A message oriented middleware for mobility

Jean-Francois Desjeans Gauthier
School of Computer Science, McGill University, Montreal

October, 2010

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of M.Sc

Contents

Abstract

Recent advances in embedded technologies have enabled cell phones to become powerful multihomed computing devices and hosts to a wide range of applications. However, communication between the device and the outside world is still a complicated task because the device is mobile, the connection is intermittent, the signal strength varies greatly and the communication options are heterogeneous. Previous work has suggested that communication with these devices is simplified and enhanced with the use of message oriented middleware (MOM).

MOMs generally allow the exchange of small messages on an IP network using communication patterns such as notification, request/reply and publish/subscribe. However, mobile devices also support non-IP networks such as the short message service (SMS). SMS can be seen as a MOM that is administered by a Mobile Network Provider (MNO). The advantage is its ubiquity and privileged access to the status of the device. Additionally, mobile devices can be reached simultaneously on multiple networks, but MOMs do not support this functionality.

In this work, we introduce a mobility middleware that will improve current MOM. Our mobility middleware uses a utility-based scheme for automatically choosing one of three network types: MNO, WiFi and the SMS. The selection scheme makes a decision using context information from the net-

work, the user preferences, the application preferences and the infrastructure. Additionally, delay-tolerant application can use our mobility middleware to delay messages in order to improve the utility. We also guarantee FIFO ordering and at-most-once delivery, at-least-once delivery or at-least-once processing as required. Finally, we implement our solution with a Java ME client and a Java middleware and evaluate the performance impact of adding our mobility middleware to a MOM.

Sommaire

Des avancements technologiques récents sur les systèmes embarqués ont permis aux cellulaires de devenir des puissants appareils et hôtes d'une panoplie d'applications. Cependant, la communication entre l'appareil et l'extérieur est encore une tâche compliquée parce que l'appareil est mobile, la connexion est intermittente, le signal varie et les options de communications sont hétérogènes. Des travaux passés ont suggéré que la communication avec ces appareils est simplifié et augmenté par l'utilisation d'un intergiciel par envoie de messages (MOM).

Les MOMs permettent l'échange de petits messages sur un réseau IP tout en utilisant plusieurs modèles de communication tels que la notification, la transmission sur demande et la publication-souscription. Toutefois, les cellulaires supportent aussi les réseaux non IP tel que le service de minimessages (SMS). Le SMS peut être considéré comme un MOM qui est administré par un fournisseur de service sans-fil. L'avantage du SMS est sont ubiquité et sont accès privilégié au statu du cellulaire. De plus, les téléphones mobiles peuvent communiquer sur plusieurs réseaux simultanément, mais les MOMs ne supportent pas cette fonctionnalité.

Dans ce travail, nous introduisons un intergiciel pour la mobilité permettant l'amélioration des MOMs suivant le protocole AMQP pour utilisant avec les téléphones intelligents. Notre intergiciel pour la mobilité utilise

un modèle d'utilité pour choisir un canal de communication approprié entre les fournisseurs de service sans-fil, Wifi et le SMS. Le modèle d'utilité fait une décision à partir de l'information du contexte courant du réseau, des préférences de l'usager, des préférences de l'application ainsi que de l'infrastructure. De plus, les applications qui supportent un délai peuvent utiliser notre intergiciel pour la mobilité pour introduire un délai et améliorer l'utilité. Nous garantissons l'ordre premier entré et premier sortis des messages ainsi que la livraison au plus une fois, au moins une fois ou le traitement au moins une fois tel que requis. Finalement, nous implémentons un client Java ME ainsi qu'un intergiciel Java et nous évaluons l'impact sur la performance de l'introduction de notre intergiciel pour la mobilité à un MOM.

Acknowledgements

I would like to thank MITACS and SAP for providing financial support to complete the work of this thesis. I would also like to thank Huaigu Wu as my contact at SAP who has supported me throughout this work. Finally, I would like to thank Bettina Kemme as my advisor at McGill for her countless advice, unrelentless help and belief in my work.

Chapter 1

Introduction

The goal of this research project is to design and implement a MOM that supports various communication patterns: event notification, request/reply as well as publish/subscribe interface. Using the publish/subscribe communication paradigm, users can subscribe to predefined channels and also publish messages (called notifications or events) on these channels. All users subscribed to a channel will receive the events published on it. Most implementations use a central event server that accepts subscriptions and notifications, and forwards notifications according to the subscriptions it has received previously. MOMs provide various degrees of reliability in regard to the delivery of published events (at-most-once, at-least-once, exactly-once), can keep histories of publications, and can provide a whole set of advanced features.

Several protocols already exist around MOMs, notably the Advanced Message Queuing Protocol (AMQP), the Extensible Messaging Presence Protocol (XMPP) and the Restful Message Service (RestMS). However, most existing MOMs have been developed for the wired world and will not work properly and reliably if some of the components are mobile devices that have intermittent connectivity, relatively low bandwidth capacity and carry the concept

of costs per data transfer. Thus, the goal of this work is to develop a mobile-aware MOM that is able to explore all possible communication mechanisms and that can dynamically and autonomously adjust its connectivity to the mobile devices, and the quality of service to the current configuration.

Our work is based on the AMQP because of the availability of enterprise-level open-source implementations such as RabbitMQ. We decided to implement mobility-related aspects in a mobility middleware that is placed between the mobile device and the MOM. A mobility middleware faces several challenges, Firstly, mobile clients have access to different wireless technologies. For instance, they can have a data plan with their service provider, they can access a locally available wireless network (Wi-Fi access), or they can transfer data via SMS. These network types differ in their coverage, cost, bandwidth, energy consumption, latency and reliability. Many devices are multimode and can support the data transmission on more than one of these networks at a time. However, switching dynamically between the networks is difficult and currently not easily supported. Also, it is not easy to determine which, at a given time, is actually the best network type to be used. Another challenge is the frequent disconnections and handoffs that occur when devices are relocating. Providing the quality of service known from the wired world is challenging or might not even be possible in this case. A mobility middleware has to adjust to these differences.

Motivated by these challenges, our research aims to build a mobility middleware with the following capabilities.

- It provides mobile clients several connection possibilities:
  - Standard Transmission Control Protocol over Internet protocol (TCP/IP) connection through the service provider and through wireless hotspots. In both cases the mobility middleware is able

to handle the fact that a single mobile device might frequently disconnect and reconnect with a different IP address.

  - Message exchange via SMS. SMS has characteristics that are very different to IP. In particular, while devices are typically not capable of accepting connections, and thus, cannot receive messages before initiating a connection by themselves, SMS can be sent asynchronously to mobile devices. This can be exploited in some cases. Also, some mobile users do not have a data plan, so SMS is the only option if there is no other Wi-Fi access. Even if there is a data plan, depending on the service coverage, it is possible to have SMS but not data coverage. Depending on the plans chosen, SMS might be cheaper or more expensive than a data transfer. In general, SMS is much slower, and can transfer less data than IP.

- Clients can dynamically switch between their communication channels without loss of service. Furthermore, the client can use many of these communication channels at a time.

- The mobility middleware uses a utility-based scheme to select the appropriate communication channel depending on the current context such as network conditions in terms of bandwidth and latency, user preferences in terms of cost and energy, application preferences in terms of delay and infrastructure in terms of access point positioning.

- It provides the same delivery guarantees as the standard MOMs: FIFO ordering and delivery guarantee such as at-most-once delivery, at-least-once delivery and at-least-once processing.

- It offers a scheme where messages can be delayed if an application is delay tolerant. This allows the mobility middleware to improve the utility of the transmission of a message by waiting for better network conditions to be present before exchanging a message.

Chapter 2

Background and Related Work

2.1 Mobile Network Environment

Current cell phones or mobile hosts (MHs) have access to different wireless technologies such MNO, WiFi and Bluetooth which are heterogeneous in terms of their coverage, cost, bandwidth, energy consumption, latency and reliability. Figure 2.1 [2, 23, 59, 63] shows approximate values for each of these properties depending on the technology that is employed. Obtaining accurate numbers for these properties depends on many factors on which we will not focus in this document. For data prices we use values for the Canadian MNO Rogers [64]. Figure 2.2 shows the interactions between each of these technologies.

One standard body has focused mostly on high throughput and short range wireless transmission. This body, the WiFi alliance, specifies the IEEE 802.11 specification which has seen its first deployment in 2000. Communication between a WiFi enabled MH and a device on the Internet is done via TCP/IP.

The Bluetooth specification focuses on low power and very short range wireless transmission. The Bluetooth Special Interest Group has managed

Technology	MNO		WiFi		Bluetooth
Revision	2	3	b	g	2
Down link (Mbps)	0.08	0.384	11	54	1
Coverage (m)	1000	1000	100	100	10
Energy Consumption (mW)	600	1500	1000	1000	120
Cost ($ / month)	30$/6GB	30$/6GB	-	-	-
RTT (s)	10	0.5	0.25	0.25	0.25

Figure 2.1: Approximate Coverage, Cost, Energy Consumption and Throughput of MNO, WiFi and Bluetooth technologies

the standard since 1998. Communication between a Bluetooth enabled MH and a device on the Internet is also done via TCP/IP.

MNOs have implemented various technologies over the years. Two standards bodies have emerged to try and consolidate them. A first collaboration group started in 1992 between implementers of the Global System for Mobile communications (GSM) technology currently has 80% of the handset market [58]. A second collaboration group started in 1993 between implementers of the Code Division Multiple Access (CDMA) technology has around 12% of the handset market. These two groups have agreed to implement similar features and interoperability. They have released several generations of technologies which are each characterized by higher bandwidth and higher power consumption. Given these tradeoffs and the cost of putting the infrastructure in place, all these generations are expected to stay in operation for many years to come. MNOs offer connectivity over large distance in populated urban and rural areas. The other major feature they are providing is interconnection with other networks such as the Public Switched Telephone

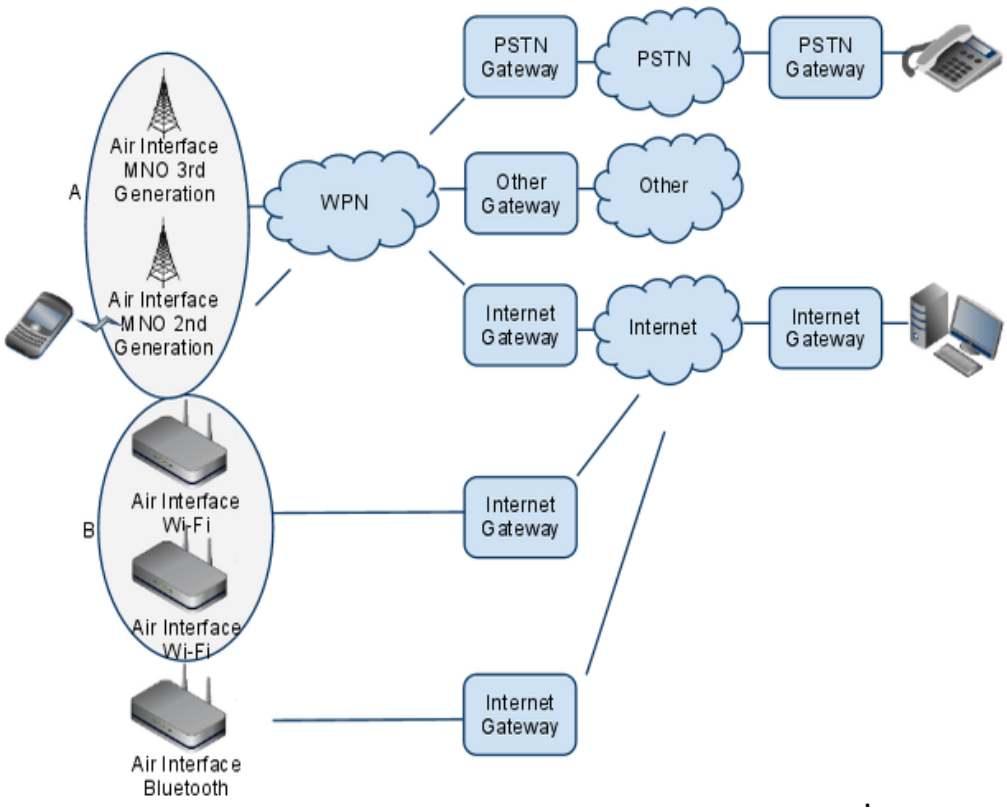

Figure 2.2: Interaction between MNO, WiFi and Bluetooth

	Horizontal Handover	Vertical Handover
Initiation	Triggered by one event (failing signal strength)	Triggered by multiple events (failing signal strength, need for higher bandwidth, lower cost, etc)
Selection	Choose among one parameter (Signal strength)	Choose among many parameters.
Execution	Same method can be applied every time.	Different method applied depending on technology
Adaptation	No adaptation necessary.	Adaptation desired.

Figure 2.3: Differences between horizontal and vertical handovers

Network (PSTN) which gives access to landlines worldwide (voice), the Internet (data, voice) and other MNO. Each handset that connects to the network is authenticated and the air interface they access is recorded. This information is used to locate the MH on the network as well as physically on the planet and for billing. Communication between a MH on a MNO and a device on the Internet is done via TCP/IP or the Short Message Service (SMS).

2.2 Handover

Since the coverage of a given wireless access point (AP) is finite, MHs must use a number of APs while moving. A handover is the process of transferring a call or a data session from one AP to another.

In a hard handover the source AP is released before the target AP is engaged. This means that the CC is broken before a new one is established. In a soft handover the source and the target AP are engaged before transferring the CC. This allows two simultaneous CCs where data/voice can be exchanged. We will only consider soft handover in this thesis.

We define the following handover related states:

- Initiation: Current context change that triggers the start of the handover process. Handover initiation is activated to maintain or improve the service.
- Selection: Choice between two or more APs to satisfy current context.
- Execution: Processing changes required to go from one AP to another. This includes updating the position of the handset, changing its address, changing the technology, rerouting, authenticating, etc.
- Adaptation: Modification to the traffic between two hosts to handle the current context. This includes adding delay, compression, filtering, etc.

A handover can be classified further into horizontal and vertical handovers. A vertical handover happens when a MH goes from one technology to another (e.g., Figure 2.2 handover 'A') while a horizontal handover is a transfer between two AP of the same technology (e.g., Figure 2.2 handover 'B'). Figure 2.3 shows how horizontal and vertical handovers differentiate from one another in each of the handovers described above. In this thesis we focus on vertical handovers where the AP change is instigated by multiple events provided a good cost/performance benefit.

Context A vertical handover is initiated by changes in context. We define context similarly to how it was presented in [20]. The network context (1) represents the current status of the network in terms of cost, bandwidth, energy consumption, latency and reliability. The application context (2) is the quality of service requirements defined on an application basis such as delay, error rate and priority. The user context (3) represents preferences a user may have in terms of data cost, energy consumption and mobility patterns. Finally, the infrastructure context (4) encompasses information about the AP location and coverage. For example, the machine servicing a

request from a MH could be chosen based on its distance to the MH's AP to lower the latency. Furthermore, one can make adaptation based decisions (e.g. delay) based on the time of day and the user's AP pattern (e.g. WiFi around 14:00 at work).

Selection Scheme There are five approaches that support the selection process in vertical handovers. The first approach focuses on choosing APs that will maximize or optimize one particular parameter of the network context such as monetary cost or energy cost. The second approach uses a cost or utility function. A function can capture a whole range of parameters of network context. The selection algorithm then chooses APs that maximizes these functions by obtaining the highest value possible in a utility model and the lowest value possible in a cost model. A third approach consists of specifying thresholds for network context values that cannot be crossed by the system. Once a threshold has been reached, the system must respond appropriately. The fourth method, a rule based approach, is similar to thresholds in that boundary values are specified for parameters of network context. However, these rules can additionally dictate appropriate actions to be taken when such a threshold is reached. Rule based systems are typically stateful and rules can modify the state of the selection scheme. Furthermore, rule based systems are not limited to the network context, but they can also specify rules about any type of context. Rules are typically stored in a database and they are specified by a rule specific language. The language allows administrators to easily change and add rules to the system. Finally, the fifth approach uses artificial intelligence (AI) to select the appropriate AP. AI is a combination of algorithms and heuristics used by the AI community that is adapted to use values of network context.

Selection Decision In a vertical handover, the selection decision can be taken and implemented at various entities. In a mobility middleware approach, a machine is between the server and the MH. The MH establishes

one or more communication channels to the mobility middleware via one or more APs and the mobility middleware gathers the current context. The mobility middleware then takes care of choosing the appropriate communication channel to reach the MH. In a MH approach, the MH opens one or more communication channels, gathers the current context and decides which communication channel it should use to transfer data with. More complex approaches can communicate with the AP to gather additional context information that cannot be obtained from the mobility middleware or MH approach only. Finally, network based approach uses many mobility middlewares typically at different physical locations. The mobility middlewares communicate with each other and coordinate to choose an appropriate mobility middleware and communication channel to communicate with a particular MH. Our work will focus on a hybrid between the MH and the mobility middleware approach. That is the decision of which communication channel to choose is made by the MH and the mobility middleware. Context information from the AP is not taken into account. In the rest of the document, the terms communication channel and AP are used interchangeably.

2.2.1 Soft Handover

In a soft handover different transmission strategies can be used to take advantage of having access to multiple APs.

- Multicast: The MH transmits copies of the same data to multiple APs in hope that at least one copy will reach its target. When the target has received a message once it can safely discards any copies. This method ensures the latency between the MH and the target is at its lowest because the target will use the first copy it receives.

- Scheduler Based: The MH segments the data it needs to send and sends it to the best AP according to the current context. If an AP has reached its bandwidth limit then the MH simply sends segments

to the second best AP. The target can receive segments out of order and it has to reorder them. This approach ensures the best use of the available bandwidth across multiple AP because the MH will simply send messages until all the AP have reached their bandwidth limit.

- Flow Based: The MH transmits data from an application to the best AP. If an AP has reached its bandwidth limit then he can send data from another application to the second best AP. The target receives data in order.

MHs that can support transmission to many APs are multimode or multihomed. MHs can support transmission of data on the MNO, SMS, WiFi and Bluetooth simultaneously. When a MH purposefully uses multiple APs to improve its transmission capabilities without the need to handover, the term multihoming is used. Our work will focus on scheduler based multihoming.

2.3 Mobile Applications in the Enterprise

Addressing mobile users is increasingly important for enterprises. In the past, laptops were the MH of choice. Software companies built specialized software for these MHs so they could support long periods without connectivity. SAP's software server side architecture for these MHs is shown in Figure 2.4. The backend can be accessed through several methods. Online laptops and desktops can access the backend through TCP connections or through Web Services. Prior to a disconnection, laptops can synchronize relevant information between a database on the device and a middleware called the Data Orchestration Engine (DOE). Synchronization is done by copying database tables from the middleware to the device. When offline, the laptop makes changes locally to the database on the device. When it goes back online it sends those changes to the middleware. Since the laptop operates with a database locally, additional business logic must be deployed on the

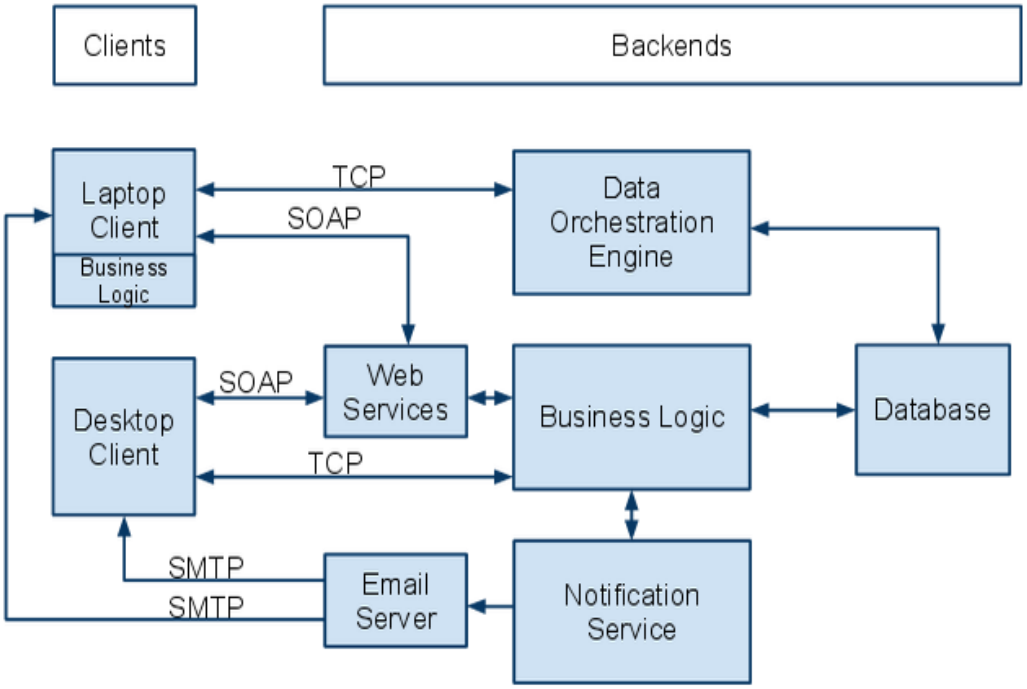

Figure 2.4: Old Mobile Architecture

device. Furthermore, the data on the laptop and the DOE is staged. Staging is the act of copying a subset of a database for a particular user. This way the whole database does not need to be copied to the laptop. The data on the laptop cannot be guaranteed to be up to date at any time. Finally, when online, notifications can be delivered via email.

This decade has been marked by a spur in usage of smartphones. A smartphone is a cell phone that offers superior connectivity and computing capabilities. A specialized high level mobile operating system runs on the device which deals with the constraints of energy, computing, memory, connectivity and gives access to the array of hardware available such as Global Positioning System, compass, specialized radio, accelerometer, gyroscope, camera, speakers, microphone, keyboard, Central Processing Unit and Graphics Processing Unit. While text entry remains a challenge on the small physical or virtual keyboard, these devices are ideal for multimedia consumption and light text editing. The major advantage of smartphones over laptops is their size which allows users to carry them on at all times. Given that worldwide smartphone sales have now reached 17% of phone sales [55] and that their computing capabilities are quite advanced, they are now an attractive platform to develop for. In the developed market such as in the United States, this trend is more pronounced with currently 21% of sales being smartphones and projections to 50% by the end of 2011 [57]. These smartphones with limited storage are connected most of the time and are disconnected only for brief periods of time. The connection is intermittent because of the changing wireless conditions such as signal strength, bandwidth, proximity to an AP and mobility of the device. This is different from laptops and desktop which can be connected with a wire or wirelessly but tend to be immobile. In this case, the connection lasts for long periods of time with long periods without connection. Large synchronization operations between the laptop and the DOE work well because of the high bandwidth and reliable connection.

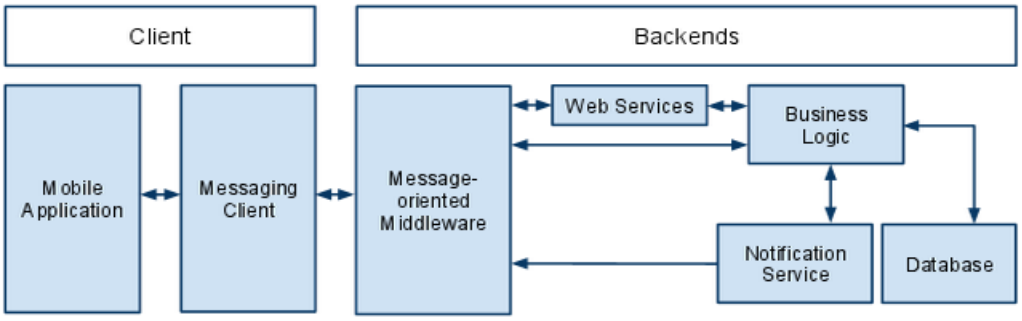

Figure 2.5: TANGO Architecture

However, when the connection is intermittent, asynchronous data exchanges small in size are preferred. TANGO [56] discusses these problems and proposes a message-oriented middleware (MOM) as the central component of the architecture to solve some of these issues. In the TANGO architecture shown in Figure 2.5, the mobile application communicates with a messaging client which abstracts the communication between the backend and the client. The messaging client also interacts seamlessly with web services and notifications from the backend by going through a MOM. However, TANGO does not consider in detail the usage of MOMs in the mobile environment. In particular, it does not fully explore all connectivity possibilities and does not consider in detail how they could be combined to provide a flexible MOM.

2.4 Communication Basics

Inter-process communication is done by two simple primitives. The *send(destination, message)* primitive is called by the sending process and the *receive()* primitive is called by the receiving process. Information between these two processes is exchanged via a communication channel. Each primitive can be blocking (synchronous) or non-blocking (asynchronous). A blocking send does not return until a confirmation has been received that the other site has received the information while a non-blocking send returns immediately. A blocking

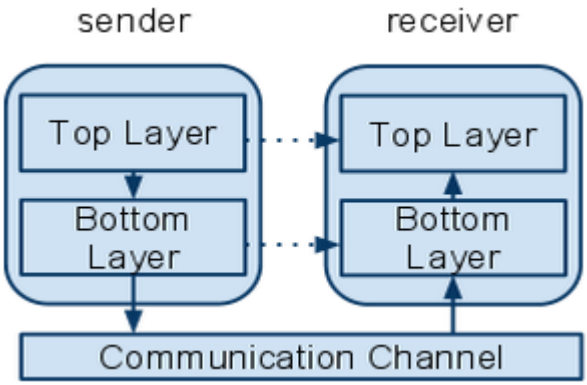

Figure 2.6: Simplified Layered Network

receive does not return until data has been received while a non-blocking receive returns immediately whether data is present or not.

Layered Network Model Communication between processes can typically be described as a layer approach. Each layer provides some service to the layer above. While each layer perceives sending a message directly to a receiver at the same layer, a message actually traverses all layers down at the sender and all layers up at the receiver. Note that as depicted in Figure 2.6, if a layer of the sender sends a message to the same layer at the receiver, it actually calls the send primitive of the layer below. Furthermore, at the receiver, when a layer has a message ready for the layer above, we say it *delivers* the message to the layer above. While the upper layer *receives* it from the lower layer.

A more complete model displayed in Figure 2.7 has 6 layers and each layer has its own functionality. The physical layer defines at the electrical and physical level how a machine can communicate with its environment. This definition includes voltage and particular transmission techniques used to send and receive information. The data link layer defines protocols to communicate between machines on a local network. This communication

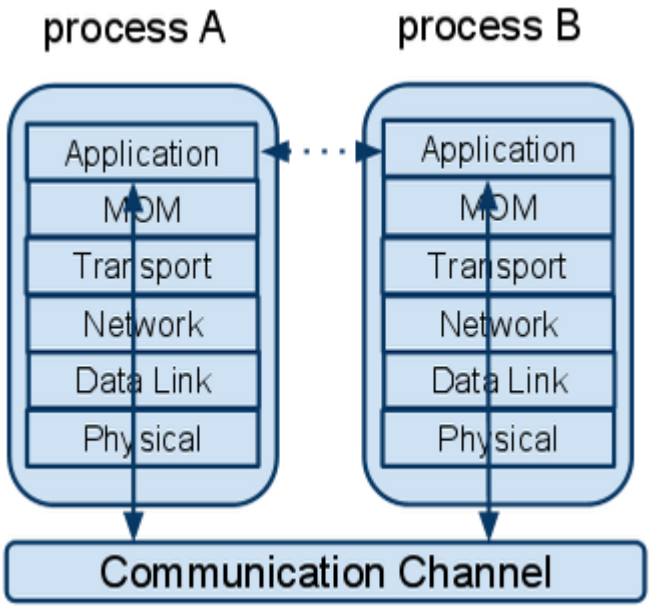

Figure 2.7: Layered Network

can be from one machine to another or from one machine to many (broadcast). This layer typically takes care of guaranteeing the accuracy of data transmitted by the physical layer. The network layer takes care of sending sequences of information via one or more networks. Its main function is to route the information correctly from the source to the destination. A well known network layer is the IP. The transport layer provides services on top of the other layers such as connection, error recovery, retransmission, segmentation and reassembly. The best known transport layer is the TCP. The MOM layer takes care of transmitting messages from one host to one or more hosts by using abstract destination addresses instead of single host IP addresses. Finally, the application layer takes care of creating and consuming content.

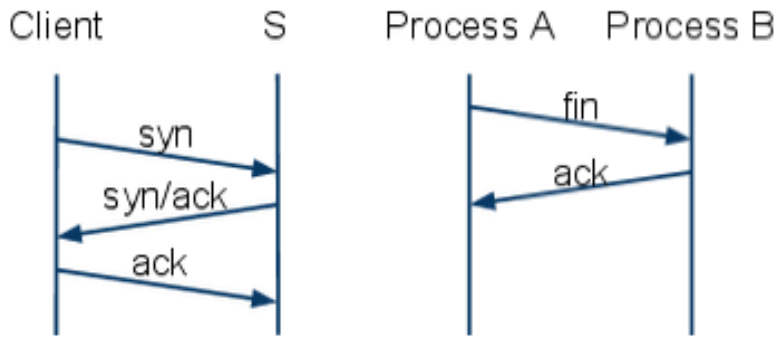

Figure 2.8: TCP Connection Protocol

2.5 Transmission Control Protocol

The TCP is a very important connection oriented protocol on the Internet. While the IP takes care of the direct transmission of information from one computer to another computer at the network layer, TCP handles other important functions at the transport layer. Each host on the Internet is identified by a unique IP address (x.x.x.x where x represent a digit from 1-255) and can receive many connections. Each unique connection to a host is identified by a port number (1-65535). TCP connections are point-to-point from one IP/port combination to another. The TCP three-way handshake connection protocol and two-way handshake disconnection protocol are shown in Figure 2.8. The server is the process who is waiting/listening for a connection while the client is the process who initiates the connection. The initial request is followed by acknowledgements from both sides. Any of the two processes can terminate the connection. TCP provides reliable and First In First Out (FIFO) delivery of segments. Each segment is a series of headers as well as a payload. Other services provided by TCP include segment size control, error detection, flow control and congestion control.

2.5.1 Reliability and Failure Model

Reliable communication protocoles typically guarantee the delivery of messages if both sender and receiver are 'correct', and are defined through two properties:

- Validity: Any message sent by a correct process is eventually delivered to a correct process.
- Integrity: The message received is identical to the one sent, and no messages are delivered twice.

The question now arises what 'correct' means and in this thesis we present communication paradigms that use two different definitions of correctness based on availability of processes. At any given time a process can be available, i.e. it is running and able to execute the protocole tasks, or it is unavailable, e.g., because it has crashed or is currently without network connectivity. Generally, one can assume that processes can restart and reconnect. So unavailability is usually temporary.

Some communication patterns, such as TCP, define a correct process as one that is available throughout the protocol execution. That is, in TCP both sender and receiver must be up at the same time to exchange messages. Other paradigms, that we will see later, only require the receiver to be eventually available, that is, while the sender is sender the receiver might not be available. But it will still receive the message once it becomes available again for sufficient time.

2.5.2 Reliability of TCP

TCP provides validity and integrity as defined above if both sender and receiver are available.

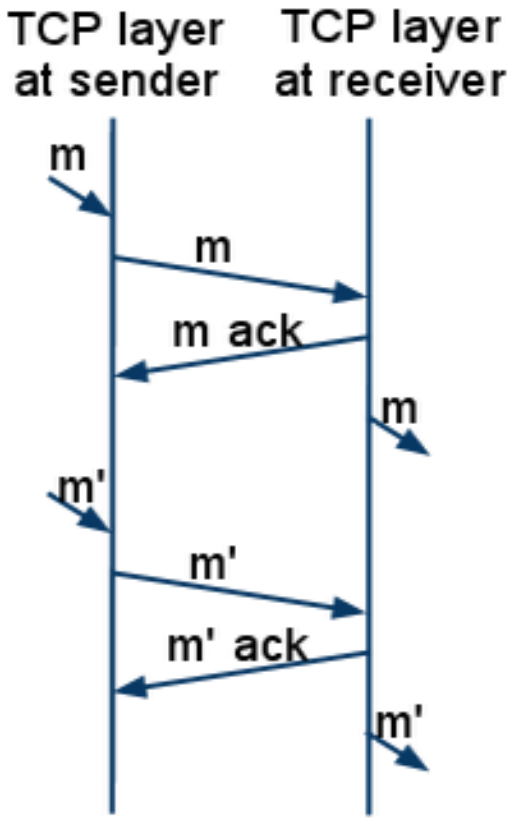

Figure 2.9: Positive Acknowledgements

Validity In order to guarantee validity despite message loss, TCP uses three types of acknowledgement schemes. Acknowledgements are messages sent by the TCP layer of the receiver side to respond to a message sent by the sender side. The TCP layer of the sender keeps track of each message it sends and waits for the acknowledgement to return before forgetting about the message. In a *positive acknowledgement* scheme as shown in Figure 2.9 the sender must wait for the acknowledgement *m ack* of the receiver to return before sending the next message *m'*. In a *cumulative acknowledgement* scheme as shown in Figure 2.10 the sender can *pipeline* *m* and *m'* before receiving one acknowledgement for each message received *m ack* and *m' ack*. In a *selective acknowledgement* scheme the sender can pipeline *m1*,

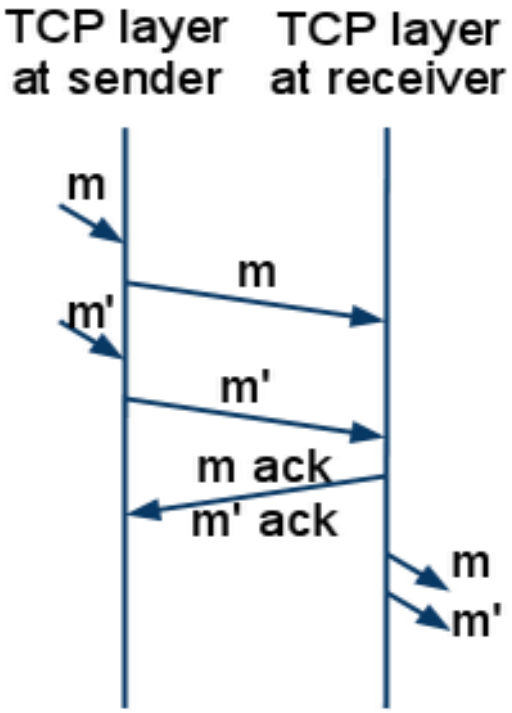

Figure 2.10: Cumulative Acknowledgements

m2 and *m3* before receiving the acknowledgement. The acknowledgement is specified as the first and last message of a range (e.g. *m1* and *m3*). In a *negative acknowledgements* scheme the sender can pipeline *m* and *m'* before receiving the acknowledgements. The receiver only sends acknowledgement when a message has not been received. Negative acknowledgements are not used by TCP. Finally, TCP implements acknowledgements with timeouts. When it sends a message it waits for the acknowledgement to return. If the acknowledgment does not return after the specified timeout, TCP will resend the message. These mechanisms guard against message loss.

Integrity In order to guarantee integrity the TCP layer adds a sequence number to each message. The sequence number is an ordered continuous range of unique numbers. The TCP layer at the receiver keeps track of the messages it receives. If two messages with the same sequence number arrive, the second one will be discarded. Furthermore, the TCP layer adds a checksum to each message which allows detection of certain errors.

In summary, if there are no process crashes, i.e., both the sender and the receiver are correct, a message sent by the sender is guaranteed to be received by the receiver exactly once and as it was sent.

2.5.3 Ordering

Ordering describes in which sequence messages are delivered to the receiver. TCP guarantees FIFO ordering of messages. If a sender sends messages *m1* and then *m2* a receiver will receive first *m1* and then *m2*. As mentioned in Section 2.5.2, TCP includes a sequence number in each message. When the TCP layer of the receiver receives message *m2* it will detect that it missed a message and wait for previous message *m1* and then deliver *m1* before *m2*. Figure 2.11 shows this property.

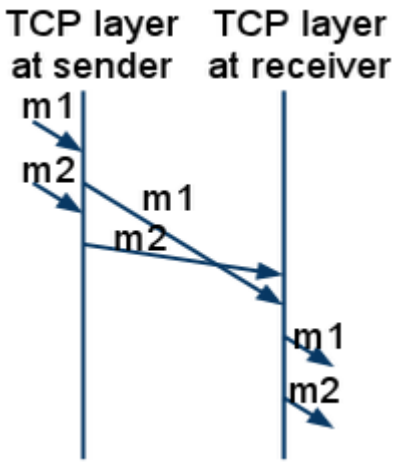

Figure 2.11: FIFO Ordering

2.6 Short Message Service

The SMS is one of the MNO' service. A messaging middleware called the SMS Center (SMSC) provides advanced messaging functionality to the application layer. The source MH and destination MH are each identified by their phone number. Each MH is also identified by a port number where the default is 0. These exchanges are point-to-point from one number/port combination to another. Each message is 160 7-bits characters or 140 bytes of binary data. The source MH first sends the message to the SMSC where it is queued. A queue is simply a FIFO data structure that stores messages. The SMSC takes care of handling and routing messages in transit. The SMS is widespread and heavily used. Reported counts put the volume per month in 2005 to 200 billion messages in Europe, 304.14 billon in China [53] and 9.8 billion in US [60]. For Rogers [64], the cost is 0.15$ per message or an unlimited bundle for 15$ per month [62].

2.6.1 Guarantees

TCP is used between the source MH and the SMSC and between the SMSC and the destination MH. However, the failure model of the SMS is not the same as TCP. A process, an MH or the SMSC, can crash and recover. That is, the process is unavailable but later becomes again available. Furthermore, network partitions can occur and eventually be resolved. That is a correct process is defined as one that might be unavailable but will eventually recover. The SMSC stores messages durably on disk to protect these messages against an SMSC crash. If the destination MH is unavailable temporarily, a message is kept on the SMSC for a certain time that is set according to the policy of the MNO. The maximum length of the queue is also a variable controlled by the MNO. Once the destination MH is online, the SMSC forwards the message. If the destination MH does not become available again before the holding period on the SMSC is reached then the message is deleted. Therefore validity is not achieved, as a correct process that becomes available after the validity period will not receive the message. Note however, that in a direct comparison of TCP, SMS actually provides stronger guarantees as it works in a different failure model. In TCP, if the sender and the receiver are not available at the same time then messages cannot be transmitted at all, but SMS supports short periods of unavailability. Integrity, however, is not respected because SMS does have any mechanisms to protect against duplicates or reordering in case of crash or disconnection. According to log traces of a provider in India [53], 73.2% of messages reach recipients within 10 seconds, 17% within a minute and 5% in more than an hour and a half. 5.1% of messages never reach the recipients.

2.7 Advanced Message Queuing Protocol

The AMQP is a protocol specification for MOMs. It is designed so that multiple implementations can inter-operate across enterprise's business system

and between organizations. Its development was initially started in 2004 for the finance industry by several companies such as JPMorgan Chase and Co, iMatrix, Red Hat, IONA technologies, TWIST Process Innovation and Cisco Systems. However, the working group now includes 20 companies in various domains. The specification delivers features such as routing, reliability as well as some ordering properties that an implementation of the specification must fulfill. The MOM is used as an abstraction layer for communication between processes, consumers and producers, on different machines, operating systems and network protocols. The application layer describes processes which create (Producers) or destroy (Consumers) messages. The MOM layer takes care of sending messages between these entities. AMQP uses TCP at the transport layer between hosts. For the remainder of this document we will be referring to AMQP version 0.8. The choice was made given availability of implementations and not based on functionality. More recent version includes AMQP 0.9, AMQP 0.91 and AMQP 1.0. In this section of the document we do not intend to copy the whole specification, but to inform of the architectural and functional specifications relevant to solve our particular problem.

Although the specification does not force any particular architecture it is mostly used for client/server topology. This is the architecture we will be considering. In a client/server architecture the clients, producers and consumers, communicate with a server, the MOM. Producers, consumers and the MOM are typically located on different physical machines.

2.7.1 Communication Patterns

AMQP is flexible and can enable many message patterns. We define four that are of interest in this work.

Point-to-Point

A producer sends a message to a consumer. This pattern is useful when a producer wants to execute an operation (write request), send a notification or stream some data to a consumer. Point-to-point message assume that the sender and the receiver are available at the time of transmission just as TCP does.

Request/Reply

A producer sends a message to a consumer (Request). The consumer then sends a message to the producer (Reply). This pattern is useful when a process wants to get data from another process (read request) or he wants to execute an operation (write request) and get the result. Request/reply again assumes that both the producer and the consumer are available throughout the message exchange.

Publish/Subscribe

In this pattern the consumer shows its interest for messages using a certain scheme or subscription language. Conversely, the producer publishes messages using the same scheme. This means that the producer and the consumer do not hold references to each other, but rather to a particular scheme. This allows a changing number of processes, producers and consumers, to participate in an interaction without knowing each other. This property is referred to as space decoupling. The publish/subscribe paradigm also allows for the participating processes to be present at different time. A producing process might send a message while a consuming process might be unavailable. The message is only received later by the receiving process. This is similar to what we have discussed in the guarantees provided by SMS. This property is referred to as time decoupling. Finally, producers send messages asynchronously and consumers receive messages asynchronously. The exchange

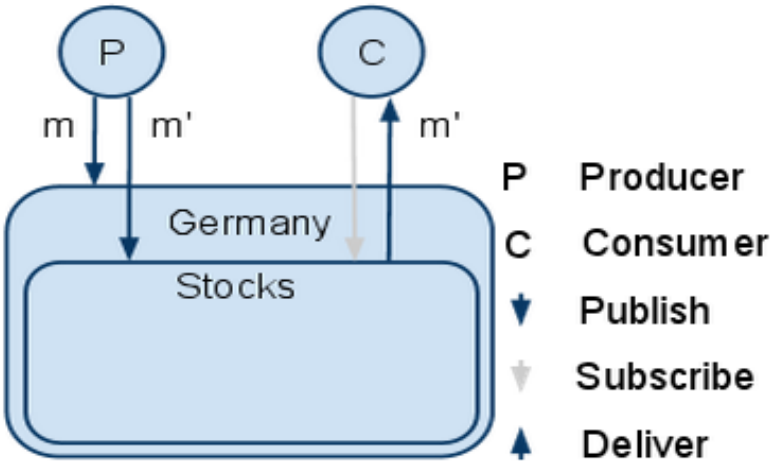

Figure 2.12: Hierarchical Topics in Publish/Subscribe

of information between the producer and the consumer is not synchronous and this is why we refer to this property as synchronization decoupling. Again, this is similar to what we have discussed in the guarantees provided by SMS.

The subscription languages can be classified into three categories: topic (also known as group, channel or subject), type (also known as content-based) and content (also known as content-based with pattern matching). The topic subscription is the most common and it is the one we will review. In a topic based subscription, each message is classified by a single keyword or topic (e.g. 'stocks'). Producers publish messages to a topic and consumers receive messages from the topic they are registered to. Many publishers can publish to the same topic and many consumers can consume the same topic, all receiving all messages that are being sent on that topic. Topics can be defined in a hierarchy. In Figure 2.12 the topic 'stocks' is a sub topic of 'germany'. If a consumer registers to the topic 'stocks' then it receives messages published to this topic alone. However, if a consumer registers to the topic 'germany' then it would receive messages published to 'germany'

and ‘stocks’.

Store and Forward

Similar to publish/subscribe a producer send messages to a topic and consumer can register to a topic. If the consumer is unavailable the message is stored on the MOM until the consumer becomes available. Different to publish/subscribe, messages sent to a topic are shared among the consumers that have registered to the topic. This is done in a round robin fashion. That is, if messages m and m' are published to a topic to which consumers c and c' have subscribed to, each of the consumers will receive one of the messages.

2.7.2 Entities Architecture

This section describes the various entities that exist in an AMQP implementation to provide interoperability. Modular components are connected as a chain to form the messaging interaction.

- A *producer* is a process that creates messages and sends them to the MOM. It belongs to the application layer.
- A *consumer* is a process that receives messages from the MOM. It also belongs to the application layer.
- A *message* is composed of several headers such as the routing key and a body that carries the content. Messages are exchanged between the producer, MOM and consumer in the form of one or more frames. A frame is the binary representation of a segment of a larger message.
- The *routing key* is a string that represents a virtual destination address for a message. The string can contain one or more tokens delimited by periods (‘.’) (e.g. ‘germany.stocks’).

- A *message queue* is a named FIFO data structure that stores messages. Queues have several properties such as durability and shareability. Queues can be consumed by one consumer for the point-to-point, request/reply and publish/subscribe communication pattern or shared by many consumers in a round robin fashion for the store and forward communication pattern. Finally, a message is removed from a queue only when it has been *successfully delivered* to a consumer. The queue is part of the MOM layer.

- An *exchange* receives messages from one or more producers, inspects their routing keys and routes them to one or more *message queues*. Exchanges do not store messages. They take care of routing multiple identical copies of a message to all the appropriate queues. The exchange is also part of the MOM layer.

- A *binding* is a link between an exchange and a message queue. So while the exchange does the routing, the binding describes to which queue the exchange should send the message. A message queue can be bound by multiple bindings.

- A *binding key* is a string that defines the binding (e.g. 'germany'). The string can contain one or more tokens delimited by periods ('.') (e.g. 'germany.stocks'). The string can also contain the star ('*') wild-card which matches a single word and the pound wild-card ('#') which matches zero or more words (e.g. 'germany.*' or 'germany.#').

- A *physical host* can contain multiple *virtual hosts* or AMQP instances.

- A *connection* is the communication channel used between two hosts.

AMQP defines several types of exchanges to define the routing algorithm. The *direct exchange* reads a message's routing key and does an exact match to a queue's name. For example a message with routing key 'stocks' would

be routed to a single queue named 'stocks' if such a queue exists. When the match is successful the message is transferred to the queue. The *topic exchange* reads a message's routing key and matches it to a binding's binding key. In this case the routing key is referred to as the topic and must be matched to the binding key. Topics are defined into hierarchies by periods. In our previous example 'germany.stocks', 'germany' is the upper most topic while 'stocks' is a sub topic of 'germany'. If a binding key is defined as 'germany.*' then messages sent with the routing key 'germany', 'germany.stocks', 'germany.bonds' will be matched successfully and the messages sent to the queues that are bound to this binding. If a binding key is defined as 'germany', then messages sent with the routing key 'germany.stocks' will not be matched.

AMQP provides two types of interfaces. One interface creates exchanges, binding and queues (i.e. it allows for the management of the infrastructure). The other interface provides all the methods for the message exchange between producers and consumers. The producer publishes messages via the *publish(exchange,routingKey,message)* message. The consumer initially registers its intent for messages of a queue via the *consume(queueName)* message. The MOM sends messages to consumers via the *deliver(m)* message which upon reception by a consumer triggers an event. The event is triggered at the application layer of the consumer where the message is processed. The publish/deliver methods are used for all four patterns as described in section 2.7.1

2.7.3 Reliability Features

Depending on the message pattern different failure and availability models are assumed. Point-to-point and request/reply guarantee message delivery only when producers, consumers and MOM are available during the message exchange. Publish/subscribe and store and forward assume producers to

be available until the message is received by the MOM. The receiver has to be eventually available, but can be initially unavailable. The MOM can crash (be unavailable), but it is expected to recover and become available again. However it is expected to be available when a producer produces a message. To defend against temporary unavailability AMQP provides several reliability features as described below.

Durable queues Queues on the MOM can store messages durably on disk. In comparison to non durable queues, this measure helps protect messages against MOM crashes.

Acknowledgement The consumer configuration will influence the time at which the MOM considers a message as successfully delivered.

- In the first configuration, the consumer does not require acknowledgements. The MOM removes the message from the queue once the TCP send primitive was successful.
- In the second configuration, the consumer requires acknowledgement. After processing a message by the application layer the consumer sends an acknowledgement. The reception of this acknowledgement by the MOM will mark successful delivery. AMQP supports positive, cumulative and negative acknowledgments schemes. Contrary to TCP, the MOM will only resend a message when the consumer requests for a redelivery. There is no acknowledgment timeout involved on the MOM. Note that the acknowledgement is not created by the MOM layer. The application layer of the consumer has to take care of creating and sending this acknowledgement.

Acknowledgements between the MOM and the consumer allow for at-least-once processing despite consumer failures.

Transactions Transactions are operations which allow two or more processes to synchronize to provide atomicity and durability of several messages.

- Atomicity: Refers to one or more requests that appear to processes in an interaction as a single operation which can either fail or succeed.
- Durability: A receiver has received a request only when it is stored durably on disk. This request can be recovered on process crash.

When there are only two processes, a consensus algorithm such as the one phase commit (1PC) can be used to provide the properties of a transaction. Figure 2.13 depicts the sequence of messages that are exchanged in such a case. In AMQP transactions are provided by the use of three methods. The *start()* command indicates the beginning of a transaction. The producer then sends one or more messages to the MOM using the *publish()* command. Finally, the producer completes the transaction with the *commit()* command. When the MOM receives the *commit()* command it processes the messages and sends back its answer (success or failure via *commit-ok*). If the operation was a success than the MOM must put the messages in a durable queue before sending the response. If the operation was a failure than the MOM discards the messages and sends the response. The ordering of the messages is determined at the time they are published and not at the time of the commit. In figure 2.13, any crash at the MOM at time 3, 4 and 7 can be recovered by the producer by restarting the transaction. However, if the MOM crashes at time 8 or the 'commit-ok' response is lost, the producer does not know if the transaction was successful. In that case the producer will need to retry the transaction. The semantic is that messages are atomically and durably stored at-least-once.

End to end semantic

Let's look at the achievable semantics for message delivery from the producer to the consumer in the case where the consumer or the MOM crash or are

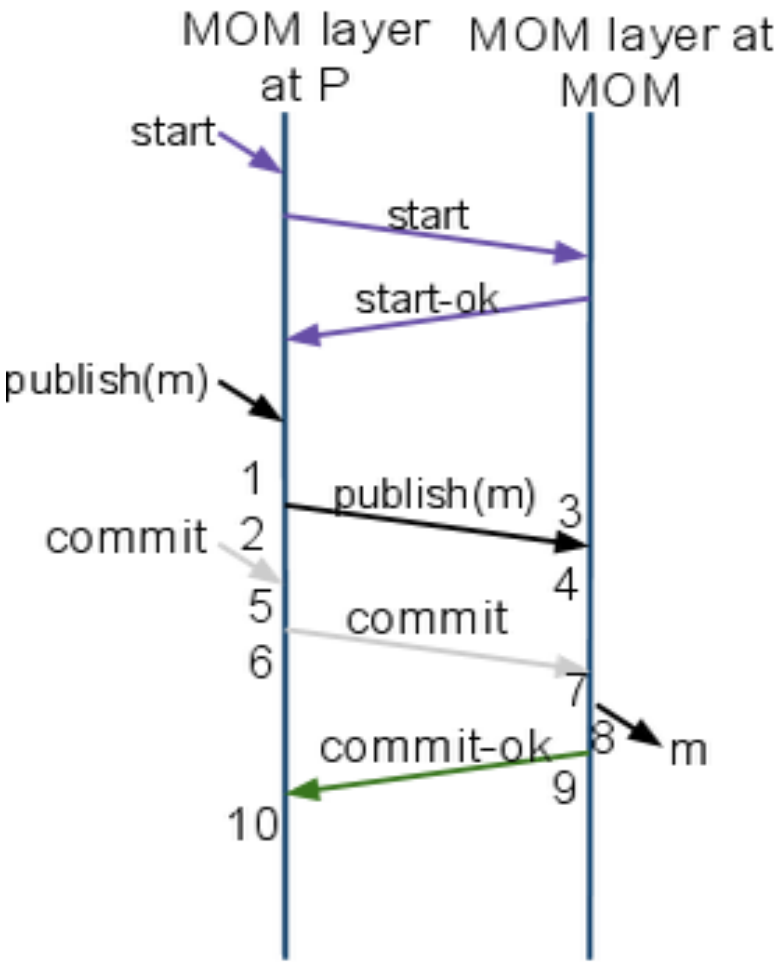

Figure 2.13: One Phase Commit

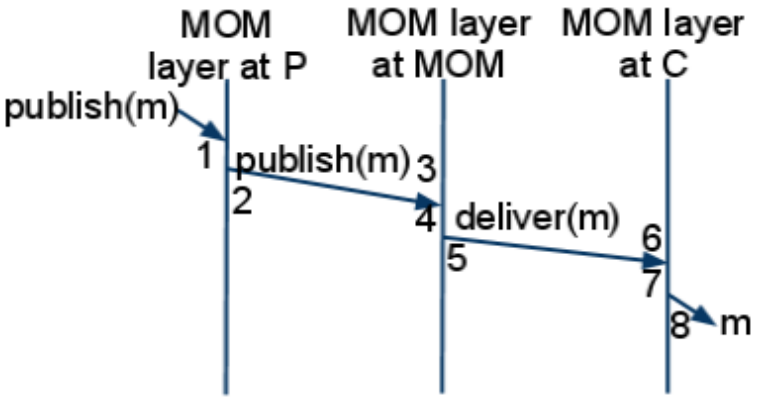

Figure 2.14: At-most-once delivery

unavailable for a certain period of time but eventually recover.

At-most-once delivery This semantic means that a message is received at-most-once by a consumer. In this case, the producer publishes messages to the MOM without using transactions and the queue is not durable. The consumer consumes messages without acknowledgments. Figure 2.14 depicts the exchange between the producer and the consumer. The producer simply sends a message to the MOM and the MOM sends the message to the consumer. If the MOM crashes after receiving the message at time 4 or the consumer crashes at time 6 or 7, then the message is lost. If there are no such crashes then the message is received once by the consumer. That is, in case of all entities being available, exactly once is provided.

At-least-once delivery This semantic means that a message is received at-least-once by a consumer. In this case, the producer publishes messages within a transaction and the queue is durable. The consumer sends an acknowledgment before processing a message. Figure 2.15 depicts the exchange between the producer and the consumer. The producer publishes a message in a transaction that, as we've seen in section 2.7.3, is stored durably and atomically at-least-once at the MOM after time 8. Messages stay stored until

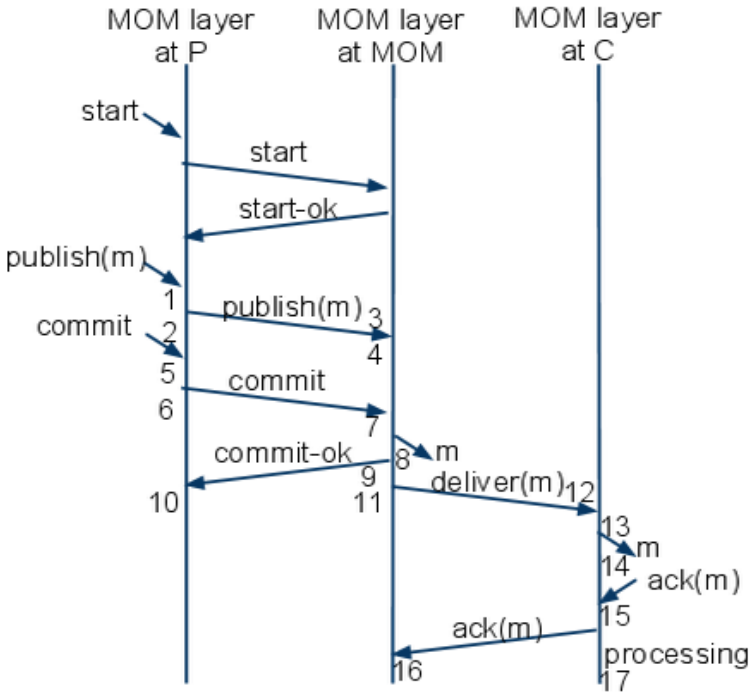

Figure 2.15: At-least-once delivery

the MOM receives an acknowledgement at time 16. This prevents message loss from MOM crashes between time 9 and 11. The MOM then sends the message to the consumer. The acknowledgement sent by the consumer guarantees that any crash at time 12, 13 and 14 can be recovered. When the consumer recovers, it simply sends a request via the *redeliver()* interface to receive any messages that have been sent by the MOM but for which the MOM has not received the acknowledgement. When the consumer crashes at time 15 or 16 the message has not yet been processed. The message is guaranteed to be delivered to the consumer at-least-once, but not to be processed. If the consumer and the MOM are available throughout the protocol then the semantic is exactly-once.

At-least-once processing This semantic means that the message is received and processed by the consumer at-least-once. In this case, the producer publishes messages within a transaction and the queue is durable. The consumer sends an acknowledgment once it has completed processing a message. Figure 2.16 depicts the exchange between the producer and the consumer. The producer sends a message to the MOM in an at-least-once manner using transactions. The MOM then sends the message to the consumer. The acknowledgement sent by the consumer guarantees that any crash at time 12,13,14 and 15 can be recovered by sending a *redeliver()* request. When the consumer crashes at time 16, the message has already been processed by the application layer of the consumer. Message redelivery will cause the message to be processed a second time. The message is guaranteed to be processed at-least-once. Again, if all processes are available throughout the protocol exactly-once is obtained.

2.7.4 AMQP Communication Patterns

AMQP makes the distinctions between two types of requests. First, an immediate request where a producer sends a message to a queue, a consumer

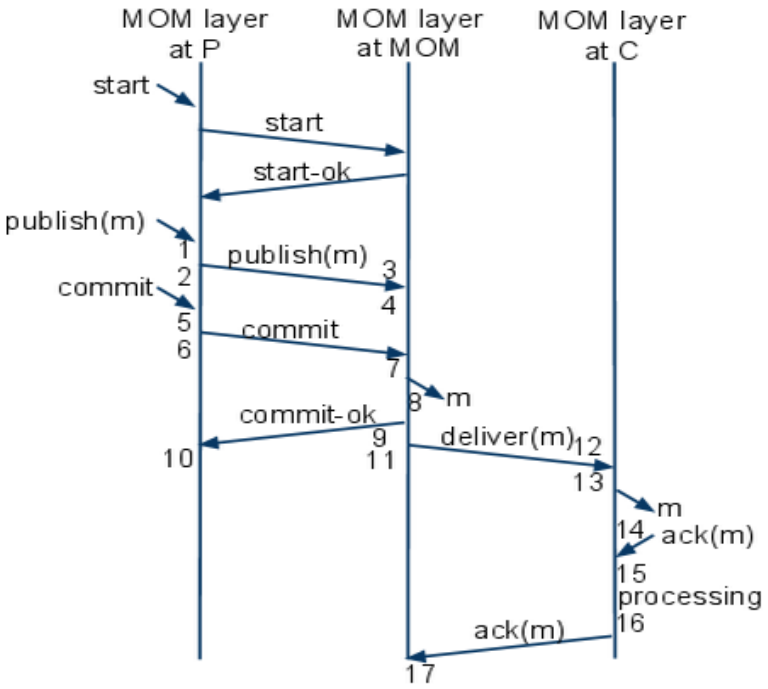

Figure 2.16: At-least-once processing

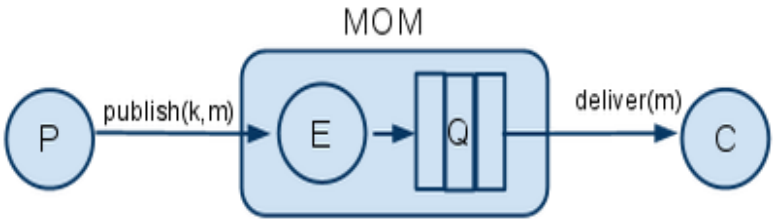

Figure 2.17: Point-to-Point

is attached to that queue and consumes it. If no consumer is present then the message is discarded. This is the proposed type for point-to-point and request/reply communication patterns. That is reliable delivery is provided if all processes are available throughout the message exchange. Second, a non immediate request where a producer sends a message to a queue, a consumer is not currently attached to that queue, but is later available to consume it. This is the proposed type for publish/subscribe and store and forward communication patterns. Reliable delivery is provided even if the consumer is temporarily unavailable.

Point-to-Point

Figure 2.17 depicts the components required to implement the point-to-point communication pattern as described in 2.7.1. A direct exchange E and a queue Q need to be created. The producer P sends a message m to the MOM with routing key k where k is equal to Q. The consumer C consumes the queue.

At-most-once delivery semantic is chosen. The ordering in such a case is FIFO.

Request/Reply

Figure 2.18 depicts the components required to implement the request/reply pattern. A direct exchange E and queues Q and Q' need to be created.

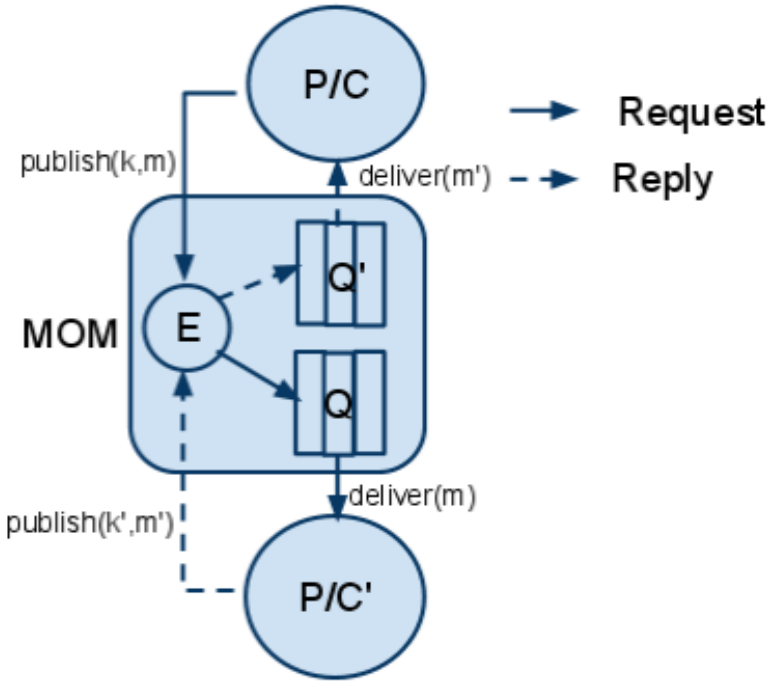

Figure 2.18: Request/Reply

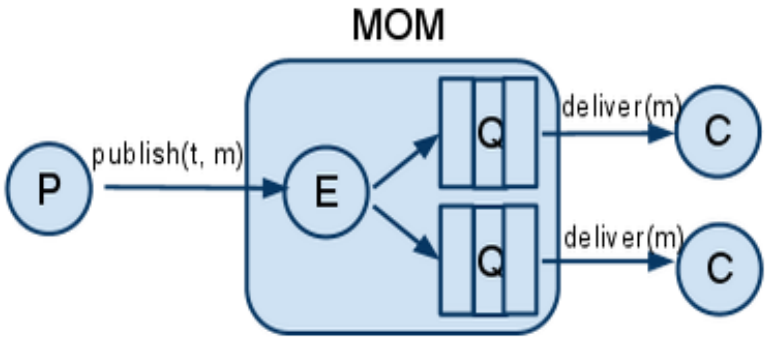

Figure 2.19: Publish/Subscribe

Producer and consumer P/C needs to send a request m with return address Q' and routing key k where k is equal to Q. Producer and consumer P/C' needs to send reply m' with routing key k' where k' is equal to Q'.

At-most-once delivery is chosen. The delivery in such a case is FIFO. If an application requires at-least-once delivery then it can easily be followed by having a timeout on the producer that waits for the response to come back. If no response returns then the producer would send a new request. Duplicate detection is needed at the consumer.

Publish/Subscribe

Figure 2.19 depicts the components required to implement this mechanism. Each client has its own durable queue and binds it to the topic they subscribe to. The binding is associated with the corresponding exchange. In the example, consumers C and C' subscribe to the same topic. The producer P sends a message to this topic.

This type of pattern requires at-least-once delivery. The ordering in such a case is FIFO.

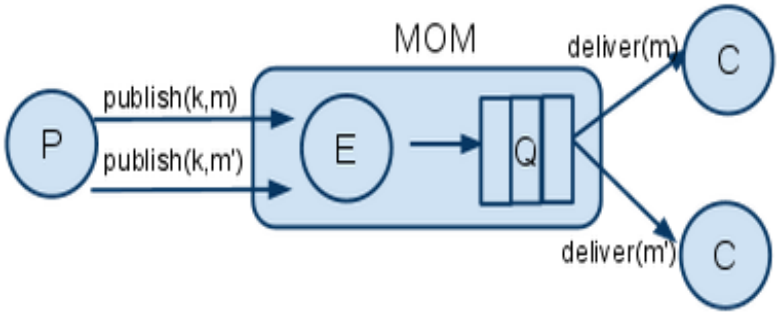

Figure 2.20: Store and Forward

Store and Forward

Figure 2.20 depicts the components required to implement this mechanism. A direct exchange E and a single queue Q need to be created. The producer P sends a message m to the MOM with routing key k where k is equal to Q. The consumer C consumes the queue. If there is more than once consumer registered with the queue, messages are delivered in round-robin.

This type of pattern requires at-least-once delivery. The ordering in such a case is weak-FIFO. The MOM sends messages in a round robin fashion to each consumer sharing a queue. However, when consumer C crashes a message can be redelivered to any of the consumers (e.g. C'). The message then arrives out of order. Figure 2.21 illustrates this issue.

2.7.5 AMQP Queue Management

AMQP offers two types of application specific queue management. First, each message is assigned a priority by the producer. The priority is represented by a number from one to nine where higher priority is represented by higher number. The MOM reorders messages in a queue from the highest to the lowest priority without losing ordering within the same priority level. Second, the producer assigns a TTL in milliseconds to each message. When

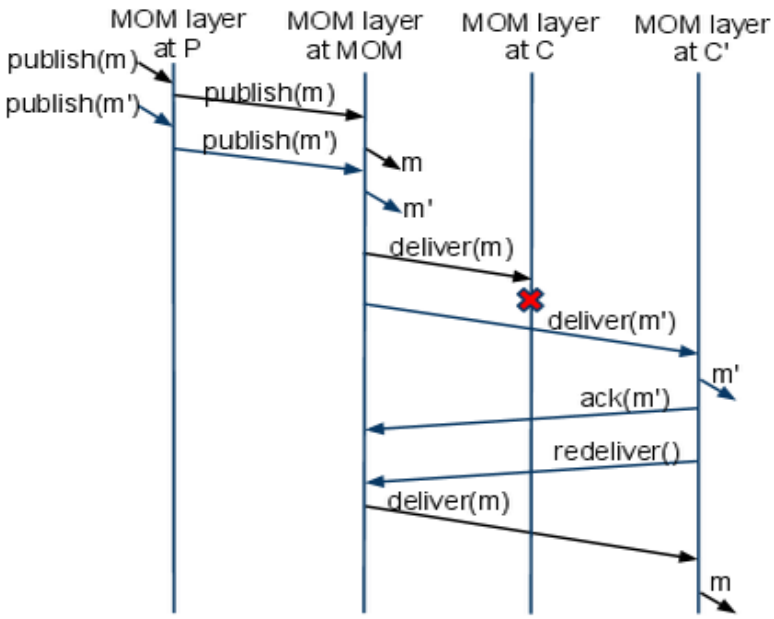

Figure 2.21: Ordering delivery with queue sharing

a message is received by the MOM a timestamp is inserted in the header of the message. The TTL in milliseconds is then added to the timestamp to obtain the maximum time at which the message should be delivered. The MOM deletes messages which are over this limit.

2.8 Related Work

2.8.1 Publish/Subscribe

Publish/subscribe [27] is a well known problem that has been looked at extensively by previous work. Existing systems are classified either by their subscription language or by their architecture. Previous work from A. Carzaniga et al. [37], has already done extensive review of related work in this field. We will not repeat that here given that the focus of our research will be on the delivery on those events and not improving the internal working of

		Architecture		
		Centralized	Client/Server	Peer-to-peer
Subscription Language	Topic	ToolTalk [43]*	NNTP [44]* JEDI [51]* TIB/Rendez-vous [36]* JMS [33] IBus [35] IBM's MQSeries [38] BEA's WebLogic [39] AMQP [26] XMPP [42]	Narada [34]*
	Type	Elvin [45]*	Keryx [46]* Yu et al. [47]*	Gryphon [48]* Hermes [48]*
	Content	GEM [49]* Yeast [50]*	Siena [37]*	Siena [37]*

Table 2.1: Publish/subscribe systems categorized by architecture and subscription language

these systems. Businesses have developed various standards to allow interoperability between MOMs. The most prominent one, Sun's JMS API [33], is an API that is accommodated by many commercial implementations such as IBM's MQSeries [38], Microsoft Message Queue (MSMQ), TIBCO's TIB/Rendezvous [36], Softwired's iBus [35], and BEA's WebLogic [39]. Researchers have also supported this API, see Hermes [40], Siena [37] and Narada [34]. Since JMS is at the API level, interoperability is not guaranteed. Other more recent standards have defined protocols to solve this problem. AMQP [26] is a protocol that focuses on publish/subscribe and other communication patterns. XMPP [42] is an XML instant message protocol with presence notifications. Presence notifications inform connected users of the other users' connection and disconnection status. With the new protocol enhancement XEP-0060, XMPP also handles publish/subscribe. That being said, AMQP is better for our needs since it is out of the box very efficient, simple to work with and has many free industrial strength implementations available. But of course, none of these solutions has been designed with mobility problems in mind.

Table 2.8.1 categorized all these system by architecture and subscription language. A star (*) indicates that the data was taken from referenced table

in [37].

There has been some work in developing publish/subscribe systems for mobile environments. Some selected publications are by Jacobsen et al. [28], A.P. Buchmann et al [29], L. Fiege [30], Y. Huang et al. [31] and Q. Yuan et al. [32]. However, they typically consider only one communication channel type or completely abstract from it. Instead, the focus is on building broker networks in ad-hoc networks (e.g., [32]), or handling frequent disconnection and location changes (e.g., [5], [29], [30] and [31]). Jacobsen et al. present a system where different communication channels can be supported, but there is no discussion on how to choose the communication channel or an evaluation of how the different communication channels fair. Coming back to business solutions, mobility middleware have been built for the communication between MHs and publish/subscribe systems. Such mobility middleware include Softwired iBus/Mobile [35] and MQ Everyplace [41]. These mobility middlewares offer support for MH specific communication channels such as MNO and SMS, but they do not give support for handover between these heterogeneous technologies.

2.8.2 Communication Channel Selection

Table 2.8.2 provides a summary of the existing communication channel selection approaches. An IEFT standard called Mobile IP [21, 22] is probably one of the important solutions to the mobility problem. It provides location independent routing of IP traffic to a MH by relaying data using a non mobile proxy. However, it is incomplete in many regards as it assumes there is only one way to reach a MH. Furthermore, it does not handle vertical handovers nor does it take into account current context such as user preferences. It also cannot be used over non IP networks or services such as SMS. L. Chen et al. [1] improve upon Mobile IP by introducing a way to achieve seamless handover with lower latency. However it does not solve any of the

Ref id (Year)	Scheme	Context	Decision	Initiation	Selection	Execution	Adaptation	Multilmode
[3] (98)	No Scheme	2	Proxy	No	No	Yes	No	Flow based
[12] (99)*	Cost	No	Middleware	No	Yes	Yes	No	Multicast
[13] (00)*	AI	No	Middleware	Yes	No	No	No	No
[14] (01)*	AI+Rule	No	Middleware	Yes	Yes	No	No	No
[16] (02)*	Rule	No	Network	Yes	No	No	No	No
[5] (03)	Optimization	4	Network	Yes	No	No	No	No
[17] (03)*	Rule	1,3,4	Network	Yes	No	No	No	No
[15] (03)*	Rule	1,3,4	Network	Yes	No	No	No	No
[18] (03)*	Threshold	1	Middleware	Yes	No	Yes	No	No
[19] (03)*	Rule	No	Middleware	Yes	No	No	No	No
[4] (04)	Optimization	1,2,3	MH	Yes	Yes	Yes	No	No
[20] (04)*	Rule	1,2,3	Middleware	Yes	Yes	Yes	Yes	No
[1] (05)	Threshold	1,3	MH	Yes	Yes	Yes	No	No
[9] (05)	Rule	1,2,3	Middleware	Yes	Yes	Yes	Yes	No
[7] (05)	Rule+Utility	1,2,3	Middleware	Yes	Yes	Yes	Yes	Scheduler
[2] (06)	Rule+Utility	1,2,3,4	Middleware	Yes	Yes	Yes	Yes	Scheduler
[8] (06)	Optimization	1,2	MH	Yes	Yes	No	Yes	No
[6] (07)	Optimization	1,4	MH	Yes	Yes	No	Yes	No
[11] (08)	Optimization	1,4	MH+AP	Yes	Yes	Yes	No	No
[10] (09)	AI	1	MH	Yes	Yes	Yes	Yes	Flow Based

Table 2.2: Comparison table of communication channel selection approaches

before-mentioned problems.

There exists considerable work in regard to choosing the right communication channel. One of the first attempts was done by a modification to TCP with a proxy based architecture by D. Maltz et al. [3]. The solution focuses on how the IP data should reach the MH by using a novel technique called TCP slice. However, they purposefully choose to not select a particular communication channel decision scheme by citing previous work by B. D. Noble et al [25]. Wang et al. [15] have proposed cost-functions to do the selection on a mobile IP proxy based architecture.

A. Seth et al. [2] added queues for messages to unreachable hosts. This allows applications to deliver data with a maximum user specified delay and utility-based on the currently available communication channels. Selection is modeled via rules as well as via utility-functions where their values decay with time. Messages with deadline are ordered from furthest deadline to nearest before being transferred assuming the worst case scenario. If a better communication channel becomes available, the system switches to the new communication channel and the worst case completion time is then recomputed. This simple policy does not take into account the current network

context or infrastructure context. While [2] use SMS as a control channel, we question why they have chosen not to use it as a data exchange channel as well. Finally, the authors choose to employ scheduler based multihoming to improve throughput, but they did not investigate the cost associated with such a technique. Legacy applications communicate with MHs through the mobility middleware where plugins are added to support each of these applications. New applications store data to be transferred as flat files in the operating system. This data transfer model similar to FTP is arguably simpler then the publish/subscribe paradigm but less powerful.

Utility-function based approaches are fairly limited in their expression and can become quite complex by adding context support. A. Qureshi et al. [7] have tried to add flexibility to utility-functions by packaging them as objectives in which a "context" defines filters to apply to the traffic and a "goal" defines various quality of service requirements. The absolute value of the "utility" can then be compared to other objectives that have been selected. Unfortunately, the flexibility of the language used by the authors does not reduce the complexity of writing the policies.

There have been many attempts to use artificial intelligence to select a communication channel such as J. Makela [13] (Neural Network), Chan et al. [14] (Fuzzy Logic) and T. Alpcan et al. [10] (Markov Decision Process and Clustering). There are several problems with using these techniques. First, it is nearly impossible for the IT department to understand why a certain decision was taken by the system. Second, they add a lot of computation to the decision process and could be impracticable in terms of scalability and time to decision. Third, these approaches fail to take into account the user preferences in the decision process. Fourth, they require priming before being able to be used. Finally, in many situations the decision process is straightforward and the use of these approaches becomes excessive.

Probably the most common approach is to use a rule system. It is rich enough to define appropriate policies for context support while being easy to configure for IT personnel. N. Fikouras et al. [18] focused on link layer information (signal strength) to help in the handover decision process. They fail to present a solution that offers network context support during AP selection because information from other layers is necessary. P. Vidales et al. [20] have worked on implementing decision support and context support by using a rule system. However they fail to account for multihomed devices or the infrastructure context. In [9], P. Vidales et al. augment the solution by adding a new kind of automata called "Finite State Transducer with Tautness Functions and identities". Although this solution solves problems around rule conflict management, this approach complexifies the rule elicitation process. Simpler and more common solutions explicitly assign a priority to each rule.

The previous approaches have provided various decision schemes at the mobility middleware and MH. Others have focused on the impact of resource allocation at the infrastructure level. Vadalachos et al. [16] used a policy system to manage a network of mobility middlewares. K. Jean [15] and K. Yang [17] have also used policy systems to move routers closer to the MH. While novel, these approaches do not provide support for mobility middleware or MH based handover decision. Furthermore, the resource allocation does not take into account the network context.

A few researchers have explored specific decision optimization scheme. A. Seth et al. [4] make the communication channel selection on the MH either online using four different schemes (e.g. max throughput estimate or closest AP), or offline with the full knowledge of surrounding infrastructure. H. Chen et al. [6] focus on theoretical monetary cost minimization of data delivery where the exact infrastructure properties must be known. Although the problem is similar to the NP-hard multi choice knapsack it becomes solvable by adding various constraints. They also study the effect of multiple routes as

well as prefetch when the communication channel bandwidth is insufficient. O. Ormond et al. [8] also mention that the monetary cost minimization is NP-hard and propose a piece-wise linear user utility-function that depends on time and monetary cost which they label Consumer Surplus. They compare their approach with Always Cheapest. Finally T. Pering et al. [23] concentrate on minimizing energy consumption of data delivery on MH by switching between Bluetooth and WiFi. In [11] they improve their work by implementing communication between the AP and the MH to determine the current load on the AP before switching. Although these are the types of high level goal we would like to specify at the policy level, none of these approaches describe architectures for context and decision support in handover of heterogeneous networks.

Chapter 3

Mobility Middleware

3.1 General Architecture

We choose to implement a mobility middleware based architecture where the mobility middleware takes care of doing the communication between the AMQP MOM and the MHs. As depicted in figure 3.1, it acts as a MOM level router in the system. As mentioned previously, related work has favored this architecture for client/server MOMs because it allows a clean separation of concerns between the mobility related connectivity and the MOM. One of our goals was to keep the AMQP protocol with as little changes as possible to allow good interoperability with other MOMs. This architecture facilitates the

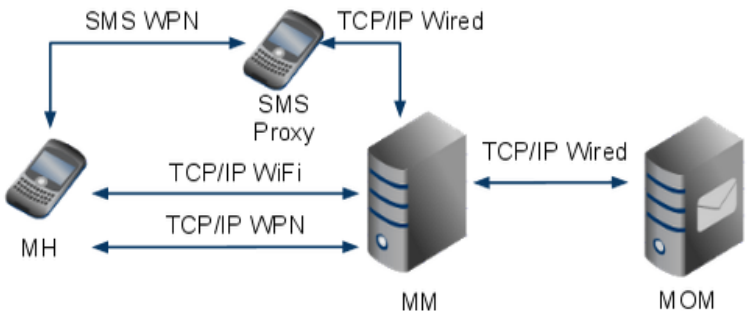

Figure 3.1: Mobility Middleware Architecture

reuse of the mobility middleware with any other AMQP implementations. In addition to handling mobility related connectivity, our mobility middleware takes care of selecting the appropriate communication channel depending on the current context. Figure 3.1 shows these different communication channels possibilities.

A MH can communicate via three mechanisms. Communication can be done with TCP/IP whereby the MH uses WiFi or the MNO air interface to the reach the mobility middleware. Communication can be done through SMS. Then, a cell phone is used as a SMS proxy to send and receive the messages. This SMS proxy is wired directly to the mobility middleware and does the conversion between TCP/IP and SMS.

3.2 Reliability and Ordering

3.2.1 TCP

As long as the mobility middleware does not reorder messages, using TCP between the mobility middleware and the MH is enough the guarantee the AMQP ordering.

However, adding the mobility middleware in the architecture is a liability in regards to message delivery in case of failure. A process has ownership of a message until it delivers it to another process. We first investigate a mobility middleware relay scheme. In order to provide this functionality, the mobility middleware is a consumer of queues on the MOM forwards these messages to MH consumers. The mobility middleware also receives messages from clients, producers and consumers, and sends them to the MOM. Figure 3.2 shows the mobility middleware relay scheme from the MOM to the consumer without and with acknowledgments between the MOM and the consumer. When a new message is available on a queue, the MOM first sends the

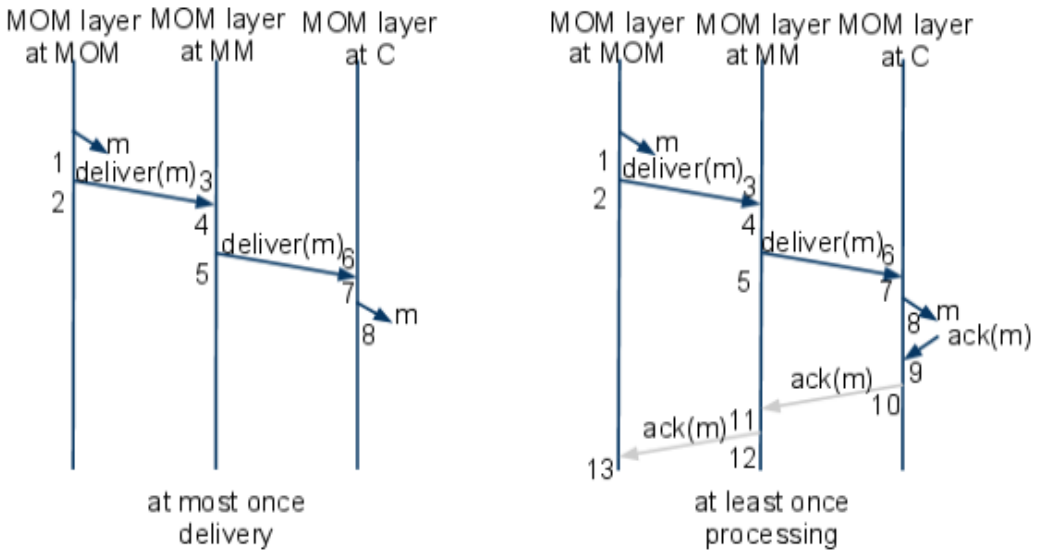

Figure 3.2: Mobility Middleware Relay

message to the mobility middleware. Without using the acknowledgements, the message responsibility is transferred to the mobility middleware as soon as the TCP send primitive is successful. The message can be lost because of a mobility middleware and consumer crash at time 4 and 7 respectively. If there are no such crash the message is delivered. The message delivery guarantee is at-most-once. When using acknowledgements the MOM only successfully removes the message after it has received an acknowledgement from the consumer at time 13. Crashes at the mobility middleware at time 4, 5 and 11 and at the consumer at time 7, 8 and 9 will prevent the exchange from being completed. In this case, the consumer sends a redelivery request to try to obtain the message again. The delivery guarantee of this exchange is at-least-once. From this, we gather that the addition of the mobility middleware in the architecture does not change the reliability semantic of AMQP, but introduces additional failure scenarios at the mobility middleware.

One solution to minimize the message loss at the mobility middleware would be to improve it so that it takes better ownership of the message after reception at the MOM layer by mimicking the MOM in terms of durability and acknowledgements. If the MOM has durable queues, the mobility middleware also has durable queues; if the consumer has acknowledgements, the mobility middleware also uses acknowledgements. When a new message is available on a queue, the MOM sends the message to the mobility middleware which stores it in a local queue. The mobility middleware then sends an acknowledgement to the MOM and sends the message to the consumer. When the mobility middleware receives an acknowledgement from the consumer then it removes it from its local queue. Figure 3.3 displays this concept with acknowledgements. The message responsibility is transferred from the MOM to the mobility middleware at time 6 and from the mobility middleware to the consumer at time 14. This technique has the advantage of releasing the MOM earlier from maintaining messages for consumers as they can be removed once they are stored at the mobility middleware.

Due to drawbacks of the second solution, we have chosen to implement the mobility middleware relay approach.

3.2.2 SMS

As we have seen in Section 2.6.1 the SMS guarantee is best effort and can cause duplicates, reordering and message loss. This is a problem when using AMQP over SMS because the specification requires a transport that has well defined reliability and ordering properties. Oliver [52] proposes a protocol similar to NETBTL [54] to ensure reliability and ordering of SMS messages in their SMS-NIC implementation. NETBTL was designed for high throughput of bulk data in high delay unreliable networks. In order to improve transmission rates, NETBTL exchanges between the sender and the receiver multiple aggregates of data called buffers. Transmission between both pro-

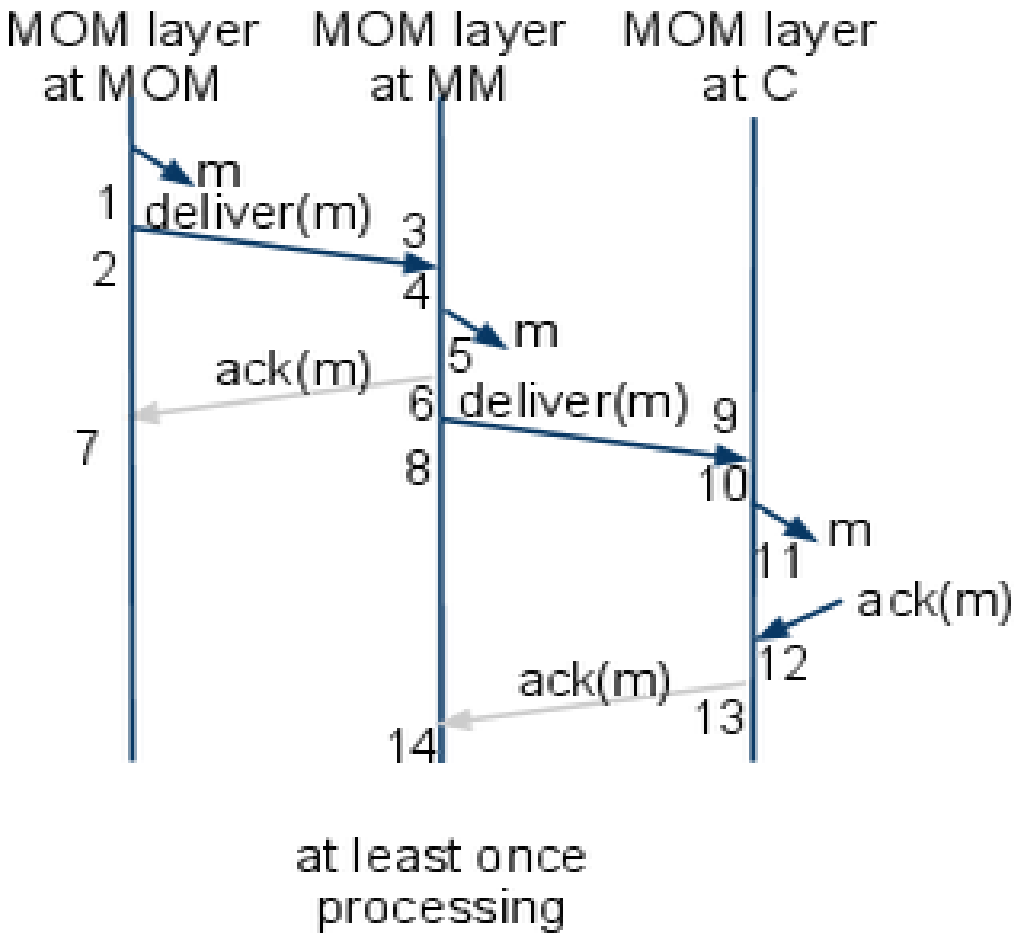

Figure 3.3: Mobility Middleware Responsibility Transfer

cesses attempt to synchronize both of these buffers. The sender breaks a buffer into segments and sends them to the receiver. Each segment contains a sequence number to allow for ordering. When the final segment arrives or after the specified timeout the receiver sends a selective acknowledgement for the entire buffer. The timeout is equal to the number of segments times the average delay for each segment. Once the buffer is completely received it can be passed to the upper network layers. The receiver must keep a history of all sent NETBTL commands (e.g. acknowledgements) in case of message loss. The sender requests acknowledgements when a timeout has been reached.

[52] chooses to modify this protocol for the exchange of a single buffer of 32KB segmented to the size of an SMS message. This removes the need for inter-buffer transfer coordination. The initial message by the sender contains the size of the data, a sequence number for each complete buffer exchange and the first part of the data. The receiver acknowledges the first message which allows the sender to bulk send the rest of the buffer. Instead of using an average, the timeout is reset to a static value of 200 seconds (four times the mean SMS delay) each time a new message is received. Each time the timeout is reached within a buffer exchange this static value is doubled. It returns to normal when the exchange terminates.

We adjust this approach. AMQP does two types of data transfer. The first type of messages is small in size and is used as AMQP commands. For example the creation of a queue, an exchange, a subscription to a topic or acknowledgements. Such messages fit in an SMS and should be exchanged as soon as possible. Furthermore, each of these messages requires a response from the MOM. This already acts as an acknowledgement and we do not require any additional acknowledgement. Since a client can only send one of these messages at a time and must wait for the response, ordering is not an issue. Duplicates are not an issue either because these messages are idempotent. For these reasons these messages can be sent using SMS without

any further reliability added as this is done implicitly already.

The second type is a content message which is routed from the producer to the consumer. Each message is typically a few KB. To add reliability we use the NETBTL mechanism and we map each message to a buffer.

3.3 Multihoming

As mentioned in Section 2.1 we consider multimode MHs, but AMQP does not directly support multihoming. Each AMQP connection is from the source address to the destination address. One of the main tasks of our work is to design an infrastructure that allows the exploitation of various communication channel between clients, producers and consumers, and the MOM.

We address this by implementing most of this functionality into a small mobility layer on top of the MOM layer at the client and at the mobility middleware. By separating this functionality from the AMQP protocol we can keep the interface of AMQP clients, producers and consumers, the same. The first operation an AMQP client must do is open a communication channel to the MOM. AMQP allows a host to open multiple TCP/IP connections to the same MOM. To this we add the ability to open SMS communication channels. When the client wants to use multihoming it opens a communication channel for each technology that should be used concurrently (e.g. WiFi and SMS). When a producer sends a message, the mobility layer selects the appropriate communication channel to transmit the message. On an incoming message from a MH, the mobility middleware relays the message to the MOM.

When using a scheduler based approach for multihoming such as described above, one must take care of reordering messages to obtain FIFO ordering.

For example, a first message m is sent via SMS and second message m' is sent WiFi. Latency on WiFi is much lower than on SMS and as such, message m' might arrive before message m. Our mobility layer adds a unique sequence number to each message so that it can reorder these messages in the correct order. This technique adds latency that might be inappropriate for latency sensitive applications, because the mobility layer must hold message m' until message m arrives. For this reason, one can direct the mobility layer not to reorder messages for latency sensitive applications at the cost of loosing ordering.

Contrarily to a producer, a consumer is bound to an AMQP connection. Whenever a MH sends a *consume(queueName)* message on an AMQP connection, the MOM knows this connection is interested in receiving messages from a queue. If a connection fails then the MOM will stop sending messages to that MH. However, multimode MHs can receive messages on many communication channels. If a communication channel fails, but another one is still opened, then we would like to continue receiving messages. For this reason, if a client wants to receive messages from a queue on two different communication channels (e.g. TCP/IP and SMS) then it must send the *consume(queueName)* message on both of these communication channels. The MOM will then see two consumers consuming one queue and deliver messages to these communication channel in a round robin fashion. If one of the communication channel fails then the MOM will continue sending all messages on the other communication channel. This approach however, causes two problems. As we have established in Section 2.7.4 consumers sharing a queue with acknowledgments have weak-FIFO ordering. Furthermore, each AMQP connection or communication channel receives equal shares of messages from that queue. If many hosts are consuming a queue the host having the most connections would receive a greater proportion of the messages.

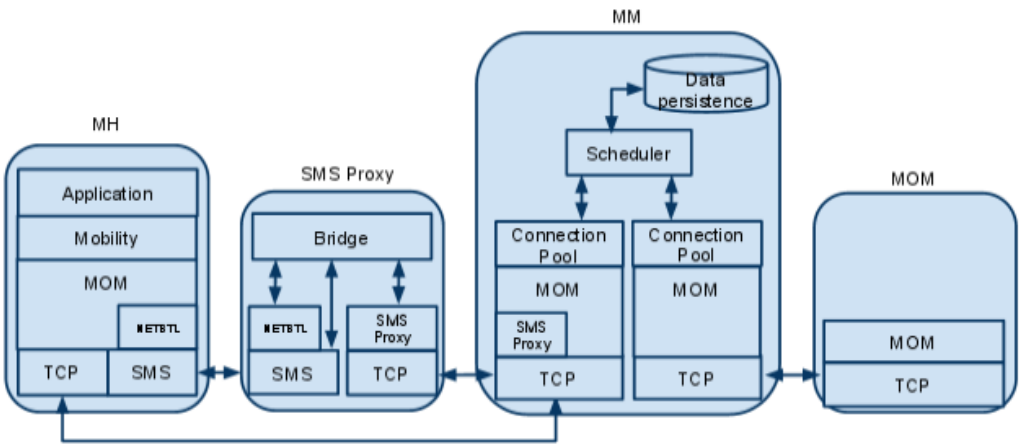

Figure 3.4: Mobility Middleware Entities Architecture

We solve these problems at the mobility middleware. When the mobility middleware receives a *consume(queueName)* message from a MH it first checks if this MH is already consuming this queue. If it does then the mobility middleware registers the MH's interest to this queue on this communication channel and drops the message. This way the MOM does not receive this second request and sees only communication channel per consumer to which all messages for this client are sent. The mobility middleware is then in charge to relay the message of one of the communication channels to the client. When the communication channel of a consumer fails, the mobility middleware checks if another communication channel from the same MH has requested consuming the same queue. If the MH has done so then the mobility middleware takes care sending a *consume(queueName)* message to the MOM for that communication channel to receive further messages.

3.4 Entities Architecture

The architecture is depicted in Figure 3.4. The MH has very little additional logic compared to the original AMQP protocol. The mobility layer takes

care of message reordering, transmitting context information and choosing a communication channel to transmit a message. The MH has several communication channels to the mobility middleware (WiFi, MNO and SMS). If a message exchange is via SMS it goes through the SMS proxy. As explained in Section 3.2.2 this communication can be done by simple SMS messages or with the NETBTL protocol. The SMS proxy is connected via the mobility proxy via TCP/IP and simply takes care of relaying messages with the mobility middleware. A simple protocol called 'SMS Proxy' takes care of handling this communication.

From client to mobility middleware When the mobility middleware receives a message from the client (either directly through TCP/IP or through the SMS proxy) it relays it to the scheduler. There are two types of messages that the scheduler can receive. First, a regular content message that is targeted at a topic or a queue. In this case the scheduler relays the message to the MOM. Second, a context message that is targeted at the scheduler. These messages are created by the mobility layer of the MH and they contain information about the current context (network, application, user and infrastructure). This information helps the scheduler to decide on the appropriate communication channel for further content message. We will discuss context messages and the relevant tasks at the scheduler below. In this case, the scheduler reads these messages, consumes and processes them and persists the information.

From MOM to mobility middleware Upon reception of a message from the MOM the mobility middleware relays the message to the scheduler. The scheduler takes care of choosing an appropriate communication channel to transmit the message according to the current context. Finally, the scheduler takes care of gathering network related statistics such as the amount of bandwidth used, the number of SMS messages sent and the monetary cost incurred by each MH. This information is also persisted.

3.5 Handover

Network Statistics Past work has either evaluated network performance data statically where data is first collected and then reused or dynamically by continuously collecting data. Static models in general are not appropriate for MHs and wireless networks because the environment changes rapidly and widely. Wireless networks are prone to network contention because the bandwidth is limited and shared between many users. Dynamic network modeling can be done intrusively where data is injected into the network or non-intrusively by simply observing regular traffic and calculating the performance.

In our work we focus on detecting network contentions on a communication channel and remediating appropriately by adapting our network selection algorithm. Accurate value estimations for latency and bandwidth are not required. We propose a simple non-intrusive method that monitors current AMQP traffic on TCP. A header is added to each message sent by the mobility middleware containing the current time and sending throughput. The consumer replies with a context message containing the time the message spent at the consumer, the mobility middleware timestamp, the receiving and the sending throughput. Upon the reception of this message, the mobility middleware can calculate the latency by taking the time difference between the current time and the original timestamp minus the time spent at the consumer. Finally, by looking at the throughput difference the mobility middleware knows if there is contention. The context message containing the statistics can be optimized in a number of ways. The information can be piggybacked on any message returning to the mobility middleware (control messages, content message, etc). If such message is not sent, the consumer will wait a certain amount of time before sending back the statistics. The mobility middleware then calculates the moving average for both of these values.

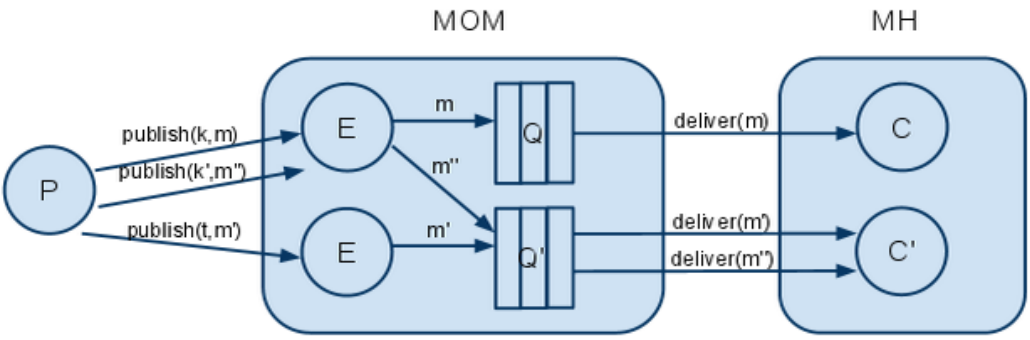

Figure 3.5: Multiple Queues

Multiple Queues As we have described previously a consumer consumes one queue. In this section we will consider that one or more queues are created for each MH. This means that a MH has one or more consumer processes at any time. When a producer wants to send a message to a MH it must first choose a communication pattern. Each of the different communication patterns directed to an MH will go through either the same queue or different queues. Figure 3.5 depicts this situation. A point-to-point message m is sent by producer P with routing key k where k is equal to Q. A publish/subscribe message m' is sent by producer P with topic t where queue Q' is bound to topic t. Finally, a store and forward message m" is sent by producer P with routing key k' where k' is equal to Q'. Topic t has more than one queue bound to it. Queues Q and Q' contain messages that will be received by the same MH. This allow messages to be separated into sets with different properties. A user can then apply different transmission preferences such as limits, selection and adaptation for each of these set as described below. For example the queue Q is assigned messages that need to be transmitted as soon as possible. Message m is actually a reply to request executed by the MH and the user is ready to pay a higher cost for this message, because he wants the answer right away. Queue Q' is assigned messages that can be delayed. Message m' and m" are notifications from different sources that the user must eventually receive. However, they do not

have to be received right away so the MOM can wait to deliver these messages at a time where the network conditions are favorable to the transmission.

The separation of messages can simply be done at the communication pattern level where point-to-point and request/reply messages are stored in queue Q. These patterns are typically used when a messages needs to be transmitted right away. Furthermore, publish/subscribe and store and forward messages are stored in the Q'. These patterns are typically used to notify a process and do not need to be received right away. However, more complex associations are also possible. Publish/subscribe messages can be assigned to queue Q or Q' depending on the needs of the application transmitting the message. Queue Q receives urgent notifications sent by a publish/subscribe mechanism while Q' would receive non urgent notifications. Finally, a single MH can consume more than two queues. Each additional queue has different properties.

3.5.1 Limits

In our work, the MH can specify certain requirements such as data transfer, message and costs limits on a daily, weekly or monthly basis (e.g. maximum 1000 SMS messages per months, 5 megabits per day on 3G). While most data plans are on monthly basis a MH might decide to enable data usage for a limited amount of time. Furthermore, finer grained limits might be useful for situations such as limiting the amount of transmissions on a device each day for energy management or other purposes. The MH specifies several sets limits where each set is assigned to a different queue. In our previous example, a user could set a monthly limit for bandwidth used by non urgent notifications. This would make sure that urgent notifications get the appropriate share of the bandwidth. The MH also specifies global limits that have to be respected by a set of queues. Finally, the MH specifies limits that are assigned to a communication channel. For example, a limit setting the

amount of bandwidth used in a month on a given communication channel to respect monthly allotment from the MNO.

3.5.2 Selection

When there is more than one communication channel available, one must choose which to use to send data. We use the same cost-function approach as in [12], but we modify the parameters of this function. We combine monetary cost (C_n), energy consumption (E_n), latency (L_n) and coverage (V_n) to evaluate each communication channel. We show the cost calculation details that were introduced in [12].

$$f_n = w_c * ln(C_n) + w_e * ln(E_n) + w_l * ln(L_n) + w_v * ln(1/V_n)$$

The higher the cost (f) for a network the worse it is. Depending on the wireless technology the monetary cost can be counted by a certain amount of SMS messages or a certain amount of bandwidth. The MH specifies the monetary cost per megabyte or per message for each communication channel. The energy consumption represents the amount of energy required to transmit on the technology. We use static values such as displayed in Figure 2.1. Four weights (w_c,w_e,w_l and w_v), whose sum is equal to one, allow for customization of the cost-function. The MH specifies the values for each of these weights. An always cheapest approach would be denoted by (1,0,0,0) while an equally balanced approach would be denoted by (1/4,1/4,1/4,1/4). The logarithm function is used to compare proportional differences (e.g. twice as expensive versus twice as energy efficient). As in [12], we compare two cost-functions by subtracting one by the other.

$$f_1 - f_2 = w_c * ln(C_1/C_2) + w_e * ln(E_1/E_2) + w_l * ln(L_1/L_2) + w_v * ln(V_2/V_1)$$

A negative value means that f_1 is better than f_2 while a positive value means the contrary.

We show an example between SMS and MNO for the transmission of a 512 bytes message. The monetary cost for an unlimited SMS bundle is 15$ per month. We consider that a user would not send more than 10000 SMS messages and put the cost at 0.0015$ per message. To transfer 512 bytes we need to send four SMS messages of 140 bytes. The cost of this transaction is 0.006$. On a MNO, the cost is 30$ for 6GB per month. The cost of this transaction is $30/6/1024/1024/2 = 0.000002384$$. We assume equally balanced weights.

$$f_1 = 0.25 * N(0.0015) + 0.25 * N(600) + 0.25 * N(10000) + 0.25 * N(1000)$$

$$f_2 = 0.25 * N(0.000002384) + 0.25 * N(1500) + 0.25 * N(250) + 0.25 * N(1000)$$

$$\begin{aligned} f_1 - f_2 = 0.25 * ln(0.0015/0.000002384) + 0.25 * ln(600/1500) \\ +0.25 * ln(10000/250) + 0.25 * ln(1000/1000) \end{aligned}$$

$$f_1 - f_2 = 2.13$$

In our example the MNO (f_2) is better than SMS (f_1).

The MH can specify several sets of parameter weights. Each set is assigned to a different queue. This means that when a communication channel must be chosen, the scheduler first looks at which queue the message comes from. It then takes the appropriate set of weights to rank the communication channels.

3.5.3 Adaptation

Each application can use one or more communication patterns as identified in Section 2.7.4 which can be categorized into two types. First, *non delay tolerant* type which is a pattern that is intended to be immediately transferred from the sender to the MOM and then to the receiver with relatively low latency. The data that is exchanged is time sensitive and delay in its delivery lowers its usefulness. Point-to-point and Request/Reply are patterns of this type. Second, *delay tolerant* type which is a pattern intended for eventual delivery from the sender to the receiver. The data that is exchanged is not time sensitive and should eventually be transferred. Email, SMS, instant messaging, non time sensitive notifications and task distribution are examples of delay tolerant applications. Store and forward and publish/subscribe are patterns of this type. In this type the MOM can delay information purposefully in order to optimize the cost according to the current context. For example a MH is in transit and it currently has access to a MNO. The mobility middleware has a delay tolerant message to send to the MH and it knows that in 10 minutes the MH will arrive at its destination where WiFi is available. Since, the cost of this new network is cheaper, the mobility middleware purposefully waits 10 minutes before sending the message to the MH. Previous work has shown the potential of taking an always cheapest [6] or maximum utility [4] approach using full knowledge of the infrastructure and of the mobility pattern of the user.

In a real world implementation full knowledge is not available. The wireless signal depends on many factors which influence its strength and availability. Furthermore, the exact position of the MH is difficult to find. However, people usually have predictable habits (e.g. work, commuting, home, etc.) and the network conditions for each of these locations are highly correlated. It is possible to determine the probability of having access to a given network at a given time given the position of the MH. Previous work such as [61]

has shown how such information can be gathered by using the MH's GPS, cell tower IDs or link layer strength information. However, this approach requires high energy consumption, exchanging information with a server or algorithm training. In this work, we simply ask the user to manually input these probabilities in calendar format (e.g. Monday 9:00 to 18:00 90% chance to have access to WiFi). This information is then sent the scheduler. These probabilities are then used to determine with how much delay a message can be delivered.

$$D_n = P_n * D_{max}$$

If only one network is available and the probability of having access to any other network is 0, then messages are sent immediately. If there is a probability of obtaining a better network, then the maximum delay D_{max} allowed by the MH is proportional by its probability P_n. In the case where there is more than one better network then the one with the highest probability is kept. For example, the MH currently has access to MNO, but has a 90% probability to have access to WiFi in the next 10 minutes. Furthermore, the data to be transmitted can be delayed for a maximum of 10 minutes according to the preferences of the user. Then the delay is simply $0.9 * 10 = 9 minutes$. If after the waiting period the MH still does not have access to WiFi than it transmits the message using the current available network.

The ordering for an AMQP queue is FIFO. By allowing certain messages in a queue to be delayed we would lose ordering between non delayed and delayed messages. For this reason a queue is only assigned either delay tolerant or delay intolerant messages. For example, a point-to-point message is sent to the delay intolerant message queue while a publish/subscribe is sent to the delay tolerant queue. While we have categorized communication patterns into delay tolerant and delay intolerant they are not bound to these types. A MH can use publish/subscribe in a delay intolerant manner as

well. The MOM simply needs to be configured to route to a delay intolerant queue. The MH specifies one or more maximum delay preferences (e.g. 5 minute and 60 minutes). Each maximum delay preference is assigned to a different queue.

3.5.4 Initiation

The MH initially connects with SMS to the mobility middleware. SMS is a connectionless protocol and the MOM layer 'connection' can last for the remainder of the device's lifetime. The MH is in charge of opening and closing communication channels to the mobility middleware at any time using TCP or another technology. The mobility middleware accepts all communication channels. It also closes communication channels when an exception happens or if a limit is reached. The client is notified accordingly so it does not reconnect. Whenever the MH needs to transmit a message it will choose one of the available communication channels. If a new communication channel becomes available the MH will open a new connection to the mobility middleware on that technology. Whenever the mobility middleware needs to transmit a large amount of data to a MH and the mobility middleware does not have a high bandwidth communication channel available, it will send a message to notify the MH so it opens a new communication channel if possible.

3.5.5 Transmission Algorithm

With several pieces of the algorithm described, limits, communication channel selection and adaptation, we propose an algorithm for the transmission of messages. This algorithm is used both at the scheduler and at the mobility layer of the MH whenever a message needs to be sent. Figure 3.6 depicts this algorithm. On the mobility middleware, when a new message needs to be sent, the scheduler first verifies if the limits for the queue have been reached. If they have we simply do not transmit the message. If they have not we

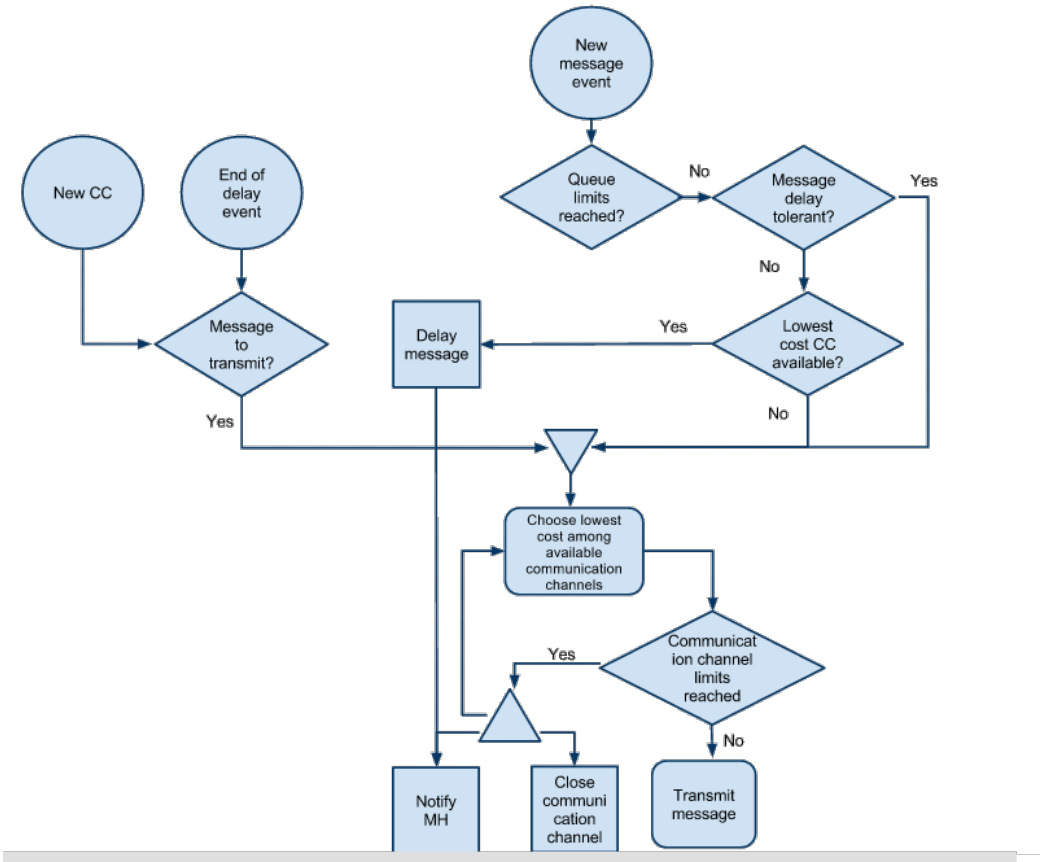

Figure 3.6: Transmission Algorithm

continue the algorithm with two major cases.

In the first case, the queue is delay intolerant. The scheduler looks at the available communication channels to the MH and finds the lowest cost one according to the cost-function and the weights provided for the queue. Once the scheduler has found the lowest cost communication channel, it verifies if the limits for the communication channel have been reached. If the limits have been reached, the scheduler notifies the MH, closes the communication channel and looks at the remaining available communication channels. If the limits have not been reached, the scheduler transmits the message.

In the second case, the queue is delay tolerant. The scheduler looks if the lowest cost possible communication channel for the MH is available. To find the lowest cost possible communication channel, the MH provides the best values of monetary cost, energy consumption, latency and coverage for each different technology (SMS_{best}, MNO_{best}, $WiFi_{best}$). For example, a MH considers that a $WiFi_{best}$ is free, it uses 1000 milliwatts of energy consumption, has 250 microseconds of latency and has a 100 meters of coverage. Additionally, the scheduler has values for each of the current communication channels ($SMS_{current}$, $MNO_{current}$, $WiFi_{current}$). The scheduler then removes the 'best' communication channels that have no chance of appearing according to the probability set in Section 3.5.3. The scheduler compares the remaining communication channels with each other using the cost-function approach described in Section 3.5.2 (e.g. $WiFi_{best}$ vs $MNO_{current}$, $WiFi_{current}$ vs $MNO_{current}$) and finds the lowest cost communication channel. If one of the 'current' communication channel has the lowest cost than we know that we have the lowest cost possible communication channel.

If the scheduler has access to the lowest cost communication channel, then the message is transmitted immediately as we described for the delay intolerant case. If the scheduler does not have access to the lowest cost commu-

nication channel then the message is delayed. The scheduler will transmit this delayed message when one of two events happens.

- A better communication channel becomes available.
- The message has been delayed for the maximum amount of time specified by the adaptation algorithm.

Chapter 4

Performance Evaluation

We evaluate the impact of our mobility middleware and the communication channel selection on the Round Trip Time (RTT) of a message from the producer to a consumer on the same MH. The producer embeds a timestamp in each message and publishes it to the MOM on a non durable queue. The MOM then sends it to the consumer and no acknowledgements are exchanged. The consumer takes the difference between the current time and the time in the message to obtain the RTT. The producer does not wait for the message to return to the consumer before sending the next message. We do this operation for different message loads and throughput. The load can be read as 60 messages in 60 seconds (at the rate of 1 message per

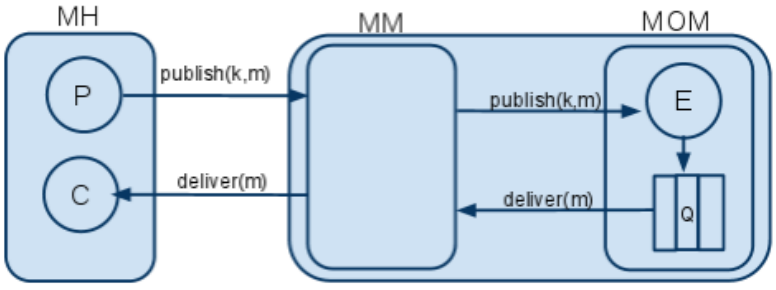

Figure 4.1: Performance Evaluation Entities one communication channel

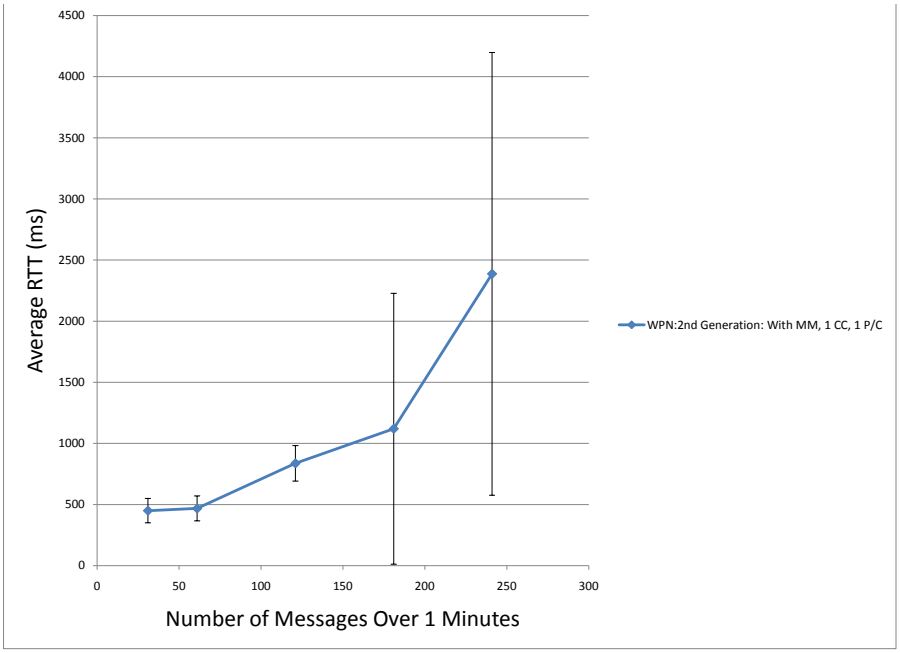

Figure 4.2: Average RTT vs message load using the MNO 2nd generation for one producer, one queue and one consumer during one minute

second). Figure 4.1 shows the entities present for this test. The MH is an older model Blackberry 8700c. The mobility middleware and the MOM are collocated on the same machine, Intel dual core 2.2 GHz with 2 GB of RAM on Windows XP. Figure 4.2 shows the average RTT for varying message loads using a second generation MNO transmission technology on the Rogers network. The RTT starts at 450ms with a standard deviation of 100ms and grows quickly. Figure 4.3 shows the average RTT for varying message loads using SMS. The RTT starts at around 15000ms with a standard deviation 400ms. Obtaining precise performance results for our mobility middleware on a mobile phone is not feasible. As shown in Figure 4.2 and Figure 4.3,

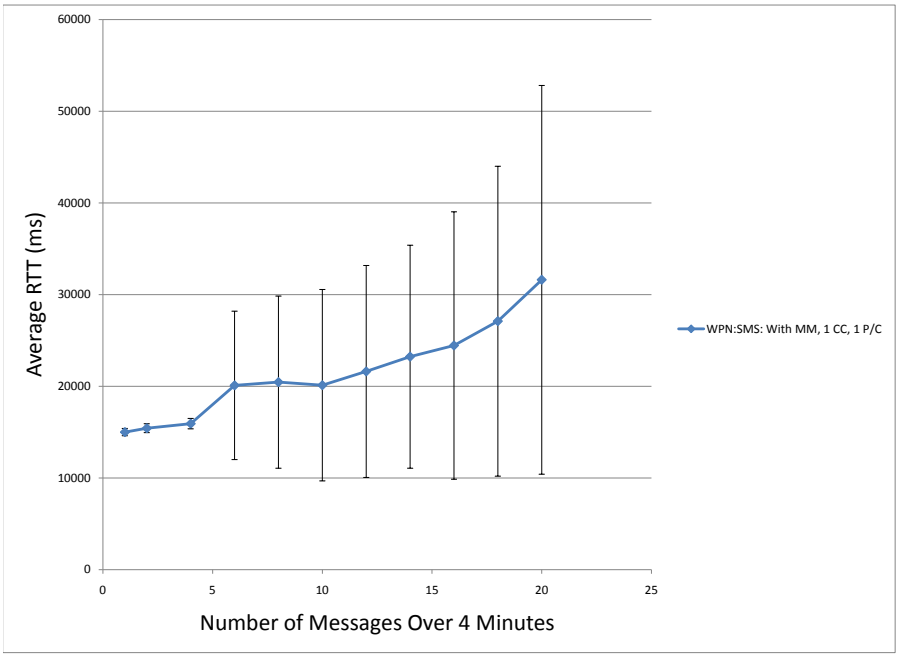

Figure 4.3: Average RTT vs message load using the SMS for one producer, one queue and one consumer during four minutes

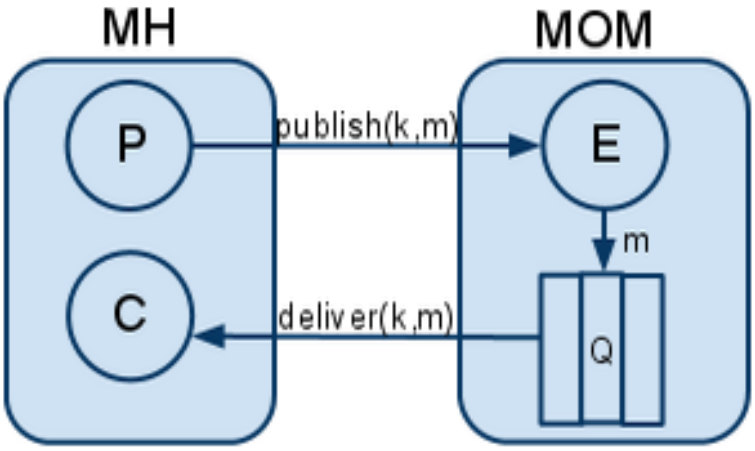

Figure 4.4: Performance evaluation entities without mobility middleware and one communication channel

the RTT and standard deviations are very high. This is due to the nature of the wireless communication channel, sharing the AP with other users and the fact that no precise time facilities exists on the MH. The results were obtained after performing the average between three different runs.

We use a desktop computer to obtain more precise results. The producer and consumer are located on a machine with the same specification as the MOM machine using the same sending scheme as above. We test four different configurations. The first configuration is depicted in Figure 4.4. The MH and the MOM communicate directly to obtain the base RTT between the two hosts. The second configuration is depicted in Figure 4.5. The MH has three communication channels which are each linked to a different consumer. The MOM sends the messages in a round robin fashion to each consumer. The third configuration is depicted in Figure 4.1. The MH communicates with the MOM through the mobility middleware. The mobility middleware simply relays messages from the MH to the MOM and vice versa. Finally, Figure 4.6 depicts the last configuration. The MH has three communication channels, each with different network context. The mobility middleware

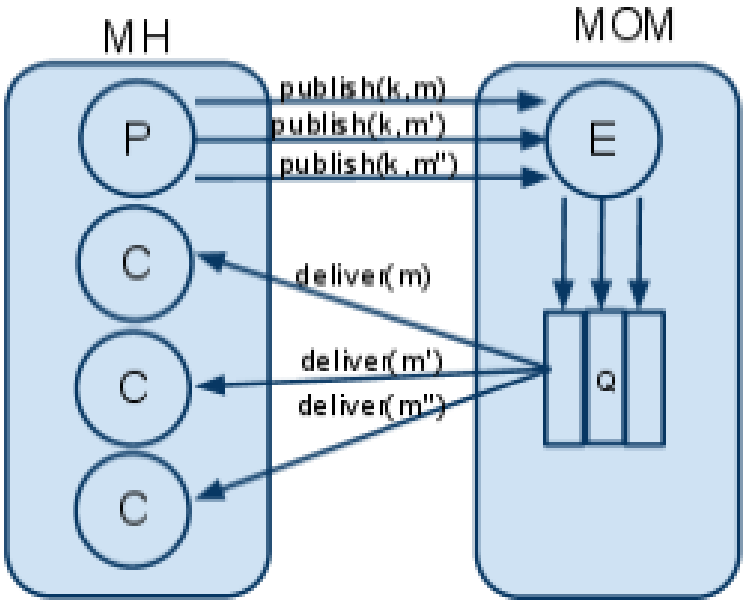

Figure 4.5: Performance evaluation entities without mobility middleware and three communication channels

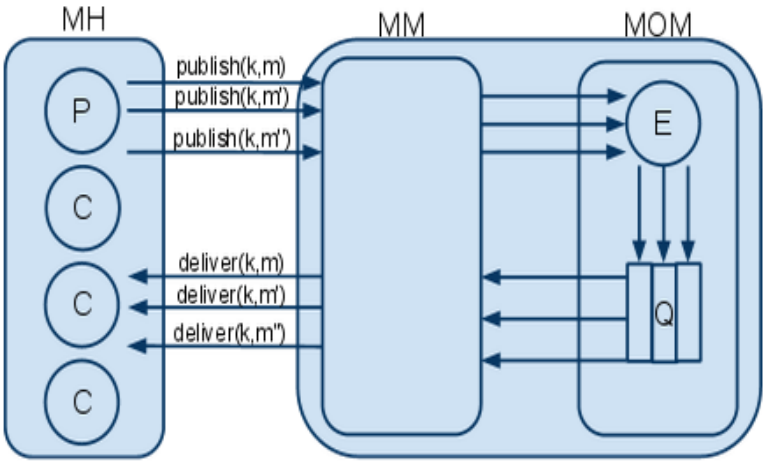

Figure 4.6: Performance evaluation entities with mobility middleware and three communication channels

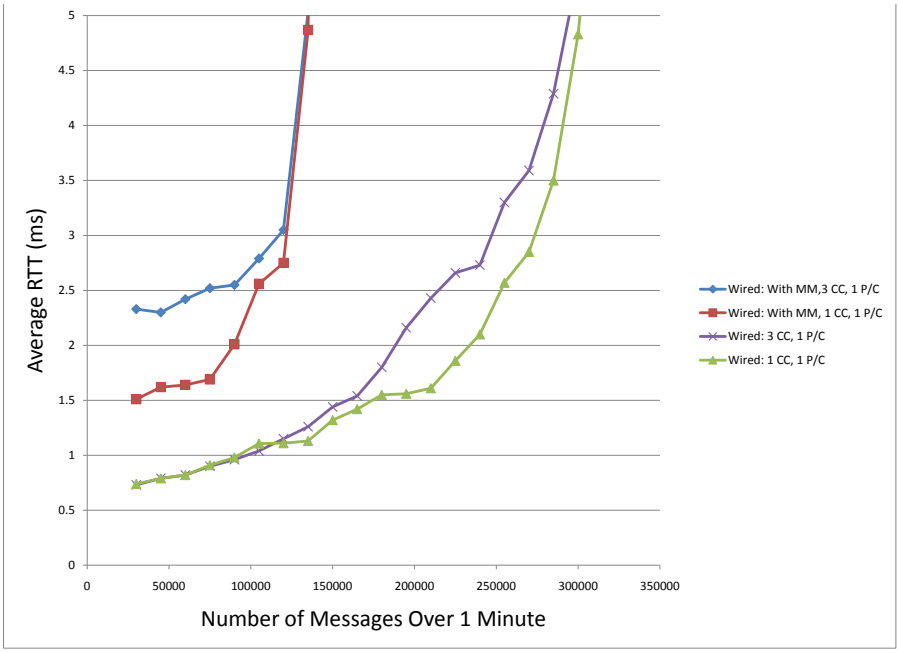

Figure 4.7: Average RTT vs message load using a wired connection for one producer, one queue and a varying amount of consumers with and without our mobility middleware during one minute

finds the lowest cost communication channel and sends the message. Since we are on a desktop, only one type of communication channel is available. We differentiate each communication channel by labeling each connection differently (e.g. 'WiFi', 'SMS', 'MNO'). When the mobility middleware receives a connection it associates one of the available labels to a predefined network context. We used the same fixed values defined in Figure 2.2 with an equal weight for each member of the cost function (monetary cost, energy consumption, latency and coverage). The label 'WiFi' obtains the best connection. Figure 4.7 shows the average RTT for varying message loads using

a wired connection between the two hosts. The base RTT is 0.73 ms for both the first and second configuration. At 135000 messages, the latency for the second configuration is slightly higher than the first configuration. We associate this difference with thread switching latency at the MH. Each consumer and producer is handled by a different thread. Since each message is sent to a different consumer, the MH must keep switching between different threads which increases the latency. In the third configuration, the addition of the mobility middleware doubles the base latency (1.51 ms). The overhead of updating data structures and traversing the mobility middleware is around 0.78 ms. This overhead is quite large for several reasons. The mobility middleware must first unserialize each message in order to analyze its headers. These messages must then be reserialized before being sent to the MOM. Additionally, the mobility middleware has access to two pools of connections. One pool contains connections to each MH while the other pool contains the same amount of connections directed to the MOM. Each connection is mapped one to one from the MOM to the MH. The mobility middleware must first determine the source connection and then the source user of each message before looking up in its internal structure where the message should be routed. It must then update the appropriate data structures (e.g. limits, bandwidth, counters, etc.). Finally, the last configuration has a RTT around 2.33 ms. The cost of choosing the lowest cost communication channel is 0.82 ms. This overhead is also quite large. The selection algorithm must compare each connection against each other to find the best available one and then it must determine if it is the best possible connection again doing a similar comparison. Additionally, the algorithm must look up limits, user preferences and delay to compute this result. In every case the latency will slowly go up until the machine becomes CPU bound. We think it would be possible to improve these results by optimizing the data structures we use for each of these operations.

Chapter 5

Conclusions and Future Work

5.1 Conclusion

In this work, we have used a MOM in a mobile context by adding a mobility middleware. This allowed the MOM to communicate with a MH using either IP networks such as WiFi and MNO, or a non-IP network such as SMS. Furthermore, we have used a utility function that takes into account monetary cost, latency, coverage and energy cost to choose between multiple communication channels. We have also added several thresholds that should be respected by each communication channel. Additionally, we have introduced a delay scheme that can be used in delay-tolerant applications. This scheme allows the mobility middleware to improve the utility of sending a message by waiting for better network conditions. These algorithms are used to transmit data on many of these communication channels simultaneously while keeping FIFO ordering if required. We offer the same guarantees offered by AMQP such as at-most-once delivery, at-least-once delivery or at-least-once processing depending on the application requirements. The solution we implemented introduces considerable latency, but we are confident that this could be further improved.

5.2 Future Work

Our solution focuses on the traditional client/server architecture. We choose this approach given the readily available implementations. Other approaches such as peer-to-peer structure, could provide an interesting field of study for this work. peer-to-peer have been studied by previous work in many different environments, but not in this particular setting. In particular, developing a peer to peer message exchange solution based on IP (WiFi, MNO) and non-IP network (SMS, Bluetooth) using a protocol such as AMQP would provide very interesting properties. A MH could continue to exchange messages with devices in its surroundings while being disconnected from the Internet. These devices could in turn relay these messages to other devices or on the Internet. This approach could provide limited connectivity to devices disconnected from the Internet.

In our work we have focused on calculating the utility of the transmission of a message by a single MH. However, when multiple MHs use shared APs the network conditions can degrade rapidly. Another approach would use a global knowledge of the system and calculate a global utility for all the devices using the same AP. This approach would increase the utility for a group of MH and improve the transmission performance.

In this thesis, we have developed a system that allows MHs to communicate through various communication channels with the internet. However we have not studied the security implications of our design. While we use known protocols (TCP, AMQP, SMS) that have well studied security issues and solutions (transport layer security, encryption, certificates and authentication), our current design does not take those into account. Further research into this area would be required before implementing our system into a production environment.

List of Figures

List of Tables

Bibliography

[1] L. Chen, T. Sun, and M. Gerla, *USHA : A practical vertical handoff solution*. In Proc. of Multimedia Services Access Networks (MSAN), 2005

[2] A. Seth, M. Zaharia, S. Keshav, and S. Bhattacharyya, *A policy-oriented architecture for opportunistic communication on multiple wireless networks*. [Online], Available: http://blizzard.cs.uwaterloo.ca/keshav/home/Papers/data/06/ocmp.pdf, 2006.

[3] D. Maltz, and P. Bhagwat, *MSOCKS: An architecture for transport layer mobility*. In Proc. of IEEE Infocom, p.p. 1037-1045, 1998.

[4] A. Seth, N. Ahmed, and S. Keshav, *Mobility decisions in heterogeneous wireless access networks*. Manuscript, University of Waterloo, 2004.

[5] Y. Chen, K. Schwan, and D. Zhou, *Opportunistic channels: mobility-aware event eelivery*. In Proc. of the 4th ACM/USENIX International Middleware Conference, 2003.

[6] H. Chen, H. Wu, S. Kumar, and N.-F. Tzeng, *Minimum-cost data delivery in heterogeneous wireless networks*. IEEE Trans. Veh. Technol. 56(6), 3511–3523, 2007.

[7] A. Qureshi, and J. Guttag, *Horde: separating network striping policy from mechanism*. In Proc. MobiSys, 2005.

[8] O. Ormond, and G.-M. Muntean, *J. Murphy economic model for cost effective network selection strategy in service oriented heterogeneous wireless network environment*. In Proc. of the Network Operations and Management Symposium 2006.

[9] P. Vidales, J. Baliosion, J. Serrat, G. Mapp, F. Stejano, and A. Hopper, *Autonomic sytem for mobility support in 4G networks*. In IEEE Journal on Selected Areas in Communications, vol. 23, pp. 2288–2304, 2005.

[10] D. Tsamis, T. Alpcan, J. P. Singh, and N. Bambos, *Dynamic resource modeling for heterogeneous wireless networks*. In Proc. of IEEE International Conference on Communications (ICC), 2009.

[11] Y. Agarwal, T. Pering, R. Want, and R. Gupta, *SwitchR: Reducing system power consumption in a multi-clients, multi-radio Environment*. In Proc. of IEEE International Symposium on Wearable Computing (ISWC), 2008.

[12] H. J. Wang, R. H. Katz, and J. Giese, *Policy-enabled handoffs across heterogeneous wireless networks*. In Proc. IEEE Workshop. Mobile Comp. Sys. and Apps., pp. 51–60, 1999.

[13] J. Makela, *Handoff decision in multi-service networks*. In Proc. of the Eleventh IEEE International Symposium on Personal, Indoor, and Mobile Radio Communication (PIMRC 2000), 2000.

[14] P. Chan, *Mobility management incorporating fuzzy logic to heterogeneous IP environment*. IEEE Communications Magazine, 2001.

[15] K. Jean, K. Yang, and A. Galis, *A policy based context-aware service for next generation networks*. IEE the Eighth London Communication Symposium, 2003.

[16] N. Vardalachos, J. Rubio, A. Galis, and J. Serrat, *A policy management system for hybrid networks*. In Proc. of The London Communications Symposium, 2002.

[17] K. Yang, A. Galis, and C. Todd, *Policy-driven mobile agents for context-aware service in next generation networks*. In Proc. of IFIP Fifth International Conference on Mobile Agents for Telecommunications (MATA 2003), 2003.

[18] S. Aust, N. A. Fikouras, D. Protel, C. Gorg, and C. Pampu, *Policy based mobile IP handoff decision (POLIMAND), Internet Draft, Work in progress*. [Online], Available: http://www.ietf.org/internetdrafts/draft-iponair-dna-polimand-00.txt, 2003.

[19] K. Murray, R. Mathur, and D. Pesch, *Intelligent access and mobility management in heterogeneous wireless networks using policy*. In Proc. of the First ACM International Workshop on Information and Communication technologies, pp. 181–186, 2003.

[20] P. Vidales, R. Chackravorty, and C. Policroniades, *PROTON: A policy based solution for future 4G devices*. In Proc. of The Fifth IEEE International Workshop on Policies for Distributed Systems (POLICY 2004), June 2004.

[21] C. Perkins, *IP mobility support in IPv4 (RFC 3344)*. August 2002, [Online], Available: http://www.ieft.org

[22] D. B. Johnson, C. E. Perkins, and J. Arkko, *Mobility support in IPv6 (RFC 3775)*. [Online], Available: http://www.ieft.org., 2004.

[23] T. Pering, Y. Agarwal, R. Gupta, and R. Want, *CoolSpots: Reducing power consumption of wireless mobile devices using multiple radio interfaces*. In Proc. ACM/Usenix Mobisys, 2006.

[24] G. Fitzpatrick, S. Kaplan, T. Mansfield, D. Arnold, and B. Segal, *Supporting public availability and accessibility with Elvin: experiences and reflections*. In Computer Supported Collaborative Work: the Journal of Collaborative Computing, pp. 15–51, 2000.

[25] B. D. Noble, M. Satyanarayanan, D. Narayanan, E. J. Tilton, J. Flinn, and K. R. Walker, *Agile application-aware adaptation for mobility*. In Proc. of the 16th ACM Symposium on Operating System Principles, 1997.

[26] *Advanced Message Queuing Protocol*. [Online], Available: http://www.amqp.org.

[27] Patrick Th. Eugster, Pascal Felber, Rachid Guerraoui, and Anne-Marie Kermarrec, *The many faces of publish/subscribe*. ACM Comput. Surv. 35(2): 114–131, 2003.

[28] G. Cugola, and H.-A. Jacobsen, *Using publish/subscribe middleware for mobile systems*. Mobile Computing and Communications Review (SIGMOBILE) 6(4):25–33, 2002.

[29] M. Cilia, L. Fiege, C. Haul, A. Zeidler, and A. P. Buchmann, *Looking into the past: enhancing mobile publish/subscribe middleware*. DEBS, 2003.

[30] L. Fiege, F. C. Gartner, O. Kasten, and A. Zeidler, *Supporting mobility in content-based publish/subscribe middleware*. Middleware: 103–122, 2003.

[31] Y. Huang, and H. Garcia-Molina, *Publish/subscribe in a mobile environment*. Wireless Networks 10(6): 643–652, 2004.

[32] Q. Yuan, and J. Wu, *DRIP: A dynamic VoRonoi RegIons-Based Publish/Subscribe protocol in mobile networks*. INFOCOM:2110–2118, 2008.

[33] *Java Message Service (JMS) Specification*. [Online], Available: http://java.sun.com/products/jms/.

[34] *The Narada event brokering system*. [Online], Available: http://grids.ucs.indiana.edu/ptliupages/projects/narada/.

[35] *Softwired iBus messaging*. [Online], Available: http://www.softwired-inc.com/.

[36] *TIBCO. TIB/Rendezvous white paper*. [Online], Available: http://www.rv.tibco.com.

[37] A. Carzaniga, D. Rosenblum, and A. Wolf, *Achieving scalability and expressiveness in an Internet-scale event notification service*. In Proc. of the Nineteenth ACM Symposium on Principles of Distributed Computing, ACM Press, 2000.

[38] *IBM MQ Series*. [Online], Available: http://www.ibm.com/software/ts/mqseries/.

[39] *WebLogic*. [Online], Available: http://www.bea.com/products/index.shtml.

[40] P. Pietzuch, and J. Bacon, *Hermes: a distributed event-based middleware architecture*. In Proc. of the 1st International Workshop on Distributed Event-Based Systems with IEEE ICDCS, 2002.

[41] *WebSphere MQ Everyplace*. [Online], Available: http://www-01.ibm.com/software/integration/wmqe/

[42] *Extensible messaging and presence protocol*. [Online], Available: http://xmpp.org/

[43] A. M. Julienne, and B. Holtz, *ToolTalk and open protocols, inter-application communication*. Prentice Hall, Englewood Clis, New Jersey, 1994.

[44] B. Kantor, and P. Lapsley, *Network news transfer protocol — a proposed standard for the stream-based transmission of news*. Internet Requests For Comments (RFC) 977, 1986.

[45] B. Segall, and D. Arnold, *Elvin has left the building: A publish/subscribe notification service with quenching*. In Proc. of AUUG97, Brisbane, Queensland, Australia, 1997.

[46] M. Wray, R. Hawkes, *Distributed virtual environments and VRML: an event-based architecture*. In Proc. of the Seventh International WWW Conference (WWW7), Brisbane, Australia, 1998.

[47] H. Yu, D. Estrin, and R. Govindan, *A hierarchical proxy architecture for Internet-scale event services*. In Proc. of WETICE, 1999.

[48] 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 The 19th IEEE International Conference on Distributed Computing Systems, 1999.

[49] M. Mansouri-Samani, and M. Sloman, *GEM: A generalized event monitoring language for distributed systems*. IEE/IOP/BCS Distributed Systems Engineering Journal 4, 96–108, 1997.

[50] B. Krishnamurthy, and D. S. Rosenblum, *Yeast: A general purpose event-action system*. IEEE Transactions on Software Engineering 21, 845–857, 2002.

[51] 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, 2001.

[52] E. Oliver, *Exploiting the short message service as a control channel in challenged network environments*. In Proc. of the third ACM workshop on Challenged networks, 2008.

[53] P. Zerfos, X. Meng, S. H. Wong, V. Samanta, and S. Lu, *A study of the short message service of a nationwide cellular network*. In Proc. of 6th ACM SIGCOMM on Internet Measurement, 2006.

[54] D. D. Clark, M. L. Lambert, and L. Zhang, *Netblt: a high throughput transport protocol*. SIGCOMM Computer Communication Review, 17(5):353–359, 1987.

[55] B. Tudor, and C. Pettey, *Gartner says worldwide mobile phone sales grew 17 per cent in first quarter 2010*. [Online], Available: http://www.gartner.com/it/page.jsp?id=1372013, 2010.

[56] Huaigu Wu, Louenas Hamdi, and Nolwen Mahe, *TANGO: A flexible mobility-enabled architecture for online and offline mobile enterprise applications*. International Conference on Mobile Data Management (MDM), 230-238, 2008.

[57] R. Entner, *Smartphones to overtake feature phones in U.S. by 2011*. [Online], Available: http://blog.nielsen.com/nielsenwire/consumer/smartphones-to-overtake-feature-phones-in-u-s-by-2011/, 2010.

[58] *GSM World. market data summary*. [Online], Available: http://www.gsmworld.com/newsroom/market-data/market_data_-summary.htm, 2009.

[59] Y. Agarwal, R. Chandra, A. Wolman, P. Bahl, K. Chin, and R. Gupta, *Wireless wakeups revisited: energy management for VoIP over Wi-Fi smartphones*. In Proc. of the Fifth International Conference on Mobile Systems, Applications, and Services (MobiSys), 2007.

[60] *Us Wireless Quick Facts*. [Online], Available: http://www.ctia.org/consumer_info/service/index.cfm/AID/10323l.

[61] A. Rahmati, and L. Zhong, *Context for wireless: context-sensitive energy-efficient wireless data transfer*. In Proc. of MobiSys, 2007.

[62] *Rogers text messaging add on*. [Online], Available: http://www.rogers.com/web/content/add-ons/text-messaging.

[63] *Rogers data plan*. [Online], Available: http://www.rogers.com/web/Rogers.portal?_nfpb=true&_pageLabel=WLRS_Plans&category=data.

[64] *Rogers*. [Online], Available: http://www.rogers.com.