Università degli Studi di Padova

FACOLTA' DI SCIENZE MM. FF. NN.
Corso di Laurea Magistrale in Informatica

TESI DI LAUREA

A Delay Tolerant Solution for P2P file sharing in MANETs

Relatore:
Chiar.mo Prof. Claudio Enrico Palazzi

Laureando:
Armir Bujari

Anno Accademico 2009/2010

Abstract

Mobile phones have already evolved from simple voice communication means into a powerful device able to handle multimedia documents, personal productivity applications, and all sort of connections to the Internet. Due to their low cost, they are growing in popularity and might eventually become the dominant mode by which users interconnect; e.g. by establishing opportunistic P2P ad-hoc connections exchanging information of interest just passing by each other. It is hence expected to see a popular application such as file sharing to become widely utilized even in this context.

Despite this, P2P applications for Mobile Ad-hoc Networks (MANETs) in contrast to the wired Internet counterparts are still in their infancy. Due to the low density of mobile nodes or their limit in wireless radio range, continuous network connectivity cannot be sustained between wireless mobile phones, thus embodying a delay/disruption tolerant network (DTN). Indeed DTNs are used in supporting communications in situations with intermittent connectivity, long or variable delay, and high error rates - characteristics that common them with the wireless mobile world. They use (i) an asynchronous communication model (modeled after email) rather than rely on end-to-end communication and (ii) message replication techniques in order to maximize the probability of data delivery to destination.

In this context, we have created M2MShare a P2P file sharing application for local ad-hoc disconnected networks. Our approach is based on an application layer overlay network where overlay routes are set up on demand by the search algorithm, closely matching network topology and transparently aggregating redundant transfer paths on a per-file basis. Moreover, M2MShare adopts the idea of a DTN into the mobile world addressing node density issue by providing means for an asynchronous information exchange similar to that of DTNs. It models the idea of a DTN in an infrastructure less environment where both source of the request and destination of the data are the same entities and intermediary nodes (servants) store-delegate-and-forward back the requested data. This forwarding path is dynamically established, each individual node locally selects the next best suited hop based on a metric defined at the protocol level and this process is entirely user-transparent. Mobility is not seen as an obstacle instead, we leverage user-device mobility to address the node density problem.

1 INTRODUCTION 3

2 TECHNOLOGY BACKGROUND 15

2.1 DELAY/DISRUPTION TOLERANT NETWORKING (DTN) 15
2.2 P2P SYSTEM DESIGN INSIGHTS 19
2.2.1 Graph theoretic perspective 21
2.2.2 Graph considerations 23
2.2.3 Mobility models 24
2.2.4 Unstructured Overlays 27
2.2.4.1 Basic routing in unstructured overlays 28
2.1.4.2 Case study: Gnutella file sharing system 32
2.2.5 Structured Overlays 35
2.2.5.1 A designer space perspective 36
2.2.5.2 Case study: Chord overlay 38
2.3 BLUETOOTH 40
2.3.1 Architecture 41
2.3.2 Piconet and scatternet 43
2.3.3 Device inquiry and service discovery 46
2.4 JAVA 2 MICRO EDITION (J2ME) 47
2.4.1 Configurations and profiles 48
2.4.2 Connected Limited Device Configuration (CLDC) 48
2.4.3 MIDlet 49

3 RELATED WORK 51

3.1. SEVEN DEGREE OF SEPARATION (7DS) 51
3.2. OPTIMIZED ROUTING INDEPENDENT OVERLAY 54
3.3. DTN-BASED AD-HOC NETWORKING TO MOBILE PHONES 59
3.4. SEARCH 60

4 M2MSHARE 65

4.1. SEARCH 67
4.2. DELAY TOLERANT ASPECT 69
4.2.1 Servant election strategy 72
4.2.2 Peer slot management policy 77
4.3. TRANSPORT MODULE 78
4.3.1 Queuing mechanism and scheduling policies 79
4.3.2 Task execution and lifecycle. 83
4.3.3 Task encoding, communication protocol 84
4.3.4 Transfer Recovery 95
4.3.5 File division strategy 96
4.4 ROUTING MODULE 102
4.5 MAC MODULE 103
4.5.1 PresenceCollector service 105
4.5.2 Broadcast service 111

5 CONCLUSIONS 113

REFERENCES **121**

FIGURES AND TABLES INDEX **127**

ACKNOWLEDGEMENTS **130**

Chapter 1

Introduction

For years, humans have been building a global communications network set eventually to bring all members of our species within range for potential communication and to support new types of networked applications. The totality of this network is growing, resulting in an increasingly connected world. While the network's core – what we generally think of as the Internet – is highly connected and well suited for routing via conventional routing algorithms, the network's expanding frontiers have infrastructure that suffers from intermittent connectivity and changes in topology that can be difficult or impossible to predict. Examples include the infrastructure-challenged environments, the interplanetary networks whose nodes are tasked with the exploration of our solar system but also the more conventional mobile networks used in developed countries which are an area of active research.

A Mobile Ad-hoc Network (MANET [1]) is a collection of mobile nodes that dynamically self organize in a wireless network without using any pre-existing infrastructure. Due to their distributed nature, its networked applications typically employ the peer-to-peer (P2P) paradigm rather than client-server. In these scenarios, mobile users have to rely on node and service discovery and applications that cope with network disruptions. It is hence reasonable to explore other means of communication and network organization.

Each network is adapted to a particular communication region, in which communication characteristics are relatively homogeneous. Spanning two network regions requires the intervention of an agent that can translate between incompatible networks characteristics and act as a buffer for mismatched networks delays. A Delay/Disruption Tolerant Network (DTN [2]) is an overlay on top of regional networks, including the Internet. They support interoperability of regional networks by accommodating long delay between and within regional networks, and by translating between regional network communications characteristics. In providing these functions, DTNs can accommodate the mobility and limited power evolving wireless communication

devices. Indeed, MANETs and DTNs share common characteristics (Section 2.1) and seem a perfect match.

While research into routing in mobile environments is not new, researchers have for many years assumed traffic and node movement to be random. In reality, however, mobile nodes are of course used by people, whose behaviors are better described by social models. This has opened up new possibilities for routing, since the knowledge that behavior patterns exist allows better routing decisions to be made [36, 37]. In this thesis we exploit this idea of social relations between users operating mobile wireless devices and provide a proof of concept implementation, adopting a DTN type solution for the mobile disconnected networks.

Beyond personal computers: P2P networking has emerged as a viable business model and systems architecture for Internet-scale applications. Although its technological roots trace back through several decades of designing distributed information systems, contemporary applications demonstrate that it is an effective way to build applications that connect millions of users across the globe without reliance on specially deployed servers.

Public attention to P2P applications came first from highly popular file-sharing systems, in which users across the globe share content with each other. The social perception of the acceptability and benefits of content distribution through P2P applications has been irrevocably altered. Nearly 10 years after the World Wide Web became available for use on the Internet, decentralized P2P file sharing applications supplanted the server based Napster [4] application, second-generation protocols such as Gnutella [5], FastTrack [6], and BitTorrent [7] in which there is no central directory and all file searches and transfers are distributed among the corresponding peers were born. The era of fully decentralized P2P systems began.
Subsequently there has been growing interest in improving on these systems as well as considering new designs to attain better performance, security and flexibility. Today it is anticipated that P2P technologies will become general-purpose, mobile ad hoc networking, information delivery, social networking and personal communications applications in the future [3].

Though the majority of P2P applications today are running on desktops, the use of P2P technology is not restricted to desktop computing. In fact, P2P appears to be *an even better fit* for the capabilities and usage patterns of networked consumer electronics (CEs). For example, Figure 1.1 illustrates a P2P scenario involving streaming media with peer casting of live video to a set of different viewing devices.

On a limited scale, P2P is already being used for such personal devices. Because P2P means that a user's personal devices can interoperate without requiring a server, it is attractive to many consumers. Several industry specifications have been developed for devices to connect peer to peer and to share resources and services. For example, the Universal Plug-and-Play (UPnP [10]) standard defines protocols for devices in the home network to directly advertise their services to other devices and for other devices to discover and use these services. Bluetooth is another standard for wireless devices to locate other devices and share services.

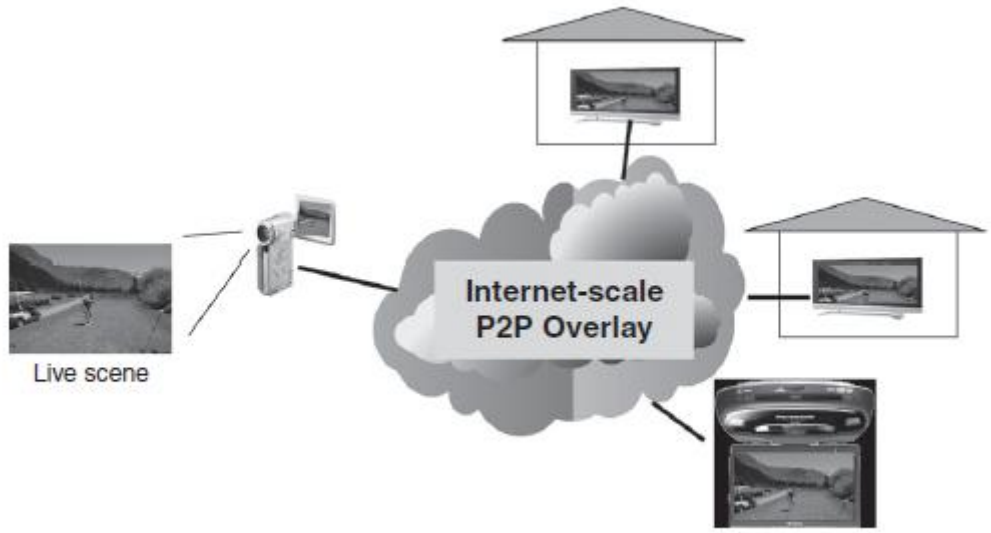


Figure 1.1 - A P2P streaming scenario involving networked consumer electronics devices as peers [17].

Sharing services between devices can expand the capability of the device without changing its cost (Figure 1.2). For example, the keypad of one device (Figure 1.2A) might be a more capable way of controlling a second device while displaying its output on a third device with a larger screen. Using local P2P connections, devices can also share storage (Figure 1.2B) or services such as instant messaging (Figure 1.2C).

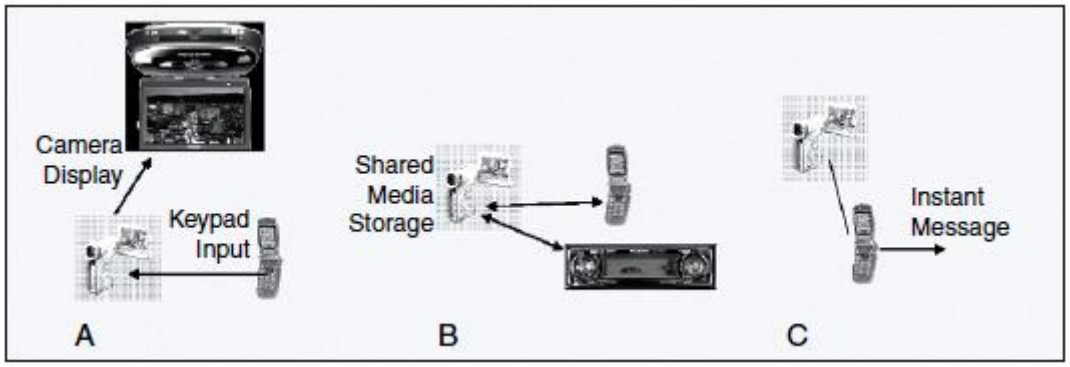


Figure 1.2 – P2P device composition [17].

The application scenarios shown before are intriguing but rather simple in their nature. In this work we impose a much higher goal – that of finding some suitable system features and characteristics for the deployment of a P2P file sharing application into the mobile world and provide a proof of concept implementation. More specifically, we see the mobile phones as our target platform since they have evolved from simple voice communication means into powerful device able to handle multimedia documents, personal productivity applications, and all sort of connections to the Internet. It is hence expected to see a popular application such as file sharing to become widely used even in this context. Indeed, the combination among mobile users and file sharing seems a match made in heaven: people could exchange files of interest just passing by each other.

We continue our discussion by introducing what MANETs are and the challenges they pose in designing a P2P system for this type of infrastructure.

Mobile ad-hoc networks. The P2P model can operate in both Internet-scale and ad hoc networks such as in MANETs, those formed spontaneously when a group of friends join their devices together. A mobile ad hoc network (MANET) is a collection of mobile nodes that dynamically self organize in a wireless network without using any pre-existing infrastructure. In a MANET, the applications are typically P2P rather than "client-server". Moreover, a MANET is often built to support a specific application, thus the networking is application-driven. For these reasons, it is often referred to the ad hoc networking done in a MANET as P2P networking. This however is not quite consistent with the general meaning of P2P in the broader Internet context. In the Internet, a P2P network is basically an overlay network justified by the need for

specialized functions that are not possible or cost-effective in the IP layer. These functions must be performed at the middleware or application layer. Classic examples of Internet overlay networks are: multicast overlays (which overcome the lack for multicast support in the IP routers), unstructured P2P systems such as Gnutella, BitTorrent, and; structured P2P systems such as Pastry [11], Chord [12]. MANETs are based on a network architecture that is quite different from the wired Internet architecture. In fact, a typical MANET architecture reminds us of the architectures introduced in the '80s to interconnect LANs, with level 2 routing which is based on MAC addresses. The MANET routing functionality is non-hierarchical on the level-2 and, thus, is several layers below the functionality offered by a P2P network. This implies that in a MANET the P2P concept will be implemented on top of level-2 routing. There is a definite need for overlay/P2P networking in a MANET, for the following reasons: (a) the MANET routing layer is often inadequate to provide the services needed by sophisticated mobile applications, and; (b) the unpredictability of the radio channel combined with the mobility of the users can pose major challenges to routing, requiring upper layer intervention.

As an example of wireless overlay, consider a large ad hoc network deployed to overcome a major natural disaster. A sad and very timely example is the South Asia Tsunami disaster. The ad hoc network may be a combination of heterogeneous technologies – from satellites to ground ad hoc radios and improvised cellular and mesh network services. Different teams are formed – a few teams will cooperate in a particular mission. For instance, three or four different teams may search for survivors; others will be in charge of distributing food and supplies; there will be medic's teams providing first aid and medications; engineering teams for reconstruction; police teams preventing looting, etc). These teams move and operate as groups. They must coordinate their operations. They require multicast and possibly content based routing. It thus makes sense to develop "team based multicast" and "content based routing" schemes at the user level, on top of the very basic routing service provided by the "instant", ad hoc, heterogeneous infrastructure.

As another example, consider a "delay tolerant" file sharing application that includes hosts partly in the Internet and partly on peripheral wireless ad hoc networks. Wireless nomadic users can rapidly change their connectivity to the Internet from Kbps (say

GPRS) to Mbps (say, 802.11). Occasionally, the users may become disconnected. The use of the standard network routing protocols may lead to inefficiencies, violation of delay constraints and possibly retransmission of large portions of the file. A P2P overlay network can keep track of connectivity among the various hosts. The overlay network can extend to wired, wireless and ad hoc network segments. It can predict disconnection/reconnection dynamics and can exploit them to deliver files efficiently and within constraints (for example, using intermediate proxy nodes for "bundle" store-and-forwarding alike DTN [2]). As the wireless, mobile network structures grow large (e.g., battlefield, urban vehicular grid, etc), different applications may emerge with different customized routing requirements. Moreover, some MANETs may grow as an "opportunistic" extension of the wired Internet. In this case, some of the opportunistic ad hoc network users will want to participate in Internet applications already supported by a P2P overlay in the Internet (e.g. games, file sharing). This will again create the need to extend the P2P concept to wireless. The picture below (Figure 1.3) shows the building blocks of a P2P system and the potential application scenario

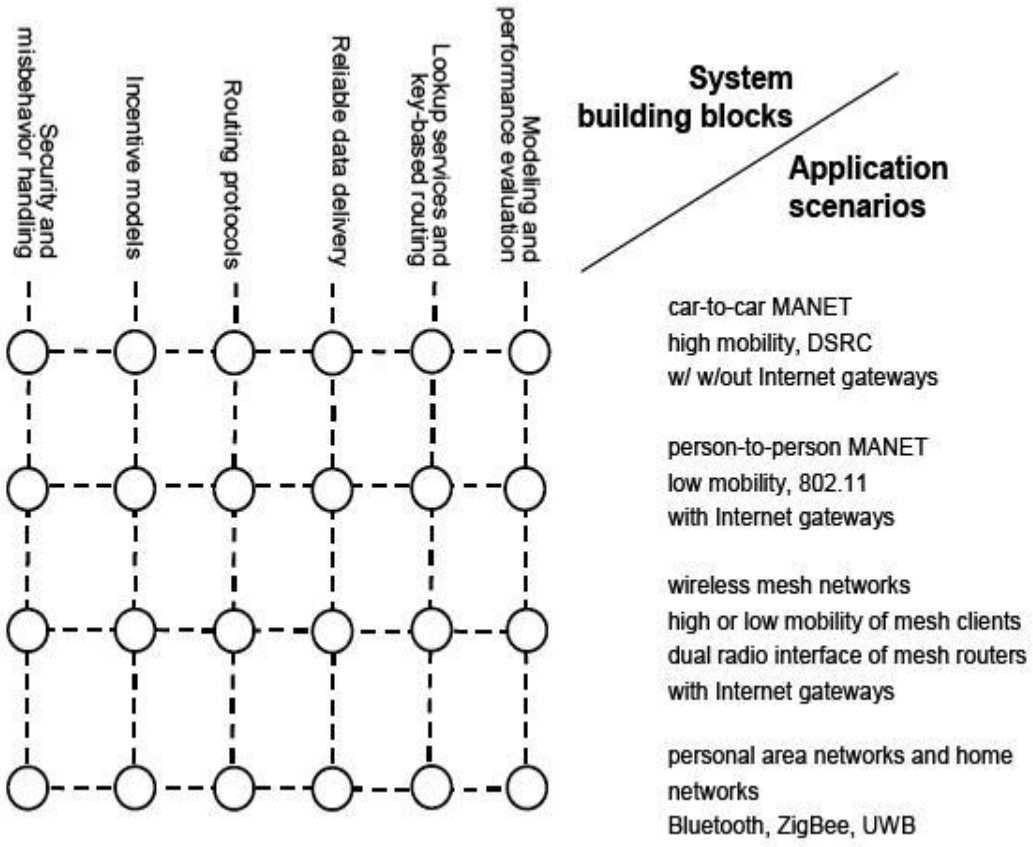


Figure 1.3 - System building blocks and application scenarios of P2P MANET [40].

Issues to address: As stated before, portable electronic devices are providing functionality previously restricted to desktop computers. Such devices can interconnect using broadband wireless network interfaces. Local storage and compute capacity are sufficient for storing and playing high-quality audio and movie files. Due to their low cost, such devices are growing in popularity and may at some time become the dominant mode by which users reach the Internet, in a way we could consider them as an *opportunistic extension* of the today's Internet. Consequently, they will have an important role in future P2P overlays.

Mobile devices have some important characteristics that differentiate them from desktop computers and that affect their interaction with the classical P2P wired infrastructure overlays. Due to their **mobility**, the amount of time a peer participates in the overlay will be more limited and any departure will result in interruption of on-going activities. In addition, in case where static connections are established, any peers that were neighbors of the departed peer may now have out of date routing information. As an example, consider a pre-established wireless Bluetooth overlay and two participating peers moving in opposite directions with 1m/s, this will result in 10 seconds of link duration. If churn had to be dealt accordingly in the wired overlay case, here it is much more of a problem as mobility shortens peer lifetime. Furthermore, an important issue is that of **peer density** which is expected to be low and this can undermine the utility of a P2P system; e.g. for a file sharing software content search is a key feature, having low peer density in such a case directly influences the population of data content in the overlay.

Mobile devices have energy issues, thus the need to **preserve energy** [13] is essential. Power management involves a combination of techniques, including network adapters that can trigger power resume of the host while offloading certain network activity and network protocols that reduce power consumption. In current overlay designs, a mobile peer that goes into power-saving mode is treated as a node that has left the overlay.

Another, not less important characteristic that differentiate the wireless from the wired world is the **communication technology**. The wireless medium as opposed to the wired communication technology is known to be error-prone and bandwidth limited. Limited bandwidth alone could mean lower data quantity transferred, adding to this the mobility factor and low peer density we might need other ways of reliable synchronous or asynchronous data transfer (see further).

Concluding, mobile environments face multiple challenges and traditional solutions for P2P over fixed networks may need to be redesigned when applied in the mobile world.

Proposal (M2MShare): Indeed, current mobile phones have multiple communication technologies built into them (e.g. WiFi, Bluetooth). M2MShare exploits this feature to create a P2P overlay network, which would allow automatic exchange of files and data among phones.
However a single device actively uploading and downloading files using WiFi will quickly become out of power due to battery consumptions. Even UMTS does not represent the best choice as its use is generally associated with a cost. Instead, the best option to guarantee quick connectivity is represented by Bluetooth[12]: it allows from 1Mbps up to 3Mbps of connectivity (even more in v3.0 of protocol specification), it is free, it has a range of about 10, hence sufficient for data exchange among customers in a store or commuters in a bus.

In this thesis, we present a special-purpose approach for P2P file sharing tailored to MANETs. Once you run M2MShare it will automatically initiate a search by broadcasting a query request toward other Bluetooth enabled devices. Once the answer/s is received, data found to match the criteria will be automatically requested (when unique file id is known). All this process is done without user mediation; it is entirely user-transparent, the user only needs to specify the data identifiers, search keys or unique file id needed. Key features of M2MShare are its ease of use and autonomy. In fact, once the initial preferences are set up no further user interaction is needed. Given that energy consumption is a problem for handheld devices the notion of active sessions was introduced. An active session is a period of time in which the software is functional and performs its duties. Such periods are configurable through the graphical user interface.

It addresses the problematic stated before, such as **(1)** mobility and its impact on overlay operations, by providing a mechanism that has little or no overhead in building new routes and maintaining those already established **(2)** new transfer protocol equipped with a synchronous and asynchronous data transfer that addresses peer density and data reachability problem.

(1) - Mobile ad hoc networks and P2P file sharing systems both exhibit a lack of fixed infrastructure and pose no a-priori knowledge of arriving and departing peers.
The operation of most P2P systems in the wired Internet depends on application layer connections among peers, forming an application layer overlay network. In general, these connections are static, i.e., a connection between two peers remains established as long as both peers dwell in the system. The maintenance of static overlay connections is the major performance bottleneck for deploying a P2P file sharing system in a MANET.

In fact the overlay network topology does not reflect the underlying MANET topology neither in terms of connection layout nor in connection lifetime. Whereas the overlay network topology is static in the timescale of a node's dwell time in the system, the MANET topology changes much more frequent due to node mobility. This induces significant control overhead for connection maintenance, resulting in increasing network traffic and decreasing search accuracy. M2MShare comprises of an algorithm for construction and maintenance of an application-layer overlay network that enables routing of all types of messages required to operate a P2P file sharing system, i.e., queries, responses, and file transmissions.

Overlay connections are set up on demand and maintained only as long as necessary, closely matching the current topology of the underlying network. The routing algorithm resembles the Ad Hoc On Demand Distance Vector (AODV [15]) routing protocol and the Simple Broadcast protocol: a query request message is flooded in the overlay and is propagated for a certain number of hops, after that, the message is discarded. Peers keep history of previously seen messages so that they will accept every message only once.

(2) - At the transport layer a new file transfer strategy is implemented and empirical results are given to prove its effectiveness under certain conditions and usage patterns. Along with synchronous file transfer (source directly in reach area) it also provides an asynchronous transfer mode; Traditional ad-hoc networking itself is usually not sufficient because peer density may be far too low to establish a network layer end-to-end path between two communicating peers. Even if a sufficient number of mobile devices is around, their owners may not be willing to cooperate to save their own resources e.g., energy or because they fear misuse; device heterogeneity may inhibit

interworking; and radio range and interference may limit communications. For a file sharing application peer density and participation in the overlay is crucial as data popularity, reachability presumably will be higher. M2MShare addresses some of these problems by providing means for an asynchronous information exchange similar to DTN.

DTNs are used in supporting communications in situations with intermittent connectivity. They use asynchronous communication (modeled after email), rather than rely on end-to-end communications using packets. DTN architecture presumes a pre-existing infrastructure, such as routers that store/forward messages (bundles) toward other infrastructure end-points, delegating them the responsibility; whereas M2MShare models the idea in an infrastructure less environment. This idea was worth exploiting as we can delegate queries and data transfer to other device expanding the search area and reaching other content outside the current established overlay. In a way we exploit node mobility to resolve the data reachability problem.

Not all peers in the overlay act as servant, a peer can behave as such only if it has been chosen by a another peer. This process is user-transparent and is handled at the protocol level. A metric is defined for electing one device instead of others as servant and a relation of such kind is (i) non symmetric and (ii) subject to local decisional process of a particular peer. Also, delegated tasks and remotely stored data do have a lifetime, after which they are deleted; lifetime might be one active session or a multiple of active sessions. This ‘’expiry unit‘’ is relative and subject to local decisions of that servant.

Outline:

This work investigates the porting of a DTN type solution in the mobile wireless world and provides a proof of concept implementation. On our judgment, the added values and contributions are the following:

1. Providing an overview of the solution and practical design considerations to other researchers/practitioners that might be interested in developing a similar project.

2. Demonstrating a new paradigm of use of P2P solutions that matches file sharing with mobile users, allowing them to exchange files with each other, thus fostering new applications.

3. Combining and providing a proof of concept solution for porting DTN type communication in the mobile world.

4. Providing a protocol that dynamically establishes forward routes (delegations) along the destination path whose activity is entirely user-transparent.

5. Defining the criteria and implementing a solution for single-hop task delegation to specific overlay peers (servant) extending the search reach area to other disconnected overlay networks.

The remainder of this thesis is organized as follows. Chapter II gives some insights on P2P overlay networks and overlay organization techniques, furthermore introducing the reader to the adopted technologies such as, Bluetooth, and the software development platform. Chapter III summarizes related work in the area of P2P systems in MANETs. In Chapter IV, we present the M2MShare protocol stack and its components. Finally, concluding remarks are given

.

Chapter 2

Technology background

This chapter provides a general overview of the technologies adopted during this work in order to establish an adequate background for the reminding chapters. We begin by introducing the DTNs, explaining what they are and why there is an emergent need for such type of solutions in an increasingly connected world. Next, we discuss about the P2P overlay/peer organization and as we will see there are different ways to do this. A graph theoretic perspective is introduced to provide additional clarity to some concepts and terminology along with some traditional routing algorithms adopted by several overlays. Furthermore, we review some mobility models used for measuring routing performance algorithms.

The Bluetooth architecture and the underlying network topology are explained, in addition to basic Bluetooth actions like *device discovery* and *service discovery*. Finally, we introduce the reader with the development platform where software implementation was done.

2.1 Delay/Disruption Tolerant Networking (DTN)

The Internet has been a great success at interconnecting communication devices across the globe. It has done this by using a homogeneous set of communication protocols, called the *TCP/IP* protocol suite [21]. All devices on the hundreds of thousands of subnets that make up the Internet use these protocols for routing data and insuring the reliability of message exchanges.

Connectivity on the Internet relies primarily on the wired links (Figure 2.12), including the wired telephone network, although new wireless technologies such as short-range mobile and satellite links are beginning to appear. These links are continuously connected in end-to-end, low-delay paths between sources and destinations. They have low error rates and relatively symmetric bidirectional data rates.

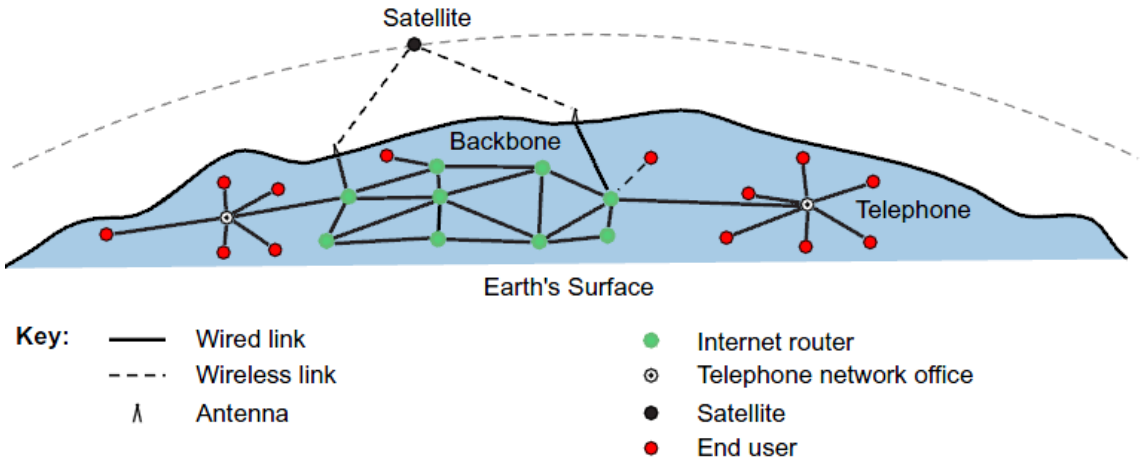


Figure 2.1 – Internet end-to-end communication links [2].

Communication outside of the Internet- where power-limited mobile wireless, satellite, and interplanetary communications are developing - is accomplished on independent networks, each supporting specialized communication requirements. These networks do not use Internet protocols and they are mutually in-compatible - each is good at passing messages within its networks, but not able to exchange messages between networks.
Each network is adapted to a particular communication region, in which communication characteristics are relatively homogeneous. Spanning two network regions requires the intervention of an agent that can translate between incompatible networks characteristics and act as a buffer for mismatched networks delays. Here comes at play the DTN type networking.

A DTN is a network of regional networks (Figure 2.13). It is an overlay on top of regional networks, including the Internet. DTN support interoperability of regional networks by accommodating long delay between and within regional networks, and by translating between regional network communications characteristics. In providing these functions, DTNs accommodate the mobility and limited power evolving wireless communication devices.

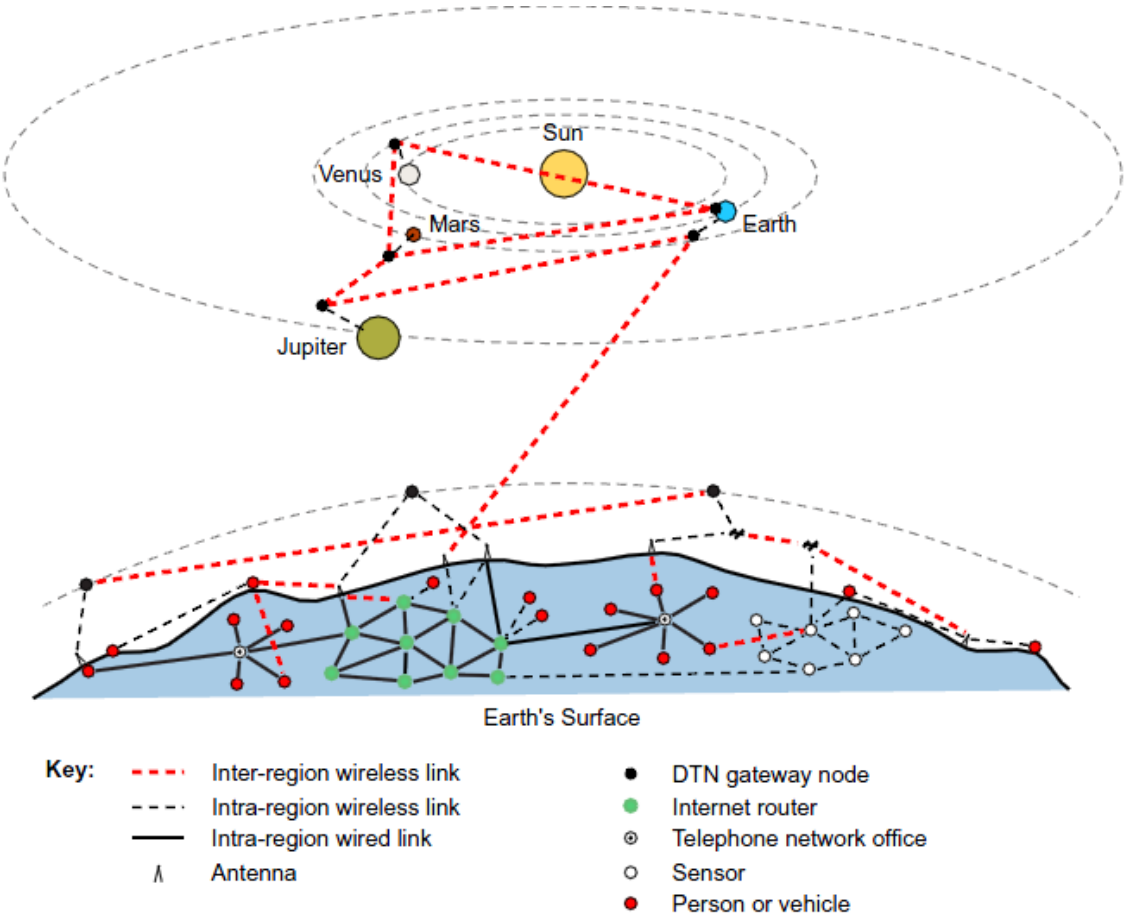


Figure 2.2 – DTN overlay, connecting different regional homogeneous networks providing means for intra-region communication [2].

Many evolving and potential networks do not confirm to the Internet underlying assumptions. These networks are characterized by:

- *Intermittent Connectivity*: If there is no end-to-end path between source and destination - called network partitioning - end-to-end communication using the *TCP/IP* protocols does not work. Other protocols are required.

- *Long or variable delays*: In addition to intermittent connectivity, propagation delays between nodes and variable queuing delays at nodes contribute to end-to-end delays that can defeat Internet protocols and applications that rely on quick return of acknowledgements or data.

- *Asymmetric Data Rates*: The internet supports moderate asymmetries of bidirectional data rate for users with cable TV or asymmetric DSL access. But if asymmetries are large, they defeat actual protocols.

- *High error rates*: Bit errors or links require correction (added redundancy and processing time) or retransmission of the entire packet. For a given link-error rate, fewer retransmissions are needed for hop-by-hop than for end-to-end retransmission.

DTNs overcome these problems by using a *store-and-forward message switching* (Figure 2.14). This is an old method, used by pony-express and postal systems since ancient times. Whole messages (entire blocks of application-program user data) - or pieces (fragments) of such messages - are moved (forwarded) from a storage place on one node to a storage place on another node, along a path that eventually reaches destination.

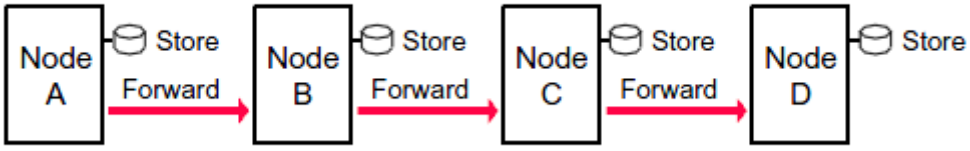


Figure 2.3 – Store-and-forward packet switching between DTN routers.

Store-and-forwarding methods are also used in today's voicemail and email systems, although these systems are not one-way relays (as shown above) but rather star relays; both the source and destination independently contact a central storage device at the center of the links.
Furthermore, DTN routers need persistent storage for their queues as opposed to Internet routers that use short-term storage provided by memory chips. This for the following reasons:

- A communication link to the next hop map not be available for a long time.
- One node in a communicating pair may send or receive data much faster or more reliably than the other node.

- A message, once transmitted, may need to be retransmitted if an error occurs at an upstream (toward destination) node or link, or if an upstream node declines acceptance of a forwarded message.

DTNs are still an area of research and a lot of other features would be worth mentioning but are outside of the scope of this work. For more insights refer to [2].

2.2 P2P system design insights

Peers in P2P applications communicate with each other using messages transmitted over the Internet or other types of networks, as in our case ad-hoc networks. The protocol for a P2P application is the set of different message types and their semantics, which are understood by all peers. The protocols of various P2P applications have some common features. First, these protocols are constructed at the application layer of the network protocol stack. Second, in most designs, peers have a unique identifier, which is the peer ID or peer address. Third, many of the message types defined in various P2P protocols are similar. Finally, the protocol supports some type of message-routing capability. That is, a message intended for one peer can be transmitted via intermediate peers to reach the destination peer.

To distinguish the operation of the P2P protocol at the application layer from the behavior of the underlying physical network, the collection of peer connections in a P2P network is called a P2P overlay. Figure 2.1 shows the correspondence between peers connecting in an overlay network with the corresponding hosts, devices, and routers in the underlying physical network.

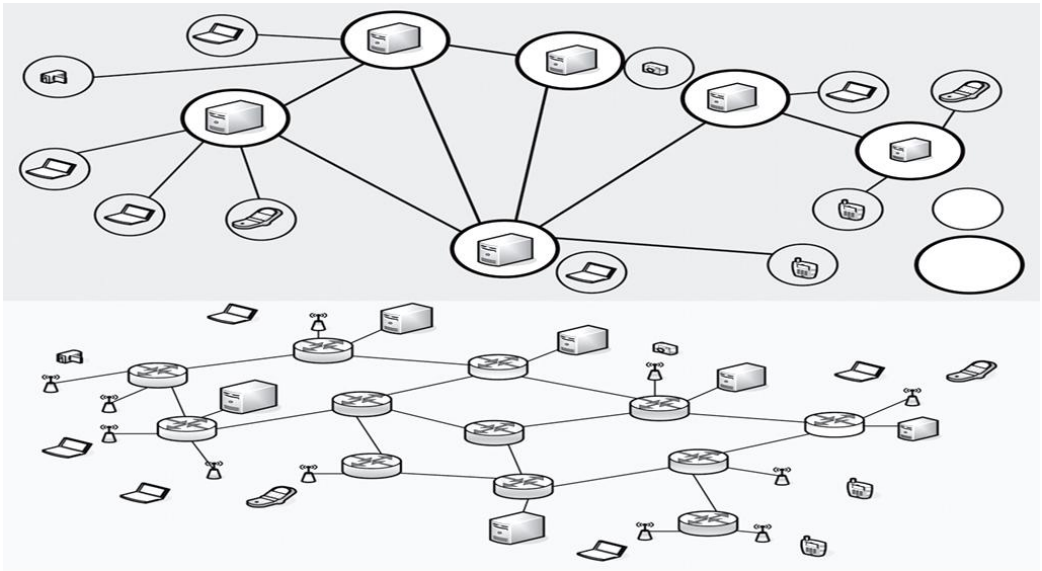


Figure 2.4 - Peers form an overlay network (top) that in turn uses network connections in the native network (bottom). The overlay organization is a logical view that might not directly mirror the physical network topology [17].

As mentioned earlier, the practice of overlay networks predates the P2P application era. For example, protocols used in Internet news servers and Internet mail servers are early examples of widely used overlays that implement important network services. These specialized overlay networks were developed for various reasons, such as enabling end-to-end network communication regardless of network boundaries caused by network address translation (NAT). Another important reason for the use of overlays is to provide a network service that is not yet available within the network. For example, multicast routing is a network service that to date has been only partially adopted on the Internet. Multicast routing enables a message sent to a single multicast address to be routed to all receivers that are members of the multicast group. This is important for reducing network traffic for one-to-many applications such as video broadcasting or videoconferencing. Since multicast routing is not universally supported in Internet routers, researchers developed an application layer capability for multicast routing called application layer multicast (ALM) or overlay multicast (OM).

There are a number of ways to look at P2P overlays, from user to application developer, system designer, and researcher. Similarly, there are a variety of relevant conceptual, theoretical, and implementation perspectives to consider.

Below a graph-theoretic notation is introduced to provide additional clarity to some concepts and terminology. A graph-theoretic view is helpful to explain functions such as overlay multicasting, mobility and security that depend on the semantics of the overlay. Another use of this perspective is as a step toward proving certain aspects of an overlay algorithm. Examples include:

- Stability. Does the rate of change of state of the peers follow the rate of membership change?

- Convergence. Is a message guaranteed to reach any peer in a bounded number of steps? Does the routing state stay within a certain accuracy bounds?

- Boundary conditions. At what churn rate does the algorithm become unstable or non convergent?

These questions are difficult to answer in general and have been studied in most cases for specific algorithms.

2.2.1 Graph theoretic perspective

A P2P overlay can be viewed as a directed graph G = (V,E), where V is the set of nodes in the overlay and E is the set of links between nodes. Nodes are located in a physical network, which provides reliable message transport between nodes. Each node p has a unique identification number pid and a network address nid (this two numbers can coincide). An edge (p,q) ϵ E means that p has a direct path to send a message to q; that is, p can send a message to q over the network using q's nid as the destination.

It is desirable that G be a connected graph. Maintaining connectedness and a consistent view of *G* across all nodes is the job of the overlay maintenance mechanism.

Due to peers joining and leaving the overlay, the overlay graph G is dynamic. As an approximation (Figure 2.2), the overlay proceeds through a temporal sequence G_i (V_i, E_i), G_{i+1} (V_{i+1}, E_{i+1}),

G_{i+2} (V_{i+2}, E_{i+2}), … When a peer *p'* joins G_i at time i, the overlay join operation causes the overlay to become G_{i+1} (V_{i+1}, E_{i+1}), where $V_{i+1} = V_i \cup \{p'\}$ and $E_{i+1} = E_i \cup \{(p', m)\} \cup \{(n, p')\}$.

That is, the join operation adds *p'* to the set of nodes and adds at least one incoming and at least one outgoing link between *p'* and some other node in the overlay. Likewise, when a peer *p'* leaves G_i at time i, the overlay join mechanism causes the overlay to become G_{i+1} (V_{i+1}, E_{i+1}), where $V_{i+1} = V_i - \{p'\}$ and $E_{i+1} = \forall m,n\ E_i - \{(p',m)\} - \{(n, p')\}$. That is, the leave step removes p' from the set of nodes and removes all incoming and outgoing links between p' and the remaining nodes in the overlay.

The overlay ''join and leave'' operations are part of the overlay maintenance mechanism. Peers *p* and *q* are adjacent in the overlay if they have an edge (p, q) or (q, p) in E. A join or leave operation by peer *p* directly affects peers to which *p* is adjacent and may indirectly affect the state of other peers. That is, there is a direct effect on node *q* if (q, p) is added to *q's* routing table as a result of *p* joining the overlay, or if (q, p) is removed from *q's* routing table as a result of *p* leaving the overlay. If there is no direct effect on *q*, there may still be an indirect effect on *q* that causes it to change the organization or contents of its routing table.

Overlay maintenance is the management of these changes through the exchange of information about G between peers.

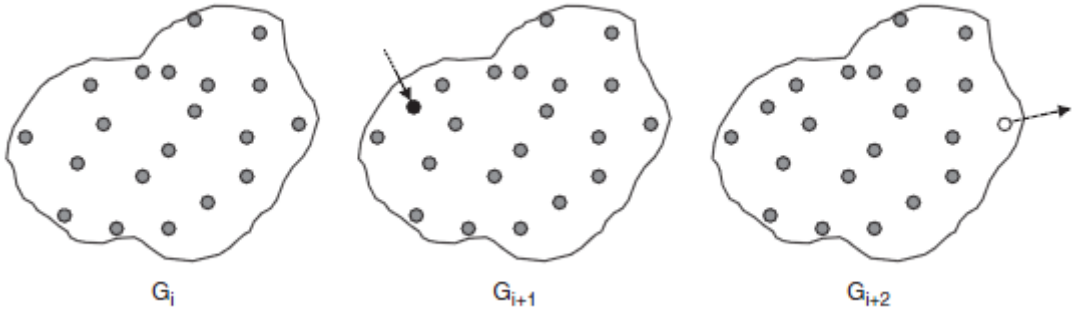


Figure 2.5 - Overlay shown as a sequence of membership changes [17].

The preceding temporal sequence model of G is an approximation because it assumes that peer membership changes are sequenced and that the overlay maintenance for a specific membership change completes before the next join or leave begins. It is possible that several nodes may join or leave the overlay simultaneously. In a large overlay, it is usually the case that overlay maintenance operations due to a specific membership change will take an extended period of time and will overlap many other overlay maintenance operations. Also, if a large number of peers leave the overlay simultaneously, it could cause the overlay to form one or more partitions.

G is a global view on overlay node membership and routing. Each peer has a local view of overlay node membership and connectivity, which is its routing table. Ignoring dynamics, each peer has a routing table R_p (V_p, E_p) ≤ G.

A peer routing table lists all nodes and edges to which it has a direct link. Each node entry in the routing table includes both its pid and nid.

Given the approximated overlay dynamics (G_i, G_{i+1}, G_{i+2}, . . .), it is possible that two peers *p'* and *p''* could have routing tables that are out of sync not only with each other, that is, $R_{p''} \leq G_i$ and $R_{p''} \leq G_{i+1}$, but also with respect to the current overlay state G_{i+n}. Further, $R_{p'}$ and $R_{p''}$ could have inconsistent views that are not subsets of any G_j. These inconsistent views are due to lack of global synchronization in the overlay and delays in propagating membership state changes throughout the overlay.

The routing procedure for peer *p* to send a message to some peer *u* is:

- If (p, u) ϵ E_p, then send the message directly to u.

- Otherwise, select one or more edges e = (p, q) from E_p according to a routing criteria that converges, that is, moves the message closer to the destination at each step.

Each edge from one peer to another peer that a message traverses in G is called a hop. Message path length from source to destination is frequently measured in terms of number of hops.

2.2.2 Graph considerations

Graph geometry affects routing performance, maintenance, and resiliency. The graph geometry defines the basic static graph topology. Example geometries that have been used in various P2P overlay designs include rings, de Bruijin graphs [17], hyper cubes, butterflies in the context of structured overlays and power-law graphs [17], random graphs, small-world model in the context of unstructured overlays. In this subsection we define several important graph properties that impact on the issues stated before.
The graph *diameter* is the maximum distance between any two nodes in the graph. From an overlay routing perspective, graph diameter provides a worst-case path length for sending a message between any two nodes in the overlay under static conditions. Consequently, graph diameter is an important parameter for comparing various geometries that might be used in an overlay.

Each peer in the overlay has a number of adjacent peers to which it has a direct connection. The number of such adjacencies is the *degree* of the peer. Outgoing degree is the number of adjacencies from the current peer to its neighbors. Incoming degree is the number of adjacencies to the current peer from its neighbors. A routing path is the sequence of edges or hops from the peer sending the message to the peer which is the target of the message. If each peer had complete information about all other peers in the overlay, each P2P message would take at most one hop. However, all peers would need to maintain an O(N) routing table size, and each join and leave event would need to propagate to all other peers in the overlay, creating a large maintenance load. This maintenance load would grow with both the size of the overlay, N, and the churn rate.

On the other hand, if a node knows only the link to its successor on a ring, routing state and maintenance load would be nominal but routing performance would be O(N).
Since neither of these two operating points is generally practical, there has been a great deal of interest in exploring the state versus overhead tradeoffs (Figure 2.3) of various algorithms.

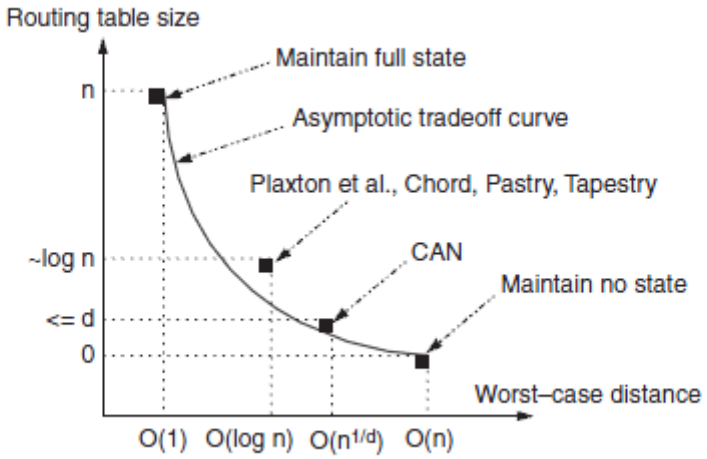


Figure 2.6 - Asymptotic tradeoff between peer-routing table size and overlay diameter # 2003 IEEE.

Our provided overlay organization does not establish static connections with neighboring nodes. Connections between nodes in our system are established on demand and maintained as long as necessary therefore reducing maintenance operations overhead to a minimum. When mobility comes into play a way to characterize the routing performance is to study routing algorithms under particular mobility models.

2.2.3 Mobility models

Nodes in MANETs are free to move randomly. Thus the network's wireless topology may be unpredictable and may change rapidly. MANETs are expected to be used in various applications with diverse topography and node configuration. Widely varying mobility characteristics are expected to have a significant impact on the performance of the routing protocols.

The overall performance of any wireless protocol depends on the duration of interconnections between any two nodes transferring data as well on the duration of

interconnections between nodes of a data path containing n-nodes. These parameters averaged over entire network are called as *Average Connected Paths*.

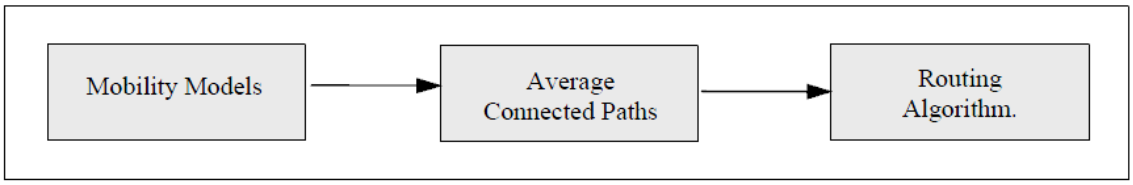


Figure 2.7 - Relationship between protocol performance and mobility model.

The mobility of the nodes affects the number of average connected paths, which in turn affect the performance of the routing algorithm. Different mobility models can be differentiated according to their spatial and temporal dependencies.

- *Spatial dependency:* It is a measure of how two nodes are dependent in their motion. If two nodes are moving in same direction then they have high spatial dependency.

- *Temporal dependency:* It is a measure of how current velocity (magnitude and direction) are related to previous velocity. Nodes having same velocity have high temporal dependency.

Given below are the descriptions of two mobility models with detailed explanation for how they emulate real world scenario. Each description is accompanied by a screenshot to give a visual representation of node movement in the model.

Random Waypoint: Is the most commonly used mobility model in research community. At every instant, a node randomly chooses a destination and moves towards it with a velocity chosen randomly from a uniform distribution $[0,V_{max}]$, where V_{max} is the maximum allowable velocity for every mobile node. After reaching the destination, the node stops for a duration defined by a ‘’pause time’’ parameter. After this duration, it again chooses a random destination and repeats the whole process until the simulation

ends. Figures 2.5 illustrate examples of a topography showing the movement of nodes for Random Mobility Model.

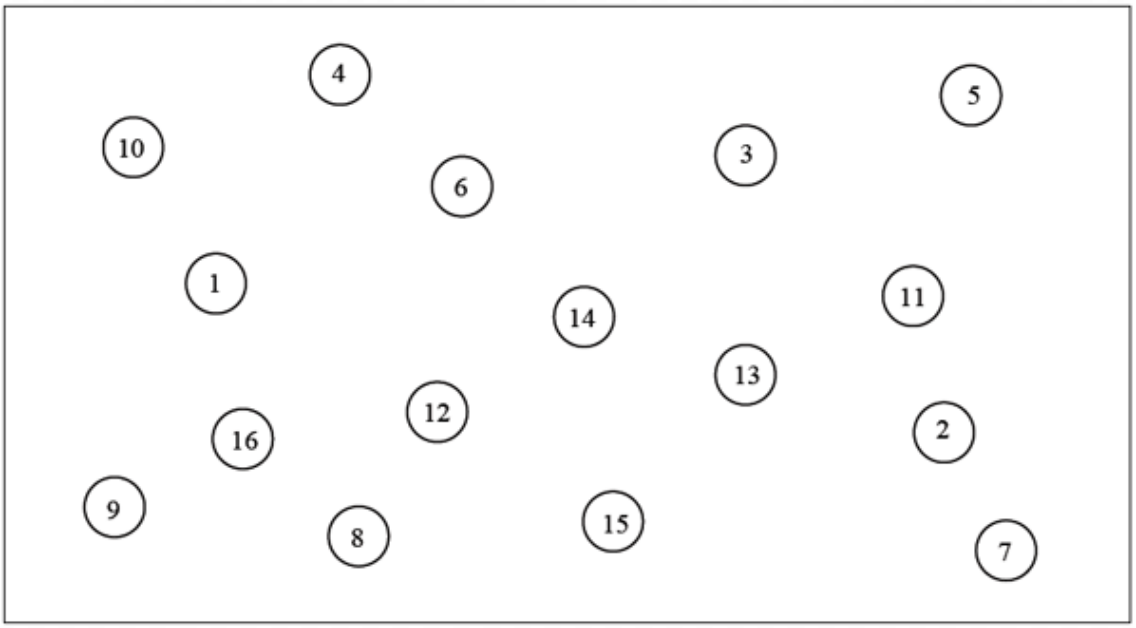


Figure 2.8 - Topography of nodes for Random mobility model.

Manhattan mobility model: This model is used to emulate the movement pattern of mobile nodes on streets. It can be useful in modeling movement in an urban area .The scenario is composed of a number of horizontal and vertical streets. Given below, Figure 2.6 is an example topography showing the movement of nodes for Manhattan Mobility Model with seventeen nodes. The map defines the roads along the nodes can move and imposes geographical restrictions to nodes movement. It is composed of a number of horizontal and vertical streets. The mobile node is allowed to move along the grid of horizontal and vertical streets on the map. At an intersection of a horizontal and a vertical street, the mobile node can turn left, right or go straight with certain probability.

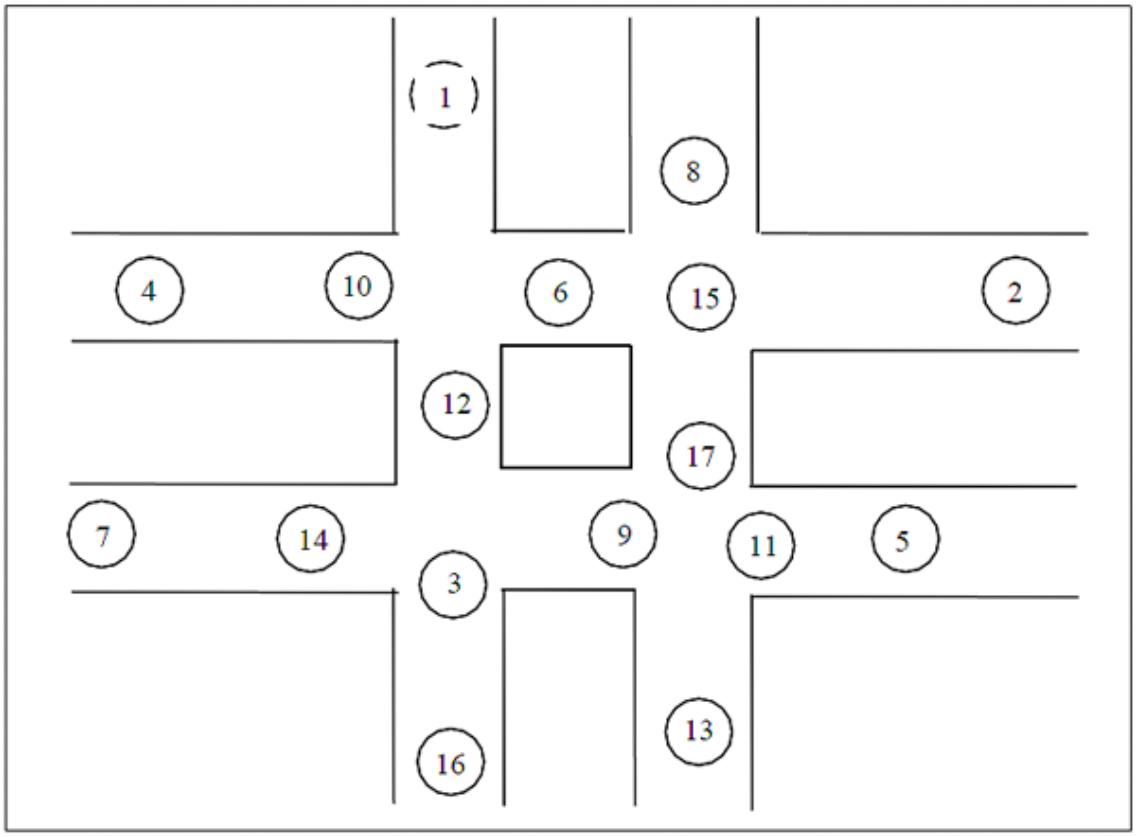


Figure 2.9 - Topography of nodes for Manhattan mobility model.

2.2.4 Unstructured Overlays

In this section we look at the way file and similar information sharing works when the P2P application does not impose much if any structure on the interconnection of the peers. Such approaches are classified as unstructured overlays.

An important observation about the nature of information sharing in P2P applications is that P2P involves primarily individuals as both information publishers and users. Thus there is a social dimension to P2P information sharing that is derived from the social relationships between the individuals who comprise the P2P system. These relationships might be explicit, as in sharing photographs with friends or family, or implicit, such as sharing based on common interests or common properties.

2.2.4.1 Basic routing in unstructured overlays

Let's assume that each peer keeps a list of other peers that it knows about. We can call these peers the neighbors. If neighbor relations are transitive, we have connectivity graphs such as the one shown in Figure 2.7A. In this particular graph, peers have between two and five neighbors in the overlay. The number of neighbors a peer has is called the degree of the peer. Increasing the degree of the peers reduces the longest path from one peer to another (the diameter of the overlay) but requires more storage at each peer. Once a peer is connected to the overlay, it can exchange messages with other peers in its neighbor list. An important type of message is a query for specific information. The query contains the search criteria, such as a filename or keywords. Since we do not know which peers in the overlay have the information, we could try sending a query to every peer we know. If the neighbor peers do not have the information, they can in turn forward the request to their neighbors, and so on. A few checks are needed to prevent messages from circulating endlessly.

First, in case the message loops back or is received over more than one path, each peer can keep a list of message identifiers that it has previously received. If it sees the same message again, it simply drops the duplicate message. Second, so that peers do not have to remember messages for an arbitrary time, which would require a continuously growing amount of storage, each message has a time-to-live (TTL) value that limits its lifetime. The TTL value of a message is set by the message originator and decremented by 1 at each peer that receives the message. When the TTL value of a message reaches 0, it is no longer forwarded.

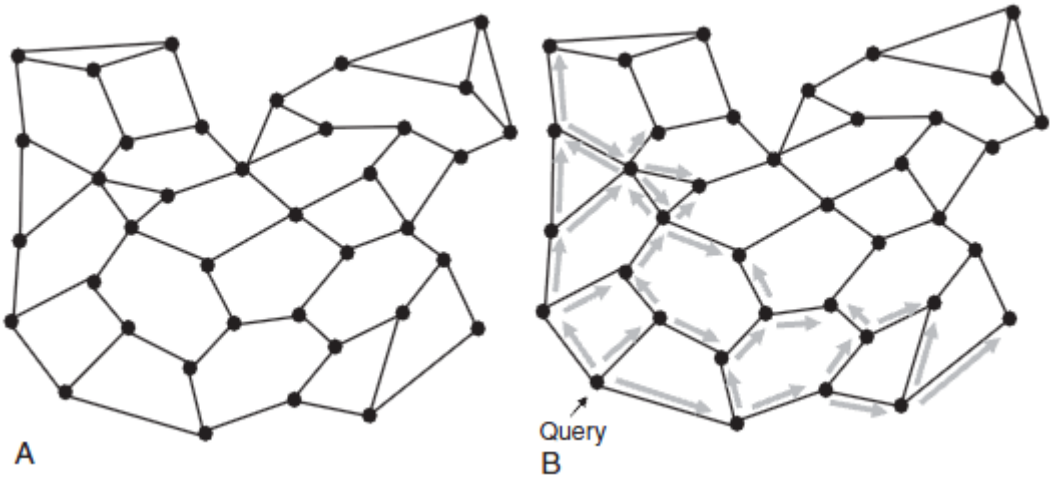


Figure 2.10 - (A) Unstructured topology showing connections between peers and (B) query flooding to four hops [17].

Simple Flooding: This simple query algorithm described before is called flooding (Figure 2.7B) and is shown in the following pseudo-code:

```
FloodForward(Query q, Source p)
        // have we seen this query before?
        if(q.id ϵ oldIdsQ) return // yes, drop it
        oldIdsQ = oldIdsQ U q.id // remember this query
        // expiration time reached?
        q.TTL = q.TTL – 1
        if q.TTL  = 0 then return // yes, drop it
        // no, forward it to remaining neighbors
        foreach(s ϵ Neighbors) if(s ≠ p) send(s,q)
```

Each peer has a list of neighbors. It initializes its list of neighbors when it joins the overlay, for example, by getting a copy of the neighbor list of the first peer in the overlay that it connects to. Over time it can add and remove peers from its neighbor list. To refresh and update its neighbor list, it can send requests to current neighbors asking for their neighbors. It can also use queries from nodes it hasn't seen before to add to its neighbor list. It removes neighbors when they are unresponsive to keep-alive messages.

Iterative deepening or expanding ring: When the query is satisfied at some peer that receives the query message, a response message is sent to the requesting peer. If the object is found quickly, the flooding mechanism nevertheless continues to propagate the query message along other paths until the TTL value expires or the query is satisfied.

Generally this creates substantial redundant messaging, which is inefficient for the network.

One way to alleviate this redundant messaging is to start the search with a small TTL value. If this succeeds, the search stops. Otherwise, the TTL value is increased by a small amount and the query is reissued. This variation of flooding is called iterative deepening or expanding ring and is particularly effective for significantly replicated objects. The pseudo-code for expanding ring is shown (Table 2.1) below for both the peer sending the search request (left) and a peer receiving and forwarding a request (right).

Sending Peer	Forwarding Peer

ExpandingRingRequest(SearchTerm st)
q.st = st
q.TTL = minTTL // first try a small TTL
// send search request to neighbors
foreach(s ϵ Neighbors) send(s,q)
// wakeup and do a retry if request fails
lastTTL = minTTL
setTimer(ERRequestRetry,100)
 	
ERRequestRetry()
// have we exceeded the permitted TTL?
// if so, then stop
if (lastTTL > maxTTL) return
// no, increase TTL and try again
lastTTL = lastTTL + 1
// send it to neighbors
foreach(s ϵ Neighbors) send(s,q)
// wakeup and do a retry if request fails
setTimer(ERRequestRetry,100* (lastTTL-minTTL))
	ExpandingRingForward(Query q, Source p)
	// is it an expansion of a previous //query?
	if(q.expansion = false) {
	// no, have we seen this query before?
	if(q.id ϵ oldIdsQ) return // yes, drop it
	}
	// remember this query
	oldIdsQ = oldIsQ U q.id
	// expiration time reached?
	q.TTL = q.TTL – 1
	if q.TTL = 0 then return // yes, drop it
	// no, forward it to remaining neighbors
	foreach(s ϵ Neighbors) if(s 6¼ p) send(s,q)

Table 2.1 - Pseudo-Code for Expanding Ring.

Objects that are sparsely placed in the overlay might not be found at all for some queries. The ability to guarantee that an object can be found if it exists in the overlay is an important feature for many applications and has motivated the structured overlay

approach discussed in the next section. An alternate strategy taken in unstructured overlays is to control the replication and placement of objects and the neighbor relationships to increase the likelihood of finding the objects.

Random walk: To avoid the message overhead of flooding, unstructured overlays can use some type of random walk. In random walk (Figure 2.8A), a single query message is sent to a randomly selected neighbor. The message has a TTL value that is decremented at each hop. If the query locates a node with the desired object, the search terminates and the object is returned. Otherwise the query fails, as determined by a timeout or an explicit failure message returned by the last peer on the walk. The initiating peer has the choice to reissue a query along another randomly chosen path. To improve the response time, several random walk queries can be issued in parallel (Figure 2.8B).

The key step in random walk is the random selection of the next hop, which avoids forwarding the query back to the node from which the query was received. The pseudo-code for Random walk follows:

```
RandomWalk(source, query, TTL)
 if (TTL > 0) {
        TTL = TTL – 1
        // select next hop at random, do not send back to source
        while((next_hop = neighbors[random()]) ≠ source)
          send(next_hop, query,TTL)
 }
```

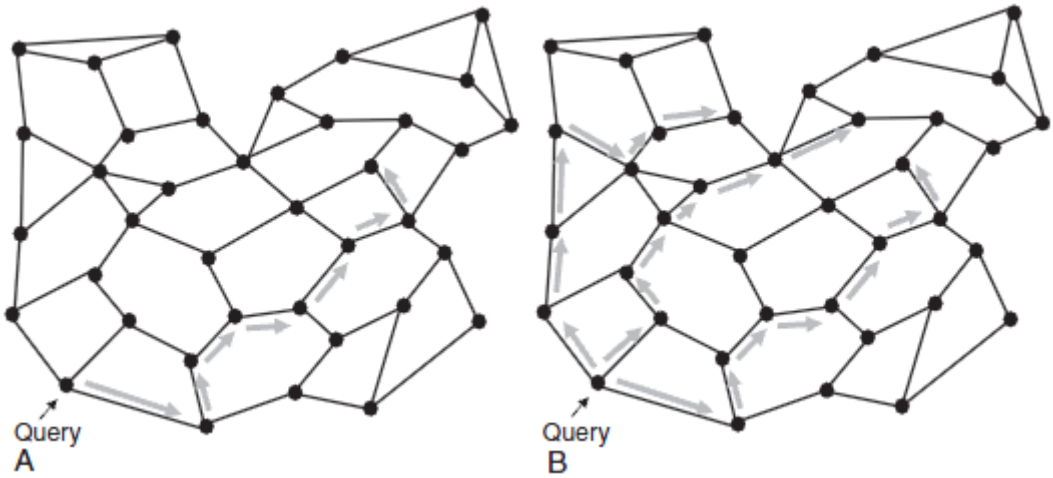


Figure 2.11 - (A) Random walk and (B) k-way parallel random walk, k = 3 [15].

2.1.4.2 Case study: Gnutella file sharing system

Gnutella [5] was the first full P2P file-sharing system and has remained one of the more popular systems to date. The earliest versions of the Gnutella protocol, through version 0.4, used an unstructured overlay with flooding for query routing. After scalability became an apparent performance issue, the most recent version of the Gnutella protocol (version 0.6) adopted a super peer architecture in which the high-capacity peers are super peers and all queries are routed, using a flooding mechanism, between super peers.

Peers are referred to as servants, which is a combination of the words server and client and super peers are referred to as ultrapeers, which are high-capacity and stable peers. Each ultrapeer maintains connections to a set of other ultrapeers. The Gnutella protocol consists of a set of basic messages (Table 2.2) and an optional set of extensions.

Each Gnutella message has the following fields:

- Message ID. A 16-byte field contain a globally unique message ID for (1) correlating response messages with the original query, (2) routing query hits from remote peers back over the original connection, and (3) detecting duplicate or misrouted messages.

- Payload type. A 1-byte field containing the message type.

- Time to Live (TTL). A 1-byte field that is decremented by each peer when it receives the message until the TTL reaches 0, at which point the message is no longer forwarded. Typically TTL is no larger than 3.

- Hops. A 1-byte field incremented by each peer receiving the message and that indicates the number of hops the message has traveled so far.

- Payload length. A 4-byte field containing the number of bytes in the remainder of the message.

- Payload. A variable-length field, the contents of which are message dependent.

After a QueryHit is received, selected files can be downloaded out of network. That is, a direct connection between the source and target peers is established to perform the data transfer.

Message Type	Meaning
Ping	Discover other hosts that are in the Gnutella network and basic information about connecting to them. If TTL=1 and Hops=0 or 1, treat the request as a direct probe of the receiving host. If TTL=2 and Hops=0, treat the request as a "crawler" ping that is collecting information about the neighbors of this host.
Pong	Reply to a ping. Provides the IP address and part number of the host and extensions supported by the peer. It may include pong responses cached from other peers. Cached entries are peers that are likely to be alive and are spread across the network; for example, by varied connections and hop-count values, these pongs are cached.
Query	Search for a file. Specifies the minimum transfer speed of the peer and the search criteria. The search criteria is text, such as a string of keywords. Search criteria of, means return an index of all files shared by the peer. A peer forwards incoming queries to all its connected peers.
QueryHit	Response to a query. A peer returns query hit responses to previously forwarded queries back along the connection from which the query was received. Contains the number of hits in the result set and, for each hit, a list of [*file index, file size, file name, and list of extensions*].
Push	Download a request for firewalled peers.
Bye	Tell the remote host that the connection is being closed.

Table 2.2 - Basic message types for Gnutella v.0.6.

Figure 2.9 shows a simplified Gnutella messaging sequence for a set of peers P1 to P6. The messaging is divided into three groups: connecting a new peer to the network, exchanging connectivity information using Ping and Pong, and query routing.

First, P1 connects to the Gnutella network by exchanging messages to peer P2. During this phase, the peers exchange information about which version of the protocol they support and what extensions they implement. The extensions are described in capability

headers. Peer P2 could reject the connection request, for example, if it does not have sufficient resources or is going offline. But in this case, peer P2 indicates, similar to HTTP messaging, a 200 OK status.

Now that peer P1 is connected to the Gnutella overlay, it may probe the network to find other neighbors to connect to. P1 issues a crawler Ping to P2 to obtain information about other neighbors. P2 responds with a Pong containing connectivity information about itself and its two neighbors P3 and P4. Later, using a flooding query, P1 issues a Query with TTL = 3 to P2. P2 responds with a QueryHit and forwards the Query to its neighbors P3 and P4 after decrementing TTL. P3 and P4 in turn return QueryHit responses to P2, and P4 forwards the Query to its neighbor P5. Since TTL = 0 at P5, the Query is not propagated any further. P5's QueryHit is returned to P4, which forwards it back along the path that it came, namely P2 and then P1.

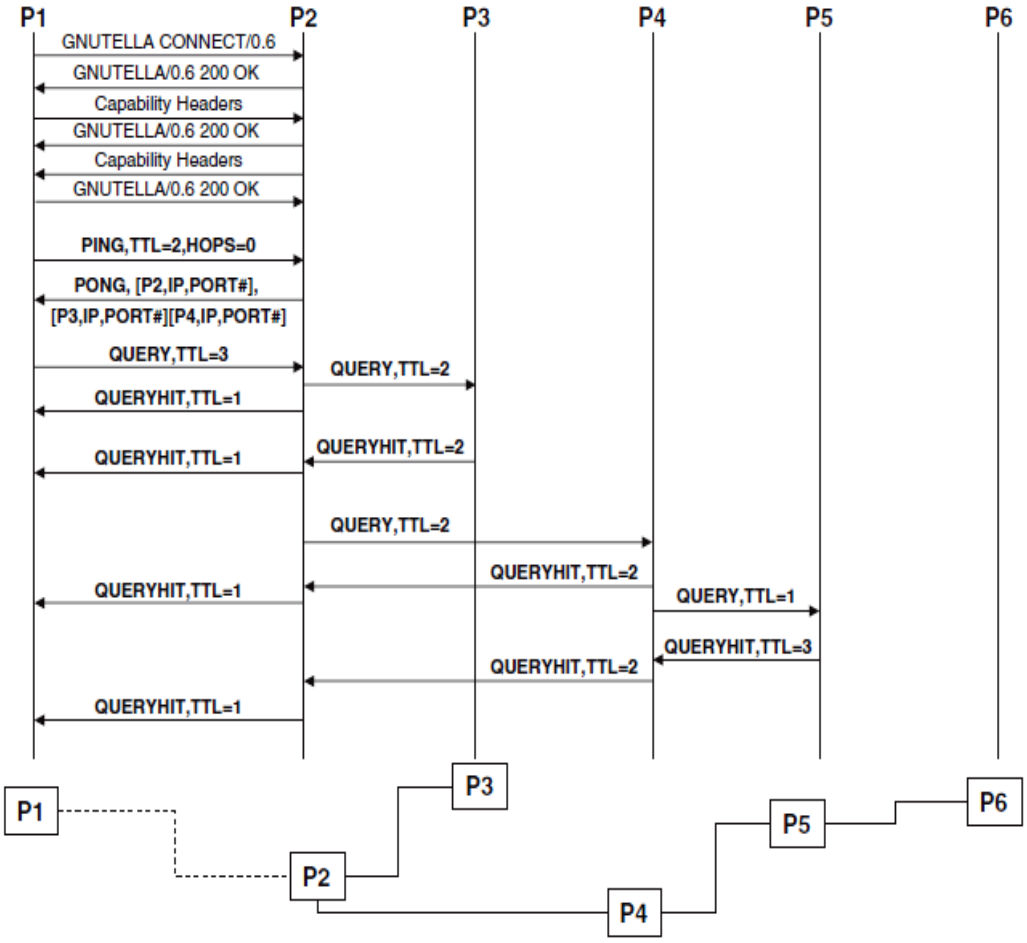


Figure 2.12 - Example Gnutella messaging using flooding-style query [17].

Current versions of Gnutella implement dynamic querying, described in previously as expanding ring search. In dynamic querying, the query is successively resent with increasing TTL values until the query is satisfied. For example, an ultrapeer initially sends the query to its leaf peers with TTL = 1. If the leaf peers do not have the file of interest or if the hit count is too low, the requesting ultrapeer sends the query to its ultrapeer neighbors, a step called a probe query. Each ultrapeer receiving the query will search its local files and its leaf peers. If the hit response is too low within a specified time period, the initiating ultrapeer enters a third step, called controlled broadcasting, in which the TTL is increased to 2 or 3 and the query is sent to one or a few ultrapeer neighbors. The larger TTL value causes the query to be sent deeper into the Gnutella network. Sending only to one or a few ultrapeer neighbors limits the amount of query traffic compared to flooding. The TTL value and number of neighbors depend on whether the results of the previous probe query indicated that the file is popular or rare.
Since these repeated query messages use the same message ID, an ultrapeer supporting dynamic querying can't discard messages coming with the same message ID as seen previously. This capability is enabled if the ultrapeer supports the probe query extension and sees a subsequent query message with a larger TTL value.

2.2.5 Structured Overlays

The earliest P2P systems used unstructured overlays that were easy to implement but had inefficient routing and an inability to locate rare objects. These problems spawned many designs for overlays with routing mechanisms that are deterministic and that can provide guarantees on the ability to locate any object stored in the overlay. A large majority of these designs used overlays with a specific routing geometry and are called structured overlays. At the same time, many unstructured overlays have incorporated some degree of routing structure in order to improve search efficiency. In addition to structured and unstructured overlay categories, there are hierarchical/hybrid[16] models, where routing and overlay maintenance is delegated to more capable peers called super peers.
Another difference between structured and unstructured overlays is that the former support key-based routing such that object identifier are mapped to the peer identifier address space and an object request is routed to the nearest peer in the peer address

space. P2P systems using key-based routing are frequently called *distributed object location and routing* (DOLR) systems. A specific type of DOLR is a Distributed Hash Table (DHT [18]) in which the identifiers are computed using a consistent hashing function and each peer is responsible for a range of the hash table. In a DHT, the object ID space and the peer ID space are the same, and each peer *p* with predecessor *q* is responsible for sid's such that $pid_q < sid \leq pid_p$. On average, the number of objects p is responsible for is |S|/|V|. The DHT has at least two operations, insert(sid, s) and s = lookup(sid), which use key-based routing to store and retrieve objects in the DHT, respectively, according to their sid. In key-based routing, the object key or sid is used as the overlay address for the DHT messages. Thus an insert message with destination equal to sid's is routed to that peer that is responsible for sid. Peers at intermediate hops forward the insert message using the sid as the destination address.

2.2.5.1 A designer space perspective

The designer of an overlay can view the design problem as selecting an operating point in a design space. Understanding the design space is important for finding solutions that might be advantageous for certain uses. Determining your application scenario can help you choose between existing design spaces and adopting one routing algorithm instead of another. Several researchers have proposed general-purpose frameworks or models. Here we'll follow the reference model proposed by Alima et al [20] shown in Figure 2.10.

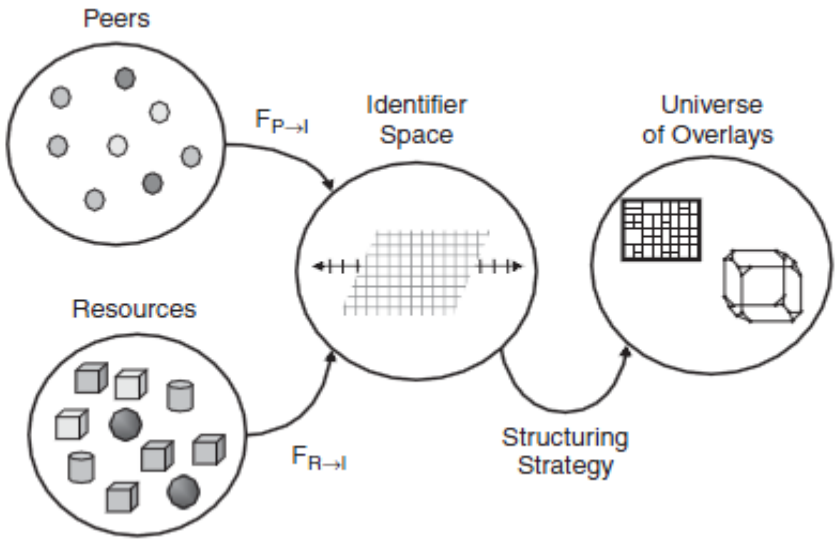


Figure 2.13 - P2P reference model for mapping peers and files into an metric address space [20].

First there is an identifier space to which all object and peer IDs are mapped. The identifier space has size, an ordering relationship, a distance relationship, and an equality test. Since it is used for addressing, it must be efficient for storage and routing. The number of peers and resources that are represented in the identifier space could be very large, so it must be scalable. It must be location independent so that moving the device doesn't affect its addressability. For message routing to converge, the identifier space should support a metric such that at any position in the address space some distance to the target can be computed. At a minimum, this distance function should be 0 when the target is the same as the current node, and it should be greater than or equal to 0 if it is a different node.

The mapping of peers to the identifier space is important for locality properties of the overlay with respect to the underlying network. The mapping of peers to an identity in the identifier space is denoted by $F(P) \rightarrow I$. The mapping should be done so that each peer has a unique identity in the identifier space. Each peer is responsible for a subset of the identifier space. In practice, the usual approach is for a peer to cover the range down to its predecessor in the identifier space. If I_1, I_2, and I_3 are the identities of three adjacent peers, I_2 is responsible for the range $I_1 < I_2 \leq I_3$.

Mapping resources to the identifier space is done to facilitate discovery of those resources and is denoted by $F(R) \rightarrow I$. A typical mapping scheme distributes resources randomly in the identifier space—for example, using a hash function. Other schemes could be used to produce locality or clusters for supporting range queries or enabling associative semantic search.

The overlay has geometry, the static structure of the graph. There is a wide range of possible geometries. Important properties guiding the selection of the geometry include the diameter of the graph, the connectedness of the graph, and the distribution of node degree across the nodes in the graph. The structuring strategy integrates the geometric model with a routing strategy and a maintenance strategy. In the section below, we give a concrete example, Chord that shows how the address space could be organized and the overlay structure that emerges.

2.2.5.2 Case study: Chord overlay

Chord [12] is a protocol and algorithm for a P2P distributed hash table. It specifies how keys are to be assigned to nodes, and how a node can discover the value for a given key by first locating the node responsible for that key.
Node keys are arranged in a circle [Figure 2.11]. The circle cannot have more than 2^m nodes. The circle can have ids/keys ranging from 0 to 2^m − 1.
IDs and keys are assigned an m-bit identifier using what is known as consistent hashing. The SHA-1 algorithm is the base hashing function for consistent hashing. The consistent hashing is integral to the probability of the robustness and performance because both keys and IDs (IP addresses) are uniformly distributed and in the same identifier space. Consistent hashing is also necessary to let nodes join and leave the network without disrupting the network.

Each node has a *successor* and a *predecessor*. The *successor* to a node or key is the next node in the identifier circle when you move clockwise. The *predecessor* of a node or key is the next node in the id circle when you move counter-clockwise. If there is a node for each possible ID, the *successor* of node 2 is node 3, and the *predecessor* of node 1 is node 0; however, normally there are holes in the sequence, so, for example, the *successor* of node 153 may be node 167 (and nodes from 154 to 166 will not exist); in this case, the *predecessor* of node 167 will be node 153. Since the *successor* (or *predecessor*) node may disappear from the network (because of failure or departure), each node records a whole segment of the circle adjacent to it, i.e. the K nodes preceding it and the K nodes following it. One *successor* and *predecessor* are kept in a list to maintain a high probability that the *successor* and *predecessor* pointers actually point to the correct nodes after possible failure or departure of the initial *successor* or *predecessor*.

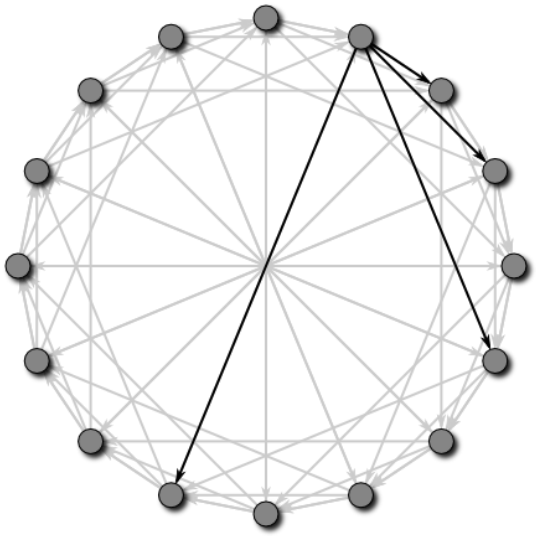


Figure 2.14 - A 16-node Chord network. The "fingers" for one of the nodes are highlighted [10].

For nodes, this identifier is a hash of the node's IP address. For keys (data content), this identifier is a hash of the key. There are many other algorithms in use by P2P, but this is a simple and common approach. A logical ring with positions numbered 0 to 2^m -1 is formed among nodes. Key k is assigned to node *successor*(k), which is the node whose identifier is equal to or follows the identifier of k. If there are N nodes and K keys, then each node is responsible for roughly K/N keys. When the (N+1)th node joins or leaves the network, responsibility for O(K/N) keys changes hands.

If each node knows only the location of its successor, you could perform linear search over the network for a particular key. This is a naive method for searching the network. Chord implements a faster search method.

A faster search will require each node to keep a *finger table* (evidenced in black color in Figure 2.11) containing up to m entries. The i-th entry of node n will contain the address of successor(n + 2i).

With such a finger table, the number of nodes that must be contacted to find a successor in an N-node network is O(log N) [12].

The routing algorithm, the next hop decision for find the desired content, key is shown below.

```
NextHop(dest)
        // if dest is in range of this peer and its successor, then
        // the successor is responsible
        if (dest ϵ (this_peer,successor]) return successor
```

```
// otherwise, search the finger table for
// highest predecessor of dest
for i = log2N downto 1
if (finger[i] ϵ (this_peer,dest)) return finger[i]
return this_peer
```

2.3 Bluetooth

Bluetooth is a low cost, low power, short-range radio technology intended to replace cable connections between cell phones, PDAs and other portable devices. It can clean up your desk considerably, making wires between your workstation, mouse, laptop computer etc. obsolete. Ericsson Mobile Communications started developing the Bluetooth system in 1994, looking for a replacement to the cables connecting cell phones and their accessories. The Bluetooth system is named after a tenth-century Danish Viking king, Harald Blåtand, who united and controlled Norway and Denmark. The first Bluetooth devices hit the market around 1999.

The Bluetooth SIG [14] is responsible for further development of the Bluetooth standard. Sony Ericsson, Intel, IBM, Toshiba, Nokia, Microsoft, 3COM, and Motorola are some of the companies involved in the SIG. The composition of the Bluetooth SIG is one of the major strengths of the Bluetooth technology. The mixture of both noticeable software and hardware suppliers participating in the further development of the Bluetooth technology ensures that Bluetooth products are made available to end users.

In 2008, at the Mobile World Congress, the Bluetooth Special Interest Group formally announced its intent to harness the speed of 802.11 with an innovative method of radio substitution. This “Alternate MAC/PHY” would allow the well known Bluetooth protocols, profiles, security, and pairing to be used in consumer devices while achieving faster throughput with momentary use of a secondary radio already present in the device. Successively on April 2009, The Bluetooth SIG announced the release of the new Bluetooth high speed specification, Version 3.0 + HS, to enable consumers to take advantage of higher data rates.

2.3.1 Architecture

The Bluetooth specification aims to allow Bluetooth devices from different manufacturers to work with one another, so it is not sufficient to specify just a radio system. Because of this, the Bluetooth specification does not only outline a radio system but a complete protocol stack to ensure that Bluetooth devices can discover each other, explore each other's services, and make use of these services.

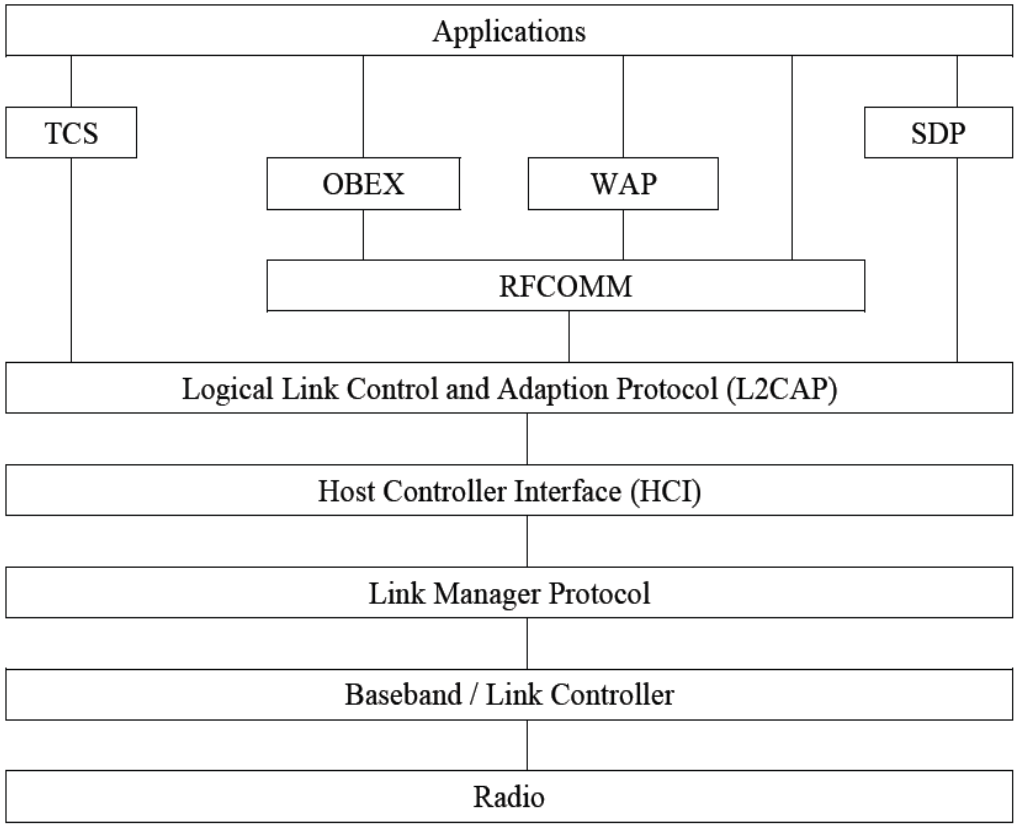


Figure 2.15 - The Bluetooth protocol stack [14].

The Bluetooth stack is made up of many layers, as shown in Figure 2.15. The HCI is usually the layer separating hardware from software and is implemented partially in software and hardware/firmware. The layers below the HCI are usually implemented in hardware and the layers above the HCI are usually implemented in software. Note that resource constrained devices such as Bluetooth headsets may have all functionality implemented in hardware/firmware. Table 2.3 gives a short description of each layer shown in figure above.

Layer	Description
Applications	Bluetooth profiles guide developers on how applications should use the protocol stack
Telephony Control System (TCS)	Provides telephony services
Service Discovery Protocol (SDP)	Used for service discovery on remote Bluetooth devices
WAP and OBEX	Provide interfaces to higher layer parts of other communications protocols
RFCOMM	Provides an RS-232 like serial interface
L2CAP	Multiplexes data from higher layers and converts between different packet sizes
HCI	Handles communication between the host and the Bluetooth module
Link manager Protocol	Controls and configures links to other Devices
Baseband and Link Controller	Controls physical links, frequency hopping and assembling packets
Radio	Modulates and demodulates data for transmission and reception on air

Table 2.3 - Description of Bluetooth protocol layers.

The interested reader will find further information about the layers of the Bluetooth stack in the Bluetooth book by Bray and Sturman [22] and in the Bluetooth specification [14].

Application developers do not need to know all the details about the layers in the Bluetooth stack. However, an understanding of how the Bluetooth radio works is of importance. The Bluetooth radio is the lowest layer of Bluetooth communication. The Industrial, Scientific and Medical (ISM) band at 2.4 GHz is used for radio communication. Note that several other technologies use this band as well. Wi-Fi technologies like IEEE 802.11b/g and kitchen technologies like microwave ovens may cause interference in this band [23].

The Bluetooth radio utilizes a signaling technique called Frequency Hopping Spread Spectrum (FHSS). The radio band is divided into 79 sub-channels. The Bluetooth radio uses one of these frequency channels at a given time. The radio jumps from channel to channel spending 625 microseconds on each channel. Hence, there are 1600 frequency hops per second. Frequency hopping is used to reduce interference caused by nearby Bluetooth devices and other devices using the same frequency band. Adaptive Frequency Hopping (AFH) is introduced in the Bluetooth 1.2 specification and is useful

if your device communicates through both Bluetooth and Wi-Fi simultaneously (e.g. a laptop computer with both Bluetooth and Wi-Fi equipment). The frequency hopping algorithm can then avoid using Bluetooth channels overlapping the Wi-Fi channel in use, hence avoiding interference between your own radio communications.

Every Bluetooth device is assigned a unique Bluetooth address, being a 48-bit hardware address equivalent to hardware addresses assigned to regular Network Interface Cards (NICs). The Bluetooth address is used not only for identification, but also for synchronizing the frequency hopping between devices and generation of keys in the Bluetooth security procedures.

2.3.2 Piconet and scatternet

A *piconet* is the usual form of a Bluetooth network and is made up of one *master* and one or more *slaves*. The device initiating a Bluetooth connection automatically becomes the master. A piconet can consist of one master and up to seven active slaves. The master device is literally the master of the piconet. Slaves may only transmit data when transmission-time is granted by the master device, also slaves may not communicate directly with each other, all communication must be directed through the master. Slaves synchronize their frequency hopping with the master using the master's clock and Bluetooth address.

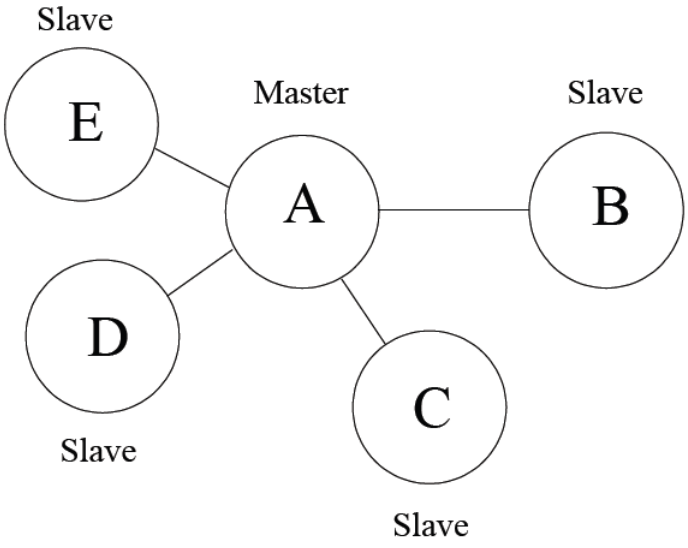


Figure 2.16 - A typical piconet.

Piconets take the form of a star network, with the master as the center node, shown in Figure 2.16. Two piconets may exist within radio range of each other. Frequency hopping is not synchronized between piconets, hence different piconets will randomly collide on the same frequency.

When connecting two piconets the result will be a scatternet. Figure 2.17 shows an example, with one intermediate node connecting the piconets. The intermediate node must time-share, meaning it must follow the frequency hopping in one piconet at the time. This reduces the amount of time slots available for data transfer between the intermediate node and a master, it will at least cut the transfer rate in half. It is also important to note that none of the Bluetooth specification defines how packets should be routed between piconets, hence implementation is vendor dependant.

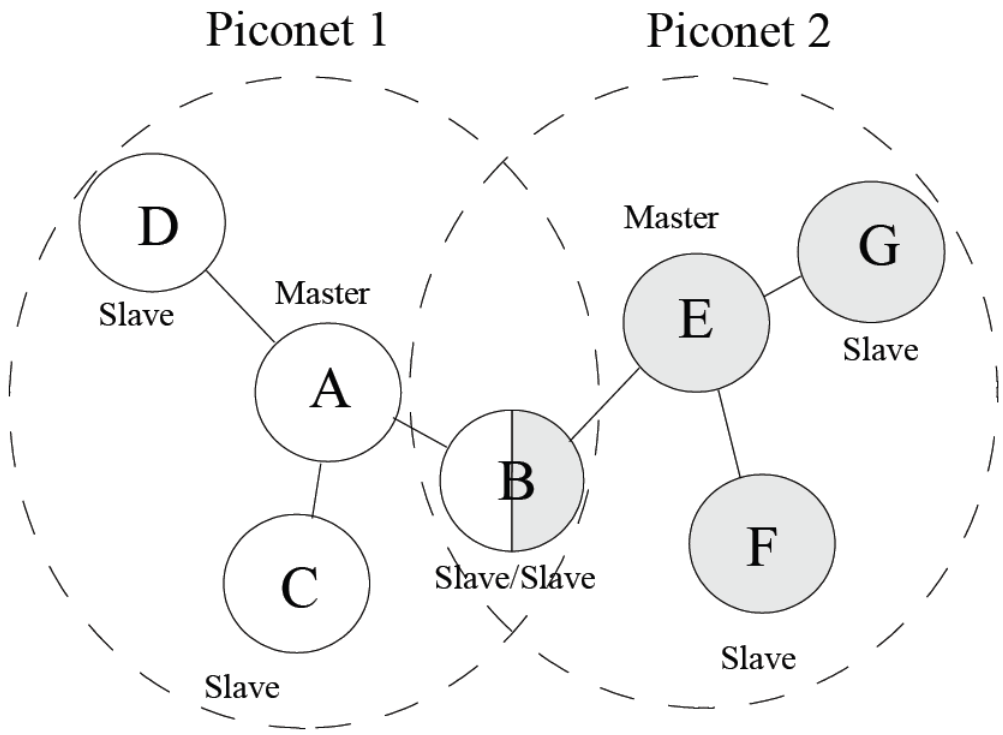


Figure 2.17 - A typical Scatternet.

Role-switching enables two devices to switch roles in a piconet. Consider the following example: You have two devices A and B. Device A connects to device B, hence, device A becomes the master of the piconet consisting of devices A and B as shown in Figure 2.18.

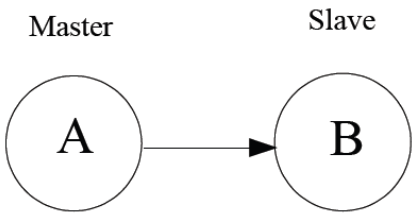


Figure 2.18 - Piconet with two nodes.

Then a device C wants to join the piconet. Device C connects to the master device, A. Since device C initiated the connection it will automatically become the master of the connection between device C and device A. We now have two masters, hence, we have two piconets. Device A is the intermediate node between these piconets, being the master for device B and the slave for device C, as seen in Figure 2.19.

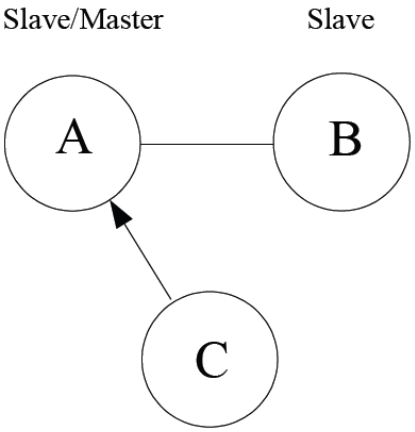


Figure 2.19 - Scatternet with three nodes.

Figure 2.20 shows that a role-switch between device A and device C will give us one piconet where A is the master and both B and C are slaves. We see that when a new device wants to be part of a piconet we actually need a role-switch to make this happen, else we get a scatternet.

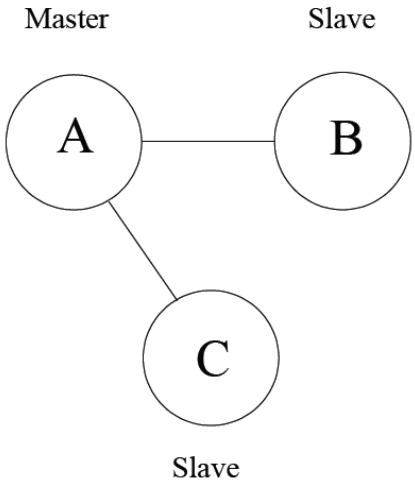


Figure 2.20 - Piconet with three nodes

2.3.3 Device inquiry and service discovery

Due to the ad-hoc nature of Bluetooth networks, remote Bluetooth devices will move in and out of range frequently. Bluetooth devices must therefore have the ability to discover nearby Bluetooth devices. When a new Bluetooth device is discovered, a service discovery may be initiated in order to determine which services the device is offering.

The Bluetooth Specification refers to the device discovery operation as inquiry. During the inquiry process the inquiring Bluetooth device will receive the Bluetooth address and clock from nearby discoverable devices. The inquiring device then has identified the other devices by their Bluetooth address and is also able to synchronize the frequency hopping with discovered devices, using their Bluetooth address and clock.

Devices make themselves discoverable by entering the inquiry scan mode. In this mode frequency hopping will be slower than usual, meaning the device will spend a longer period of time on each channel. This increases the possibility of detecting inquiring devices. Also, discoverable devices make use of an Inquiry Access Code (IAC). Two IACs exist, the General Inquiry Access Code (GIAC) and the Limited Inquiry Access Code (LIAC). The GIAC is used when a device is general discoverable, meaning it will be discoverable for an undefined period of time. The LIAC is used when a device will be discoverable for only a limited period of time. Different Bluetooth devices offer different sets of services. Hence, a Bluetooth device needs to do a service discovery on

a remote device in order to obtain information about available services. Service searches can be of a general nature by polling a device for all available services, but can also be narrowed down to find just a single service. The service discovery process uses the Service Discovery Protocol (SDP [24]). A SDP client must issue SDP requests to a SDP server to retrieve information from the server's service records.

2.4 Java 2 Micro Edition (J2ME)

This section gives an overview of the J2ME [25] technology, its architecture is described in general and some of the main components are introduced. J2ME applications are also discussed, and it is explained how they are made available to end users.

J2ME is a highly optimized Java runtime environment, aimed at the consumer and embedded devices market. This includes devices such as cellular telephones, Personal Digital Assistants (PDAs) and other small devices.

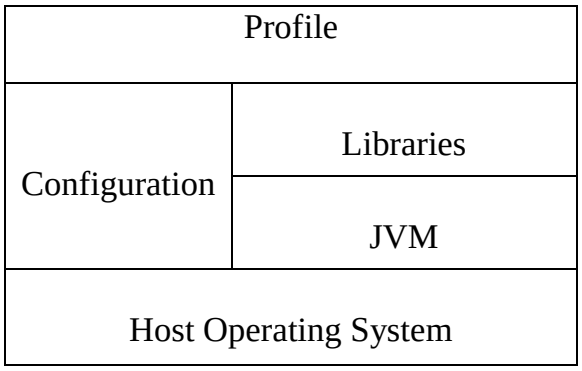


Figure 2.21 - High level view of J2ME.

Figure 2.21 shows the J2ME architecture. Java 2 Standard Edition (J2SE) developers should be familiar with Java Virtual Machines (JVMs). Profiles and configurations are introduced in J2ME and will be outlined in next section. The OS will vary on different mobile devices. Some devices run the Symbian OS, others run some other OS developed by the manufacturer. It is therefore up to the manufacturers to implement a JVM for their specific platform compliant with the JVM Specification and Java Language Specification.

2.4.1 Configurations and profiles

Mobile devices come with different form, features and functionality, but often use similar processors and have similar amounts of memory. Therefore *configurations* were created, defining groups of products based on the available processor power and memory of each device. A configuration outlines the following:

- The Java programming language features supported
- The JVM features supported
- The basic Java libraries and Application Programming Interfaces (APIs) supported

There are two standard configurations for the J2ME at this time, Connected Device Configuration (CDC [26]) and Connected Limited Device Configuration (CLDC [27]). The CDC is targeted toward powerful devices like Internet TVs and car navigation systems. The CLDC is targeted toward less powerful devices like mobile phones and PDAs. The vast majority of Java enabled mobile devices available to consumers today use CLDC. The CDC will therefore not be discussed. The interested reader can find more information about CDC on Sun Microsystems' CDC product website.
A *profile* defines a set of APIs which reside on top of a configuration and offers access to device specific capabilities. The Mobile Information Device Profile (MIDP [28]) is a profile to be used with the CLDC and provides a set of APIs for use by mobile devices. These APIs include classes for user interface, persistent storage and networking. Specifications, APIs and other J2ME related information can be found on Sun Microsystems' J2ME website.

2.4.2 Connected Limited Device Configuration (CLDC)

The CLDC is the result of a Java Community Process expert group JSR 30 consisting of a number of industrial partners. The main goal of the CLDC Specification is to

standardize a highly portable minimum-footprint Java application development platform for resource-constrained, connected devices.

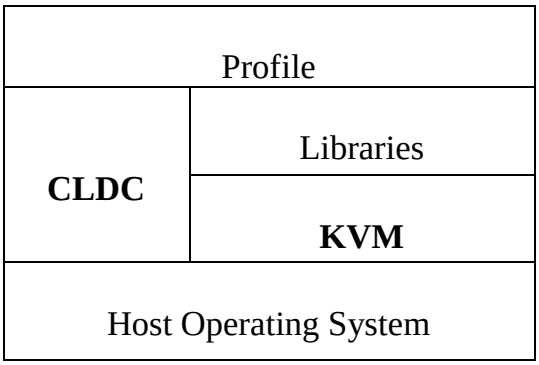


Figure 2.22 - CLDC position in the J2ME architecture.

Figure 2.22 shows that CLDC is core technology designed to be the basis for one or more profiles. CLDC defines a minimal subset of functionality from the J2SE platform. Hence, the CLDC does not define device-specific functionality in any way, but instead defines the basic Java libraries and functionality available from the Kilo Virtual Machine (KVM). The KVM got its name because it includes such a small subset of the J2SE JVM that its size can be measured in kilobytes. It is important to note that the CLDC does not define any optional features. Hence, developers are sure their applications will work on any device with a compliant CLDC implementation.

2.4.3 MIDlet

MIDP applications are called MIDlets [35]. Understanding the MIDlet life-cycle is fundamental to creating any MIDlet. The life-cycle defines the execution states of a MIDlet and the valid state transitions. Whether emulated or real, mobile devices interact with a MIDlet using their own software, which is called *Application Management Software* (AMS). The AMS is responsible for initializing, starting, pausing, resuming, and destroying a MIDlet. To facilitate this management, a MIDlet can be in one of three states which are controlled via the MIDlet class methods, and every MIDlet extends and overrides. These states are active, paused and destroyed as shown in Figure 2.23. Further, the AMS provides the runtime environment for a MIDlet. It enforces security, permissions, and provides system classes and scheduling.

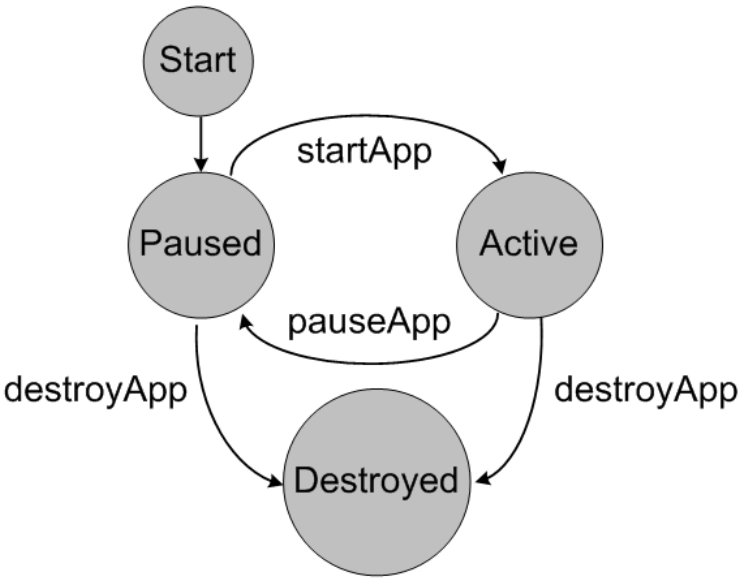


Figure 2.23 - MIDlet lifecycle [31].

Chapter 3

Related work

Establishing P2P file sharing for mobile ad hoc networks requires the construction of a search algorithm for transmitting queries and search results as well as the development of a transfer protocol for downloading files matching a query. The underlying design assumptions for most P2P systems are quite dissimilar from the routing architecture's developed for MANETs. They are characterized by low-bandwidth, high error rates of the wireless medium and low computation power for each node. Energy preservation is also an important consideration to take into account. Node mobility and the continually changing network topology pose challenges to scalability and the design of both structured and unstructured overlays in MANETs. These issues involve a variety of design dimensions from the way nodes interconnect (overlay geometry), how search criteria are specified and performed, and how the system and its information are retrieved.

The typical approach in designing such systems is to have in mind a usage pattern and the mobility scenario of participating peers in the overlay; doing so clarifies the requirements and ''hostilities'' that the routing and transfer protocol respectively might be subject to.

In this chapter we review existing solutions that influenced some design aspects of M2MShare, focusing on specific system properties rather than describing them entirely.

3.1 Seven degree of separation (7DS)

7DS [29] is a unique system and the first real attempt to deploy P2P in MANETs build in Columbia University's Internet Real-Time Lab. It allows wireless users to exchange data in a local ad-hoc disconnected network. This technique allows for successful exchange of relevant information based on two realistic assumptions - (1) that devices move in and out of the network, eventually connecting to the Internet and sending out information and bringing in new information and (2) there is a high probability that the information that is being searched for exists on a device in the near vicinity. (The

second assumption is based on the fact that there are globally popular data items that most users would be interested in.)

When the node density in a network decreases below the level necessary to sustain ad-hoc networks, communication can succeed only by leveraging node mobility and transitioning to message-based communications. In the 7DS project, two core Internet services are emulated, namely web access for information retrieval and email for delivering messages from mobile nodes to the Internet. 7DS implements and evaluates a system that leverages search, feedback and propagation limits to build a scalable system that can deliver data to and from mobile nodes. It makes data exchange in disconnected networks possible by providing an application-level set of protocol services that will enable exchange of information between peer devices. It enables dynamic information exchange by using a proxy server, a multicast query system, a search engine, and a transport entity. With these entities, 7DS performs efficient and transparent data exchange among peers in the absence of a network connection. Data exchange with the larger Internet occurs when peers enters or exits the peer network.

7DS Protocol: The system uses a very lightweight protocol involving simple XML messages for exchanging queries and responses with peers. It works seamlessly and transparently: in the absence of an Internet connection, the 7DS platform automatically queries its peers, retrieves the requests and presents the user with the data he has requested. A key component of the architecture is the proxy server, which listens to incoming HTTP requests. Based on the type of request and whether the device is connected to the Internet or not, the proxy server decides to serve the request from the cache, the Internet or through querying other 7DS system nodes via the 7DS multicast engine (Figure 3.1). The proxy server serves as the interface between the user, the Internet and other 7DS peers.

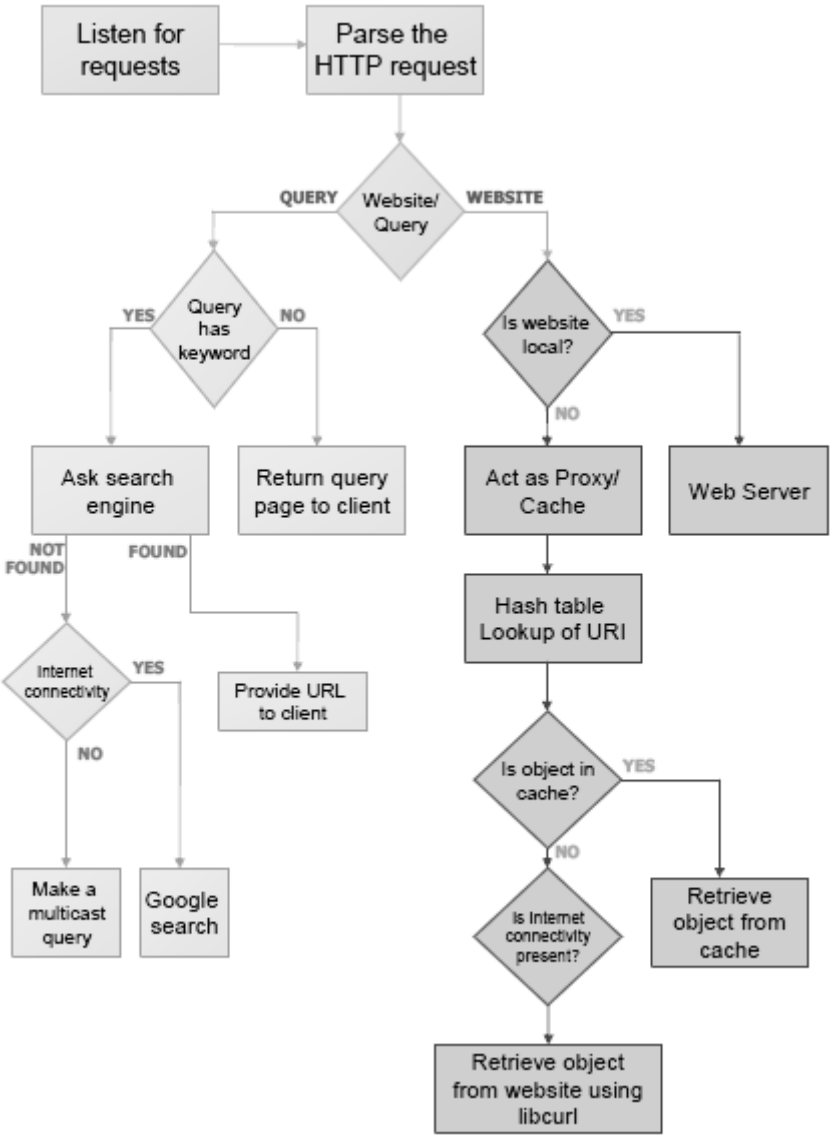


Figure 3.1 - The algorithm of the 7DS proxy server for handling HTTP requests [29].

Comparison: 7DS was developed as a platform to address the connectivity problem and node density in mobile networks, providing a store-carry-forward communication and by doing so they leverage mobility to add new information to the network. By store-carry-forward is meant that a user downloads the content elsewhere, stores it in its local storage and wherever content is requested it is forward toward the destination.

Our system too, exploits mobility to reach content in other disconnected networks by the way this is done is different. A client peer *delegates tasks* such as queries or data download to particular peers in the overlay called servants. When the servant accomplishes the task, it is ready to forward the output of that task to the client peer next time they encounter. We leverage the servant mobility and expect him to return

back the required output. This process can be further generalized allowing servants to act as client for the just-delegated task, meaning that delegations can be forwarded along a variable-hop forwarding path.
The process of choosing one peer instead of others to act as servant is entirely user-transparent and done at the protocol level. Our approach follows a store-and-forward model taken by DTNs and adopted for the mobile world where source and destination are both the same entities and servants act as DTN routers which store-accomplish the delegated task and *hopefully* forward back along the chain the resulted task output.

3.2 Optimized routing independent overlay

Optimized routing independent overlay (ORION [30]), is a special-purpose approach for P2P file sharing tailored to MANET. It comprises of an algorithm for construction and maintenance of an application-layer overlay network that enables routing of all types of messages required to operate a P2P file sharing system, i.e., queries, responses and file transmissions.

ORION designers identify the maintenance of static overlay connections as the major bottleneck for deploying a P2P file-sharing system in MANET instead, the overlay connections are set-up on demand and maintained only as long as necessary (e.g. until file transfer is completed or disconnection occurs). It provides an efficient algorithm for keyword based file search by combining application-layer query processing with techniques known from Ad Hoc on Demand Distance Vector (AODV) and Simple Broadcast Protocol for MANET. Below are shown some graphs that demonstrate how ORION overlay organization outperforms the *Off-the-Shelf* approach consisting of nodes operating the Gnutella protocol on top of TCP using AODV as underlying MANET routing protocol.

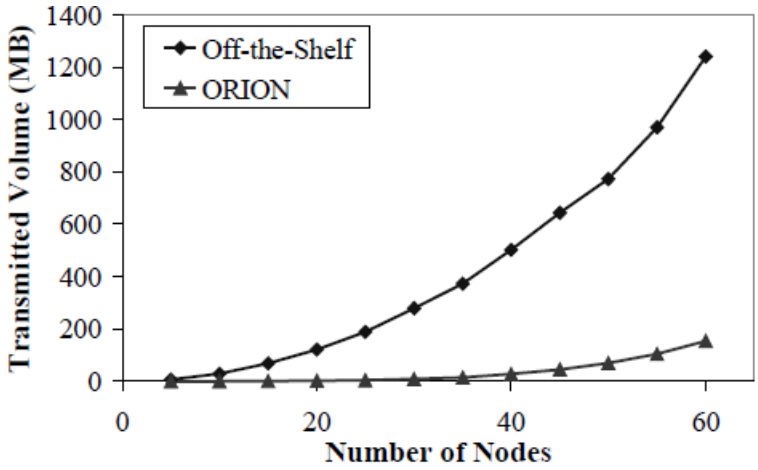


Figure 3.2 – Protocol overhead vs Size [30].

The amount of data transmitted counts the overall traffic generated in the network between nodes after a search is performed for an increasing number of nodes. As can be seen in the Figure 3.2 ORION outperforms the of-the-shelf by more than one order of magnitude for a MANET with a larger number of nodes. Therefore, we can conclude that P2P file sharing systems based on static application-layer connections are not applicable even for environments with few nodes while ORION easily scales to large scenarios.

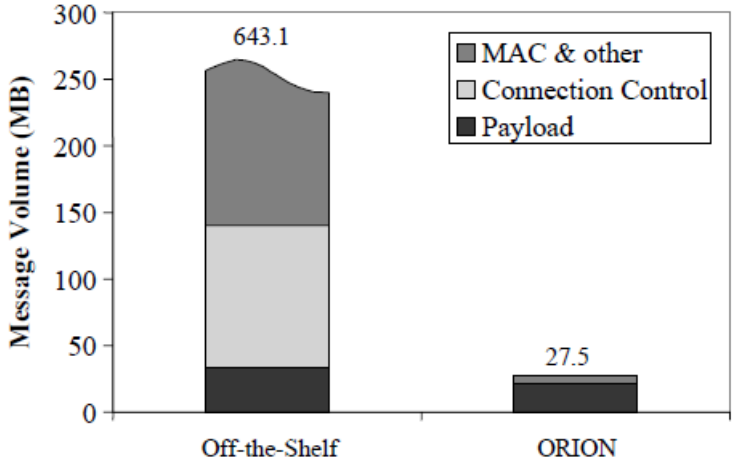


Figure 3.3 – Breakdown to message types [30].

Figure 3.3 gives insight in the composition of messages contributing to the overall message volume generated by both approaches. Message volume in the off-the-shelf approach is dominated by control traffic from MAC, routing and transport layers. Only 5% of the overall traffic is used for exchange of query and response messages (payload), which should be the main purpose of the search algorithm in a P2P

application. In contrast, the fraction of this type of traffic exceeds 79% of the total message volume for ORION. Since the amount of payload is similar in both approaches, this experiment clearly shows that the main deficiency of the off-the-shelf approach is not the inefficient query mechanism but the overhead of maintaining static overlay connections. It is arguable that the results obtained with the off-the-shelf-approach could also valid for any P2P file sharing systems using static overlay connections, not only for Gnutella.

We continue by giving some insights on the ORION building blocks and their modus operandi.

Orion Indexing: Each mobile device maintains a local repository, consisting of a set of files stored in the local file system and provides searching capabilities for all files in the repository. To this date, in the product paper there are no insights on how files are locally indexed and globally identified by the search algorithm, it is simply stated that files are associated with a unique file identifier.

Orion Routing: ORION maintains two routing tables, a response routing table and a file routing table. The response routing table, as in AODV is used to store the node from which a query message has been received as next hop on the reverse path. In this way, a node is able to return responses to the enquiring node without explicit route discovery. The file routing table stores the alternative next hops for file transfers based on the file identifier.
To illustrate the operation of the search algorithm we consider the scenario shown in Figures 3.4 to 3.7. The mobile nodes are shown in circles and the rectangles near the nodes show parts of the local repositories and the file routing table respectively. Node A issues a query matching files 1, 2, 3 and 4 (Figure 3.4). The query message is distributed by link-layer flooding. On its way through the network the query message sets up reverse paths to node A in the response routing tables of all intermediate nodes.

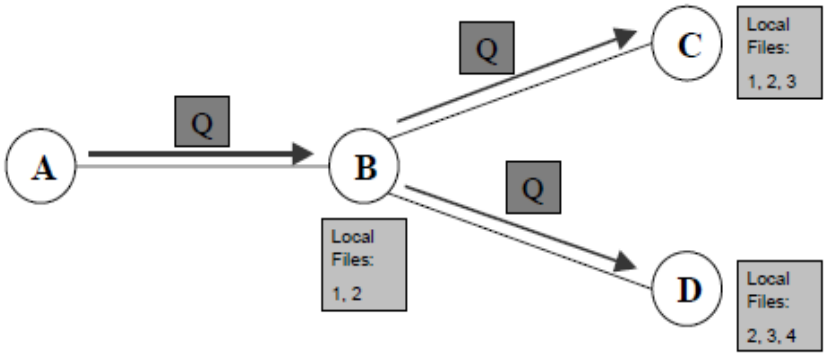


Figure 3.4 - Node A floods a QUERY message with keywords matching to files 1 to 4 [26].

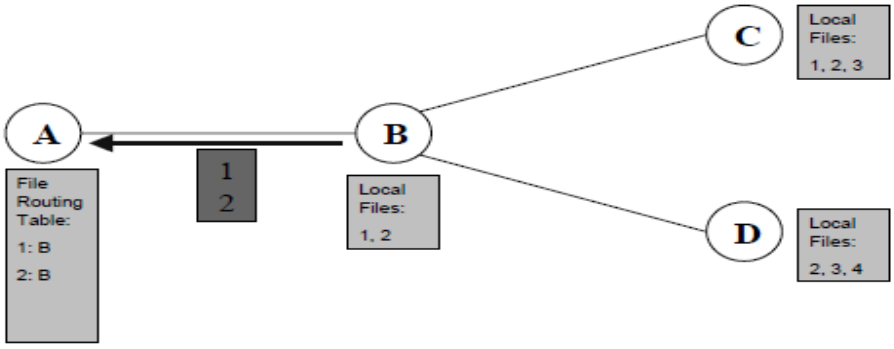


Figure 3.5 - Node B sends a RESPONSE message with identifiers of local files [26].

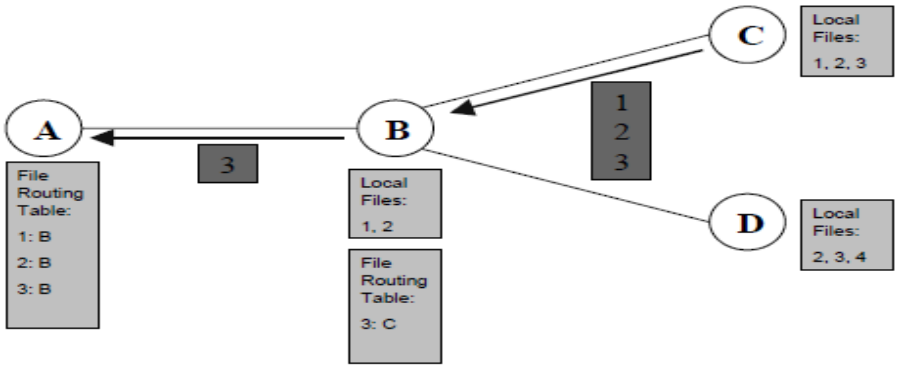


Figure 3.6 - Node C sends a RESPONSE message with identifiers of local files, Node B filters and forwards RESPONSE of C [26]

.

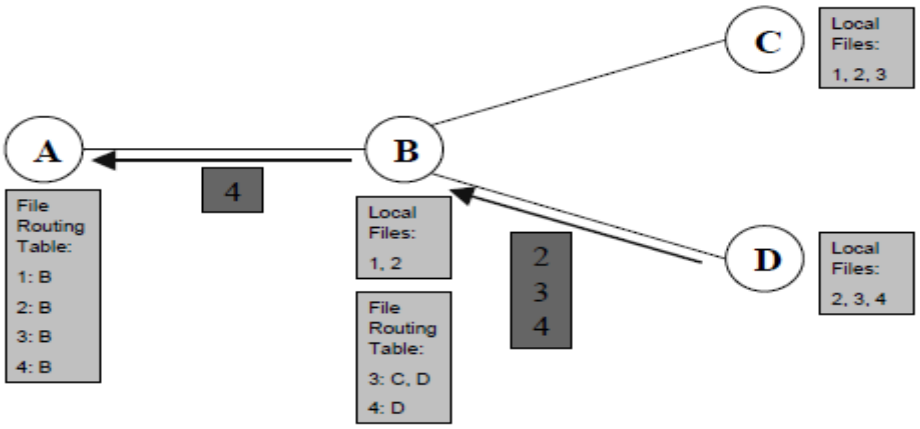


Figure 3.7 - Node D sends a RESPONSE message with identifiers of local files; Node B filters and forwards the RESPONSE message [26].

After searching the local repository, node B sends a response message containing the identifiers of files 1 and 2 to the next hop in node A's direction, i.e., directly to node A (Figure 3.5). Additionally, node C sends a response message with the identifiers of files 1, 2, and 3 in node A's direction, i.e., to node B. Before forwarding the message to node A, node B examines the contained result. File 3 is so far unknown to node B. Therefore, node C is stored as feasible next hop for this file in the file routing table. Node C is not stored as next hop for files 1 and 2, because node B stores these files itself. Therefore, node B will never request any of these two files from node C. Instead of relaying the complete response message to node A, node B sends a reduced response message to node A containing only the new file identifiers (Figure 3.6). Similar to node C, node D answers the query with the identifiers of matching local files with a response message to the next hop in the direction to node A, i.e., to node B. As before, node B stores node D's files in the file routing table and sends an own additional response. After the query phase, node A may chose one of the four matching documents for download.

Orion Transport: The ORION transfer protocol utilizes the routes given by the file and response routing tables for transmission of control- and data packets. The file routing table may store several redundant paths to copies of the same file. Due to changing network conditions, the sender of a file might change during a file transfer. Therefore, control over the transfer is kept on the receiver side. Thus, opposed to TCP the ORION transfer protocol does not maintain an end-to-end semantic. For transfer, a file is split into several blocks of equal size. Since the maximum transfer unit of the mobile

network is assumed to be equal between all neighboring nodes, the block size can be selected such that the data blocks fit into a single packet. The receiver sends a DATA_REQUEST message for one of the blocks along the path given by the file routing tables. Once the DATA_REQUEST reaches a node storing the file in the local repository, the node responds with a DATA_REPLY message, containing the requested block of the file.

Comparison: The overlay organization technique of our implemented solution was borrowed by this project - connections are established on demand and maintained as long as necessary, and test results prove its efficiency over wired P2P overlays. What ORION misses to address is node or peer density which for a file sharing application is crucial as it is strictly correlated to the population of files in the overlay. Using a DTN store-and-forward mechanism we address peer density problem and leverage node mobility by reaching data content available on other disconnected networks. We explain how this is achieved in section 4.2.

3.3 DTN-based ad-hoc networking to mobile phones

This work [31] indirectly addresses a significant problem that arises in file sharing systems deployed in MANETs which is that of content search and content 'reachability' during the search phase. Indirectly addresses the problem, because the main purpose of this work was not meant for our application scenario but instead for simple content forwarding; that where locally generated and stored content has to be transferred to another location or destination of user's choice. All this is done first trying to establish (whenever possible) ad-hoc connections with other in-reach devices, delegating them the responsibility of data transfer to destination. After the timeout expires and the file was not delivered to destination the infrastructure based communication (cellular network, e.g. UMTS) is used instead.

The main idea that motivates this approach of file delivery is that traditional ad-hoc networking is not sufficient as node density may be too low to establish a network layer end-to-end path between two communicating peers. Instead of adopting a synchronous communication between end peers an asynchronous information exchange using Delay-tolerant Networking type communication (DTN) is introduced.

On the sending side, when invoked the application checks whether new outgoing items were created by the user (e.g. new image or video captured) and, if so, offers the item for transmission to the user. The user specifies the intended recipient, by selecting phone number or email address from the address book and decide which item(s) to send along. The user can further specify which interfaces shall be tried in which order and how long the user is willing to wait for successful DTN delivery before falling back to infrastructure communications, e.g., for (multimedia) messaging.

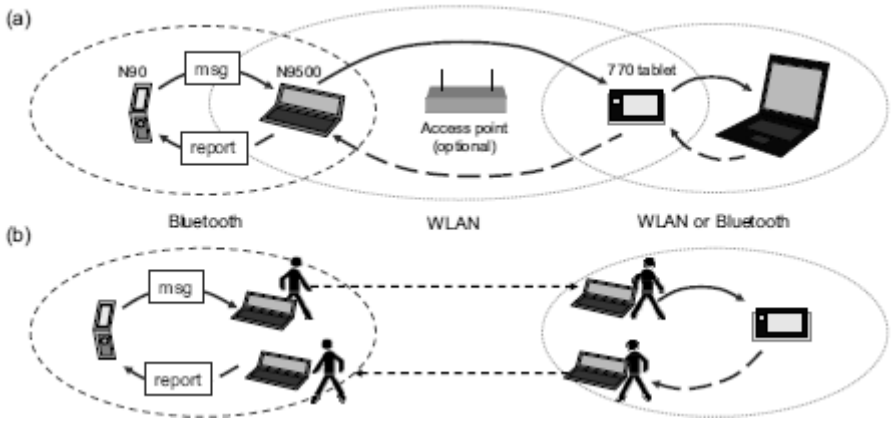


Figure 3.8 - DTN demonstrator setup [27].

Figure 3.8 shows a possible usage scenario; both types of mobile phones are equipped with digital cameras and allow recording of still and moving images. Both may serve as information source and DTN routers (store-and-forward messaging). The same applies to other devices shown in figure expect that they do not generate content. Forwarding messages is done (a) by overlapping radio ranges of different hops and (b) by means of physical carriage in a users device until another device comes in reach. In both cases, a confirmation will be returned upon successful DTN-base delivery. If success is not reported within the specified period, a fallback to operator infrastructure services is initiated.

3.4 Search

Search is an intrinsic function of most P2P overlays, and the overlay geometry and routing protocol is often designed to make search efficient. There are many types of

search mechanism, from keyword and simple pattern matching to information retrieval and content-based retrieval.

An efficient searching scheme needs to take storage, indexing, query, and retrieval into consideration. In other words, how and where objects are stored and the way the objects are indexed, queries are formulated and matched, accurate queries are matched, and fast objects are retrieved can have substantial impact on the efficiency and effectiveness of a system's data searching capability.

Most existing P2P indexing schemes can be categorized into local, centralized, distributed, or hybrid indexing types. Figure 3.9 illustrates centralized, local, and distributed types of indexing systems. Sample hybrid indexing configurations are shown in Figure 3.10.

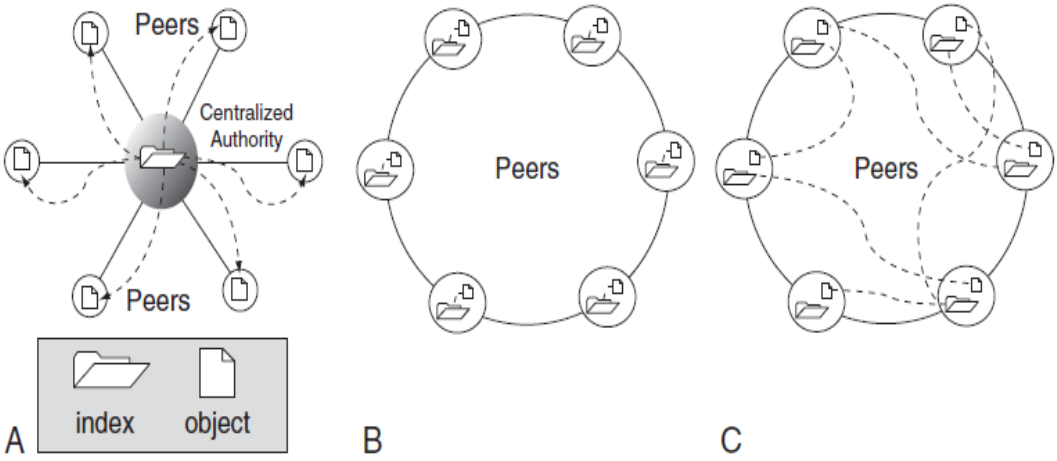


Figure 3.9 - Illustration of (A) centralized index, (B) local index, and (C) distributed indexing [17].

Centralized indexing: With a centralized index (Figure 3.9A), the index is kept on a centralized server. Object lookup is done by searching over the index on the central server to obtain the location of the target object. Napster the father of the today's P2P content distribution system, is a classical example of centralized indexing.

In Napster, peers share files stored locally on their hard drives. Text-based content description, such as the title of a song, is then generated, indexed, and stored by the Napster server. A file hash, is generated by computing the hash of the initial 300kb content. This value uniquely identifies the file in the overall system. Each peer in the Napster network uses the Napster client software to connect to the centralized Napster

server. Peers connected to the server can submit keyword-based queries for a particular audio file.
A list of matching files along with the description and location of the file is sent to the peer from the server. The peer then tries to connect to the peer with the desired file and transfers the target content in a P2P fashion.

Local indexing: In a local index-based system (Figure 3.9B), each peer keeps the directory of its own data objects locally. The early Gnutella design is a typical example of local index-based P2P system. Local object indexing can be used to support complex query search. When a peer generates a query, it conveys the query to peers in the network to locate the desired object, most often through flooding or random walk, as discussed earlier. Forwarding of query messages is stopped when the desired object is found or when the Time-to-Live (TTL) value expires.
Compared to centralized indexing schemes, localized indexing can create more network load since queries potentially have to be sent to many peers in the overlay. But since query processing is distributed across many peers in parallel, there is inherent scalability. If any peer node is faulty, it affects the objects stored locally but not the overall search mechanism of the network. Thus, localized indexing offers advantages over centralized schemes in terms of system scalability and reliability.

Distributed index: Distributed index (Figure 3.9C), on the other hand, distributes the index over all peers. Pointers to a single object may reside in multiple indices located in different peer nodes, most often in neighborhood peer nodes. Freenet [8] is one of the earliest P2P systems in this category. It uses content hash keys to identify files. Queries are forwarded to neighborhood peers based on a peer routing table until the target object is found or the TTL threshold is reached. Subsequently, Mache [32] proposed to place additional index entries on successful queries. Freenet offers only key-based indexes, but FASD [33] suggests to cache lists of weighted keywords and to use Term Frequency Inverse Document Frequency (TFIDF [34]) measures along with inverted indexes to offer keyword-based searching capability in distributed indexing systems. It is worth mentioning here that an inverted index is a popular data structure used in databases and information retrieval. It maps a search key such as a word to a list enumerating documents containing the key.

The list may also comprise the locations of occurrence within each document. Inverted indexing allows efficient implementation of Boolean, extended Boolean, proximity, relevance, and many other types of search algorithms. It is one of the most popular index mechanisms used in document retrieval. Noticeably, an inverted index is not limited to distributed indexing for P2P systems. Today, distributed indexing is one of the most popular P2P indexing schemes seen in the literature.

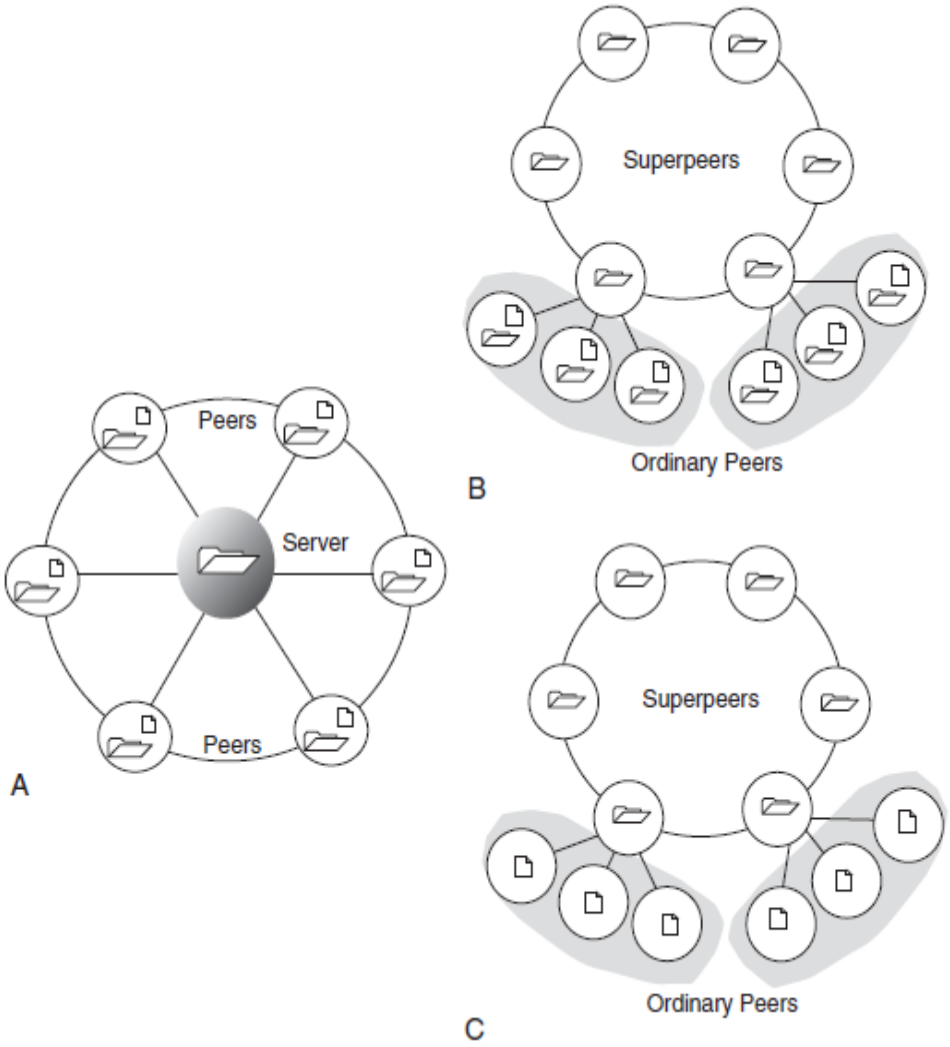


Figure 3.10 - Illustration of sample hybrid indexing configurations [17].

Hybrid index: intends to take advantage of the query efficiency of centralized indexing and the scalability of localized and distributed indexing. For instance, Figure 3.10C presents an example of hybrid indexing under a hybrid P2P overlay.

In this case, superpeers maintain the indices in a distributed manner. A query from an ordinary peer is sent to its superpeer to retrieve the location of the desired object. The

superpeer forwards the query to other superpeers if it does not find the desired object in its own index. Information about the desired object is sent back to the ordinary peer via its superpeer. Thereafter, the query issuer may directly contact the peer with the desired object.

Chapter 4

M2MShare

In previous chapters we focused on the concepts, algorithms, and abstractions of P2P networks. Here we are concerned with the design of a peer as a system. We stated the difficulties that P2P systems encounter in environments like MANET and given some insights of existing solutions and how they tackle some of the problematic. As noted, work in P2P MANET is diverse and touches on various aspects of computer science and communication engineering. In this work we had to focus our attention on some aspects rather than all of them. A protocol stack was designed, providing core functionalities that a P2P system must have. More specifically, in this chapter we review the protocol stack from a top-down perspective, discussing them in the following order:

1. *Search module.* Indexing strategy and the flexibility it provides for adding other complex query models from simple keyword to range queries (Section 4.1).

2. *DTN module.* Responsible for servant election therefore for task delegation. As said earlier in this thesis, studies in routing algorithms for challenging environments such as MANETs have a social dimension built in [36, 37] - knowing that behavior patterns exist allows better routing decisions to be made. We develop this idea of social relations existing between users operating mobile wireless devices in order to dynamically build a DTN path from source to destination. We present the reader with the decisional process involved during the module design, proceeding further by illustrating the actual implementation and finally give some concluding remarks (Section 4.2).

3. *Transport module.* Provides the task queuing mechanism and task lifecycle management. We discuss about the necessity of resource management given the fact that mobile devices are resource constraint and extend our solution to this. An important part of this section is the communication protocol that individual

tasks implement along with a detailed analysis of the data packets exchanged during communication. Finally, we introduce the file division strategy and provide some test results that demonstrate its efficiency over other division strategies (Section 4.3).

4. *Routing module*. Provides message forwarding capabilities in our overlay network and implements a controlled flooding technique alike AODV. Here we discuss about overlay organization - how overlay connections are established and maintained along with some insights on some implementation details (Section 4.4).

5. *MAC module:* Provides service discovery and message broadcast facility for peers in our network. In this section our focus is concentrated in a fundamental service called *PresenceCollector* which periodically gathers presence information about in-reach area devices. We discuss the importance of the *periodic* inquiry and introduce a theorem that can be used to compute the average transferred data quantity between two devices. Finally, we conclude our discussion by introducing the *BroadcastService* giving some implementation considerations (Section 4.5).

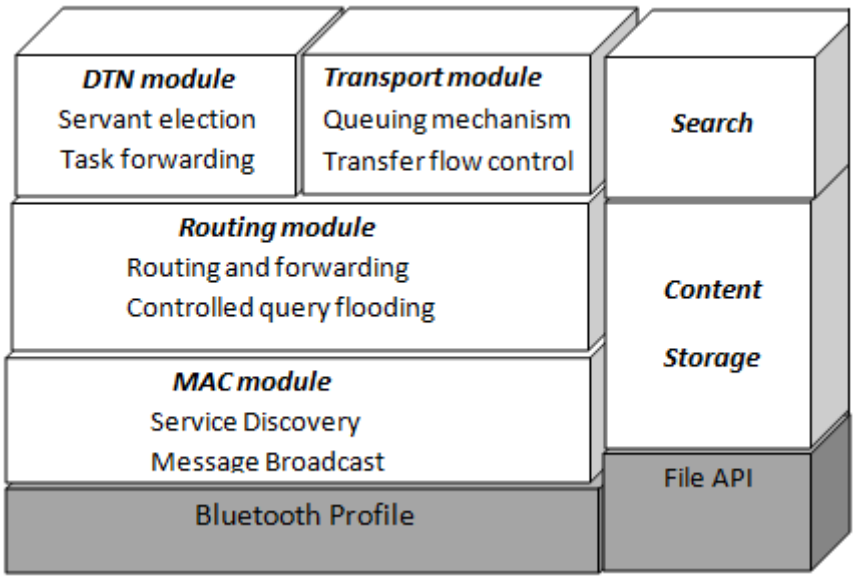


Figure 4.1 - M2MShare peer architecture. Blocks shown in gray are proprietary.

On our judgment, the added values and contributions of M2MShare are the following:

1. Demonstrating a new paradigm of use of P2P solutions that matches file sharing with mobile users, allowing them to exchange files with each other, thus fostering new applications.

2. Ease of use and autonomy. In fact, once the initial preferences are set up no further user interaction is needed.

3. Extending the maximum communication range of Bluetooth operating devices by establishing an Bluetooth overlay network capable of message routing between devices not in-reach area.

4. Combining and providing a proof of concept solution for porting DTN type communication in the mobile world.

5. Providing a protocol that dynamically establishes forward routes (delegations) along the destination path by exploiting social relations existing between users operating mobile wireless devices, whose activity is entirely user-transparent.

6. Defining the criteria and implementing a solution for single-hop task delegation to specific overlay peers (servant) extending the search reach area to other disconnected overlay networks.

4.1 Search

Each mobile device, peer in our system, maintains a local repository consisting of a set of files stored in the local file system and the user can choose which files to share with others through the software graphical interface. A file is represented by simple data structure called *FileDescriptor* which contains the file attributes such as name, hash, length and file location.

The indexing strategy follows the Local Indexing strategy (Figure 3.7B) earlier explained, where each device maintains his own index of shared files. Our search API provides two types of indexes:

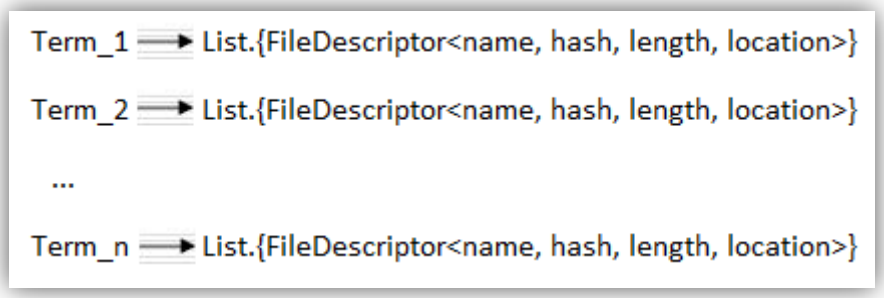


Figure 4.2 - Inverted index list. Each file descriptor is indexed under one or more indexing terms.

1. Inverted index list (Figure 4.2): This is an indexing strategy borrowed from Information Retrieval, where the index usually maintains a mapping from content, such as words or numbers, to its locations in a database file. This index in its simplicity is quite powerful as it can be extended to provide complex query modeling strategies, varying from simple Boolean queries to range and proximity queries.

   Our software implements a simple version; where each file descriptor is indexed under a finite number of terms contained in the file description text (e.g. file name). This index gives us the possibility to formulate and match keyword queries.

   An added feature in the matching algorithm, provided in the API is the *MATCHING_TRESHOLD* (*mTHresh*) ϵ *(0,1]*, which meaning is that each matching result to considered as such, should at least match *mTHresh* * *#Search_keys* query keys. The default value for this parameter is actually set to 0.5, meaning that a file needs to match at least half of the query search keywords in order to be returned as a result for that particular issued query.

2. Hash index: Each file descriptor is indexed using its file hash value. This index is provided in order to optimize operations where only a yes/no answer is need. It is the case when a query is issued with unique file identifier and only presence yes/no answer is needed in return.

4.2 Delay tolerant aspect

Traditional ad-hoc networking itself is not usually sufficient in mobile disconnected networks because peer density may be far too low to establish an end-to-end path between two communicating peers. Even if a sufficient number of mobile devices are around, device heterogeneity may inhibit interworking; radio range and interference may limit communications therefore searches for a file might take a long time to return and results might not return the requested files or even not return any files at all. For a file sharing application peer density and peer participation in the overlay is crucial as data popularity, reachability presumably will be higher.

DTNs are used in supporting communications in situations with intermittent connectivity, long or variable delay, and high error rates - characteristics that common them with the wireless mobile world, therefore it seems natural to adopt the idea of DTNs.

M2MShare achieves this by introducing an asynchronous communication mode between peers where a client peer *delegates* an unsatisfied, unaccomplished task to a *servant* peer (Figure 4.3). By delegation we mean that a task is locally encoded by client and communicated to servant where it is stored and scheduled for later execution. When servant accomplishes the task, it is ready to *forward* the output to the client peer the next time they encounter each other. By doing so we *leverage peer mobility* to reach data in other disconnected networks where they might be available.

Also, each delegated task has a TTL for which it is stored in servant's local storage and if not accomplished or forwarded until this period of time expires it is never re-scheduled. In this way, servant is prevented from scheduling in infinity the same unaccomplished task.

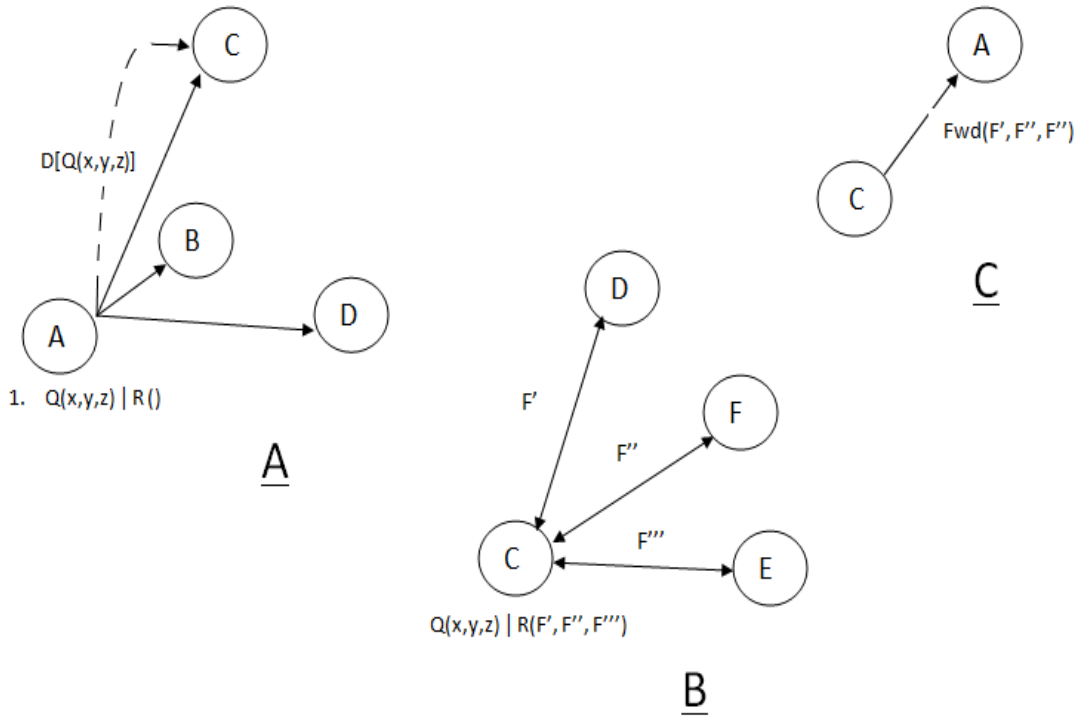


Figure 4.3 - In (A) peer A issues a query broadcast request in the established overlay network and no satisfying answer/no answer is received. Subsequently, A's protocol elects C as a servant and delegates him the unsatisfied query request. In (B) C's mobility is exploited to reach data in another disconnected network: C issues A's locally stored query broadcast request and persists the returned results (if any) in it's local storage. In (C), next time servant and client encounter each other the output of the delegated task is forwarded to client.

While in DTNs there are pre-deployed entities that store and forward data along the destination path (DTN routers), M2MShare achieves this functionality in an infrastructure-less environment - where this forward routes are established dynamically along the path to destination. In other words at each hop a client peer, which in turn might be acting as servant for another peer along the chain, dynamically chooses its servant to which delegate the task. Unlike DTNs where source and destination are different entities in our case both source of the request and destination of the data (task output) are the same entities, while servants are intermediary nodes along the chain which store-delegate-and-forward back the task output.

The picture above is a generalized view of what can be achieved by the system if delegations where allowed to be forwarded more than one hop away. In the actual implementation a servant cannot further delegate a delegated task to another peer that is, delegations are one hop only. This is achieved by queuing delegated tasks in a separate queue (see further) whose entries are not subject to delegation anymore. We chose to do

so for timing purposes only as experimenting and studying the system dynamics in the former case is more complex and requires a more detailed study.

Having introduced the overall modus operandi of the module and what it can be achieved by it, we outline some important questions and critical though that guided the module design: First of all, *which device is a good candidate for acting as a servant?* Delegating an unaccomplished task to all the peers in the established overlay is bandwidth and energy consuming therefore a criterion is needed to choose one peer instead of others. Once servant is chosen and delegated a task - *What guarantees do we have that he will return back the output of that task?* It is sound to delegate tasks to servant devices operated by mobile users whom we expect to encounter again in the future. *Are there any criteria for discriminating a good servant from a bad one?*

This section is organized as follows:

1. *Servant election strategy:* Here we introduce the actual module design and implementation, discussing in depth the questions that we stated above (Section 4.2.1).
   .
2. *Peer slot management policy:* Here we introduce the replacement and management policy applied for the servant list (Section 4.2.2).

Before proceeding further, we introduce the reader to the adopted terminology and some 'must know' configuration parameters:

- *Servant*: Elected device, to which we can delegate unaccomplished tasks.

- *Good servant*: A servant who returns back the output of the delegated task before TTL task expiry.

- *Servant list* (L): The size of the probation queue, where presence information is stored. It is a constant and its default value is set to 100.

- *Frequency threshold* (c): number of times a device need to encounter another device, before electing it as a servant for a task, $c \in [2, 4]$.

- *Active Ratio* (A_r): number of servants to which we have delegated tasks during one day of the current monitoring period (P_w).

- Expected Ratio ($E_r = L / P_w$): least number of expected servants, delegated tasks per day during this monitoring period (P_w).

- *Probation Window* (P_w): Variable that regulates the flow of probed device entries out of the servant list and imposes the goal of active entries to be expected for the next probation day. It can be seen as gained information of user routine during software operational period. Default value is set to two days, $P_w \in [2, 30]$.

4.2.1 Servant election strategy

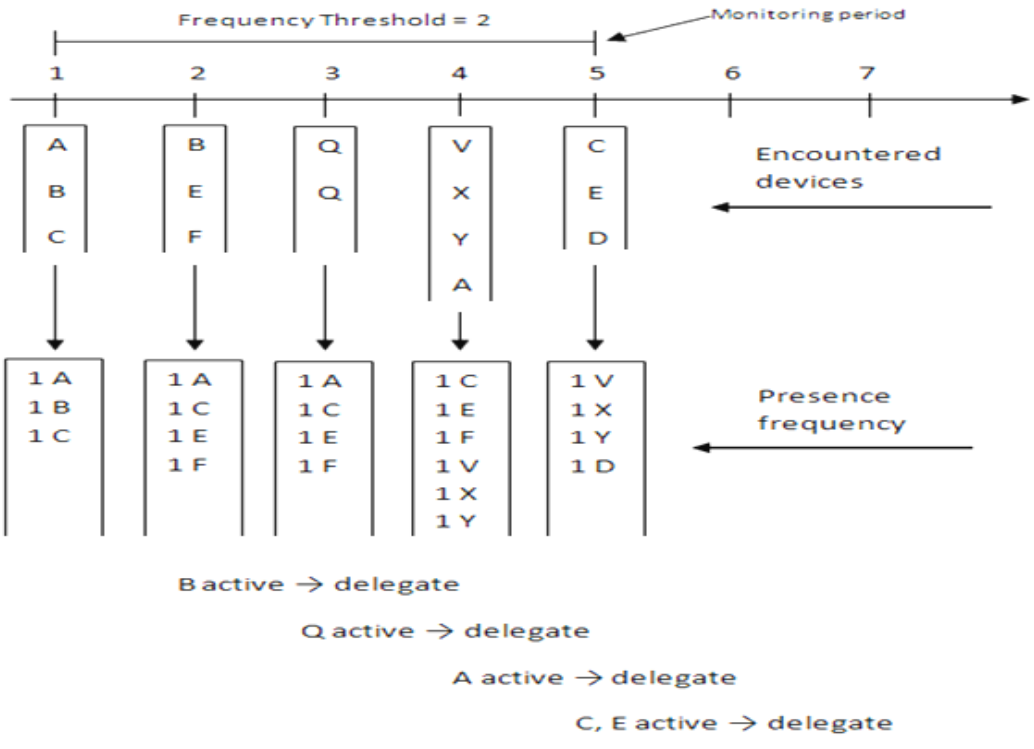


Figure 4.4 - Abstract view of DTNs functional behavior.

Mobile nodes are used by people, whose behaviors are better described by social models and earlier in this thesis we argued that the fact that behavior patterns exist

allows better routing decisions to be made. We exploit the idea of social relations between users operating mobile wireless devices and the underlying assumption behind the module design is that users have a certain routine of their own that matches others routines. E.g. user staying at his office is in communication reach with his colleagues for the duration of his work hours, user travelling by bus or train to go to work frequently encounters some people instead of others etc. A servant device is a *frequently encountered* device and the concept of frequently encountered changes in time, it adapts to the observed dynamics.

M2MShare actively collects presence information of encountered devices (Figure 4.4 rectangle), devices that are in direct reach area of communication and this job is handled by an active daemon of the system, called *PresenceCollector* (Section 4.5). Those devices that exceed the *Frequency Threshold (c)* are elected as servants to whom the system might delegate a particular unaccomplished or unsatisfied task. The delegation is considered successful if the remote device accepts the tasks based on the local applied policies.

What is a reasonable or significant value for the Frequency Threshold? By having a low *Frequency Threshold* client peer might naively elect another peer as servant, whom will not return back the output (sporadic encounter) instead, by having a high frequency the device might never give chance to a potential good servant to be elected as such. This given the reasonable assumption that established connections are opportunistic and short in time. Therefore a tradeoff has to be found.

Another important factor correlated with the servant election process is the inquiry frequency of the *PresenceCollector* service. The same reasoning made for the *Frequency Threshold* stands for the frequency of inquiry too. We only need to tune one value instead of both of them. We anticipate that the provided solution adapts the *Frequency Threshold* whereas the inquiry frequency is a configurable user value. A more detailed study of this is given in Section 4.5.

In an ideal case, where memory consumption is not a problem we would allocate new space in the servant list (L) for each device we encounter. Since memory is constrained,

this assumption is not realistic and a replacement policy needs to be applied when necessary.

What could be a sound replacement policy? Given the mobility factor, the number of newly encountered devices per day could be high and a naive replacement algorithm (e.g. FIFO) might continually register new peers replacing previous registered entries. In this way we might miss chances of electing servant's because they are continually replaced by new entries instead, we would like to monitor new entries for a *minimum period of time*, giving them a chance to serve us. Meanwhile, if the list reaches its maximum capacity new encountered devices are discarded. Also means to free allocated space of current list entries need to be provided too. Later on, we introduce the complete replacement policy.

The current implementation, at the beginning of each day imposes a 'goal' that needs to be achieved during that monitoring day. This goal is the *Expected Ratio* (E_r), the number of elected peers (servants) expected during one day of the current monitoring day.

Since, one day's activity might differ at some level from the others the systems tries to adapt the configuration parameters to the observed dynamics in order to achieve a better performance (*Expected Ratio*). Initially the ratio is computed by the default configured values, no history seen before, and at some point in time it is expected that the algorithm will reach an equilibrium where the configuration parameters (E_r, c) will be stable or will not be subject to frequent change.

To better explain how the algorithm works let's refer to the pseudo code shown below:

```
Pw_Adapt() {
    //calculate number of expected queries (ratio) per day in the actual Pw
    Er = L/Pw                    (A)
    //at the end of the probation day check if expected ratio was
      // achieved
    if(Ar ≥ Er) then
        //expectations achieved, lower probation window
        Pw = Pw  - 1     (B)
          //frequently encountering, electing servant/s: is c low?
          if(Pw < 1) then
              Pw = 2
```

```
                    c = c+1
else
        //duplicate monitoring period, lower system expectations
        Pw = Pw*2       (C)
            //frequently encountering, electing a small number of
           //servant/s: is c high?
        if(Pw > 30) then
                Pw = 30
                 c=c-1
}
```

In **(A)** the *Expected Ratio* is computed, except day 0, using the information gathered the day before; we impose a goal for today's activity based on data gathered the day before. The underlying assumption is that *user has its own routine and habits which do not change radically from day to day*.

The *Expected Ratio* is computed by performing the division between the number of servant slots in the servant list (L) and the current probation window value (P_w). There might be users operating the software that have a high number of encountered devices per day and others whom have only few of them. By dividing with P_w the algorithm can tune the parameters (P_w, c) imposing a sound goal for tomorrow's activity based on the *user's capacity of encountering other devices*.

In **(B)** the *Expected Rati*o (E_r) is achieved and we lower the monitoring period, decrementing it, imposing a higher 'goal' for next time. A monitoring period P_w=2 means that a peer is considered periodic if it is seen c times in 2 days. In this case the device is frequently encountering nodes and electing them, so everything seems going well. Frequently encountering and electing servant/s does not necessary mean that the they are returning back the output of the delegated tasks and we do not have any instrument or criteria to determine whether this is the case or not. When the monitoring period goes below its minimum value (<2) we increment the *Frequency Threshold* and probe whether this high frequency of election is induced by a probable low c.

In **(C)** we use a conservative approach, doubling the monitoring period and by doing so we lower the system expectations (ratio halved) for the next probation window. Ratio could not be achieved either because *Frequency Threshold* is too high or effectively we're encountering a small number of devices (e.g. software 'missed' all its active

sessions but one). A monitoring period P_w = 30 means that a peer is considered periodic if it is seen *c* times in 30 days. If the monitoring period exceeds its maximum value we decrement *c*, imposing a lower threshold for election and probe whether a low frequency of election is induced by a probable high *c*.

Incrementing or decrementing the *Frequency Threshold* might seem mind troubling and initially difficult to comprehend. We do so as we do not have any other feedback other than the number of active servants (*Active Ratio*) and Probation window ($P_{w)}$. This is not a fault of the DTN Module design but this is what can be achieved but the system as a whole. Consider the statements below:

- Today's good servant might not be tomorrows good servant: this might happen, because servant got lucky and found a missing file part of today's file request or query response of today's query request, but tomorrow it might not be so lucky. The established links between peers in our system are opportunistic and do not have any semantic meaning (Chapter 5).

- High frequent encountered node does not necessary mean good servant but a potential good one: frequently encountering a peer in our system is not a sound criteria to prove its worthiness for us. We might encounter a peer frequently, so might other peers. Others might have delegated him tasks and local policies of that peer might refuse our delegated task. Even more our specific file requests might never be accomplished by this peer.

- Low frequent encountered node does not necessary mean bad servant: this statement is also true as a peer we might not encounter frequently might always return back its delegated tasks.

What the algorithm does is to try and tune the configuration parameters, adapting to network dynamics day by day, and expect servant's to return back the output of that particular delegated task. There is no way to determine or assure that a servant peer is going to return back the output. We can only judge on probabilistic terms based upon the observed behavior of a particular servant peer which previously served us. This idea

is further detailed in Chapter 5 and is an interesting idea worth exploiting in future developments.

4.2.2 Peer slot management policy

The policy applied is tied to the configuration parameter P_w, which can be viewed as an observed history or intel gained during software running time. The replacement policy is periodic and scheduled at some point of each monitoring day. The criterion used to determine if a slot in the servant list must be freed is based on the actual value assumed by P_w. Each entry of the list is checked to see whether its insertion time (T_{ins}) added to the current value of P_w exceeds the current monitoring day value (T_{curr}) and if so its slot is not reserved any more. Slot is freed to give a chance to other devices as the current one had its chance and was not elected (Figure 4.5).

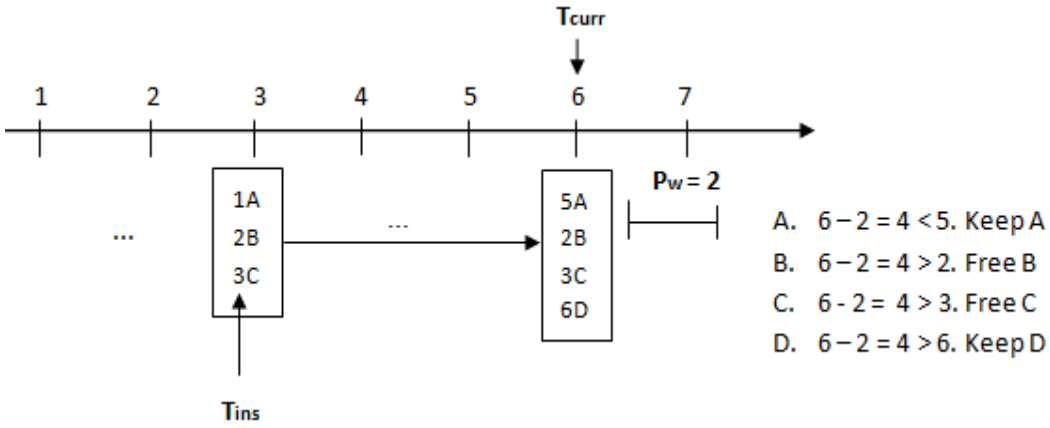


Figure 4.5 - Replacement policy demonstration scenario.

In other words the slot management algorithm decides whether previous days reserved slots are worth probing anymore based on the recently observed dynamics.

Also, earlier in this section we stated that each entry should have the opportunity to be monitored for at least a minimum time, giving it a chance to serve us. Referring to the disequation used to determine if a slot should be freed or not, the minimum time is a monitoring period of two days independently of the number of encountered devices (low or high). This since, $T_{ins} \geq 1$, where one denotes the first operational day of the software and $P_w \geq 2$ so we can deduce that $T_{curr} \geq T_{ins} + P_w \geq 3$.

Summarizing everything said the policy for servant list space management is as follows:

1. If the list is full new encountered devices are discarded.

2. Slot *x* freed when a device becomes active. We could subsequently reserve a slot for this device assigning a value of encountered times to 1. We do not follow this approach because we want to give other devices a chance. This behavior is justified by the assumption that if entry in *x* was a frequently encountered device it is going to have a chance further in time.

3. Slot freed by the periodic replacement algorithm. The periodic flow that performs the check whether to free or not a particular slot is scheduled daily and the hour is randomly chosen inside a random chosen active session. This is done not to bias the system toward devices encountered in one active session instead of others.

All this is implemented under the m2m.dtn package.

4.3 Transport module

In this section we introduce the reader to the transport module. It is the most complex building block of M2MShare as it provides facilities for task lifecycle management, from creation to termination, memory resource management along with a queuing and scheduling mechanism for tasks of our system. The arguments exposed in this section are discussed in the following order:

1. *Queuing mechanism and scheduling policies*: Here we discuss the necessity of a queuing mechanism as a pre-requisite for memory resource management. Different classes of queues and policies applied to them are introduced and described in detail. Furthermore, we give some implementation details by extending the bigger picture of the queuing and scheduling mechanism (Section 4.3.1).

2. *Task execution and task lifecycle:* In this section we describe task lifecycle from creation to termination and valid transitions by means of an state diagram. Also, here is introduced an important component of our system responsible for memory

resource management whose behavior is configurable from the graphical user interface (Section 4.3.2).

3. *Task encoding, communication protocol:* This section provides some insight on the communication protocol implemented by tasks of our system. It is shown how tasks are locally encoded, task encoding format, and remotely communicated along with auxiliary data packet exchanged during communication (Section 4.3.3).

4. *Transfer recovery:* Here is discussed an important feature that is common to the transfer protocol implemented by some tasks of our system; where relay peers, those standing between requestor and file possessor, have the means to locally repair link failures during data transfer without requestor having knowledge about this occurring (Section 4.3.4).

5. *File division strategy:* Here we expose the implemented file division strategy and provide some experimental proof of its performance compared to other division strategies (Section 4.3.5).

4.3.1 Queuing mechanism and scheduling policies

A peer device operating M2MShare might receive simultaneous incoming requests for upload and delegations from other peers, on the same time issue his own requests. Parallel requests processing means that each request being processed is handled in a separate execution flow. This modus operandi can easily cause a relatively large memory utilization that, due to the limited device resources, can cause the application to crash; In Java particularly, an *OutOfMemoryException* is thrown whenever a process executing within the JVM requires more memory than it has allocated. When such an exception is thrown, it is not recommended that the application handle the exception. Calling the garbage collector in such a case is not the correct solution because the result of doing so is unpredictable, as the garbage collector itself requires memory for execution.

The solution is to control the number of requests (tasks) being serviced at once. In order to do this, a queuing mechanism was implemented. Tasks, when created are initially queued and later processed one at a time, in a sequential manner.

Another feature that was deemed important was to distinguish between different kinds of tasks, as this would allow different policies to be applied to different types of requests. In order to manage inter and intra-queue requests, local and global policies were needed. Local policies regulate the way that tasks are handled within one specific queue, whereas global policies regulate the manner in which the various queues are processed. These considerations are implemented in the current release by providing a default processing policy.

The *QueuingCentral* is the component that implements our queuing strategy. As you can see from Figure 4.6, it has different types of queues where each of them internally queues particular classes of tasks. Below there is a brief description of each one of them.

- *dtnDownloadQueue*: are remotely issued tasks, notifying a servant is ready to forward the output of a task we previously delegated to him: DTNQueryFwd, DTNDownloadFwd.
- *dtnPending*: are remotely issued queued tasks other peers delegated to us: DTNPendingQuery, DTNPendingDownload.
- *dtnPendingUpload*: are queued finished tasks other peers delegated to us and we need to forward once we encounter them: DTNQueryFwd, DTNDownloadFwd.
- *queryQueue*: queues query requests (search key queries) configured by the user: Query_Request.
- *uploadQueue*: queues data requests tasks: Data_Request.
- *virtualFileQueue*: queues locally generated download file requests: VirtualFile.

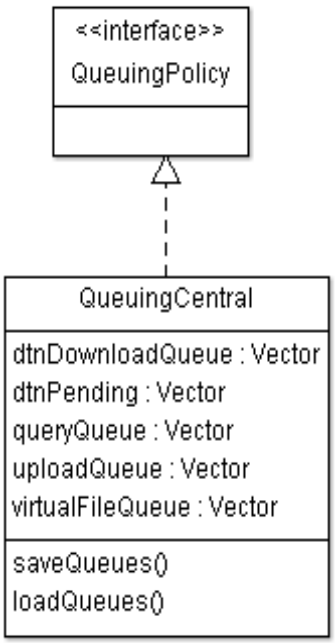


Figure 4.6 - QueuingCentral, showing different types of queues to which apply different queuing policies.

Furthermore, *QueuingPolicy* specifies storage constraints that apply to queues. Some of them are:

- *DTN_STORAGE_SIZE*: limits the local storage amount that a DTNPendingDownload might occupy when accomplished. Exceeded this amount no other type of task will be accepted. Default value is set to 4Mb.

- *MAX_PENDING_QUERIES_NUMBER*: limits the number of queued delegated queries (DTNPendingQuery) that device should locally store. Exceed this amount no other type of task we be accepted until space is freed up. Default value is set to 10.

- *UPLOAD_NUMBER_LIMIT*: limits the number of queued download requests (Data_Request) that device should handle. Exceed this amount no other type of task we be accepted until space is freed up. Default value is set to 10.

As said before, we distinguish between inter and intra-queuing policies. Local policies, those concerning how tasks are handled and fetched for execution inside a single queue are done in a FIFO fashion. To better explain intra-queuing policies we need to introduce a bigger picture of our queuing mechanism shown in Figure 4.7.

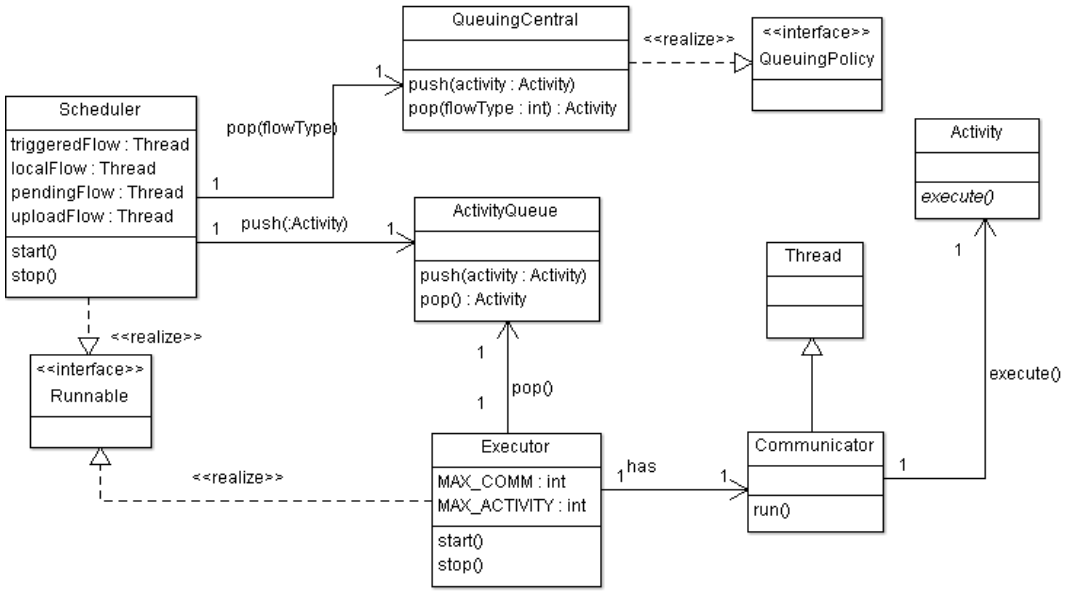


Figure 4.7 - A bigger picture of the queuing, scheduling mechanism.

The Scheduler, as its name announces is the component that schedules for execution queued tasks in the *QueuingCentral*. It is composed of four active execution flows that are:

1. *TRIGGERED_ACTIVITY_FLOW*: the queue subject to this execution flow is the *dtnPendingUpload* queue where finished delegated tasks are queued. This flow is triggered for execution each time after presence information is gathered. It checks if there is any queued task delegated to us by one of current present devices and if there is any it schedules that particular task for execution. The outcome of this execution will forward the required result to the delegator device. If there is more than one task found, the first one is picked for execution.

2. *LOCAL_ACTIVITY_FLOW*: the queues subject to this execution flow are: *dtnDownloadQueue*, *virtualFileQueue* and *localQueryQueue*. This is an active flow which alternates between this queues in the following manner showed by the pseudo code:

```
Local_Activity_Schedule() {
        If( dtnDownloadQueue is not empty) then
                Schedule all stored tasks under this queue
        Elseif (virtualFileQueue is not empty or
                localQueryQueue is not empty) then
                 pick first task from one this queues
                 schedule it for execution
                 alternate turn for next time.
}
```

   A priority is given to tasks queued under the *dtnDownloadQueue* as this are tasks that we have previously delegated to servant peers and servant/s announced that they are ready to forward the output. If no task of this kind has been notified (queued) the flow alternates between the two other queued types of tasks.

3. *PENDING_ACTIVITY_FLOW*: the queue subject to this execution flow is *dtnPendingQueue*, where delegated tasks are scheduled. If there is any queued entry the first one is picked and scheduled for execution.

4. *UPLOAD_ACTIVITY_FLOW*: the queue subject to this execution flow is the *uploadQueue*, where upload requests are queued. If there is any queued entry the first one is picked and scheduled for execution.

This intra-queuing strategy is implemented in the *QueueingCentral::pop(flowType: int)* method.

The reader might rightfully ask how are delegated tasks created and scheduled for execution. These types of tasks are created once an active entry is elected and the scheduling is handled by an execution flow inside the DTN Module. This flow alternates between *queryQueue* - unanswered queries, and *virtualFileQueue* - not completed file downloads, and respectively schedules for execution a DTNPendingQuery or a DTNPendingDownload task.

4.3.2 Task execution and lifecycle.

We earlier said that tasks are executed one by one, in a sequential manner and this is not entirely true. Tasks are scheduled for execution by a system active daemon called *Executor* (Figure 4.9). *Executor* has certain configuration parameters, configurable through the user interface, that vary its functional behavior which are:

- *MAX_COMMUNICATOR_#*: Maximum number of communicator flow's active at once in our system. *Communicator* represents the execution flow under which task is executed (Figure 4.7).

- *MAX_ PARALLEL_ACTIVITIES*: Maximum number of parallel executing activities.

We differentiate between number of communicators and activities as there is no 1-1 relation between them. A task might be accomplished by more than one *Communicator* and this is the case of a *VirtualFile* task where parallel communicators try to simultaneously download pieces of the same file from different sources in-reach area.

In Figure 4.8 is shown the complete task lifecycle from creation to termination and the valid state transitions.

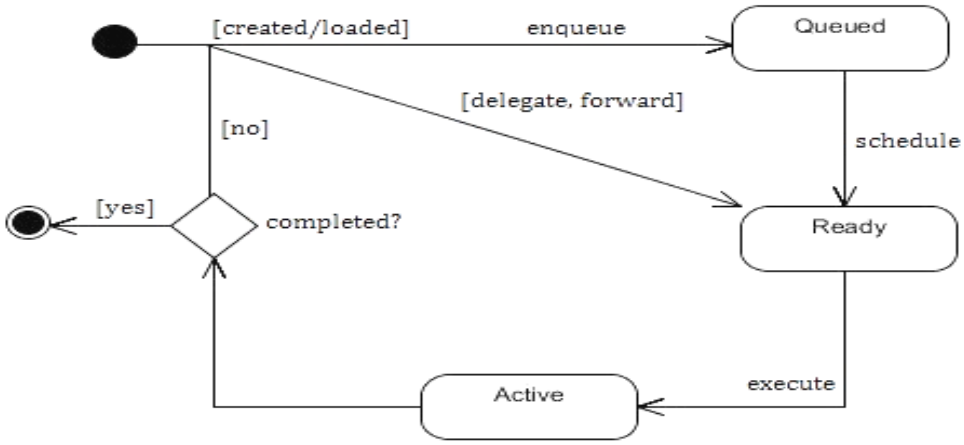


Figure 4.8 – State diagram showing task lifecycle from creation to termination and valid state transitions.

An important thing worth noting from the state diagram is the initial state. When an active session expires, the software releases all acquired resources and works in background waiting for the next scheduled active session, if there is any. Parts of the acquired resources (memory resource) are queued tasks which are removed from memory and persisted to filesystem in an xml format. When software is run the next time it automatically loads queues contents from a pre-determined storage location.

4.3.3. Task encoding, communication protocol

A task in order to be communicated remotely needs previously to be encoded in the sender side and then decoded, queued on the receiver side. In the JAVA language encoding is denoted by the term serialization and decoding by de-serialization; Serialization involves saving the current state of an object to a stream, and restoring an equivalent object from that stream. The stream functions as a container for the object. Its contents include a partial representation of the object's internal structure, including variable types, names, and values. Each serializable task in our system implements the *Serializable* interface (Figure 4.9) and provides an implementation of the serialize() and deserialize() methods.

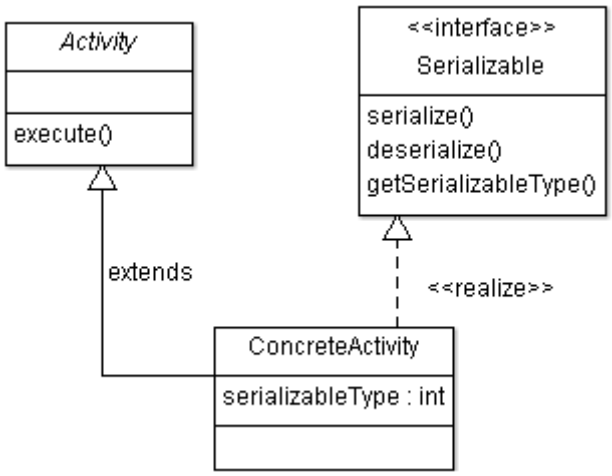


Figure 4.9 – Serializable class hierarchy.

We now introduce the data packet composition of each serializable task along with other auxiliary packets exchanged between communicating peers. For each packet type the serializable fields are given along with their size in bytes.

1. Query_Request

   - Type: identifier for this type of packet; unique value [4 bytes]
   - Type key: 0/1 value to differentiate if query is a search key or unique file id query [1 byte]
   - Criteria length: number of bytes the following fields is composed of [variable length]
   - Search criteria: unique file identifier (file hash) or search keys for this query [variable length]
   - Request id: numerical identifier for this request [8 bytes]
   - Source: uniquely identifies the device that issued this request [?max address]

   *(Source, Request Id) are used for controlled flooding as in AODV type flooding control mechanism

2. Query_Respose (auxiliary)

   - Type: identifier for this type of packet [4 bytes]

- Type key: 0/1 value to differentiate if query is a search key or unique file id query [1 byte]
- Result length: number of bytes the following field is composed of [variable length]
- Result: {0/1} if query was issued with file hash, List.{file descriptor} instead [variable length]

3. Data_Request

- Type: identifier for this type of packet [4 bytes]
- Begin byte: first of this interval to be served [4 bytes]
- End byte: last byte of this interval to be served [4 bytes]

4. Data_Response (auxiliary)

- Type: identifier for this type of packet [4bytes]
- Accepted: if data request has been accepted by remote device or not (policy appliance) [1 byte]
- Content: the requested byte content [endByte-beginByte+1]

5. DTNPendingDownload

- Type: identifier for this type of packet [4 bytes]
- Requestor: uniquely identifies the device that delegated this task [12 bytes]
- File identifier: unique file identifier [16 bytes]
- Download map: download intervals delegated (file division strategy) [max 1024 bytes]
- TTL: expiry time for this task to be completed, depends upon probation window value [1byte]

*(TTL) value is set by delegator to his current monitoring period value.

6. DTNPendingQuery

- Type: identifier for this type of packet [4 bytes]
- Requestor: device that delegated this task to me [12 bytes]
- Search keys: search keys for the delegated query [variable length]
- TTL: expiry time for this task to be completed, depends upon probation window value [1 byte]

*(TTL) value is set by delegator to his current monitoring period value.

7. DTNDownloadFwd

- Type: identifier for this type of packet [4 byte]
- Requestor: device to whom we need to forward the output of this task, previously he delegated us [12 bytes]
- Download map: output of the task to forward [1024 bytes]

*Not all requested data might have been download, we communicate download intervals to requestor first, he then communicates which one to transfer.

8. DTNQueryFwd

- Type: identifier for this type of packet [4 bytes]
- Requestor: device to whom forward the output of this task, previously he delegated us [12 bytes]
- Criteria length: number of bytes the following field is composed of [variable length]
- Search criteria: search keys for this query [variable length]
- Result length: number of bytes the following field is composed of [variable length]
- Result: query response, a list of file descriptors containing file name, identifier and length [variable length]

Below are shown some sequence diagrams demonstrating the execution and communication protocol implemented by tasks of our system. For sake of simplicity, the diagrams do not show all alternative execution scenarios.

1. Query_Request: Automatically created after user enters through user interface search keywords matching a particular file. Its execution initiates a Query_Request broadcast into the network. We previously discussed the mechanism of how this is achieved; from controlled message flooding to response gathering process.

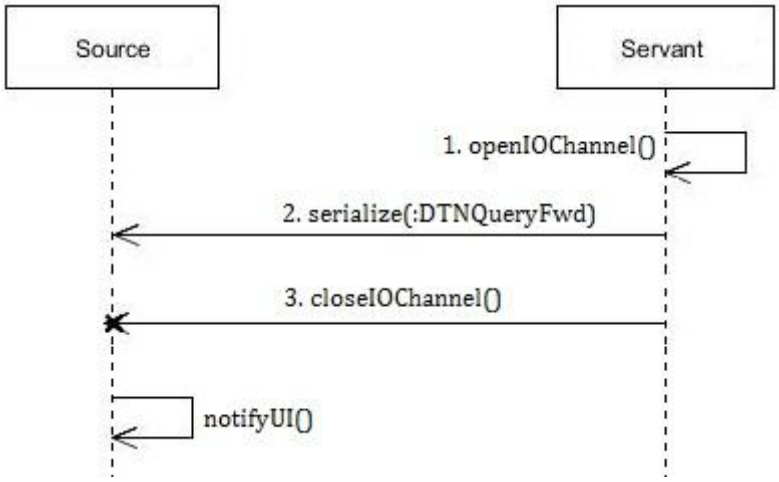


Figure 4.10 – Communication protocol by DTNQueryFwd task execution.

2. DTNQueryFwd: A task of this type might have been (i) locally issued, a remote device delegated us a DTNPendingQuery; we accomplished the task and are ready to forward the output, (ii) remotely issued, we previously delegated a DTNPendingQuery to another device and this device is notifying us that is ready to forward the output. The communication protocol implemented in each case is shown by the sequence diagram below (Figure 4.10) where case (i) is the servant and case (ii) is the source.

   Preconditions:

   1. Servant device to which we previously issued a DTNPendingQuery, issued a DTNQueryFwd notifying that it

is ready to forward the delegated task outcome, consisting of file descriptors matching the search keywords.

2. Task on the source side is scheduled and ready for execution

Step:

1. Servant device opens communication stream with source device in reach-area.
2. Servant device serializes DTNQueryFwd request which is queued on source side.

Outcome:

1. User of source device is presented with the task outcome and might issue a download request for a particular file in the list. Issuing a download request queues a VirtualFile download for the selected file.

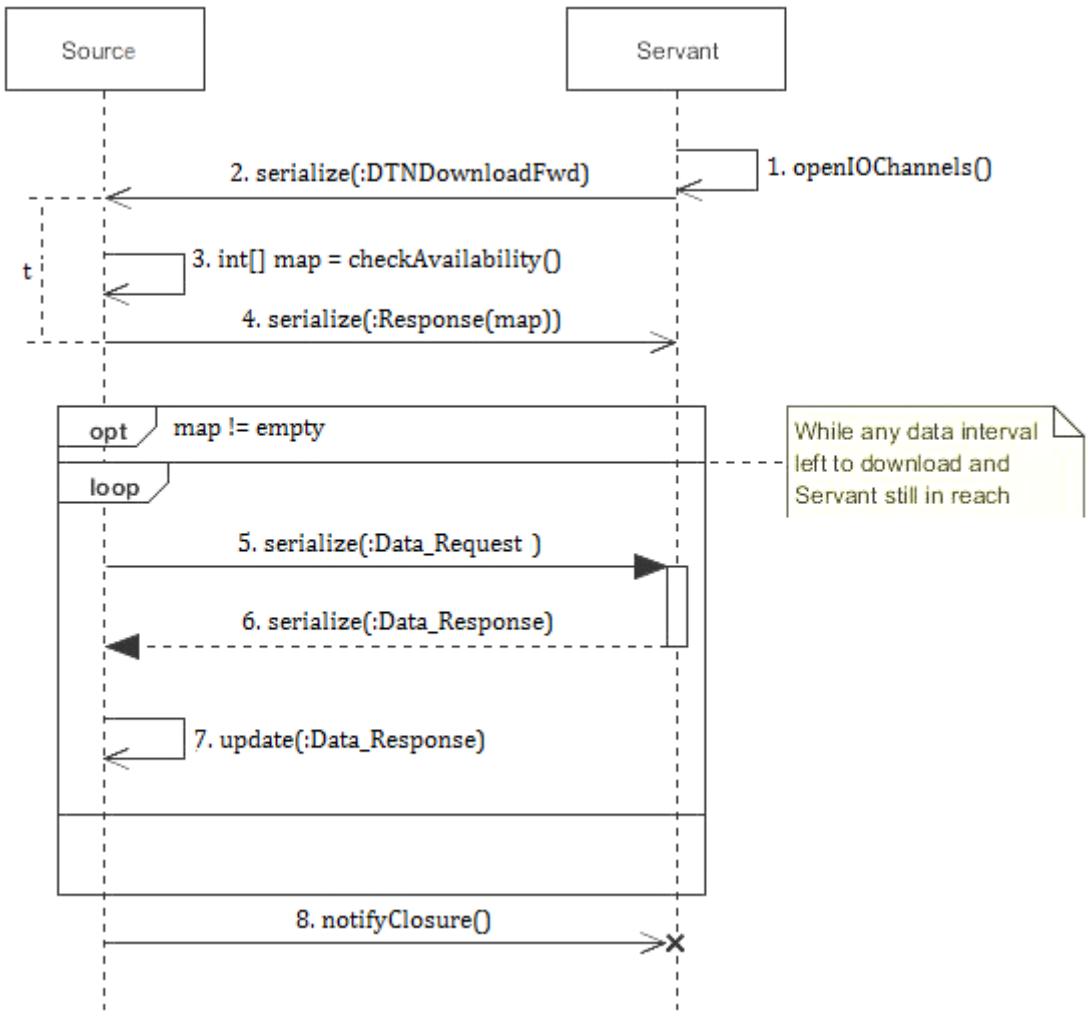


Figure 4.11 - Communication protocol implemented by DTNDownloadFwd task.

3. DTNDownloadFwd: A task of this type might have been (i) locally issued, a remote device delegated us a DTNPendingDownload; we accomplished the task and are ready to forward the output, (ii) remotely issued, we previously delegated a DTNPendingDownload to another device and this device is notifying us that is ready to forward the output. The communication protocol implemented in each case is shown by the sequence diagram below (Figure 4.11) where case (i) is the servant and case (ii) is the source.

Preconditions:

1. Servant device, to which we previously delegated a DTNPendingDownload, issued a DTNDownloadFwd notifying that it is ready to forward the delegated task outcome.

2. Task on the source side is scheduled and ready for execution while servant is still in reach.

Step:

1. Servant device opens communication stream with source device in reach-area.
2. Servant device serializes DTNDownloadFwd request which is queued on source side.
3. Source device retrieves the VirtualFile associated with the current file request and checks whether there is any data piece still missing that can be requested by servant.
4. Source has calculated missing intervals and responds back to servant.
5. Step 5 – 7: Devices begin data exchange till servant is in reach-area and all required data are finished downloading. After each download, source device updates the IntervalMap (see further) associated to the VirtualFile and persists fetched data.

Outcome:

1. Locally stored forwarded data at servant's local storage are deleted.
2. Source downloaded, persisted missing required data interval's and updated VirtualFile status (see further). If there are no intervals left to download VirtualFile::build() method is called.

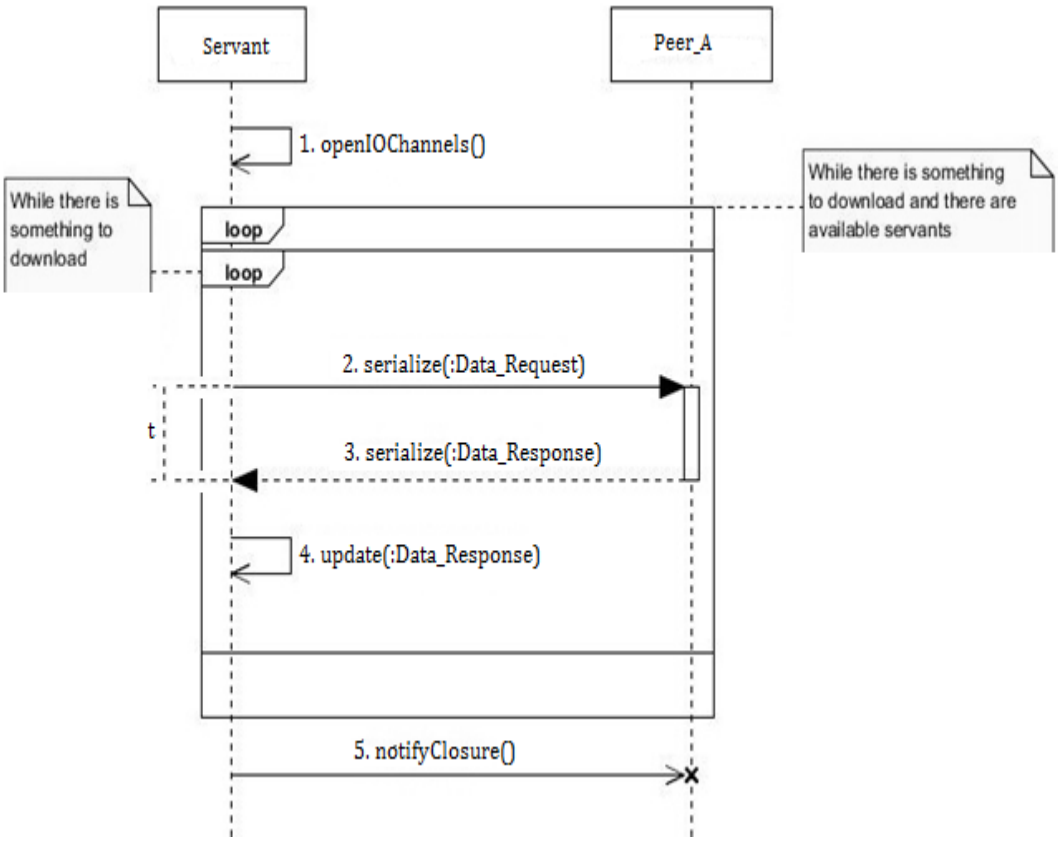


Figure 4.12 - Communication protocol implements by DTNPendingDownload task.

4. DTNPendingDownload: A task of this type might have been (i) locally created by scheduler's *PENDING_EXECUTION_FLOW* and scheduled for execution or (ii) delegated by another peer device. If task is locally created its execution consists only in serializing the data packet, communicating it to the remote source. Here we show the protocol implemented in case (ii) (Figure 4.12).

Preconditions:

1. A DTNPendingDownload task, previously delegated by another peer device (not shown), is ready for execution in the servant's device.
2. As first, a file hash query (yes/no query) is broadcasted into the network for available file sources.
3. Protocol is initiated with peer/s (Peer_A) in order to retrieve data content.

Step:

1. Servant device opens communication stream with Peer_A device which is in direct posses of data file or acts as a relay point between servant and file possessor.
2. Step 2 - 4: Devices begin data exchange till file source is in reach-area and all required data are finished downloading.

Outcome:

1. Servant stores locally the downloaded data interval's and queues a DTNDownloadFwd task. If no data was downloaded the initial task is re-queued for later execution.

5. DTNPendingQuery: Conditions of creation and scheduling are the same as those for DTNPendingDownload. The communication protocol implemented is simple and straight forward. When task is locally created its execution consists only on serializing the contents of the task remotely. If task was delegated by another device the execution consist of the following steps:

1. A Query_Request broadcast is issued with the delegated search keywords.
2. [alt] If any response was received they are gathered and a DTNQueryFwd is queued for later execution.
3. [alt] If no response was received the task is re-queued for later execution.

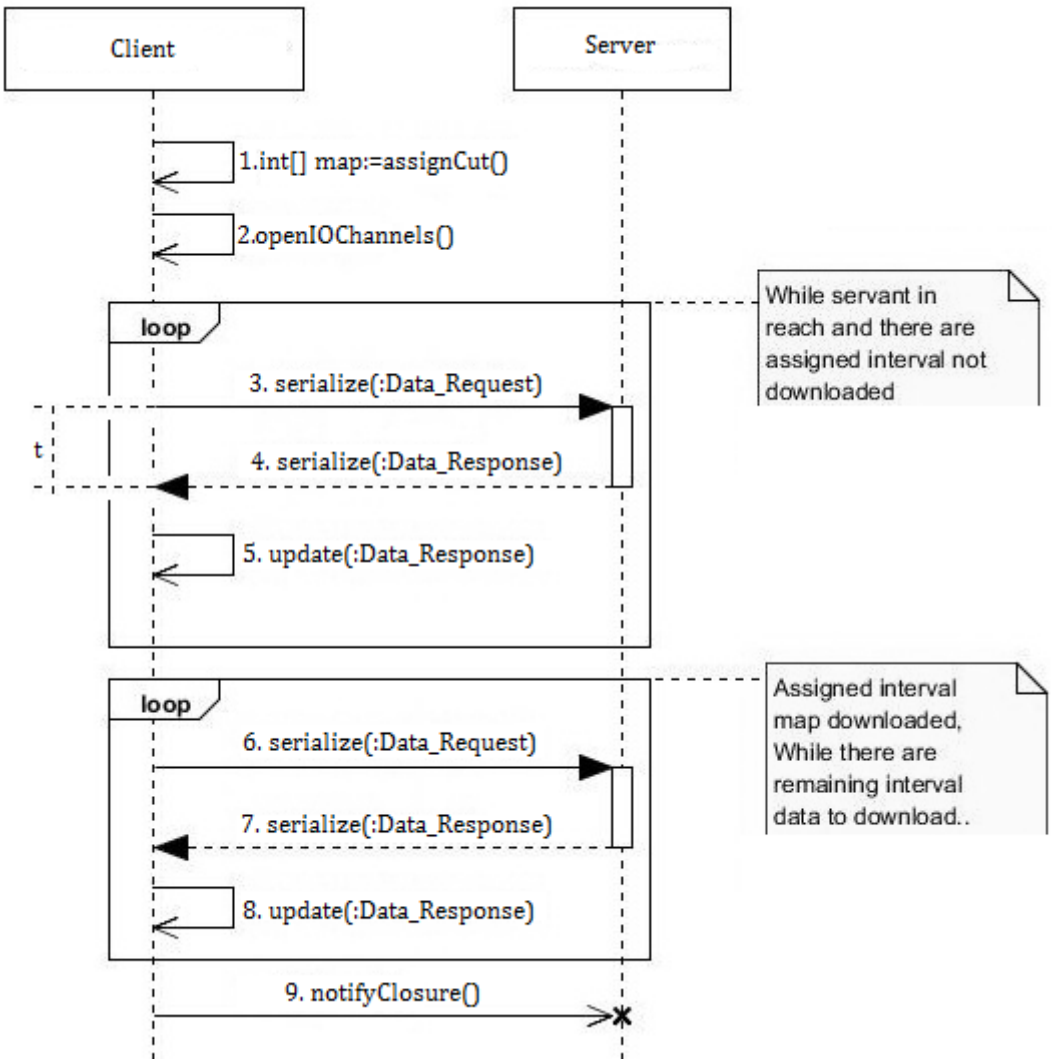


Figure 4.13 - Communication protocol implemented by VirtualFile task.

6. VirtualFile: This type of task is locally created after user issues a specific file download request. An IntervalMap (see further) with initial download interval (0, fileLength) is created, representing data intervals to be downloaded.

Preconditions:

1. A file hash query (yes/no response) is broadcasted into the network for available file sources. For simplicity only one source (Server) is represented.

Step:

1. The *Communicator,* executing the task at the client side, retrieves the assigned data piece/s (interval/s) that needs to download.
2. Client device opens communication stream with Server device which is in direct posses of data file or acts as a relay point between servant and file possessor.
3. Step 3 – 5: Devices begin data exchange for assigned interval/s till file source is in reach-area and all required data interval/s have finished downloading.
6. Step 6 – 8: Assigned data interval/s have all finished downloading, begin data exchange for other left interval/s in the IntervalMap.
9. Close connection.

Outcome:

1. Device peer downloaded, persisted missing data interval's and updated VirtualFile status (see further). If there are no intervals left to download VirtualFile::build() method is called otherwise is re-queued for later execution.

In some of the diagrams presented above is shown a signature (t), which delimits the time from first request issuing (client) and first response issued (server). Since first issued requests are first queued and later processed we do not want the issuer to hang waiting for the response.

To deal with this, the concept of timeouts was introduced. Each issued request is given a timeout. If task (encoded request) being processed receives no response for a time longer than the timeout (t), the request is terminated and rescheduled and the connection will be marked as terminated.

4.3.4 Transfer Recovery

Recall that the file routing table may store several redundant paths to copies of the same file. Due to changing network conditions, the sender of a file might change during a file

transfer. Therefore, it is essential to keep the complete control over the transfer on the receiver side. Thus, opposed to TCP the M2MShare transfer protocol does not maintain an end-to-end semantic. The receiver sends a DATA_REQUEST message for one of the pieces contained in the download map along the path given by the file routing tables. Once the DATA_REQUEST reaches a peer storing the file in the local repository, the peer responds with a DATA_REPLY message, containing the requested piece of the file.

Selecting an alternative route provides an efficient mechanism to locally resolve link failures. Consider again the example in Figure 3.4. Suppose the link between peers B and C fails during the file transfer, e.g., because C moves out of B's transmission range. As soon as B recognizes the link failure, it deletes B in its routing tables and forwards subsequent DATA_REQUEST messages to the now best-suited next hop to a peer possessing file 3, i.e. D. Thus, the link failure can locally be resolved by B without involving others, the requestor peer A.

4.3.5 File division strategy

The majority of Bluetooth file transfer applications in the market follow a client-server paradigm where devices pair with each other for all the duration of data transfer. If a disconnection takes place (e.g. because devices are not in-reach area anymore) file transfer should restart from the beginning and already downloaded data are of no use. Pairing for all the duration of file exchange is desirable if possible, but taking in consideration the mobility of users and that established connections are opportunistic and short in time, the chances of this happening reduce drastically. As said earlier we would like that software be entirely user transparent and as such a file download should run automatically whenever possible. For instance, user might enter a Cafe Shop - drink his café in say 2 minutes and during this time software is actively running and a file exchange might have started. Also, part of user daily routine might be taking the subway meanwhile the software is running and operative. A borderline situation might be that of user walking in the street where mobility is continuous and a file transfer might initiate even for an instance of time.

Some P2P software deployed on the wired internet divide the file into data chunks (Gnutella, BitTorrent), which are the atomic transferable parts. This is a good starting point but taking into consideration the possibility of delegation of data pieces and the usage patterns mentioned before we require a more flexible file division strategy. M2MShare provides a new file division strategy where a file can be downloaded in pieces and a piece size varies. The file is seen as map of non overlapping intervals of variable length that need to be downloaded (Figure 4.14).

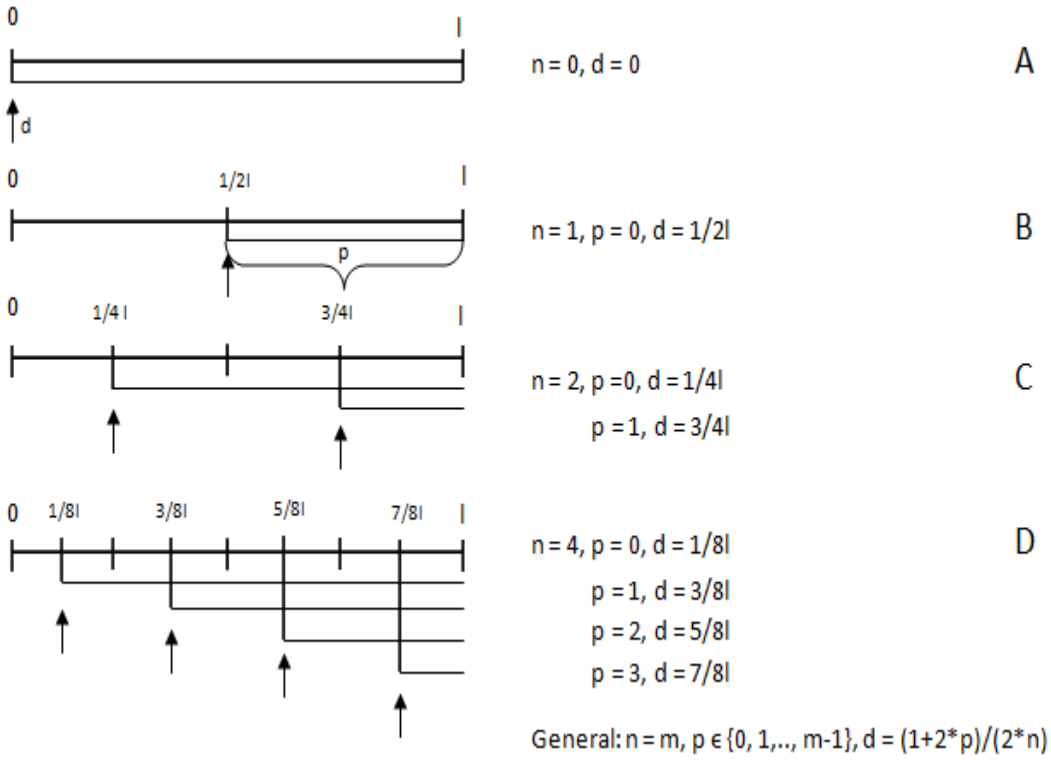


Figure 4.14 - File division strategy.

For the remainder of this section we are using the term file server to denote both the device which is in posses of the file and the device to which we delegate a file piece download.

When user chooses to initiate a file download a *VirtualFile* task (representing this process) is created and scheduled for execution. Initially there is only one interval to be downloaded that is the entire file [0, fileLength] (Figure 4.17A). Once a file server is in reach area a DATA_REQUEST is issued containing the missing data interval, in this case [0, fileLength]. If there is more than one file server in reach transfer protocol might be initiated (Section 4.5.2) with each of them. Now, to each execution flow is assigned

an interval that needs to be downloaded and each interval corresponds to one potential data piece.
To better illustrate how these intervals are computed let's consider some potential scenarios, referring to Figure 4.14:

- 2 file servers are in reach and there are available resources to launch 2 parallel execution flows:
    - To file server 1 is requested the data content in the interval [0, fileLength]
    - To file server 2 is requested the data content in the interval [1/2l, l]

- 4 file servers are in reach area and there are available resources to launch four parallel execution flows:
    - To file server 1 and 2 are requested the intervals shown before
    - To file server 3 is requested the data content in the interval [1/4l,l]
    - To file server 4 is requested the data content in the interval [3/4l, l]

The starting point of the requested interval is calculated by the following formula $d = (1+2p)/(2n)*fileLength$, where n denotes the current number of pieces the initial interval [0, fileLength] is composed off, and p denotes the next interval on the current partitioning to be fetched.

Once the assigned data interval is downloaded from the file server we would desire to use this source to download other missing intervals while in reach (Figure 4.13). Also, on the other side it is possible that a data request is not entirely satisfied (transmission was interrupted for some reason) and between the next starting point (d) and interval end point (fileLength) might be comprised more than one interval (Figure 4.15).

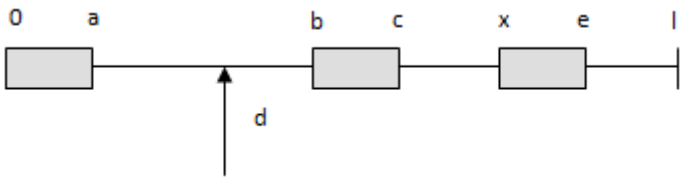


Figure 4.15 - Potential download map. Some data pieces have been downloaded. The resulting map in this case is {d;[a+1,d-1],[d,b-1],[c+1,x-1],[e+1,l]}.

To better illustrate how these maps are computed and composed let's consider again the scenarios shown before:

- 2 file servers are in reach and there are available resources to launch 2 parallel execution flows:
    - To file server 1 are requested data pieces, intervals from map{ 0; [0, fileLength] }, and 0 denotes the starting point.

    - To file server 2 are requested data pieces, intervals from map{ 1/2l; [0, 1/2l), [1/2l, l]] }) and 1/2l denotes the starting point

- 4 file servers are in reach area and there are available resources to launch four parallel execution flows:
    - To file server 1 and 2 are requested the data pieces, intervals shown before

    - To file server 3 are requested data pieces, intervals from map{ 1/4l; [0, 1/4l), [1/4l,l] }) and 1/4l denotes the starting point

    - To file server 4 are requested data pieces, intervals from map{ 3/4l; [0, 3/4l), [3/4l, l] }) and 3/4l denotes the starting point

In the general case, the download map from which issued requests are based upon has the following format:

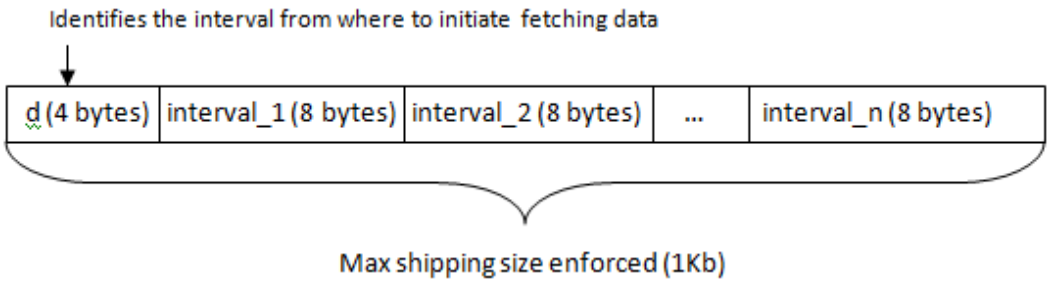


Figure 4.16 - Download map format.

For optimization purposes (redundancy, see further), in case when direct file possessors are in-reach area (*VirtualTask*), the rest of the map is requested after the initial assigned map has been transferred.

A data download might also be delegated to an active entry, just elected entry, and in such a case the entire download map is communicated to the servant device (Section 4.5.3). In the worst case scenario, where downloaded and not downloaded bytes alternate, shipping the entire map would mean transferring a data quantity of 4 times the file size (8*length/2). So a restriction in the map size is imposed and default value is set to 1Kb. The strategy of map composition is as follows:

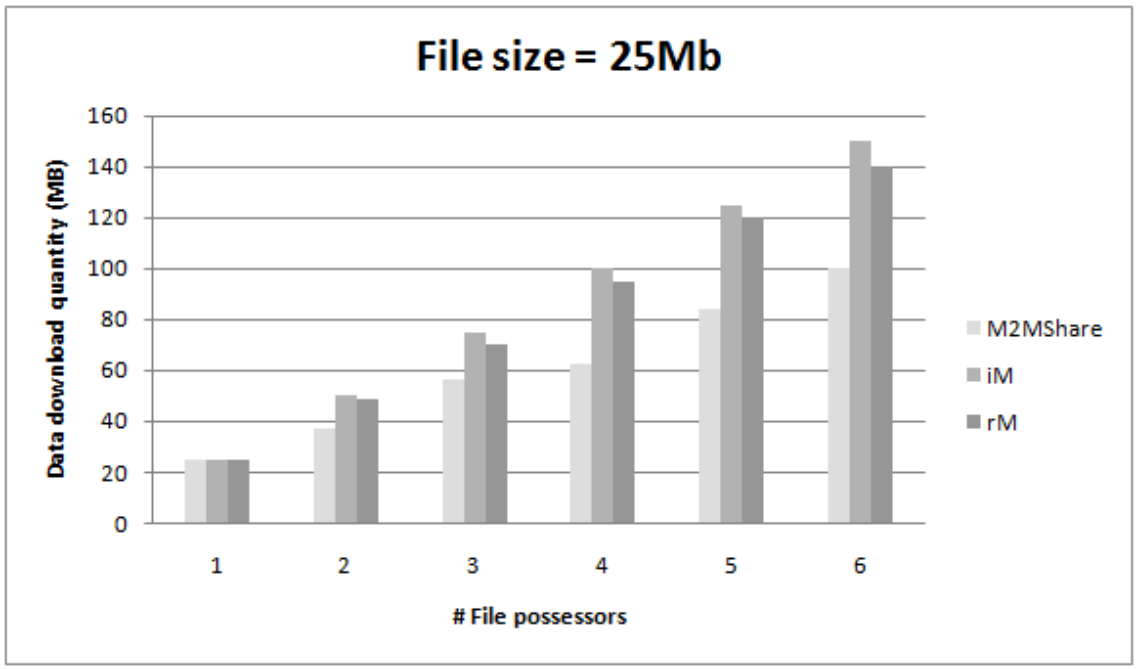
```
Map_Composition() {
        Add starting pointer
        Add starting interval
        While space not exceeded do
                Add intervals found between (startingPoint, fileLength]
        While space not exceeded do
                Add intervals found between [0, startingPoint)
```

As mentioned earlier while introducing DTNs, they use message replication techniques along the destination path in order to increase the probability of data reaching the destination. In our case the file division strategy might add redundancy during data transfer as it can happen that at concurring file servers are requested overlapping data intervals. From the scenario above (Figure 4.14) if two file possessors are in-reach area the requested download intervals will overlap by half of file size. This is the worst case scenario, file possessors present at the same point in time, where the algorithm handling the communication cannot find out this redundancy taking place while assigning

intervals. If file possessors come in different points in time, e.g. a disconnection occurred and transfer was not finished the next requested data interval consists of only free, not yet downloaded data intervals.

Moreover, the task when initially created and scheduled or rescheduled after some execution time is subject to delegation. We might delegate servant data intervals to download (*DTNPendingDownload*) and never require them back as they were previously downloaded at some point in time after the delegation occurred.
So by redundancy is meant not only redundancy on the client side only but on the system as overall for that particular file download.
The graphic below shows a comparison of our division strategy with two other division strategies which are:

- *iM*: strategy which requests at each file server the entire file.
- *rM*: strategy which initial point in the request interval is chosen randomly.



In our testing scenario file possessors are always in-reach area (star topology). If *n* is the number of file possessors the task handling the file download might initiate, if system resources are available (Section 4.3), *n* simultaneous transfers. The experiment was

repeated several times for each division strategy in the cases where n≥2 in order to increase result accuracy. As we can see from the graphics above our division strategy has the least redundancy during data transfer.
Task delegation was not considered as this scenario requires taking into consideration disconnections due to mobility or interferences that happen in real world communication. To our misfortune the adopted development platform does not provide such testing facilities (Section 5, future works).
This division strategy is implemented in the m2m.utilities.IntervalMap class.

4.4 Routing module

M2MShare establishes overlay connections on demand and maintains them as long as they are necessary; task accomplished or when disconnect occurs because devices aren't in reach area anymore. Peers in the overlay are uniquely identified by their MAC address hence peer address space is flat and analogous to that of MAC layer, whereas files are identified by a 16byte MD5 digest computed on its first 100k data bytes calculated during the indexing process.

The routing layer provides mechanisms for query routing along with a mechanism for controlled flooding of query requests. As in ORION (Section 3.3) a structure like file routing table (FRT) is used to store alternative next hops for a particular file. Another particularity borrowed by ORION is the shadowing of non direct file possessors from the origin (query issuer) point of view. From the origin stand point all files are stored by devices in-reach area which during data transfer will act as relay nodes without the origin knowing about it.
The controlled flooding mechanism is relatively simple and is described by the pseudo code shown in (Section 2.2). Its modus operandi is shown below by means of an activity diagram (Figure 4.17). Controlled message flooding is achieved by implementing the following pseudo code elements:

1. Uniquely identifying messages: This technique is borrowed from the AODV routing algorithm, where a message is uniquely identified by (i) originator of the message and a (ii) message *id* which is shipped along with the message itself. All devices

maintain an *id* which is incremented on each issued broadcast. In our system this *id* is represented by a *long* data type.

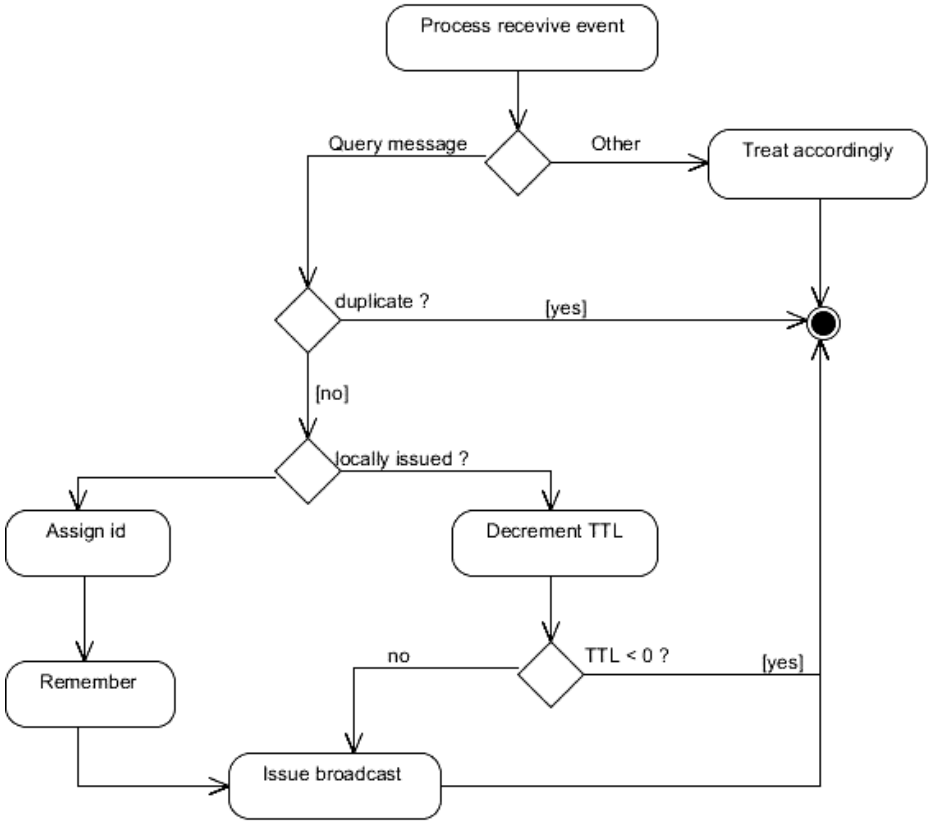


Figure 4.17 - Flow diagram for controlled message flooding algorithm.

2. Reminding messages: There is a need to keep track of previously seen queries in order to prevent flooding the messages over and over again in the network. Each time, before flooding an incoming query message a check to see if it was previously seen is performed in the data structure.

The memory consumption for both tables is controlled by imposing a limiting to their size and applying a FIFO replacement policy in case list is full.

4.5 MAC module

Service discovery protocol (SDP [24]) is the basis for discovery of services on all Bluetooth devices. This is essential for all Bluetooth models. Using the SDP device

information, services and the characteristics of the services can be queried and after that a connection between two or more Bluetooth devices may be established. It is also important to mention that a peer device, in order to be found during the inquiry process must be in inquiry scan mode, usually called visible mode.

In M2MShare this duty is delegated to a particular listener called *M2MServer*, which is in busy waiting for incoming connections. Once program is launched for execution our device is set as discoverable, visible to inquiry scans performed by devices in the reach-area. The listener is uniquely identified, by an identifier called UUID (Universal Unique Identifier) and all *M2MServer* share the same UUID. This identifier is generated in such a way that it is guaranteed to be unique across space and time. For more insight on how this identifier is generated refer to [14, page 345].

The MAC layer is responsible for message broadcasting and presence information gathering. Both services require that an inquiry for in-area devices and services they offer be previously performed.

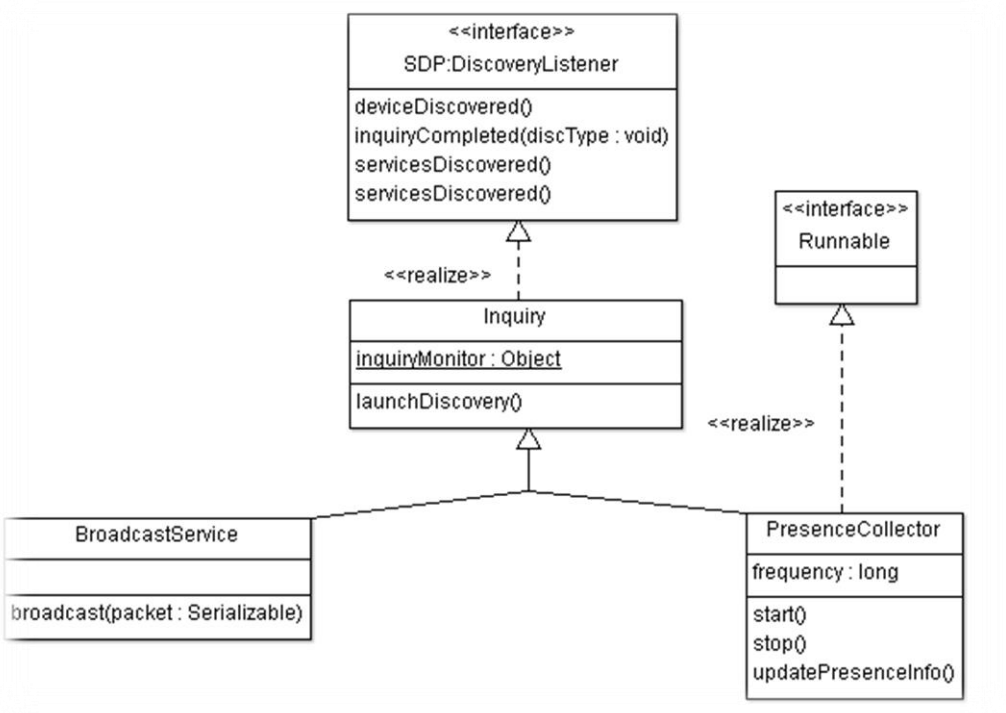


Figure 4.18 - MAC services composition.

As it is shown in Figure 4.18 *Inquiry* serves as base class to both *BroadcastService* and *PresenceCollector* classes. It implements the peer discovery mechanism, peers which

are hosting and running M2MShare. To perform its task it essentially needs two configurable parameters which are:

- M2MServiceUUID: identifier for *M2MServer*, which is actively listening for incoming connections.

- M2MSERVICE_ATTRIBUTE: identifies services attributes stored for *M2MServer*, retrieved during the inquiry process.

4.5.1 PresenceCollector service

PresenceCollector is modeled as an active daemon which periodically scans the network for in-reach area devices (peers). Its periodicity is configurable through the user interface and should be ‘‘meaningful’’. High frequency (e.g. 1sec) is not reasonable from the energy preservation point of view as software is actively running, consuming network bandwidth and device energy. Figure 4.19 shows battery lifetime of a Nokia 5730 XpressMusic cell phone running only *PresenceCollector* service with different frequency at each run. The results are quite intuitive and expected.

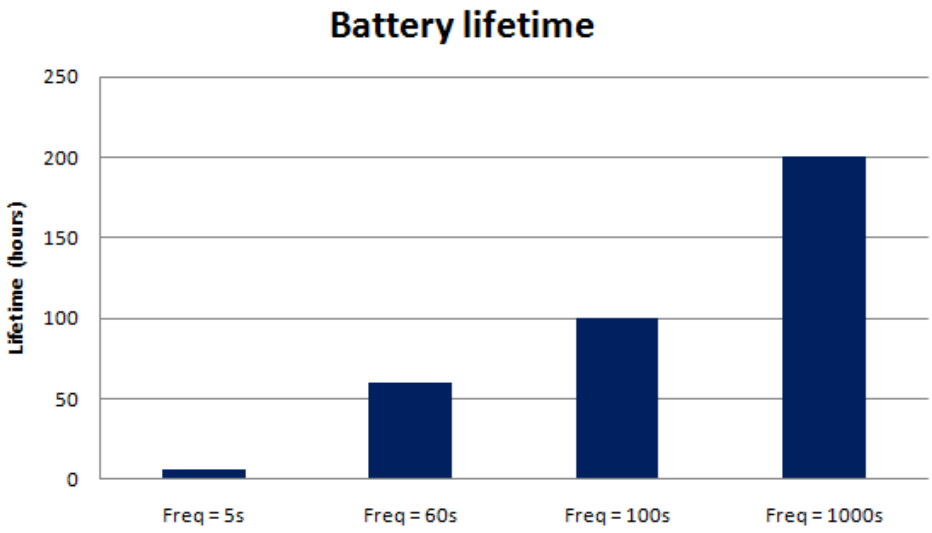


Figure 4.19 – Battery consumption of a Nokia 5730 XpressMusic running *PresenceCollector* service under different frequency at each run.

Having low frequency (e.g. 10min) instead, the device might miss the chance to elect and delegate an unaccomplished task to a potential good servant (Section 4.2). Also a servant might miss the chance to initiate an output forward of a previously delegated task while delegator device is in-reach area but not yet discovered by its periodic discovery service. Both this situations refer to a particular class of tasks in our system whose creation and execution depends directly on the inquiry frequency (Section 4.3.1).
In choosing a value for the polling frequency we need not only make energy considerations but should also (i) give task delegation and (ii) output forwarding a chance.

The inquiry frequency is strictly connected to the election of servants, therefore to task delegation. Having high inquiry frequency might naively elect and delegate a task to a peer device, a device which we might never encounter in the future. E.g. with a periodicity of 5s and an encountering period of 10s between device A and B one might elect the other as servant and delegate a task (c=2). If device is encountering frequent sporadic encounters this is not desirable and the algorithm was designed to adapt to this situation by incrementing the *FREQUENCY_THRESHOLD* (c). By doing so the algorithm imposes a much larger encounter time between devices discarding these sporadic encounters (Section 4.2).
On the other side, where the inquiry frequency is low and sporadic encounters are a representative part of the overall encounters, the algorithm will lower the *FREQUENCY_THRESHOLD* - imposing a lower encounter time between devices before election. This is not a design bug - the algorithm tries to adapt to the capacity of that particular user for encountering other devices operating the software and gives each entry in the servant list a chance to be elected.

Now we tackle the problem of data transfer, in particular that of output forwarding which is a special case; Device B has delegated another device A some particular task or vice versa A has delegated B a task and next time they encounter each other the servant notifies client that it is ready to forward the output of that particular task.
The corollary, introduced below, can be used to compute the average data quantity transferred between two devices once they sense each other's presence. We use the theorem to compute the transferred data quantity in different scenarios where software

might be active and operational. The data quantity the client (A or B) downloads depends on different factors (Figure 4.20):

a) The duration of the established link (D) between node A and node B, the amount of time that might be used for data transfer that differs from the encounter duration (T_e).

b) The bandwidth available on the client side for data transfer (B_w), which we consider to be constant during all link establishment time, neglecting factors such as interference, protocol activities and other possible on-going transfers (upload or download).

c) Time (T_c) in which the servant (A or B) discovers client and notifies it for output forwarding - devices are in communication range and know of each other's presence (communication delay).

d) Time (T_a) in which the client (B or A) executes the forwarded task, initiating a data transfer (queuing delay).

e) The frequency of periodic inquiry (T_f).

For sake of simplicity communication delays (T_c) and queuing delays (T_a) are no considered in the study. They are both considered as e negligible amount of time therefore irrelevant.

Theorem: The average time lost for data transfer between devices A and B entering in communication range with each other is $E[X] = \frac{1}{3}T_f$.

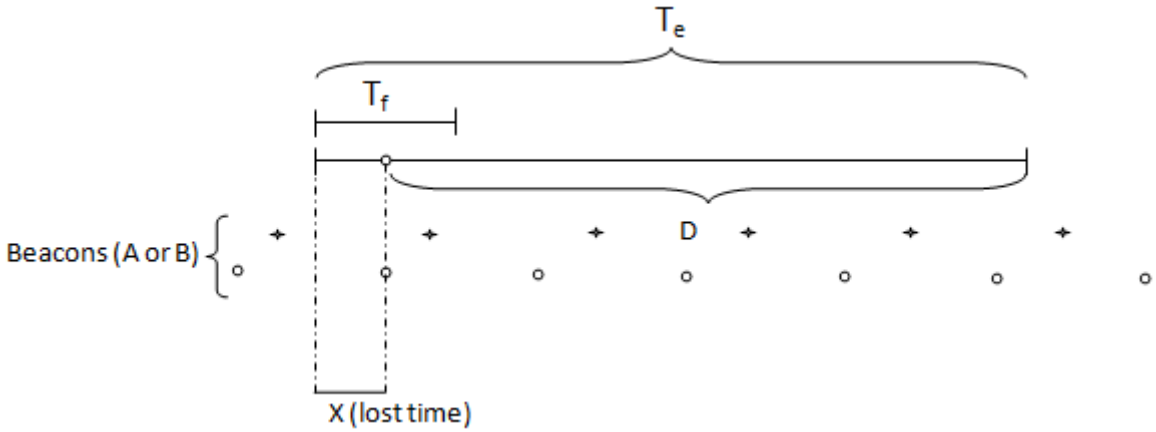


Figure 4. 20 - Servant senses client presence and notifies (round ball) him that a previously delegated task output has been accomplished.

Proof:

Let *X* denote the interval between the moment at which the distance between two nodes A and B becomes smaller than the transmission range and the moment when the first one of them sends a beacon and let X_A be the interval between the moment at which node A and node B enter in the transmission range of each other and the moment when node A transmits its beacon.

As node A and node B periodically transmit their beacons independently from each other, then X_A is independent from X_B and $X = min\ (X_A, X_B)$.

From the single node perspective the probability $P(X_A \leq t)$, where t ϵ [0, T_f] denotes the lost amount of time is:

$$P(X_A \leq t) = \frac{t}{T_f} \leftrightarrow P(X_A > t) = 1 - \frac{t}{T_f} \quad (1)$$

Since there are two nodes independently transmitting their beacons (A, B) the lost amount of time is characterized by the following probability function:

$$F(t) = P(X \leq t) = P(\min(X_A, X_B) \leq t) =$$
$$1 - P(X_A > t) * P(X_B > t) \quad =$$
$$1 - (1 - \frac{t}{T_f})^2 \quad (2)$$

Therefore $f(t) = dF(t)/dt$, where *F(t)* and *f(t)* denote the *Cumulative Distribution Function* (CDF) and the *Probability Density Function* (PDF), respectively.
In order to compute the expected time lost, we need to integrate the product with its density function:

$$E[X] = \int_0^{T_f} f(t) * t \; dt =$$

$$\int_0^{T_f} \frac{2}{T_f}\left(1 - \frac{t}{T_f}\right) * t \; dt \quad =$$

$$\frac{1}{3}T_f. \qquad (3)$$

Having calculated the average time lost between two nodes entering in communication range of each other, we now introduce a second theorem from which can be computed the average data quantity transferred.

Corollary: Given T_e, the average amount of time node A and node B are in communication range of each other, given T_f the frequency of periodic inquiry of each node and B_w the bandwidth available at each node then the average data transferred quantity is: T_d= (Te - Tf/3)B_w.*

Proof*:*

The corollary is a direct consequence of the previous theorem. Given the overall expected time of the encounter (T_e) and the expected lost time (T_f /3) after which both devices can initiate the transfer, than, the remaining time for data transfer is:

$$E[D] = \; Te \; - \frac{Tf}{3} - \; Ta - Tc \;\; (1)$$

Since we are under the assumption that *Ta* (queuing delay) and *Tc* (communication delay) are negligible amount of times then:

$$E[D] = \; Te \; - \frac{Tf}{3} \qquad (2)$$

Finally, knowing the average link duration between the two nodes we can compute the average data quantity transferred after link establishment, which is:

$$T_d = \left(T_e - \frac{T_f}{3}\right) * B_w \quad (3).$$

The data exposed in the graphics below are computed using the introduced theorem and refer to different scenarios in which software might be running and functional.

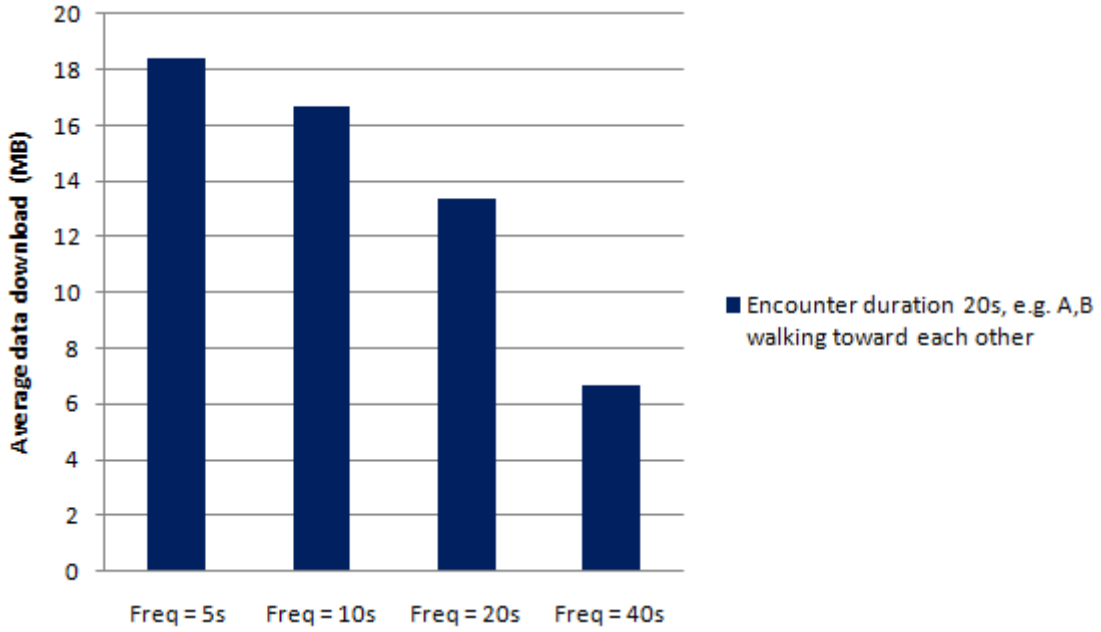


Figure 4.21 – Average data download quantity (MB) between servant and client in an encountering period of 20s.

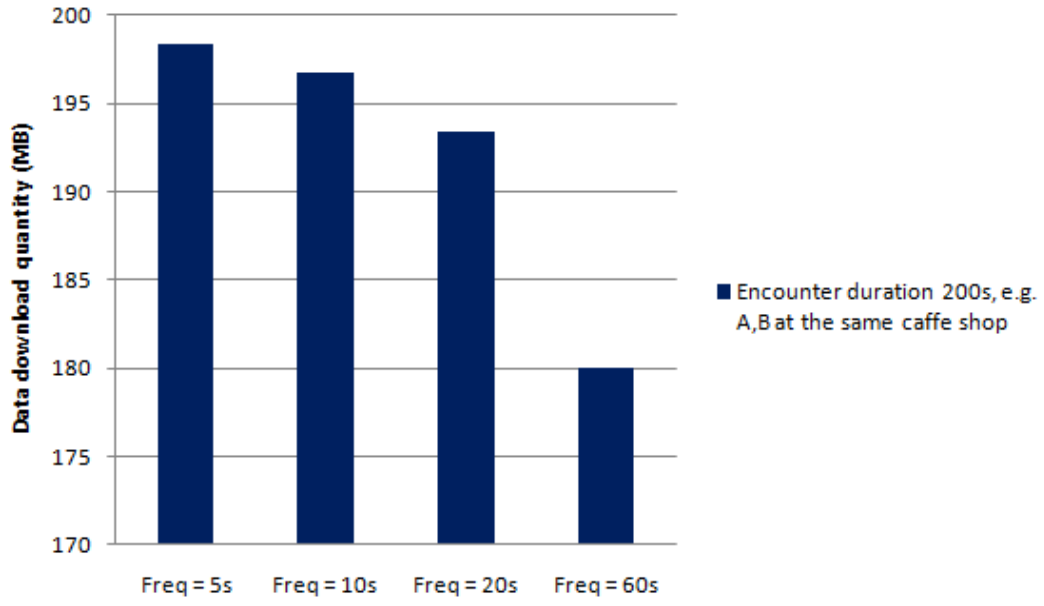


Figure 4.22 - Average data download quantity (MB) between servant and client in an encountering period of 200s.

Summarizing what said, there are different factors involved in the decisional process of choosing the frequency value for the periodic inquiry, varying from energy consumption to operational considerations. We chose these particular frequency values a we want software to be active and operational even in situations where mobility comes into play, the provided default value is set to *10s* as it is desirable that software becomes operational even in the borderline scenario where users are walking toward each other.

4.5.2 Broadcast service

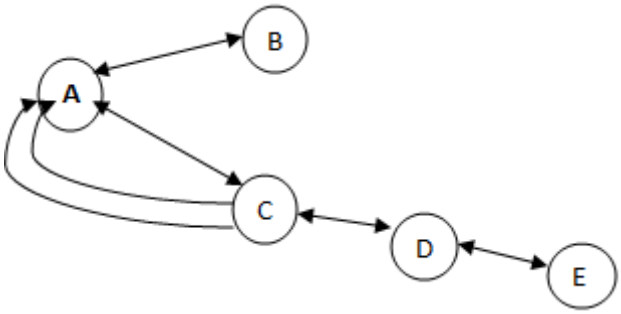


Figure 4.23 - Query_Request broadcasting. Peer C acts as a relay point between A and D,E.

As its name indicates this services provides the message broadcasting feature in an established Bluetooth network. Once a Query_Request is broadcasted into the network we expect back the responses. We cannot a-priori decide whether all responses are returned or not since, other peers in the system might act as relay points between other peers and the request issuer and we do not kwon the length of this chain (Figure 4.23). Conceptually the message response gathering should be modeled as a busy waiting process:

```
while(true) do   inputStream.ReadResponse()
```

A way to interrupt this infinite cycle should be provided, in order to fetch the gathered responses to the executable task that issued this request. This is done by imposing a time limit on each connection established between request issuer and other peers, in our case between A-B and A-C. Responses received before timer expiries are gathered and the output is returned to the issuer task, meanwhile other responses are discarded.

Another thing worth noting, is that both services this layer offers might want to access the underlying medium in the same time, e.g. issuing a message broadcast process might coincide with presence collection phase. Practically the communication medium is used and accessed sequentially. The underlying Bluetooth protocol layer (L2CAP) on which SDP is build upon handles this case by throwing an error if an inquiry is launched while another inquiry is currently on-going. M2MShare handles this situation by synchronizing parallel SDP flows with a monitor (*inquiryMonitor*)

Chapter 5

Conclusions

With the growing number of mobile devices equipped with wireless interfaces, mobile users increasingly find themselves in different types of networking environments. These environments, spanning from globally connected networks such as cellular networks or the Internet to the entirely disconnected networks of stand-alone mobile devices, impose different forms of connectivity. Due to the high mobility, communication links between mobile nodes are transient and temporarily connected. Therefore, a continuous end-to-end path between a source and a destination cannot be sustained. This is a modern and increasingly more commonly used type of DTN, which was originally intended to be used for communication in outer space but is now directly accessible from our pockets.

To this aim, we examined the experimentation that we found in literature on the P2P file sharing applications for MANETs. From this analysis we classified the main problems that arise when considering P2P architecture in this type of environment and implemented a prototype that serves as a proof of concept.

The hardest part of this work was the development of the prototype. It takes enough time and work. Initially our intent was to extend and improve a previous project that attracted our attention, namely ORION which we discussed in details in Section 3.2. ORIONs approach on establishing overlay links on demand was the predominant approach in P2P literature for MANETs, which significantly reduces overlay maintenance overhead in an established P2P network and test results demonstrate this. To our misfortune, after writing numerous emails to the authors, we did not get any response in return and given the timetable and the experience gained in developing J2ME applications we chose the former as the development platform.

As noticed, work in P2P MANET is diverse and touches on various aspects of computer science and communication engineering. In this work we had to focus our attention on some aspects rather than all of them. More specifically we presented the interested reader a special purpose P2P system tailored to MANETs which addresses some of the

problems intrinsic to their nature. Overlay connections are established on demand and little or no overhead is needed for their maintenance on the contrary to the reviewed overlay organizations conceived for the wired internet where fixed connections with neighbor peers are established.

We do not see mobility as an obstacle instead, we exploit peer mobility to reach data in other disconnected overlay networks providing along with a synchronous file transfer an asynchronous transfer mode, implementing a mechanism alike DTN (store-and-forward) where peers in the network delegate tasks to other peers (store) and wait back for their output (forward).

We argued that a peer delegating tasks to all other peers in the established network proves bandwidth and energy consuming. Therefore we defined a metric at a protocol level by which individual peers select their servants to whom they delegate tasks (Section 4.2). The criterion of selecting one peer instead of others is based upon the frequency with which one device encounters other devices in-reach area. As argued this frequency of election changes and adapts to the observed dynamics; there might be days when the device encounters a small number of other devices and others when it encounters a large number of them.
This mechanism can be further extended by allowing delegations to be delegated more than one hop away. Each servant peer in the delegation chain chooses dynamically which next route the delegation should be forwarded to.

M2MShare periodically collects presence information of in-area devices registering and probing them for a minimum period of time (day ≥ 2), giving them the chance to serve as servants. The list where this information is kept has storage constraints therefore a replacement policy was provided. We also discussed the importance of the periodic inquiry and its influence in the election process, output forwarding and provided a default value for it justified by the idea that the software needs to be operational even in borderline situations, those where mobility comes into play.
On our judgment, the innovations and contributions of this work are the following:

1. Providing an overview of the solution and practical design considerations to other researchers/practitioners that might be interested in developing a similar project.

2. Demonstrating a new paradigm of use of P2P solutions that matches file sharing with mobile users, allowing them to exchange files with each other, thus fostering new applications.

3. Combining and providing a proof of concept solution for porting DTN type communication in the mobile world.

4. Defining the criteria and implementing a solution for task delegation to particular overlay peers extending our data reach area to other disconnected overlay networks in contrast with the delegation of tasks to all participating peers of the overlay.

Another think worth noting is that key features of M2MShare are its ease of use and autonomy. In fact, once the initial preferences are set up no further user interaction is needed. This said, there is a lot of work that can be done to further improve and extend the functionalities provided by the software. Some potential future developments might be:

- Mobility scenario: Our software development platform, Netbeans 6.5 Mobility pack, does not have a proper test environment for this type of application. The first thing that needs to be done is to test the software under real mobility scenarios and see how it performs. During this work we presented some algorithms are tried to characterize their behavior by means of statistical modeling or by formalizing them in a mathematical framework or in the case of the transfer protocol and file division strategy, real tests was performed not considering mobility scenarios. This approach is essential but not sufficient. We are currently studying the feasibility of testing the software in The Opportunistic Network Environment (THE_ONE) a simulation environment which seems to serve our purpose.

- We are currently developing a statistical model in order to study the behavior of the parameters (*Pw, c*) for the servant election algorithm by simulating device encounters by means of a stochastic process. The idea of this work is to simulate the encounter of devices and study the parameters evolution from a single device point of view. Not all encounters are of the same type, we differentiate between three types of devices:

    - High frequent encountered devices: device we frequently encounter during one unit of software operational time, where unit might be an hour, active session or an entire monitoring day.

    - Low frequent encountered devices: devices which we encounter relatively frequently.

    - Sporadic encountered devices: devices which we encounter once and might never see them again.

This work is being supervised by Paolo Dai Pra, professor of statistics of Padua University.

- Indexing scheme improvement: M2MShare uses inverted index list to locally index shared files. The implementation can be further improved in these directions such as: (i) allowing dynamic index editing when a new entries are added to the index set vs entire index re-compute as provided in the actual implementation; (ii) improve stop word filtering by adding a dictionary of stop words vs considering words of length ≤ 2 as stop words, as done in the current implementation; (iii) query modeling can be further improved to provide more complex search schemes such as proximity and range queries.

- DTN nominal entries and semantic categories: The actual implementation of DTN Module tries to maximize the expected Active Ratio (r) adapting the functional parameters to the observed network dynamics. As discussed we have no judgment criteria to tell what a good servant is. We argued that frequency is not a relevant indicator and also today's good servant might not be tomorrow's good servant. A

possible future development for this module would be that of adding a list where individual previous servants are registered. This implies that a feedback of this kind should be provided to the module and the module itself should provide means for upgrading and downgrading entries of the list depending on some criteria. This potential extension is summarized below:

- The actual DTN Module is inherited and is the same in the new module.

- Add another list which keeps track of previous servants which returned back the output. The list should have a different scoring scheme attributed to each individual entry in order to measure its goodness or relevance.

- A list management technique has to be found, applying a scoring function for:

    - Upgrading an entry: once a device, who previously served us, is encountered even if not elected is delegated a task. We could immediately upgrade the entry by applying a particular scoring function or expect him to return the output first and then upgrade it by adding a contribute to its score.

    - Downgrading an entry: if device has not been seen for some period its score will gradually lower until it expires and be replaced from the list.

    - Replacement policy: in case the list is full of entries and a new entry has to be considered for insertion.

By doing so, we can delegate tasks to previous good servants and expects back the output with a greater confidence that it will return it back to us; also we enrich this list with entries given from the actual mechanism of election.
Furthermore, one might even think to group servants in thematic categories. There might be servants who are good in finding particular thematic categories of files

such as audio files from a particular writer or movie trailers from a particular producer etc. The underlying assumption is that users have a routine of their own which does not change frequently and places they go, data content they reach, might be classified in thematic category/s. By doing so, we can forward queries or content download to these entries, if a match is deduced, prioritizing them above other active entries.

A lot can be done to further improve and extend the current module implementation. Our solution might seem poor at first glance but it serves as a proof of concept and as a solid base to future developments.

- Route discovery: When a peer (client) delegates a task to another peer (servant) in the established overlay, the next time they encounter the servant will forward the task output to the client peer. The output is forwarded if devices are in-reach area. This design is justified by the assumption that users have a routine of their own which does not change frequently and most probably they will encounter again at some point and exchange the output content. This is not entirely true, imagine a scenario where:

    - Device A (client) elected device B (servant) as a servant and delegated him a task.

    - Device B completes the task and is ready to forward its output.

    - Tomorrow, users are both on the same train to work but in different compartments and are not in-reach area.

Device B will not be able to forward the output of the task and the task might even expire and be discarded at the end. A future work should be to extend the routing protocol with a route discovery mechanism as in AODV. In such a case the servant's device might initiate an explicit route discovery for the client device and client might be reachable by other devices hosting M2MShare and standing between servant and client. This feature is most desirable and is currently under implementation.

- Biased routing: Peers in the overlay can act as relay points for query responses toward the requestor (Figure 4.8). These responses can be used by peers in the overlay to gain information about file availability in the responder side. A peer might later use this gathered information to query specific nodes in the network and not flood the query to all of them. Let's see an example of what we intend:

    - Refer to configuration shown in Figure 4.8, peer A floods a query request with search keys *x, y, z*.

    - Peer B responds to the issued query with file descriptor F' containing only *x* keyword

    - Peer D responds for himself with file descriptor F'' containing keywords x, y and acts as a relay point for response given by C with file descriptor F''' containing keywords x, y, z.

    - Given that peers C acts as a relay point for (A, [D, C]), he can inspect responses and build a history of (destination, keywords).

    - Next time peer C issues directly query request with keywords x, y, z toward D as they most probably, referring to the observed data, have a match toward that direction. If D is no more in reach or returned results are not satisfied a query request flood might be issued as a second step.

References

[1]: *Mobile Ad Hoc Networks (MANETs), refer to -* http://www.antd.nist.gov/wahn_mahn.shtml

[2]: *Delay Tolerant Networking Research Group, refer to -* http://www.dtnrg.org/wiki

[3]: *D. Clark, B. Lehr, S. Bauer, P. Faratin, R. Sami, and J. Wroclawski, The growth of Internet overlay networks: implications for architecture, industry structure and policy, 33rd Telecommunications Policy Research Conference, Sept. 2005, available at http://web.si.umich.edu/tprc/papers/2005/466/TPRC_Overlays_9_8_05.pdf.*

[4]: *B. Carlsson and R. Gustavsson, The rise and fall of Napster: an evolutionary approach, in: Proceedings of the 6th International Computer Science Conference on Active Media Technology, (Dec. 18–20, 2001) J. Liu, P. C. Yuen, C. H. Li, J. K. Ng, and T. Ishida (eds.), Lecture Notes in Computer Science, Vol. 2252, Springer-Verlag, 347–354.*

[5]: *Gnutella Protocol Specification, available at: http://wiki.limewire.org/index.php?title=GDF.*

[6]: *J. Liang, R. Kumar, and K. Ross, The FastTrack overlay: a measurement study, Computer Networks, 50, 842–858, 2006.*

[7]: *B. Cohen, The BitTorrent protocol specification www.bittorrent.org/beps/bep_0003.html, Feb 2008.*

[8]: *I. Clarke, O. Sandberg, B. Wiley, and T. Hong, Freenet: a distributed anonymous information storage and retrieval system, in: Proceedings International Workshop on Design Issues in Anonymity and Unobservability, Springer, 2001.*

[9]: *D. Stutzbach and R. Rejaie, Understanding churn in peer-to-peer networks, in: Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement Conference 2006 (IMC 2006), Oct. 25–27, 2006.*

[10]: *UPnP Forum, refer to - www.upnp.org.*

[11]: *Antony I. T. Rowstron and Peter Druschel, Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems, Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms, Heidelberg, Nov 12–16, 2001, 329–350.*

[12]: *I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, Chord: A scalable peer-to- peer lookup service for Internet applications, in: ACM SIGCOMM 2001, San Diego, CA, 2001, 149–160.*

[13]: *S. Gurun, P. Nagpurkar, and B. Y. Zhao, Energy consumption and conservation in mobile peer-to-peer systems, in: Proceedings of the 1st international workshop on Decentralized resource sharing in mobile computing and networking, Los Angeles, Jul. 25–25, 2006.*

[14]: *Bluetooth SIG, Specification of the Bluetooth System version 1.1, 2001.*

[15]: *AODV Specification - IETF, RFC3561 available at http://www.ietf.org/rfc/rfc3561.txt.*

[17]: *J.Buford, H.Yu and Eng K.Lua - P2P Networking and Applications 2009 by Elsevier*

[18]: *Distributed Hash Table (DHT), refer to - http://www.linuxjournal.com/article/6797*

[19]: *B. T. Loo, R. Huebsch, I. Stoica and J. M. Hellerstein, The case for a hybrid P2P search infrastructure, in: Proceedings the 3rd International Workshop on Peer-to-Peer Systems, San Diego, CA, Feb. 26–27, 2004*

[20]: *K. Aberer, L. O. Alima, A. Ghodsi, S. Girdzijauskas, S. Haridi, and M. Hauswirth, The essence of P2P: a reference architecture for overlay networks, Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P '05), Volume 00, Aug. 31–Sept. 2, 2005, IEEE Computer Society, Washington, DC, 11–20.*

[21]: *W.R.Stevens - TCP/IP Illustrated, Volume 1- The Protocols, Addison-Wesley, April 2000.*

[22]: *J. Bray and C. Sturman, Bluetooth 1.1, Connect Without Cables, Second Edition, Prentice Hall, 2002.*

[23]: *M. S. Gast, 802.11 Wireless Networks, First Edition, O'Reilly, 2002.*

[24]: *SDP, refer to - http://www.palowireless.com/infotooth/tutorial/sdp.asp*

[25]: *J2ME platform, refer to - http://java.sun.com/javame/index.jsp*

[26]: *Connection Limited Device (CDC), refer to - http://www.jcp.org/en/jsr/detail?id=128*

[27]: *Sun Microsystems, Inc., Connected, Limited Device Configuration. Specification Version 1.0a, 2000.*

[28]: *Mobile Information Device Profile (MIDP), refer to - http://java.sun.com/products/midp/*

[29]: *S.Srinivasan, A.Moghadam, S.Hong and H.Schulzrinne - 7DS, Node Cooperation and Information Exchange in Mostly Disconnected Networks*

[30]: *A. Klemm, C. Lindemann, and O. Waldhorst, ORION - A Special-Purpose Peer-to-Peer File Sharing System for Mobile Ad Hoc Networks.*

[31]: *O.Mukhtar and J. Ott - Backup and Bypass: Introducing DTN-based Ad-hoc Networking to mobile phones*

[32]: *J. Mache, et al., Request algorithms in Freenet-style peer-to-peer systems, in: Proceedings of the Second IEEE International Conference on Peer to Peer Computing, Sept. 5–7, 2002, Linkoping, Sweden.*

[33]: *A. Z. Kronfol, FASD: A fault-tolerant adaptive scalable distributed search engine, Master's Thesis, Princeton University, available at www.cs.princeton.edu/akronfol/fasd/.*

[34]: *Term Frequency Inverse Document Frequency (TFIDF); S.Robertson - Understanding Inverse Document Frequency, On theoretical arguments for IDF, Microsoft Research.*

[35]: *MIdlet application insights, refer to - http://today.java.net/pub/a/today/2005/02/09/j2me1.html*

[36]: *I.Rhee, M.Shin, S.Hong, K.Lee, S.Chong - Human Mobility Patterns and Their Impact on Routing in Human-Driven Mobile Networks, available at http://conferences.sigcomm.org/hotnets/2007/papers/hotnets6-final108.pdf*

[37]: *S.Samal - Mobility Pattern Aware Routing in Mobile Ad Hoc Networks 2003, Master's Thesis, available at http://scholar.lib.vt.edu/theses*

[38]: *A. Tannenbaum, Computer Networks, 4th ed., Prentice Hall, 2002.*

[39]: *M. Castro, M. Costa, and A. Rowstron, Debunking some myths about structured and unstructured overlays, Proc. of the 2nd Symposium on Networked Sys. Design and Impl. (NSDI) 2005.*

[40]: *C. Xie, S. Guo, R. Rejaie, and Y. Pan, Examining Graph properties of unstructured peer-to-peer overlay topology, Global Internet Symposium, 2007.*

[41]: *Gnutella2 Developer Network, http://g2.trillinux.org/index.php?title=Main_Page, retrieved Nov. 2007.*

[42]: *M.Gerla, C.Lindemann and A.Rowstron, Microsoft Research - P2P MANETs – New Research Issues.*

Figures and Tables index

Figure 1.1 - A P2P streaming scenario involving networked consumer electronics devices as peers [17]..... 5
Figure 1.2 – P2P device composition [17]. 6
Figure 1.3 - System building blocks and application scenarios of P2P MANET [40]. 8

Figure 2.1 - Peers form an overlay network (top) that in turn uses network connections in the native network (bottom). The overlay organization is a logical view that might not directly mirror the physical network topology [17]..... 19
Figure 2.2 - Overlay shown as a sequence of membership changes [17]..... 22
Figure 2.3 - Asymptotic tradeoff between peer-routing table size and overlay diameter # 2003 IEEE. 24
Figure 2.4 - Relationship between protocol performance and mobility model. 25
Figure 2.5 - Topography showing the movement of nodes for Random mobility model. 26
Figure 2.6 - Topography showing the movement of nodes for Manhattan mobility model..... 27
Figure 2.7 - (A) Unstructured topology showing connections between peers and (B) query flooding to four hops [17]. 29
Figure 2.8 - (A) Random walk and (B) k-way parallel random walk, k = 3 [15]. 31
Figure 2.9 - Example Gnutella messaging using flooding-style query [17]..... 34
Figure 2.10 - P2P reference model for mapping peers and files into an metric address space [20]. 36
Figure 2.11 - A 16-node Chord network. The "fingers" for one of the nodes are highlighted [10]. 39
Figure 2.12 – Internet end-to-end communication links [2]. 16
Figure 2.13 – DTN overlay, connecting different regional homogeneous networks providing means for intra-region communication [2]..... 17
Figure 2.14 – Store-and-forward packet switching between DTN routers. 18
Figure 2.15 - The Bluetooth protocol stack [14]. 41
Figure 2.16 - A typical piconet..... 43
Figure 2.17 - A typical Scatternet. 44
Figure 2.18 - Piconet with two nodes. 45
Figure 2.19 - Scatternet with three nodes. 45
Figure 2.20 - Piconet with three nodes..... 46
Figure 2.21 - High level view of J2ME. 47
Figure 2.22 - CLDC position in the J2ME architecture. 49
Figure 2.23 - MIDlet lifecycle [31]. 50

Table 2.1 - Pseudo-Code for Expanding Ring. 30
Table 2.2 - Basic message types for Gnutella v.0.6. 33
Table 2.3 - Description of Bluetooth protocol layers. 42

Figure 3.1 - The algorithm of the 7DS proxy server for handling HTTP requests [29]. 53
Figure 3.2 – Protocol overhead vs Size [30]. 55
Figure 3.3 – Breakdown to message types [30]. 55
Figure 3.4 - Node A floods a QUERY message with keywords matching to files 1 to 4 [26]. .. 57
Figure 3.5 - Node B sends a RESPONSE message with identifiers of local files [26]. 57
Figure 3.6 - Node C sends a RESPONSE message with identifiers of local files,..... 57
Figure 3.7 - Node D sends a RESPONSE message with identifiers of local 58
Figure 3.8 - DTN demonstrator setup [27]. 60
Figure 3.9 - Illustration of (A) centralized index, (B) local index, and (C) distributed indexing [17]. 61
Figure 3.10 - Illustration of sample hybrid indexing configurations [17]. 63

Figure 4.1 - M2MShare peer architecture. Blocks shown in gray are proprietary. 66
Figure 4.2 - Inverted index list. Each file descriptor is indexed under one or more indexing terms. 68
Figure 4.3 - Delegation demonstration scenario. 70
Figure 4.4 - Abstract view of DTN's functional behavior. 72
Figure 4.5 - Replacement policy demonstration scenario. 77
Figure 4.6 - QueuingCentral, showing different types of queues to which apply different queuing policies. 80
Figure 4.7 - A bigger picture of the queuing, scheduling mechanism. 81
Figure 4.8 - Task lifecycle. 84
Figure 4.9 - Each serializable task implements the Serializable interface. 85
Figure 4.10 – Communication protocol by DTNQueryFwd task execution. 88
Figure 4.11 - Communication protocol implemented by DTNDownloadFwd task. 90
Figure 4.12 - Communication protocol implements by DTNPendingDownload task. 92
Figure 4.13 - Communication protocol implemented by VirtualFile task. 94
Figure 4.14 - File division strategy. 97
Figure 4.15 - Potential download map. Some data pieces have been downloaded. The resulting map in this case is {d;[a+1,d-1],[d,b-1],[c+1,x-1],[e+1,l]}. 99
Figure 4.16 - Download map format. 100
Figure 4.17 - Flow diagram for controlled message flooding algorithm. 103
Figure 4.18 - MAC services composition. 104
Figure 4. 19 – Battery consumption of a Nokia 5730 XpressMusic running *PresenceCollector* service under different frequency at each run. 105
Figure 4. 20 – Servant peer A senses B's presence at $T_{A'}$, notifying him that a previously delegated task output has been accomplished ..... **Error! Bookmark not defined.**
Figure 4. 21 – Average data download quantity (MB) between servant and client in an encountering period of 10s. 110
Figure 4. 22 - Average data download quantity (MB) between servant and client in an encountering period of 100s. 110
Figure 4. 23 - Average data download quantity (MB) between servant and client in an encountering period of 500s. ..... **Error! Bookmark not defined.**
Figure 4. 24 - Average data download quantity (MB) between servant and client in an encountering period of 1000s. ..... **Error! Bookmark not defined.**
Figure 4.25 - Query_Request broadcasting. Peer C acts as a relay point between A and D,E. 111

Table 4.1 - Overview of the protocol stack a peer device implements in M2MBluetooth.**Error! Bookmark not defi**

Acknowledgements