com/.
15. Sun Microsystems. *Java Message Service (JMS) API Specification.* http://java.sun.com/products/jms/.
16. Sun Microsystems. *JCache: Java Temporary Caching API.* http://www.jcl.org/jsr/detail/107.prt.
17. TIBCO. TIB/Rendezvous Concepts . http://www.rv.tibco.com.
18. E. Yoneki and J. Bacon. Gateway: a Message Hub with Store-and-forward Messaging in Mobile Networks. *Proc. of ICDCS Workshops MCM*, May 2003.
19. E. Yoneki. Pronto: Messaging Middleware over Wireless Networks. *4th ACM/IFIP/USENIX Int. Conf. of Middleware(Work in Progress)*, 2003.
20. E. Yoneki. Mobile Applications with a Middleware in Publish-Subscribe Paradigm. *Proc. of the Third Workshop on ASWN* , 2003.