REYKJAVÍK UNIVERSITY
HÁSKÓLINN Í REYKJAVÍK

A Gateway for Wireless Dissemination of Delay-tolerant Content

Kristján Valur Jónsson

Master of Science
January 2008

Reykjavík University - School of Computer Science

M.Sc. Thesis

A Gateway for Wireless Dissemination of Delay-tolerant Content

by

Kristján Valur Jónsson

Thesis submitted to the School of Computer Science
at Reykjavík University in partial fulfillment
of the requirements for the degree of
Master of Science

January 2008

Thesis Committee:

Dr. Úlfar Erlingsson, supervisor
Associate Professor, Reykjavik University, Iceland

Dr. Gunnar Karlsson, co-supervisor
Professor, KTH, Sweden

Dr. Björn Þór Jónsson
Associate Professor, Reykjavik University, Iceland

Dr. Gísli Hjálmtýsson
CEO, Bru Venture Capital, Iceland

A Gateway for Wireless Dissemination of Delay-tolerant Content

by

Kristján Valur Jónsson

January 2008

Abstract

Wireless ad-hoc networks are a viable alternative to the currently prevalent model of last-hop wireless links to an infrastructure network. Prior research has shown that mobility can be utilized to increase the capacity of wireless ad-hoc networks, by opportunistically utilizing peer-to-peer contacts. *PodNet* is a research project whose goal is to create a comprehensive personal broadcasting system in which participants opportunistically utilize random contacts to exchange content. This dissertation contributes to the PodNet system. A design of a multi-role node and a solicitation protocol for use in the PodNet system is presented. In particular, this dissertation considers *gateway nodes*, which procure content from sources on the Internet and provide to peers in the ad-hoc domain. This work is supported by an implementation in the form of a simulation model and a simple prototype of PodNet node software. Evaluation of the protocol and some initial explorations of a simple PodNet community are also discussed.

Gátt fyrir þráðlausa dreifingu á biðþolnu efni

eftir

Kristján Valur Jónsson

Janúar 2008

Útdráttur

Þráðlaus jafningjanet eru raunhæfur valkostur í stað þráðlausra tenginga við stoðkerfi sem algengt er í dag. Rannsóknir hafa sýnt að þegar jafningjar í slíkum netum eru hreyfanlegir má nýta tilfallandi innbyrðis tengingar milli þeirra til gagnaskipta. Hreyfanleikann má þannig nýta til að stuðla að dreifingu efnis. *PodNet* er rannsóknarverkefni sem hefur að markmiði að skapa heildstætt dreifikerfi sem nýtir tilfallandi tengingar milli hreyfanlegra jafningja til gagnaflutnings. Þessi ritgerð er innlegg í PodNet verkefnið. Hönnun fjölhæfs hnúts er sett fram og samskiptareglur eru skilgreindar. Þessi vinna er studd með útfærslu í formi hermilíkans og frumgerðar af hugbúnaði. Ein hnútategund er skoðuð sérstaklega: Það er *gátt*, sem sækir efni á Internetið og dreifir í jafningjanetinu. Einnig eru niðurstöður hermunar settar fram, auk prófana á samskiptareglum og frumathugana á hegðun PodNet hópa.

This dissertation is dedicated to my family, Fjóla, Vala Rún and Jón Valur.

Acknowledgements

I want to thank my thesis committee, especially my co-supervisor dr. Gunnar Karlsson for the opportunity to work on this project and for all his support and sponsorship. I also want thank dr. Gísli Hjálmtýsson, my former supervisor for his sponsorship during the course of this work. Finally, I thank Ólafur R. Helgason and Vladimir Vukadinović at the LCN laboratory at KTH in Stockholm for their help during my visit at LCN in the spring of 2007.

Publications

A part of the material in this dissertation was published as *"A Gateway for Wireless Broadcasting"* presented at the *3rd International Conference on emerging Networking EXperiments and Technologies (CoNEXT) Student Workshop*, held December 10, 2007 in New York, NY, USA.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The once futuristic idea of ubiquitous computing, that of of small, independent, computing devices, either carried on one's person or embedded in various everyday devices that surround us, has moved from the realm of science fiction into science fact. An increasing number of such devices play a role in our everyday lives, e.g. mobile phones and PDAs. Wireless capabilities of those devices are ever increasing, inviting a closer look at the paradigm of wireless ad-hoc networking in its various forms.

Despite the wide availability of wireless access, the infrastructure model is still prevalent, essentially relegating this powerful technology to the cable replacement role. Last hop wireless links to infrastructure are still the norm, thus placing rigid restrictions on how wireless access is applied. Present IEEE 802.11 devices almost exclusively use the wireless transport over one hop as a bridge to an infrastructure network. Similarly, cellular networks, perhaps the most ubiquitous of all present wireless devices, use a one hop wireless link between cell tower and device.

There are compelling reasons to explore alternatives to this approach. First, the explosion in the number of wireless devices may make the model unwieldy. Expecting all wireless capable devices to exchange information over infrastructure links may prove unreasonable since this would surely lead to bottlenecks forming within cells and at access points as an increasing number of devices compete for the limited bandwidth of the infrastructure access points or cell towers. Second, creation and distribution in an infrastructure-based network can be severely limited; telecommunications traffic, for example, is heavily regulated and subject to high licensing fees. In extreme cases, content to be published can be subject to restrictions and censorship. Finally, constant infrastructure connection is unnecessary for some applications, where a peer-to-peer ad-hoc exchange between individual nodes may be a much more natural approach.

1.1 Ad-hoc Networking

An alternative model is that of *ad-hoc* networking. This is a communications paradigm best described as assembling impromptu networks from independent devices on the fly, imposing few restrictions on the resulting topology. This is clearly a very natural utilization of the wireless medium, which is characterized by its broadcast nature, especially if mobility is a factor as well. This dissertation only discusses applications of ad-hoc networking to mobile wireless devices, although it certainly applies to other types of networks and devices.

Routing and reliable end-to-end services are complicated in wireless multi-hop ad-hoc networks, due to their non-hierarchical and ever changing nature. This difficulty is compounded if nodes are mobile, in which case any given network configuration is unlikely to persist for an extended period. The wireless medium itself is a further complicating factor, adding increased error potential to an already difficult problem. The flexibility of wireless mobile ad-hoc networks, however, is such that this class of systems is worth exploring as a viable alternative to the more traditional infrastructure based methods. This is especially true if we can take advantage of the mobile nature of the nodes to carry information, rather than regarding it as a problem, as is common in current infrastructure-based wireless research. Wireless ad-hoc networking can clearly compliment the prevalent infrastructure model by distributing load and preventing bottlenecks at access points or cells. For certain applications, this method of communication may replace the infrastructure model in its entirety.

1.2 The PodNet Project

The *PodNet* project (http://podnet.ee.ethz.ch) (Karlsson, Lenders, & May, 2007; Lenders, Karlsson, & May, 2007) is an ongoing effort to create a comprehensive personal broadcasting system, based on wireless ad-hoc networking, in which peers exchange content when within radio range. Node mobility is the primary force of content dissemination. The content is organized into channels, in analogy with traditional radio broadcasting, podcasting and on-line news syndication protocols. Diversity of content is enhanced by requiring nodes to carry an arbitrary number of public channels, in addition to the private channels to which they subscribe.

Nodes in the PodNet system communicate opportunistically over one hop and exchange delay-tolerant content. The mobility of nodes is utilized as the primary force of dissem-

ination; no routing mechanism is necessary, and the problems discussed previously can thus be disregarded. This method of content distribution, however, is not applicable to all types of content. The content in question must have very lenient restrictions on delivery time, order and completeness. Suitable content and delivery methods have been studied in the relatively recent field of *delay-tolerant networking*, which deals with mechanisms for coping with challenged networks, e.g. those with very sporadic connectivity.

The PodNet system is designed for an application which is similar to that of common radio stations. A user tunes his radio to a certain station and thereafter consumes any broadcast content; favorite songs may or may not be played while the user is listening. Content in the PodNet system is mapped onto virtual channels that can be roughly compared to Atom or RSS syndication feeds. A system with multiple channels is likely to have a very broad range of content. If each user were to carry only the content of personal interest, any two nodes would be relatively unlikely to acquire content of interest in a random encounter. Thus a notion of two distinct types of channels is built into the system: A node solicits content on its subscribed, or *private*, channels for its consumption. In addition, each device is required to cache an arbitrary number of *public* channels, whose purpose is to carry contents for the benefit of future contacts. This altruistic behaviour helps to maintain content diversity in the system as a whole. The PodNet makes no assurances regarding completeness or ordering of content items or constituent pieces and is thus a best-effort service.

1.3 Contribution

This dissertation presents work on a node design for PodNet. The special case of a *gateway* node is considered in particular. The gateway bridges the divide between the infrastructure world of the Internet and the ad-hoc domain by providing content procured from Internet sources to its peers. The infrastructure connection is here used as a means for injecting content into the ad-hoc domain, and thus increasing the potential for successful queries in a certain area. A possible deployment scenario is one where the gateway software runs on a high-capacity computing platform, equipped with a wired Internet connection for procuring content and a wireless interface for providing content to the ad-hoc domain. The power of current hand-held devices, however, is such that a gateway in the form of a handheld device with a highly available Internet connection is by no means precluded.

The contribution of this dissertation is as follows:

- A modular design for a gateway node is presented. The design is equally applicable to general purpose PodNet nodes, although the gateway is considered here in particular.
- A light-weight content solicitation protocol design is presented, based on the work described by Karlsson et al. (2007) and Lenders et al. (2007). The protocol is designed primarily for efficiency, rather than reliability and can be implemented at the link layer or above in the OSI stack.
- A simulation model, *podsim*, is presented. It implements a simplified version of the content solicitation protocol in a simulation setting. Multiple content channels are supported. Systems consisting of a fixed number of nodes in a square can be simulated using a random waypoint mobility model. In addition, a mechanism of dynamic node generation and mobility control was developed (Helgason & Jónsson, 2008), which enables more realistic mobility patterns.
- A prototype for a software package, implementing a multi-role PodNet node, is presented. The cache data structure, management functions and subscription services are represented in the prototype. It compliments the simulation model; together the two implementations present a cohesive proof-of-concept for the design work presented.

Full code and user manuals for the prototypes presented is publicly available.

This dissertation is organized as follows: Chapter 2 describes the background for this work. The PodNet system is introduced in Chapter 3. The system design for a PodNet node is then discussed in Chapter 4. A light-weight solicitation protocol, suitable for use in the PodNet system, is presented in Chapter 5, followed by a discussion of the simulation model in Chapter 6. Simulation results evaluating the performance of the simulator and the correctness of the protocol are presented in Chapter 7, along with some findings on the properties of a PodNet system, particularly regarding multi-channel behaviour. The prototype PodNet node software implementation is described in Chapter 8. The dissertation concludes in Chapter 9 with a summary of the work and some remarks on future directions.

Chapter 2

Background

The purpose of this chapter is to introduce the problem domain of wireless ad-hoc networking and some of the influential work which has provided inspiration for this dissertation and previous work done on the PodNet system.

2.1 Ad-hoc Networking

An ad-hoc network is generally defined as the direct interaction of two or more nodes, without the need for any intermediary routers or servers (Kurose & Ross, 2005, pp. 507). This dissertation discusses only wireless ad-hoc networking, although other means of transfer are certainly also applicable. Nodes in an ad-hoc network co-operate in a peer-to-peer fashion to exchange messages by forming impromptu relations on the fly. Ad-hoc networks can be used wherever quick and flexible deployment of a communications network is needed. An example would be connection between students in a classroom to share lecture notes, or medical and rescue personnel communications in a disaster area where local communications have broken down (Royer & Toh, 1999). This model can well compliment the more traditional infrastructure based approach, as enabling nodes to opportunistically exchange content may well help to distribute load off a cell or access point.

Pairwise connection between two peer nodes is the simplest model of ad-hoc relations. A more complex model is a multi-hop ad-hoc network, in which routing algorithms are employed to intelligently forward messages from a source to a destination. An example of a multi-hop ad-hoc network is shown in Figure 2.1. This is obviously a difficult problem, and often impossible in wireless networks. The wireless medium is notoriously error

prone and difficult in terms of quality of service (QoS) requirements. Fading effects, interference, blockage and unfavorable atmospheric conditions are just some of the problems conspiring to make the medium unpredictable, although ever more sophisticated methods are being developed to combat those problems. Allowing mobility provides another class of ad-hoc networks: *mobile ad-hoc network (MANet)*. Mobility obviously compounds the already difficult problem of maintaining some semblance of semi-permanent routes in an ad-hoc scenario.

Mobility of wireless nodes can, however, be used as an advantage for content distribution rather than the problem that it is commonly regarded as, in current infrastructure-based wireless research. As discussed by Grossglauser and Tse (2002), using mobility as an advantage can increase the availability of a wireless network. The family of Mobility Assisted Information Diffusion (MAID) protocols was discussed by Bai and Helmy (2005).

Employing node mobility and pairwise forwarding of content is often termed *store-carry-forward*. Nodes acquire and forward information during random contacts generated by their mobility. The network shown in Figure 2.2 is an example of the store-carry-forward approach, where opportunistic one-hop contacts are utilized to forward data. Restricting communications to take place over one hop essentially bypasses the routing problem of multi-hop wireless networks. Store-carry-forward can be employed to forward a specific message from a source to a specific destination (Juang et al., 2002); intermediary nodes often being oblivious to the actual nature of the content being carried. Sensor network applications commonly use inter-node contact to forward content towards a common sink. *Data mules* (Shah, Roy, Jain, & Brunette, 2003) or *ferries* (Zhao, Ammar, & Zegura, 2004) are in some cases employed to collect data from an often partitioned network of nodes and deliver to a sink for further processing, as shown schematically in Figure 2.3. These methods use mobility as an agent to collate contents from a number of stationary or mobile nodes to a common sink. The PodNet system, on the contrary, uses the mobility of users to disseminate the contents within a population, without the notion of sinks or sources. A gateway, as here presented, can be roughly compared to a source node in a data ferrying scenario, while the purpose is not to ferry the provided contents to a sink but rather to disseminate it in a population.

Epidemic forwarding models (Vahdat & Becker, 2000; Fawal, Boudec, & Salamatian, 2006) employ inter-node contacts to pass messages between nodes using a model borrowed from epidemiology. Nodes with information attempt to "infect" other nodes upon short and random pairwise encounters. Epidemic routing attempts to pass a message from a source to a specific destination through random pairwise contacts. This differs from our

work in that no routing is ever performed; nodes are assumed to pull the contents required.

Applications or systems which depend on individual mobility are of course very dependent on mobility patterns within a population. The *Haggle* project (http://www.haggle-project.org) investigates patterns in human mobility and how forwarding decisions can be optimized based on such predictions, as reported by Scott, Hui, Crowcroft, and Diot (2006); Chaintreau et al. (2006). Some findings of the Haggle project and various other networking experiments are collected in the *Crawdad* database (Yeo, Kotz, & Henderson, 2006) (http://crawdad.cs.dartmouth.edu). Other notable results include datasets collected by the *UMass DieselNet* project (Burgess, Gallagher, Jensen, & Levine, 2006). Pedestrian contact traces collected using Bluetooth devices are also reported in (Natarajan, Motani, & Srinivasan, 2007; Pietilainen & Diot, 2007).

An *Infostation* (Goodman, Borras, Mandayam, & Yates, 1997) is proposed in the *DataMan* project (http://www.cs.rutgers.edu/dataman). A ubiquitous low bandwidth wireless network is augmented with a number of isolated high bandwidth Infostations. The infostation concept is further discussed in e.g. (Frenkiel, Badrinath, Borres, & Yates, 2000; Cheverst, Davies, Mitchell, Friday, & Efstratiou, 2000; W. H. Yuen, Yates, & Sung, 2003; W. Yuen, Yates, & Mau, 2003; W. Yuen & Yates, 2003; W. H. Yuen, Yates, & Mau, 2003). Mobility awareness of infostations is discussed in (Ye, Jacobsen, & Katz, 1998). The gateway discussed here can be considered a special case of an Infostation, with a cache filled by various means and running the content solicitation protocol in the ad-hoc domain.

Peer discovery is an important subject in opportunistic networking. Contacts are in many cases short, and rapid discovery thus necessary to take advantage of the opportunities for useful content exchange. This is for example true of the PodNet system, at least in sparse scenarios. An energy efficient probing mechanism for wireless devices is presented by Wang, Srinivasan, and Motani (2007). This dissertation does not consider energy efficiency as such, but it is an actual problem which needs to be addressed in future work.

2.2 Peer-to-peer Networking

Peer-to-peer content exchange protocols, e.g. *Bittorrent*, *KaZaA* and *Gnutella* are widely used in the present day Internet and a considerable fraction of network traffic can be attributed to them. Some sources estimate that up to 35% of web traffic is due to Bittorrent

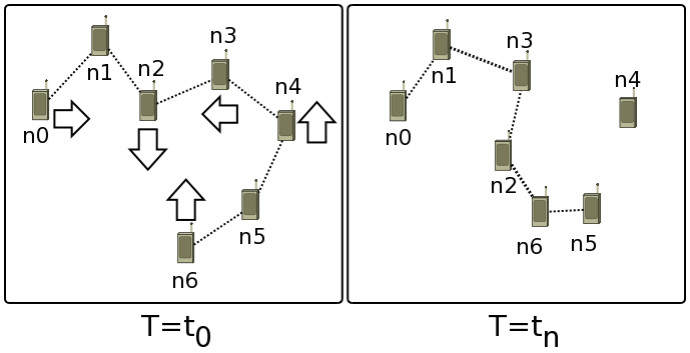


Figure 2.1: Multi-hop ad-hoc wireless network. The available communications paths may change as network nodes move or conditions change, making the configuration highly dynamic.

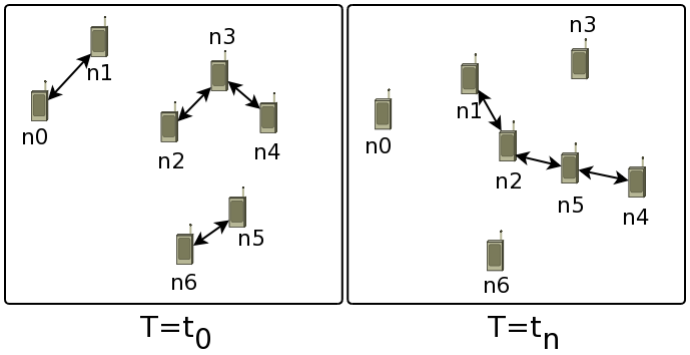


Figure 2.2: Mobility assisted content exchange over a single hop. The arrows show communications opportunities, during which content exchange of some sort can take place.

file sharing (Pasick, 2002). The stable load characteristics of peer-to-peer networks are discussed in (Gerber, Houle, Nguyen, Roughan, & Sen, 2003), noting a lack of geographic locality. A strong geographic locality can, however, be expected in wireless ad-hoc networks, as noted in the work on Infostations. The heavy load generated by a relative minority of the Internet population on peer-to-peer networks are discussed and a pricing strategy for discouraging abuse of these services proposed. The self-scaling properties of peer-to-peer networks are discussed in (Felber & Biersack, 2004).

Bittorrent (http://www.bittorrent.org) is perhaps the best known and widely used peer-to-peer filesharing protocol. It has recently gained some academic interest, e.g. as discussed by Izal et al. (2004). There are considerable similarities between the PodNet content solicitation protocol and Bittorrent, although the latter is designed to run as a wired overlay network, essentially disregarding geographic locality. The content solicitation protocol discussed here uses some of the terminology of Bittorrent, as also noted by Wacha (2007), and is similar, although less organized, in its behavior. Chunks of contents are spread amongst peer nodes in more or less random fashion, and assembled into a coher-

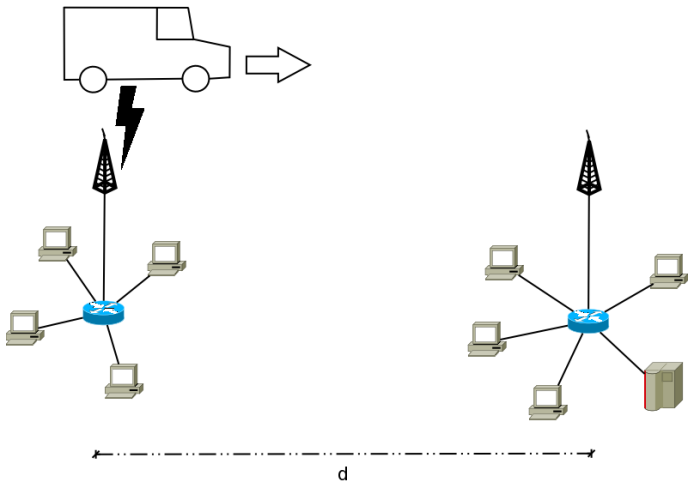


Figure 2.3: An example of a message ferry or data mule. Contents are in this case carried between isolated networks by a wireless equipped vehicle.

ent piece of content by the receiver. Both protocols are also receiver driven. The content solicitation protocol of the PodNet system cannot deliver content with any assurances of completeness. A transfer of a large file, however, is likely to succeed in Bittorrent, given a sufficient number of online *seeds*, i.e. peers sharing the required content.

Bittorrent is a *swarming* transfer protocol, meaning that packets are forwarded from a multitude of peers, thus potentially achieving better throughput and load distribution than more traditional client-server approaches. Swarming as a scalable content delivery mechanism is further studied by Stutzbach, Zappala, and Rejaie (2004, 2005).

Bittorrent is mainly notorious for its use in distributing copyrighted material without proper licensing. It can be argued that the protocol and its implementations merely provide the means of distribution for content, which the users can choose to use or abuse at will. There is nevertheless considerable resistance to the mechanism as such from copyright holders. The same principles apply to the PodNet system. Its properties are such that it can be employed for distribution of content as seen fit by any channel publisher, which may well include copyrighted material. This aspect is disregarded in the present work, but may well have to be considered in the future.

BlueTorrent (Jung, Lee, Chang, Cho, & Gerla, 2007) is a peer-to-peer file transfer application, whose goal is to employ ubiquitous Bluetooth (Haartsen, Naghshineh, Inouye, Joeressen, & Allen, 1998) devices to share content. A similar approach is taken in the Blue* project (http://www.csg.ethz.ch/research/projects/Blue_star), which has developed a range of applications using Bluetooth for peer-to-peer data exchange. There are, however, some problems employing Bluetooth, particularly the long connection and discov-

ery times (Karlsson et al., 2007), which severely limit the application of this particular technology to highly mobile users, which are the target population of the PodNet project.

The term *flash crowds* was originally coined by sci-fi author Larry Niven[1] to describe the effects of instantaneous transfer of people between locations, using the fictional concept of transfer booths. It has been employed to has been employed to describe the effects of an avalanche of requests to a particular server. This phenomenon is also known as the *slashdot effect*, originally used to describe the phenomenon of a small site being overwhelmed by requests as a result of being referenced by the popular technology news site *Slashdot*. It is now commonly used to refer to the same effect caused by other popular sites, which post links to items hosted on various servers. The resilience of peer-to-peer networks to flash crowds is discussed by Ari, Hong, Miller, Brandt, and Long (2003, 2004), where it is argued that the self-scaling and self-organizing properties of peer-to-peer systems make them highly efficient for content distribution and resilient towards flash crowds.

2.3 Delay-Tolerant Networking

The traditional wired computer networks provide almost perfect continuous end-to-end connectivity through the use of reliable high-speed nodes and multiple concurrent routes. This level of service is necessary for streaming multimedia, e.g. voice communications. but is not strictly needed for some asynchronous applications with generous tolerance for delays. The goal of continuous end-to-end connectivity is most likely unattainable in wireless ad-hoc networks, as previously stated. Some of the current communications paradigms that assume end-to-end connectivity will therefore have to be readdressed for the wireless medium.

An alternative paradigm to the traditional continuous connectivity approach, and one that is resistant to the inherent QoS problems of the wireless medium, is the relatively new research field of *delay and disruption tolerant networking*. This is the focus of the *Delay-Tolerant Networking Research Group (DTNRG)* (http://www.dtnrg.org). Topics covered in this field range over a variety of issues regarding networking and forwarding of information in extreme and performance challenged environments where the traditional model of end-to-end connectivity breaks down. In general, nodes in a network based on delay-tolerant principles store content on behalf of other nodes and forward at the next opportunity. The content being carried is assumed to tolerate very long delays and out of order

[1] Larry Niven, *"Flash crowd"*, Three trips in time and space, 1973

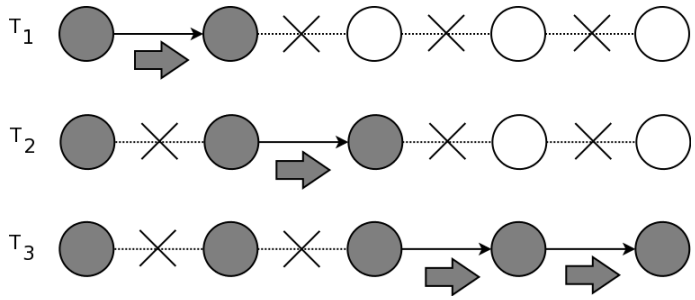


Figure 2.4: An example of opportunistic forwarding of delay-tolerant content. The network is composed of nodes which are periodically able to communicate due spatial relationships or other conditions.

delivery. No end-to-end assurances of completeness can generally be made. Any content that can be considered delay tolerant if its application does not pose QoS constraints on delivery time or ordering of its constituent pieces. An example of a challenged network in which nodes are only periodically able to communicate is shown in Figure 2.4.

An example of paradigms utilizing delay-tolerant characteristics are the ad-hoc and sensor networks which use *store-and-forward* and *store-carry-forward* techniques.

The problems associated with the wireless medium, in particular intermittent connectivity and its effect on mobile computing is discussed by Fall (2003), which presents the bundle protocol for optionally reliable message passing, using a transport layer overlay. Probably the most extreme environment for networking is space and interplanetary communications An end-to-end solution for delay tolerant interplanetary message passing is presented by Burleigh et al. (2003).

The broadcasting concept of the PodNet system is considerably simpler than the bundle protocol and related approaches, since it does not consider end-to-end message passing at all. Rather, all communications are pairwise over one hop only using the broadcasting paradigm where each node can be both a consumer and a forwarder of any given message.

2.4 Content Dissemination

The PodNet system is a mechanism for dissemination of delay-tolerant content to a population of independent mobile nodes. It may be better classified as *opportunistic networking*, rather than delay-tolerant. Similar mechanisms of content sharing have been considered in the recent years, and are summarized in this section.

The 7DS system by Papadopouli and Schulzrinne (2000, 2001a, 2001b) provides a mechanism to distribute contents in an ad-hoc manner between wireless nodes. A well-known multicast address is assumed on which mobile devices can exchange contents. A client can push a summary of its contents to this multicast group. A study on predicting base station association patterns of wireless users is discussed by Chinchilla, Lindsey, and Papadopouli (2004). In contrast, the PodNet system uses only one-hop contacts and no multicast address is thus necessary; a MAC address, UUID (Leach, Mealling, & Salz, 2005) or URI (Berners-Lee, Fielding, & Masinter, 1998) suffices to identify a node. A pull based approach is used exclusively and no push mechanism is thus necessary. An important difference is that the 7DS system is not designed to carry contents on behalf of other users. The PodNet channel concept also provides a coarser grained approach to content distribution, increasing the chances of a successful query. 7DS relies on a high degree of spatial locality of contents in pervasive computing environments, while we envision a much greater diversity of contents.

TACO-DTN (Sollazzo, Musolesi, & Mascolo, 2007) is a delay-tolerant content dissemination system. The concept of temporal utility of subscriptions and events is employed, routing events to the Infostation appropriate for the given subscriber. It uses a publish/subscribe approach, whereas our approach is purely receiver-driven.

Peoplenet (Motani, Srinivasan, & Nuggehalli, 2005) propagates information on a contact basis between mobile nodes, using a model of social networking. It is reliant on the fixed infrastructure and targeted at seeking information, rather than broadcasting, as is the intent of our system. Contact traces, including contact time, are analyzed in (Natarajan et al., 2007).

The drive-through Internet (Ott & Kutscher, 2004, 2005) employs intermittent WLAN coverage along a highway to provide intermittent network access using an infrastructure connected proxy system. It uses a drive-thru proxy with a reliable connection to the Internet and a transport layer proxy on the wireless node. The goal is to make applications unaware of the intermittent nature of the connection. A potential problem is the need for a supporting infrastructure, i.e. the two proxies, which could pose deployment problems and hinder widespread use of this technology. Another problem is that the update path from the fixed proxy to the mobile node could be arbitrarily long, posing delay problems. There is no exchange of contents between the mobile nodes (cars) in the drive-through Internet. Spawn (Das, Nandan, Pau, Sanadidi, & Gerla, 2004; Nandan, Das, Pau, Gerla, & Sanadidi, 2004) is a vehicular ad-hoc swarming protocol employing both gateways for content injection and node-to-node content exchange. A gossip protocol is used to

obtain information about content availability. Vehicular networks are further discussed by Nandan, Tewari, Das, Gerla, and Kleinrock (2006).

2.5 Node Cooperation

The importance of node cooperation in a delay tolerant network is discussed by Panagakis and Stavrakakis (2007), for a variety of routing algorithms. Although the PodNet does not rely on routing, this work nevertheless underscores the importance of cooperation. The PodNet system assumes cooperation implicitly, without considering any incentive mechanisms, by the design of the solicitation strategies and protocol, and the private/public cache concept. Compliance with the protocol is assumed in this work, in terms of user behaviour, user policies and the actual implementation of the protocol.

TCP is an example of a protocol which implicitly assumes "good" behaviour and does not use any incentives to do so (Kurose & Ross, 2005, pp. 228). This approach is based on the trust that implementers and end-users are willing to obey the rules of the protocol. This may be unrealistic, however, given the explosion of the Internet population and the increasing incentives for selfish behaviour. Methods for abusing the TCP congestion control mechanisms and ways to combat such behaviour are discussed in (Savage, Cardwell, Wetherall, & Anderson, 1999).

There may still be means for a PodNet user to control his or hers level of cooperation, e.g. the simple measure of turning the device off after having retrieved the desired contents. Some incentive based mechanisms may thus be considered in future work. The effect of cooperation in wireless content distribution networks is reported by Helgason and Karlsson (2008). A content distribution system is modeled using continuous time Markov chains. The study suggests that cooperation is most effective for channels with few subscribers. Fairness is thus dependent on cooperation in spreading less popular channels. A barter based method to promote cooperation is presented by Buttyan, Dora, Felegyhazi, and Vajda (2006). *Rational attacks*, i.e. selfish utilization of a protocol, are discussed by Nielson, Crosby, and Wallach (2005). Methods of countering *byzantine* (e.g. misbehaving or faulty), *rational* and *altruistic* nodes are discussed by Aiyer et al. (2005).

2.6 Summary

This chapter presented a brief summary of the background research for this dissertation. The PodNet concept is, however, the main foundation of this work, and will be summarized next in Chapter 3.

Chapter 3

The PodNet Architecture

This chapter describes the *PodNet* architecture in some detail. The previous work within the project is introduced, followed by a high-level description of the architecture presented in this thesis. The contribution of this dissertation to the architecture of the PodNet system is the node design described in Chapter 4.

3.1 Introduction

The foundation of this dissertation is the groundwork of (Karlsson et al., 2007; Lenders et al., 2007), which describe the architecture of a wireless content distribution system. This chapter describes this important previous work, which forms the basis for the PodNet project (http://podnet.ee.ethz.ch) and this dissertation.

A prototype implementation of a peer-to-peer content exchange protocol using IEEE 802.11 (IEEE, 2003) devices has been created at ETH in Zurich, as described in (Wacha, 2007; May, Wacha, Lenders, & Karlsson, 2007). The effect of cooperation in wireless content distribution systems is discussed in (Helgason & Karlsson, 2008). A system architecture for PodNet is outlined in (May, Karlsson, Helgason, & Lenders, 2007).

PodNet is a wireless personal broadcasting system for wireless dissemination of delay-tolerant content throughout a community of peers. Peer-to-peer content exchange is employed, using wireless ad-hoc methods. Participation in the community is entirely voluntary. Users may additionally depart any given region or turn off their devices or wireless interfaces at any time. Nodes participating in the system opportunistically take advantage of contact opportunities to exchange content. These opportunities are created by the node mobility, which is thus the principal force of content dissemination. A receiver-driven

content solicitation protocol is employed, where nodes communicate pairwise over one hop and exchange content. Consequently, sophisticated multihop communications protocols, where routes have to be constructed and maintained, are not needed.

Although no specific transport or frequency band is assumed for use by the PodNet system, the unlicensed *industrial, scientific and medical (ISM)* band is a likely match, providing free and open use of the system to all potential participants.

PodNet is designed primarily for a community of pedestrians and a rather slow node velocity is thus assumed. Typical pedestrian velocity is e.g. reported in (Wiseman, 2007). However, higher speed nodes, e.g. carried in vehicles, are not precluded and may be addressed in future work.

3.2 Channels

One novelty of the PodNet system is the organization of content, selected by a publisher, into virtual channels, each of which carries a collection of content. An analogy can be drawn with traditional radio channels; a user "listens" to a channel on its device, that is consumes the contents that are retrieved through peer-to-peer contact. The main difference is that contents on the channel can appear in any order and completeness of a feed or any individual item cannot be ensured since best effort services are assumed. The channels concept is inspired by Podcasting and syndication protocols like RSS or Atom (Nottingham & Sayre, 2005), although its application is by no means limited to those purposes. Rather, the purpose of PodNet is to provide a transport for content of any kind to application layer consumers, whose purpose is to finally interpret and display the content for end user consumption.

The PodNet system is envisioned to be a general purpose, open and free, broadcasting system. Some channels may be open to creation of contents, while others may be under editorial control of their creators. All users are able to access any published contents and are free to create their own channels. PodNet thus provides a useful alternative to the present licensed cellular systems, where publishing of content may be severely limited and even subject to censorship.

A user subscribes locally to one or more *private* channels on his mobile device, using the channel URI (Berners-Lee et al., 1998) or UUID (Leach et al., 2005). This subscription is strictly limited to the user's device; no global publish/subscribe concept is employed in the system. The channel identifier can be retrieved by using a search engine when connected to the Internet, or retrieved on a special common discovery channel; the zero channel is

built-in to enable a device to bootstrap into the network when turned on, even though it has no prior knowledge of available channels. All devices can respond to channel information queries, and may thus update their local knowledge at any time. Note, however, that a consistent set of available channels for all nodes in a population is unlikely, since each node may have encountered a different set of content providers, be it gateways or other mobile nodes.

A system with multiple channels is likely to have very diverse content. If each user were to carry only the content of personal interest, two nodes would be relatively unlikely to acquire content of interest in any random encounter. Thus the notion of *public* and *private channels* is built into the system: A node solicits content on its subscribed, or *private*, channels for its consumption. In addition, each device carries a number of *public* channels, whose purpose is to cache contents for the benefit of future contacts; solicitation strategies as discussed later can be employed to vary the participation in caching of public content. Note that this is not a forwarding of request, nor is the content cached for the particular user that generated the need, since the two are unlikely to meet in the near future. Rather, this mechanism can be viewed as building a local base of the most popular content. A node will thus with a relatively high likelihood be able to satisfy requests for the more popular content, after having participated in the system for some time.

No incentive or barter system is designed into this system, as is for example discussed by Buttyan et al. (2006). The altruistic behavior of public caching helps to ensure content diversity in the system as a whole, as discussed by Karlsson et al. (2007). A node benefits indirectly from carrying public content, although it has no direct interest in it, since it can assume that other nodes carry content it is interested in with significant probability. The premise of the system is that this is enough incentive for compliance with the prescribed protocol in the solicitation, caching and distribution of public content. The nodes thus exhibit *altruistic* behaviour by carrying public content for the benefit of the community as a whole. Participation in the PodNet community is however entirely voluntary, as stated previously. The behaviour of the users can thus be assumed to be somewhat *rational*, in the sense that they participate in the system only when interested in consuming content.

The channels concept of content distribution can be compared to the multicasting method of content distribution in the wired world, although no specific groups are formed and no subscriptions are required. Rather, the user selects a set of channels to consume from a publicly available pool of channels.

Each node maintains access count for channels and individual content items, providing a local approximation of their popularity.

Organizing the content into channels provides a coarser grained target for queries, and thus increases the chances of a successful request, rather than if a specific content item would have to be requested. Success is relatively unlikely in the latter case, unless content is highly localized, whereas any member of a population is likely to have some content of interest on a particular set of channels. The user controls the level of specialization of the request and hence the likelihood of a successful retrieval. This is further described in (Lenders et al., 2007). The most general request would be for any content on any channel. A more useful but slightly more restrictive request would be for any content on a specific channel. The granularity of this request is such that a random node in the population is likely to be able to service it. Some further restrictions can be placed on the request, such as specifying a certain maximum age for the content item. The most restrictive type of query, and the one least likely to succeed, would be for a specific item of content. An exclusion list can be provided with a request to prevent wasted bandwidth due to transfer of duplicate content.

3.3 Content

The content to be distributed is assumed to be *delay-tolerant*, that is, only very lenient restrictions may be placed on its delivery time. Further, no assurances are made regarding completeness or ordering of delivery. A best effort transport is thus assumed for content delivery, in keeping with the nature of the wireless medium and the working conditions of the devices in general. No other restrictions are placed on the type of content.

The gateway is not the only means of content provision into a population. For example, mobile device can obtain content when docked at a desktop computer or when connected to the infrastructure via a Wi-Fi or Bluetooth connection. Users can also create content directly on the device, e.g. by writing blogs published on a dedicated channel.

PodNet is designed for distribution of rather small items of content, preferably tolerant of dropouts, as no assurances can be given for delivery of a complete item. Suitable content is for example

- audio files, e.g. mp3 encoded music or speech.
- text files, e.g. blogs, news items, poems or web pages encoded in HTML.
- image files, e.g. jpeg compressed photographs or graphics.
- small binary files, e.g. software updates. Binary files should be rather small, so that an entire file can be transferred with high probability within a single average

inter-node contact. An example application of such updates is distribution of city maps to arriving passengers at airport terminals.

A channel can contain a mix of any of the categories. The radio station analogy mentioned earlier is one example were a mix of news, advertisements and music could be published. Some other applications include traffic reports interactively created and consumed by users or blog entries and exchange of comments. This content is assumed to be non-critical and a best effort service thus sufficient.

Content is fragmented into a number of smaller units, the smallest fitting into a linklayer frame. This can be viewed as proactive fragmentation, promoting fairness and increasing the potential throughput of the system. Use of short range radios implies that contacts will be short. A relatively large number of small pieces, rather than a few large ones, means that each node participating in a content transfer is on average likely to be able to solicit and receive equal amount of content. Similarly, small content fragments mean that less content is on average wasted when contact is lost. The terminology of content organization outlined by Wacha (2007) is used in this thesis:

Episode An *episode* is the actual content item and may consist of a number of *enclosures*.

Enclosure An *enclosure* is an attachment to an episode that consists of one file, e.g. an HTML file, image or audio file.

Chunk The *enclosures* of an *episode* are combined and then divided into several *chunks*. This can be compared to the fragmentation employed by Bittorrent. In the PodNet system, the chunks are passed between wireless peers during short and possibly infrequent encounters, while in Bittorrent the dynamic overlay network is used to maintain TCP connections for extended periods of time, at least on the timescale of the ad-hoc world.

Piece A chunk may be larger than allowed by the link layer transport, which requires further fragmentation into *pieces* which fit within a link layer frame.

3.4 Solicitation Strategies

The PodNet system utilizes private and public caches to aid in content availability and diversity in the network, as discussed previously. The strategies employed to choose which channels to solicit content for are thus of interest to the performance of the system as a whole, especially regarding the public channels.

A number of solicitation strategies are outlined in (Lenders et al., 2007). All are based on the premise that a node will first solicit all available content on its private channels before soliciting any public content. The defined strategies are:

No caching essentially disables the public cache causing the peer to exhibit purely selfish behaviour.

Most solicited prioritizes the solicitation of public content by the respective channels popularity. This strategy aims to increase the probability of successful future solicitations by prioritizing popular content.

Least solicited is the inverse of most solicited. Its purpose is to prevent extinction of less popular content.

Inverse proportional solicits content with a probability that is inversely proportional to a channel's local popularity. The purpose of this strategy is to mitigate the potential of the least solicited strategy for filling the public cache with unpopular feed items that may never be requested in future encounters.

Uniform prioritizes all public channels equally, soliciting randomly on all known public channels. Note that no popularity metrics are required for implementation of uniform solicitation. This strategy produced the optimum performance in the trials described by Lenders et al. (2007).

An additional solicitation strategy, *weighted solicitation*, is described in Section 4.2.1.1. This strategy varies from those listed above by using a probabilistic approach to soliciting content on private and public channels, rather than favouring strictly the private channels, as the aforementioned strategies do.

3.5 Design Parameters

The behaviour and content exchange performance of a PodNet community are largely defined by the following factors:

- Topology and population dependent factors.
- The radio range and other medium access aspects of the transport in question has considerable effect.
- The contact setup time, including node discovery, negotiation and session setup.
- The efficiency of the content solicitation protocol.

The first three factors listed above are largely outside the influence of the system designer. Topology and population factors can be influenced by strategic placement of gateways and caching nodes, but is otherwise difficult to control. The radio range of the transport in question has considerable effect. Short range radios produce relatively few contacts but at the same time cause little interference. On the other hand, long range radios produce more contacts but more interference. Short range radios are thought to be more efficient for content dissemination, given that they produce many short contacts with relatively little interference. This can be compared to the cellular phone versus traditional two-way radio communications. Discovery and contact setup times are very important, since encounters with peers must in general be assumed to be short. A long setup time means that a larger portion of the valuable contact time is wasted. Very long contact setup times of up to 10 seconds are reported by Karlsson et al. (2007) for Bluetooth devices. This is most likely unacceptably long for the purposes of the PodNet system. Discovery and connection setup times can be considered to be twofold: at the hardware and link layers and the level where the content exchange agent resides. The factors within the influence of the system designer should certainly be minimized. Radio range, discovery time and contact setup times are difficult to control unless the system is being designed from the ground up. A more likely scenario is that a ubiquitous technology, such as IEEE 802.11 or Bluetooth, would be employed.

The efficiency of the content solicitation and exchange protocol is of prime importance and the one factor that can actually be engineered by the system designer. A careful design must make the most of the opportunities for content exchange, that is the period from when nodes discover each other and until they are out of contact. During this time, nodes must exchange content information and as much actual content as possible. Overhead must thus be minimized.

3.6 Summary

The background of the PodNet system was introduced in this chapter, summarizing the most important features of the previous work by Karlsson et al. (2007); Lenders et al. (2007) for the purposes of this dissertation. The following chapter will now introduce the design of a PodNet node, based on these principles.

Chapter 4

PodNet Node Design

This chapter describes the system design of a PodNet node, which is the object of this dissertation. The base roles of a PodNet node are first introduced and discussed. A single universal node design applicable to all the base roles and their hybrids is then presented, providing added flexibility and allowing a PodNet node to take on diverse roles. Design, implementation and deployment issues are introduced and discussed as occasion arises.

4.1 PodNet Node Roles

A PodNet community consists of a collection of peers, whose roles can be roughly categorized into the following three base roles:

Roaming nodes are the prevalent node class in the population, and are generally portable wireless devices, e.g. mobile phones, PDAs or laptop computers.

Gateways provide an infrastructure bridge to the ad-hoc world by means of a permanent (or highly available) Internet connection.

Caching nodes provide added stability to the content availability in an area by soliciting and caching any content from passing nodes. This node role is defined in (Karlsson et al., 2007).

Figure 4.1 shows a schematic diagram of a PodNet community with a number of mobile nodes. A gateway pulls content off the Internet and makes it available to its peers in the ad-hoc domain. Additionally, a fixed caching node stores content from passing roaming nodes for later dissemination.

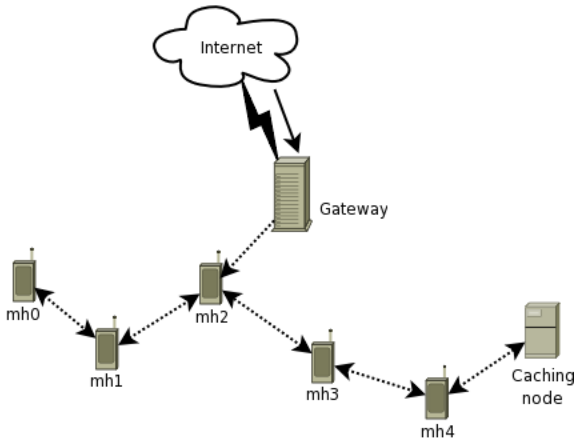


Figure 4.1: A schematic example of a PodNet community. The diagram shows a gateway with Internet connection, five mobile nodes and a single fixed caching node. All nodes participate in opportunistic content exchange.

4.1.1 Roaming Node

The main players in the system are the mobile devices, or *roaming nodes*, which are carried by users who move freely within a region. Pedestrian mobility is assumed for PodNet applications in this dissertation, as discussed in Section 3.1. Each device runs the content solicitation service and exchanges content in an opportunistic manner upon encounters with other nodes. Application layer consumers then process, format and present the content at the request of the user. Content is brought into the system by several means. A mobile device can for example obtain content when docked at a desktop computer or when connected to the infrastructure by a Wi-Fi or Bluetooth connection. Users can also create content directly on the device, e.g. write blogs and make publicly available on a dedicated channel.

4.1.2 Gateway

The *gateway* is a content provider, whose purpose is to make content procured from the infrastructure network available in the ad-hoc domain. It can thus be viewed as extending the reach of an infrastructure-based network by ad-hoc means. An Internet connection is required, which makes the gateway best deployed as a fixed platform with a wired Internet connection. A gateway should also be highly available, which requires high capacity in terms of power and storage. This further supports the fixed platform scenario. Note, however, that the deployment of a handheld gateway is by no means precluded, as will be discussed later on.

Contents are brought into the system by the gateway, either by pre-fetching or on-demand when solicited by a roaming node. An obvious choice of content are podcasts and other syndication services, although other types of content can certainly be envisioned. The contents from the various Internet sources are mapped to one or more channels in the ad-hoc domain. The gateway administrator controls this mapping and thus decides the channels and influences their contents. Gateways do, however, not solicit any content from their peers in the base role, although more complex behaviour can be envisioned as discussed later on.

A fixed, high-capacity platform is primarily envisioned as the means of deployment of a gateway node, as previously noted. The storage capacity of such a node is large compared to that of the handheld nodes. It is nevertheless not infinite and cache maintenance techniques must be carefully considered to maximize the availability of popular content.

An interesting phenomenon shown in (May, Karlsson, et al., 2007) is that content can be persistent in an area, given sufficient arrival rate. That is, a node entering an area has a high probability of obtaining content, even though the nodes that originally brought the content into the area have departed. An area can thus be said to have information storage properties, or provide *virtual storage*, given a certain minimal arrival rate and density of nodes in the system. Surprisingly low arrival rates are required to maintain persistent content. However, when the arrival rate of nodes and/or density becomes too low, the content will disappear if further measures are not taken to maintain it. Such measures may include adding gateway nodes with connection to the infrastructure network, or a caching nodes as described in the next section.

4.1.3 Caching Node

The purpose of a *caching node*, defined by Karlsson et al. (2007), is to cache contents from roaming nodes for redistribution, thus enhancing the content availability in an area. Their purpose is primarily to retain content in an area at times when the density of roaming nodes is low. Caching nodes are generally fixed, high capacity platforms, similar to a gateway node. Caching nodes do not subscribe to any channels, but collect available content from passing nodes and store for later dissemination. In effect, they carry only public channels.

4.1.4 Hybrid Roles

Although three general types of nodes are defined, the roles of nodes in the PodNet system are not necessarily as strictly defined. A number of hybrid roles can be envisioned, which can be regarded as an intersection of the base roles, as defined above. This subject is discussed in this section.

The primary difference between a roaming device and a gateway as defined above is that the handheld has a single wireless interface, but the gateway has in addition a wired infrastructure connection. This is however not necessarily the case. Handheld devices are becoming more capable and, more often than not, they have a multitude of wireless interfaces. A mobile device might thus have both a short-range wireless interface for participation in the ad-hoc domain and a 3G interface, able to connect to the Internet with minimal delay. Such a device may be viewed as a miniature gateway and is functionally not distinguishable from a fixed system.

A mobile node can certainly take advantage of contact opportunities to the infrastructure, e.g. through available IEEE 802.11 base stations, to procure content of interest. This model can be used as an alternative to the gateway model discussed in this dissertation. However, the gateway has the added benefit of transparency; a node procuring content from a gateway does so in the same manner as when communicating with any other peer in the system.

Although the content distribution system that is described here is largely designed to be altruistic, a simple extension with more selfish motivations can be envisioned. A cost-of-patience scheme can be set up such that a policy is set for the balance between the virtue of patience (waiting for content to arrive on the peer-to-peer network) and the cost of going on-line through a 3G interface and retrieving the content immediately. The retrieved content would then presumably be made available to other passing nodes in the ad-hoc domain. Such a node is essentially acting as a gateway, although the retrieval policies are entirely up to the whims of its carrier. A handoff scheme can also be envisioned in which nodes connect to an infrastructure, e.g. Wi-Fi hotspot, if available, or share content on the same interface in ad-hoc mode otherwise. A node with multiple wireless interfaces could also potentially use several interfaces in the ad-hoc world, e.g. Wi-Fi and Bluetooth, either using policies to select the optimal one at any given time or potentially maintaining several simultaneous connections. A short-range, high-speed interface, could for example, be a better choice than a lower-speed, long-range one. If no contacts appear for a given time in the shorter range, it would be beneficial to search for contacts on the longer range interface.

Apart from the infrastructure connection, the primary difference between the base roles of a gateway and a fixed caching node is that a gateway is strictly a provider, and thus never requests content from its peers. A gateway can however easily take on the role of a caching node, if enabled to request content. A caching gateway hybrid can therefore easily be configured using deployment- or run-time specified policies.

4.2 PodNet Node Design

The focus of this dissertation is the design and implementation of a *gateway* node. This is however unnecessarily restrictive, given the potential hybrid roles of a node as defined previously. Rather, it motivates a design where the role of any given node in a PodNet community can be shaped at design, deployment or run-time. The configuration can take place at design time as the programmer assembles the needed modules into a software package for a given platform. Configuration and user policies can also shape the role of a node at deployment and run times. This is the approach taken in this dissertation. Any given participant can take on a role that can be viewed as an intersection of the base roles defined previously. The gateway role can thus be viewed as a special case of a general PodNet node, defined by deployment platform and configuration, rather than a specialized implementation. This dissertation thus describes a unified PodNet node design. The role of a gateway is however considered in particular.

The main components of a PodNet node are shown in Figure 4.2. A number of subsystems are contained in an executable, compiled for the target platform. The subsystems are described shortly in the following sections. The executable runs transparently as a background task, and is potentially one of many processes running on the target platform. It must communicate with the operating system and other processes through well-defined interfaces. The *User applications interface* provides access for applications such as news readers and music players running on the device, which need to access the retrieved contents. The user applications interface is not addressed further in this design, but a likely implementation is an API providing the required services, or an interface layer like COM or CORBA. The *Host platform interface* enables the system to communicate with its hosting platform, e.g. to obtain Internet services. This can in general be assumed to be handled by the operating system services and the standard socket APIs.

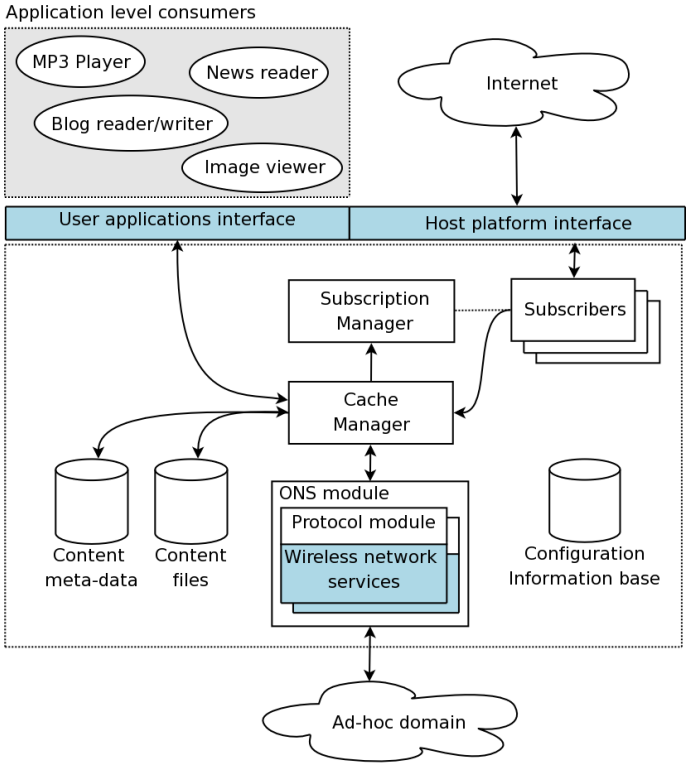


Figure 4.2: PodNet node modules

4.2.1 Opportunistic Networking Services

The *opportunistic networking services (ONS)* module runs the content solicitation protocol and provides the interface with the ad-hoc domain. An array of such modules may be employed, each handling a specific wireless interface. A common deployment scenario, given the present wireless technology, would be to support IEEE 802.11 and Bluetooth devices simultaneously. A design for a solicitation protocol, suitable for use in the ONS module, is described in Chapter 5.

The PodNet system is designed to be deployed on a variety of platforms and the protocol module should therefore be independent of the actual transport mechanism in use. An abstraction layer, *wireless networking services*, isolates the protocol module from the specifics of the transport protocol employed. This design was inspired by (Bianchi, Tinnirello, & Scalia, 2003). The actual wireless transport is thus not considered as such in this dissertation. This is a simplifying assumption, intended to limit the scope of this work. Possible scenarios must nevertheless be briefly considered. A UDP/IP transport over IEEE 802.11 is a viable high-level transport, widely supported and easily implemented. A custom link layer implementation over IEEE 802.11 or COTS radios is equally plausible. Reliable transport can be implemented by running the solicitation protocol over TCP,

although its features are rather redundant for this task and may even hinder performance (Gerla, Tang, & Bagrodia, 1999; Holland & Vaidya, 2002). A TCP based implementation of a podcasting content distribution protocol, based on (Karlsson et al., 2007; Lenders et al., 2007), is presented in (Wacha, 2007; May, Wacha, et al., 2007). This design would however need to be refactored to follow the model discussed here.

The minimum assumed capabilities of the wireless networking services layer are:

- It handles discovery of peers in the wireless domain and provides notification of peer discovery and contact breaks. At minimum a single notification should be given to upper layers at the time of discovery or contact break for a given node.
- It handles services for broadcasting (if the transport in question allows) or unicasting messages to peers.
- It handles services for receiving messages from peers.

Content is proactively fragmented into chunks and pieces as described in Chapter 3. It may be possible to disregard the payload size limitations of the transport in use, assuming the encapsulating layers are capable of handling fragmentation. In the case of UDP, fragmentation is generally handled at the IP layer. Fragmented UDP datagrams are usually discarded if a single fragment is lost (Stevens, 1994, sec.11.5) A content piece larger than the possible payload would clearly be an disadvantage in this case. This implicit fragmentation support is in general unavailable for a link layer protocol. The piece size should therefore be chosen small enough to fit within the payload limits of the transport in use. This functionality can however be integrated into the HAL wrapper described above, isolating the protocol module from the transport details. The hazards related to datagram fragmentation are further described by Kent and Mogul (1995).

4.2.1.1 Weighted solicitation

Several solicitation strategies were described in Section 3.4. A common element is that all strategies solicit all available private content first, before considering public content. This approach may, however, prove to be less suitable for a system with a provider, like the gateway. The private channels of nodes which happen to come into contact with a gateway would be provided for, while starving the public channels. Lets consider a simplified scenario of a system with a single gateway as the sole source of content, which has infinite content available on all defined channels. Lets further assume a sparse scenario, in which nodes meet infrequently. A node n_A coming into contact with the gateway will solicit content solely for its private channels, when employing any of the strategies de-

scribed in Section 3.4. Now, n_A encounters n_B, which has not previously encountered any node. Solicitation of content by n_B from n_A is relatively unlikely to succeed. n_B can obtain content content of interest with $p \approx \frac{1}{N_{private}}$ from n_A; a solicitation is therefore only successful if the two roaming nodes n_A and n_B carry an intersecting set of private channels. Another potential drawback is the bias towards private channels created by the expected short contact times. Requesting all available content on a private channel may well take up most or all of a contact and thus starve the public channels carried. The strategies listed above can thus degenerate to selfish behaviour in a considerable percentage of contacts.

An additional solicitation strategy, *weighted solicitation*, is therefore proposed. In weighted solicitation, a node will probabilistically solicit content on all carried channels. The probability of providing a chunk from a private channel $n_{pr,i}$ or a public channel $n_{pu,i}$ can be expressed as:

$$p_{pr,i} = \gamma \cdot \frac{\alpha_i}{\alpha_S}$$
$$p_{pu,i} = (1-\gamma) \cdot \frac{\rho_i}{\rho_S}$$

where $0 \leq \gamma \leq 1$ is the relative weight of the private versus public cache, and $\alpha_S = \sum \alpha_i$, $\rho_S = \sum \rho_i$ are the aggregate weights of the private and public channels, respectively. Weights are set on each device independently, thereby modifying the nature of the nodes' participation. Setting $\gamma = 1$ is equivalent to a selfish, or none, solicitation strategy, while $\gamma = 0$ is a purely altruistic mode. The level of rationality can thus be modified by varying γ between 0 and 1. This technique may result in a fairer distribution of content, as well as richer content diversity. Starvation of public channels can be avoided, while at the same time providing adequate private channel service, by choosing appropriate private/public channel weight γ. Further evaluation of this strategy is postponed until Chapter 7.

4.2.2 Cache Manager

The *cache manger* handles the content cache and the associated channels structure. The contents reside in two databases, the meta-data and contents databases. The former contains the meta-data on the specific content items and must therefore be easily searched and accessed. A relational database is therefore a natural choice for its implementation, as further shown in Chapter 8. A simpler implementation for the meta-data store is an in-memory datastructure, e.g. the STL-based implementation described in Chapter 6. The content store is generally a simple directory structure in the file system supported by the

target OS. The content can either be stored as complete *enclosures* (see Section 3.3) or pre-fragmented into the constituent *chunks* or *pieces*. The storage format of the content is seen as a policy decision at run time; a general peer would presumably rather store items as complete as possible, since its purpose is to consume them, while a gateway might benefit from pre-fragmenting enclosures for increased performance.

4.2.2.1 Channels

The private and public channel categorizations described by Lenders et al. (2007) are employed in this design. Usage statistics are collected in order to provide relative popularity metrics, as needed for cache maintenance and some of the solicitation strategies described in Section 3.4. In addition, channels can be assigned *weights* employed in the weighed solicitation, described in Section 4.2.1.1. Private channels have weights assigned in accordance with the users preference, while public channels would rather use the relative channel popularity. Channels can also be assigned timeout values, which can for example be employed by cache management algorithms to remove entire public channels and their contents if not accessed for a period of time.

Each channel can be of the following types: *Feed channels* are modeled after podcasting channels. They contain multiple unique content items, each which is typically not versioned as such. *Update channels* provide typically versions of the same content item, e.g. a software update or alarm. An item on such a channel carries a version number and timestamp. A node would thus compare the offered version number on any update channel to its own, prior to solicitation. Software updates and alarms are typically small, preferably so that each one can be transferred in its entirety within an average contact. Priorities can be assigned to increase the successful solicitation of update channels.

4.2.2.2 Cache Maintenance

Cache management and replacement strategies must be carefully considered, since device memory is in general limited, especially when considering the potentially huge availability of public content. This section considers cache management algorithms for use in a PodNet node to keep the size of content within manageable bounds. An implementation is described in Chapter 8.

The objective of the cache management algorithms is to remove some items from the cache, once it reaches a predetermined size limit. The selection of items to evict is though a rather difficult problem. The most logical choice for replacement are:

- private items, which have already been consumed and are not needed in the future.
- the least popular public items.
- the least recently requested items.

Popularity is here defined as the locally accumulated relative access count of channels and content items. Removing the least popular content items carries the obvious risk of exterminating least accessed content from the system; a relatively unpopular content item may still be of interest to a small fraction of the active solicitors. A better strategy may be to remove the least recently requested items, on the assumption that they are least likely to be solicited in the near future. Finally, some content may have associated expiration or staleness dates. It is logical to remove stale items once the cache has reached its maximum size. The criteria to use for determining which content to evict is though seen as a per-channel policy decision and not considered further here.

Public channels can in addition time out in their entirety, at which time they and their associated content are marked for removal. This measure prevents "dead" channels from being carried indefinitely by a device. Items associated with private channels can optionally be marked for removal after being consumed. Further, a private channel, whose subscription is revoked, becomes a public channel, and the associated policies apply.

The algorithm for cache replacement is as follows:

The content cache is reviewed, either periodically or by an explicit user action. Some items need to be evicted from the cache once it has reached an upper limit, C_{max}. We consider the private and public caches separately, and apply two limits $C_{max,pu}$ and $C_{max,pr}$.

Content in the *private* cache is not considered for removal, unless marked as consumed and removable. Instead, the user should be notified, once $C_{max,pr}$ is reached, that memory is low and cleanup is needed.

N items are selected for eviction from the *public* cache, if the cache size has reached the upper limit C_{max}. Items marked for removal can be safely evicted. Other selection criteria include staleness, least popular or least recently accessed. We can consider a two level eviction scheme if the node in question has a highly available Internet connection:

1. Remove the actual content item in the first pass, on the assumption that we can obtain it in a reasonable time if requested in the future. The meta-data is retained for the time being.
2. Remove the meta-data if the associated content item has already been deleted. It is however reasonable to favour removing content items, if the meta-data is small

compared to the average content item. This can be done by maintaining a n_{chance} counter which is incremented each time a meta-data record is considered for removal; the meta-data would then be removed when $n_{chance} \geq n_{max}$.

In essence, the two-level eviction scheme can be used in a gateway node, since it has by definition a highly available Internet connection. A typical peer, which obtains all content by peer-to-peer contact, would however most likely be better served by removing items along with their associated meta-data in one pass.

Note that a gateway node typically carries only public content, so eviction strategies can be automatically applied to its entire cache. The gateways cache can thus be maintained at a preset level, without any supervisor intervention.

4.2.3 Subscriptions Subsystem

The *subscription manager* handles interaction with the infrastructure world. It is the only component of the PodNet node that can be considered to be gateway role specific. A subscription manager handles one or more *subscribers*, whose purpose is to interact with a specific type of provider. Each subscriber is in this design a simple script, employing a software library for standardized access to the content cache. A prototype library, subscription manager and subscriber are discussed in Chapter 8.

The subscription manager is in its simplest form a scheduler, periodically triggering a subscriber to update content from a specific source. This can be compared to the function of common newsreaders or podcatchers. This is the approach taken in the prototype work of this dissertation. More complex adaptive or on-demand schemes can be envisioned, but are reserved for future work.

The PodNet concept is inspired by podcasting as implied by its name. Natural sources of content are thus podcasts and other XML based feeds, like those published using RSS and Atom syndication protocols. Other sources of content can however be envisioned. The subscription system should be flexible enough to be able to handle a broad range of content sources, preferably without recompiling its code. This goal is attained by the script-based approach, which makes subscribers easy to implement and adapt to various content sources. A subscriber for RSS and Atom feeds is described in Chapter 8. Other potential sources are e.g. FTP/HTTP servers and multicast subscriptions. The relatively easy task of creating subscribers for these and other forms of content providers is reserved for future work.

4.3 Summary

A high-level PodNet node design has been presented in this chapter. The next chapters of this thesis describe further design and implementation. A light-weight content solicitation protocol, suitable for use in the PodNet system, is described in Chapter 5. A two-pronged approach is taken with the prototype implementation. The main prototype is presented in the form of a simulator, *podsim*, in Chapter 6. The simulator implements the solicitation protocol of Chapter 5 in a simplified form. It thus primarily functions as a testbed for the protocol implementation and experimentation with various deployment scenarios and mobility parameters. The simulation model does, however, not address some important issues, for example the content subscription subsystems. The cache implementation is also rather rudimentary. A proof-of-concept initial prototype for a PodNet node is therefore presented in Chapter 8, where an implementation of a content cache, subscription system and a node command and control set is discussed. The ad-hoc solicitation protocol and wireless transport is however disregarded in the prototype and reserved for future work.

Chapter 5

Content Solicitation Protocol

This chapter presents a design for a light-weight content solicitation protocol, suitable for use in the PodNet system, based on the principles described by Lenders et al. (2007). The purpose of the protocol is to provide efficient services for querying, requesting and retrieving available content from an encountered peer on an ad-hoc basis.

The PodNet architecture assumes a pedestrian mobility model and short-range radios. Contacts are therefore due to occur randomly and be of widely ranging duration. It is thus important to maximize the efficiency of the solicitation protocol to utilize the brief and infrequent periods for useful content transfer, The protocol is receiver driven, meaning that pushing or flooding of contents is not employed. Nodes request contents on the channels of interest, and respond in kind to the best of their ability. Fairness of exchange is important in a peer-to-peer protocol, and is ensured in this protocol by requiring peers to take turns soliciting and providing contents. Nodes take advantage of opportunistic contact with peers in a PodNet community to exchange contents over one hop. Mobility is thus used as the primary means of dissemination of contents. Routing is not required, greatly simplifying the protocol.

The remainder of this chapter is organized as follows: The protocol is introduced in Section 5.1 and then described in detail in Section 5.2, using an ad-hoc message passing model. Extensions for enabling the pairwise mode are then discussed in Section 5.3. Utilizing broadcast for content dissemination is discussed in Section 5.4. The protocol is analyzed in Section 5.5 and the discussion concluded in Section 5.6.

An implementation is presented in Chapter 6 in the setting of the *podsim* simulation model. Further details on the protocol are presented in Appendix A.

5.1 Introduction

Efficient content transfer between PodNet nodes depends on taking maximum advantage of the brief contact opportunities presented as a result of the participants mobility. Protocol efficiency depends primarily on reducing overhead, including initial negotiation and control messages, to the bare minimum. This was previously discussed in Section 3.5. The approach taken here is to use a very simple light-weight protocol where the amount of content transferred is maximized, at the expense of reliability guarantees. An additional advantage is that a simple protocol is easier to understand and implement than a complex one.

Minimal peer state is maintained locally on each node for known peers; no synchronization of state, handshaking or acknowledgements are employed. Since all contacts may be broken unexpectedly and without any chance of mutual state cleanup, all states must be soft, i.e. time out after a reasonable period of time.

The QoS issues of the wireless medium heavily influence the reliability of content delivery. An elaborate protocol can be constructed, e.g. using the model of TCP, to guarantee delivery of messages. This would unavoidably involve considerable overhead in the form of acknowledgements and retransmissions. Another important factor is the unpredictable node mobility and its effects on the duration of contact. This cannot be accounted for in the protocol, since each user moves about autonomously and without any knowledge of ongoing transfers. The transient and unpredictable nature of the contacts may well be such that an elaborate reliable protocol would be an wasted effort, considering the added overhead, and hence reduced throughput. The protocol presented here is best-effort in nature, i.e. no measures are taken to ensure message delivery. Since applications consuming contents have varying requirements and tolerance of dropouts, the task of implementing a reliable transport mechanism of any kind is outside the scope of this work. Minimal timeout based retries at the protocol level are, however, employed to increase efficiency.

Fairness of content exchange is ensured by the peers taking turns soliciting and providing contents. The length of each turn is unspecified, but should be such that on average equal number of content pieces can be supplied to each node during a typical contact.

Solicitation strategies are discussed in Chapters 3 and 4. The strategies can however be viewed as independent of the solicitation protocol itself; a node solicits contents according to its own criteria and utilizes the protocol to communicate its wishes to its peer and to receive the available contents.

Two distinct modes of communications are considered. The first one, *ad-hoc message passing*, is conceptually simpler and thus discussed first. This mode allows a node to send solicitations to any known peer. Similarly, a node can supply content to a number of peers. There is no notion of a session or any kind of exclusive pairing in this mode. The second mode discussed is the pairwise paradigm described by Karlsson et al. (2007); Lenders et al. (2007), in which nodes form an exclusive pairwise association for a period of time.

The protocol is designed to be independent of any underlying transport solution, thus enabling the system to be deployed in a variety of settings. The remainder of this chapter assumes minimal transport support, mainly in the form of medium access; at minimum RTS/CTS collision avoidance is required. The simplifying assumption of a guaranteed mutual exclusion condition on the radio medium is in fact made.

A protocol for use in the PodNet system is described by Wacha (2007) and May, Karlsson, et al. (2007). It was designed for TCP/IP over a IEEE 802.11 wireless transport, and uses a client/server paradigm. The congestion and flow control mechanisms of TCP are, however, superfluous over one hop, and may indeed degrade performance over wireless links (Gerla et al., 1999; Holland & Vaidya, 2002). Conversely, the protocol described here is designed to be independent of the underlying transport and to use a peer-to-peer, rather than a client/server, model. It is also designed to be considerably lighter in weight.

5.2 Protocol Description

This section describes the protocol in some detail. The messages of the protocol are described briefly in Table 5.1 (refer to Appendix A for further details). The *ad-hoc message passing* communications model assumed in the discussion of the protocol in this section. The following Section 5.3 considers the necessary extensions for the pairwise paradigm described by Karlsson et al. (2007); Lenders et al. (2007).

The ad-hoc message passing mode is a very disorganized model of communications. Its mechanisms are, however, conceptually simpler than that of the pairwise one. It is thus used for the initial protocol description. In the ad-hoc message passing mode, a node can solicit contents from any known peer. Similarly, a node can supply contents to any of its known peers in response to their solicitations.

Node will be used in the following description when referring to the hosting node of a collection of state-machines, *peer* will be used when referring to its known neighbors, each of which is represented by a state-machine in the node.

Message	**Description**
Announce	Announce a node's presence and capabilities to a peer node
Pairing request	Requests a pairing with a discovered peer node
Pairing response	Response to a pairing request
Content request	Request content from a peer node
Content advisory	Advises a requesting node that all or part of a request cannot be fulfilled
Content	Encapsulates a single piece of content or part of a channels/content listing

Table 5.1: Protocol messages

The protocol can be broken up into the following phases: *discovery*, *channel update*, *content synchronization* and *content exchange*. The channel update and synchronization phases are optional as will be discussed later. Let us first consider a simple, idealized, lossless scenario, depicted in Figure 5.1. Nodes A and B meet and opportunistically exchange contents. We assume that both nodes have contents to share for the duration of the contact. After an initial discovery phase, the nodes start the protocol interaction. Either node can begin the interaction; it is simply the discoverer who begins. It sends an *announce* message to make its presence known. The receiver can elect to request channel or content meta-data information first, or request content directly, as done in this example. The receiver of the solicitation provides *n* pieces of content, sending a request of its own after the last one. The nodes will continue to take turns requesting and delivering content for the duration of the contact. A simple state-machine representing this protocol is shown in Figure 5.2. It is easy to understand and captures the essence of the protocol for this simple case. A node starts execution in the *initial* state. A node discovering a peer sends an announce and enters a *wait* state, in which it will wait for the expected request. Having received the request, it enters the *provide* state, where *n* content messages will be sent. The node then sends a request and enters the *receive* state, where it will expect a number of content messages. Conversely, a node executing in its *initial* state, which receives an announcement from a peer, will send a request and enter the *receive* state.

This simple protocol is efficient, since a high ratio of content to control traffic can be assumed. It is also fair, since both partners receive approximately equal share, assuming that each sending turn is short compared to the total contact duration. This protocol will however stall if either node does not have content to provide, or if either is strictly a provider, as no valid transitions then exist from the *provide* state. An extension is considered in the example shown in Figure 5.3. A provider node invites its peer to continue transfer by sending announce messages, instead of requests, to signal turntaking. The receiver node will then continue to receive for the duration of the contact. The provider node could for

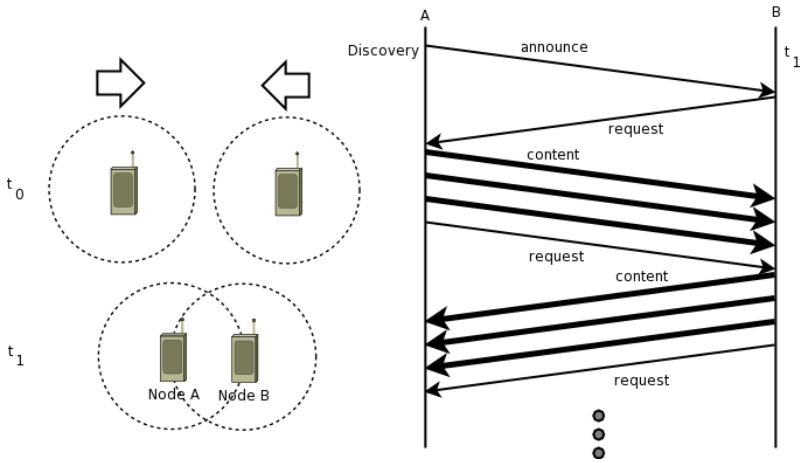


Figure 5.1: Interactions between a pair of nodes for a simple lossless scenario. This simple scenario assumes that both nodes have content to trade for the entire contact period. A single radio channel is assumed, so the content transfer is strictly half-duplex.

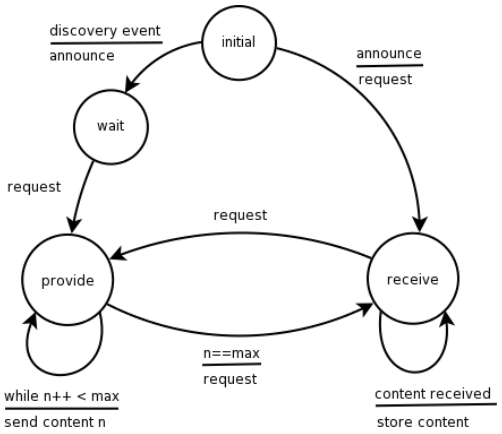


Figure 5.2: Simple content exchange state-machine for a lossless scenario. A pairwise contact is assumed. Further, both nodes have content to trade for the entire contact period.

example be a gateway, which in its base role will never request content. We must also account for the problems due to the inherent lossyness of the wireless medium and the unpredictable loss of contact due to node mobility. A further extension is shown in Figure 5.4, which considers a simple example of content exchange in a lossy environment. A request message is lost, which would potentially stall the protocol and seriously impact throughput. A timer is set when the request is dispatched and decremented until a response is received by the peer. The timer elapsing is interpreted as a lost control message. The request is retransmitted in this case, thereby restarting the content exchange.

The following sections discuss the various phases of the protocol in more details, while addressing those problems.

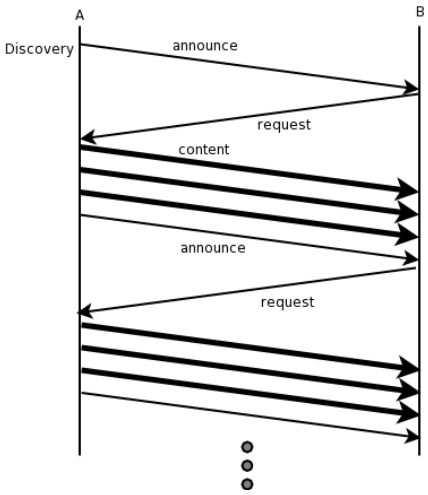


Figure 5.3: Simple content exchange state-machine for a scenario scenario in which a provider and receiver pair exchange content for an entire contact. The provider is assumed to have content available, while at the same time not requiring any from its peer. The provider uses announce messages to invite its peer to continue transfer.

5.2.1 State-Machine Representation of the Protocol

Let us briefly introduce the state-machine notation used in this chapter. A node can execute indefinitely in any state. The exception is states marked c, which are intermediary states, and require a transition to be taken immediately. No or negligible time is assumed to be spent in those states. The notation of $\frac{condition}{action}$ is used throughout, meaning that an action is executed for the given condition when the marked transition is taken. Transitions without associated actions are marked as $condition$ only. Initial states are marked with a double circle.

The protocol can be described by two state-machines: A *management state-machine*, shown in Figure 5.5, monitors the medium and discovers new peers, while a *protocol state-machine* handles interactions between the host and a discovered peer.

The management state-machine waits for notification from lower layers or received announces from unknown hosts. It spawns a new protocol state-machine for each discovered peer. Similarly, a peer loss event of any kind (timeout or explicit notification from lower layers) results in removal of the peer object and subsequent termination of transfer. A node receiving a discovery event from lower layers will announce its presence to its discovered peer by sending an announce message before spawning the peer object.

A new protocol state-machine is spawned for each discovered peer. A node can thus host multiple protocol state-machines, as shown in Figure 5.6, creating the collection $\{p_1, p_2, \ldots, p_n\}$ of peers. Each p_i represents the state of a particular known peer. A

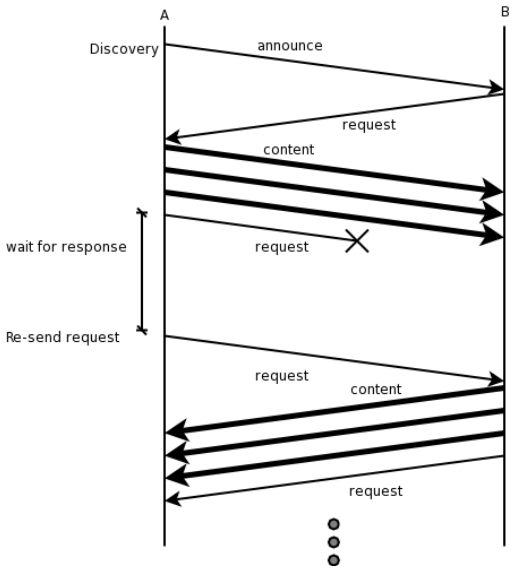


Figure 5.4: Simple content exchange between peers in a lossy environment. A request message is lost. The protocol stalls until a timer elapses in the requesting node, indicating that the request was lost. It is then retransmitted and the exchange continues.

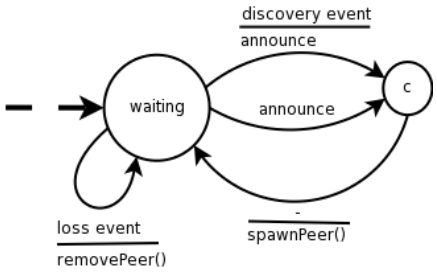


Figure 5.5: Management protocol state-machine for the ad-hoc message passing mode. This state-machine handles peer node discovery, creation and removal. A content exchange state-machine is spawned for each discovered peer and handles the actual content transfer.

protocol state-machine represents the pairwise communications between the host and a single peer. This simplifies the model considerably. A single common radio channel is assumed for the PodNet community. Each peer state and associated communications stream is considered to be independent, since the underlying MAC layer is assumed to manage access to the wireless medium, in effect creating a per-message mutual exclusion condition for the state-machine interactions in the parallel composition. A node may thus actively exchange content with multiple peers simultaneously, subject to the medium access restrictions. A single pairwise contact, represented by a single protocol state-machine will though be considered in the following sections.

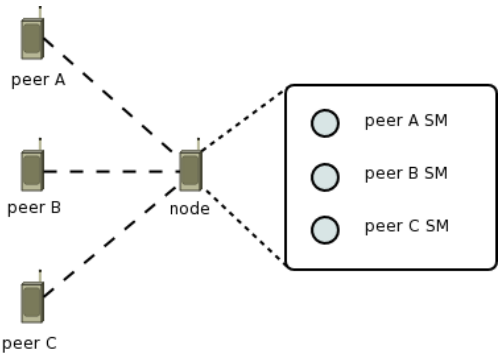


Figure 5.6: Node-peer relations. A collection of state-machines is maintained in each node, each representing a known peer. Each state-machine handles content exchange between the hosting node and a single known peer. Note that the node can maintain concurrent relations with all its peer nodes, subject to medium access restrictions.

5.2.2 Discovery and Initiation of Contact

A discovery mechanism is necessary in order for wireless nodes to be able to communicate. For example, beacons are emitted at the MAC layer in IEEE 802.11 by access points in infrastructure mode, which enable roaming nodes to discover their services, while devices in peer to peer mode emit probe messages to discover their neighborhood (Stallings, 2005, pp. 442). An application layer beaconing and discovery mechanism is described in the solicitation protocol of (Wacha, 2007; May, Wacha, et al., 2007). This mechanism operates on top of the IEEE 802.11 services and thus introduces considerable latency into the protocol. A more optimal solution, proposed in (Karlsson et al., 2007), would be to use the MAC layer beacons directly as a discovery mechanism, possibly piggybacking additional information.

The actual mechanism of discovery, however, is immaterial to this discussion. An underlying abstraction layer is assumed, as discussed in Chapter 4, which wraps the OSI protocol stack up to the layer at which the protocol operates. The following discussion assumes that the underlying services provide a notification of discovery to the protocol layer; a discovering node responds to such a notification by unicasting an announce message to its discovered peer.

5.2.3 Channel Updates

All nodes carry a common, pre-configured, discovery channel to exchange channel information. This channel is assumed to use a well-known common identifier. A node being initialized can thus request a channel update from a peer, initially populating its channel

```
<channels>
  <channel id={channel guid} name={name} description={descr}/>
</channels>
```

Figure 5.7: Simple channels meta-data listing

list. This list is then made available to administration programs, enabling the user of the device to subscribe to channels of interest. The channel list should be refreshed periodically; the most frequent updates of practical interest would be at every discovery of a new peer. This is assumed as the default functionality of the protocol. A discovering node thus requests a channel meta-data list from a new peer, utilizing a normal content request with the discovery channels well-known identifier, as described in Section 5.2.5. The receiver of the request prepares a sequence of content messages, carrying a summary of its channels as a payload. A minimal example of such a summary in XML notation is shown in Figure 5.7. The receiver of the summary utilizes it to refresh its channels list.

5.2.4 Synchronization

A node would ideally have complete knowledge of a peer's channels and the content items carried. This would in reality require considerable amount of information to be exchanged at the initiation of each exchange. In fact, this exchange could well take a considerable fraction of the valuable contact time and thus pose an unacceptable overhead. Some compromises clearly have to be made.

Compact content descriptors in the form of *Bloom filters* (Bloom, 1970) are utilized by Wacha (2007) to carry a hash of channel identifiers in a negotiation phase to establish common channels of interest, followed by a query phase in which the nodes exchange more detailed information about content items on those common channels. Bloom filters are a space-efficient hash of a dataset, which can be used to query for membership. False positives are increasingly more likely as more members are added to a set. False negatives do, however, not occur. Networking applications of Bloom filters have been surveyed by Broder and Mitzenmacher (2003).

The approach taken in this design is to optionally embed a compact descriptor, for example a Bloom filter, of the channel identifiers in the announce and request messages, thereby enabling a recipient to form an instantaneous approximate image of a peer nodes capabilities. The descriptor must obviously be small enough to be carried in a single frame, and the information must thus be rather limited. This approach does not provide synchronization of contents per se, but does nevertheless enable nodes to make intelligent choices in their solicitations and thereby reduce unsatisfiable requests. It poses very little

```
<channels>
  <channel id={channel guid} name={name} max_age={age} description={descr}/>
    <item>
      <id {item GUID} />
      <name {name} />
      <timestamp {date/time timestamp} />
      <category {item category} />
      <description {description} />
    </item>
  </channel>
</channels>
```

Figure 5.8: Simple channels meta-data listing with content summary. This example assumes that the content listing is subject to an age restriction, enabling a practical constraint on the amount of information to be transferred.

overhead when compared to the method proposed by Wacha (2007). Synchronization of any extent is, however, optional in this protocol, since a node is free to request content blindly without any prior knowledge of the recipients state.

A fuller synchronization can be implemented by extending the channel update previously described. The channel meta-data would then include some additional content information. An example of such a summary is shown in Figure 5.8. A node receiving a fuller channel and content description could thus create a very complete image of its peers contents. This approach does, however, add considerable overhead to the actual content exchange, so the potential benefits would have to be weighted against the decreased content transfer.

5.2.5 Content Exchange

The objective of the protocol is efficient content solicitation and transfer. The phase described here, the actual content transfer, is thus of primary importance in the protocol. It is also the only one, apart from the discovery phase, that is required by the protocol. Nodes take turns soliciting and providing contents, as shown in simplified form in Figure 5.1. The content exchange consists of consists of multiple turns, to maximize the fairness of the transfer. The simple protocol state-machine depicted in 5.2 is now expanded to the one shown in Figure 5.9, which will be discussed in this section. The protocol state-machine handles pairwise transfer between the hosting node and a single peer. The communications model is thus greatly simplified for the purposes of this discussion; we assume a mutual exclusion restriction on medium access by virtue of the underlying link layers access controls.

The simplifying restriction of a single content channel is employed in this discussion. The model can however easily be extended to include multiple content channels by in-

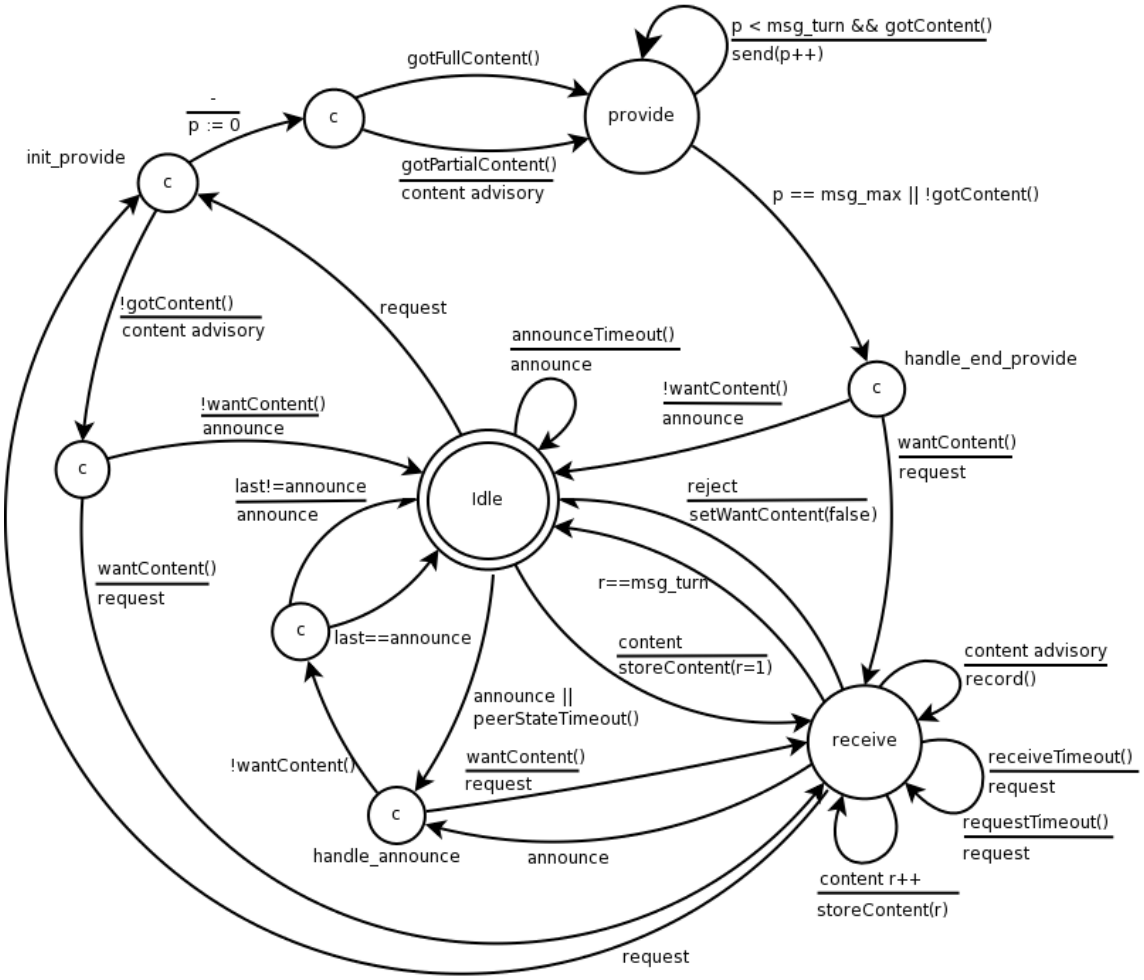


Figure 5.9: Content solicitation protocol state-machine for the ad-hoc message passing mode. The management protocol for a node spawns and initializes a content exchange state-machine in the idle state for each discovered peer. Similarly, the management state-machine will remove the content exchange state-machine for a peer after determining that the peer is unreachable.

cluding an array of channel states with the peer state, represented by the protocol state-machine.

The content exchange protocol is discussed in detail in the following sections. Refer to Table 5.1 for further explanation of the messages employed.

5.2.5.1 Idle

The *idle* state is the initial state of the protocol. A node receiving an *announce* will send a *content request* to its peer if it has contents of interest. This can be determined by examining the compact content descriptor field, or based on past history: a past rejec-

tion signifies that the peer does not have content of interest. The node responds in this case with an *announce* inviting the peer to solicit content, but only if the previous message sent was other than an *announce*. Otherwise, it transitions silently to the *idle* state. This prevents continual bouncing of *announce* messages between peers without mutually interesting content.

The node will send a *request* to its peer if it is determined to have content of interest, followed by a transition to the *receive* state. Note that a request can contain a criteria list of multiple content channels. Simultaneous solicitation for multiple channels and criteria in a single request is thus enabled, as further described in Appendix A.

A timer is employed to retransmit pending announces after a period of inactivity to provide a measure of protection against lost announces. A further timeout is provided for peer state, *peerStateTimeout*, after which a node will re-query a peer previously determined to have no content of interest.

A promiscuous mode of operation[1] is supported, in which a node can utilize broadcast or promiscuously overheard content, intended for another recipient. This is further discussed in Section 5.4. A node overhearing a content transmission will transition to the receive state and thus not try to solicit any content while the transmission is ongoing.

5.2.5.2 Receive

A node enters the *receive* state after soliciting content from a peer, or after overhearing a transmission, as previously discussed. The number of messages in the sending turn are included in the content message header (see Appendix A), providing the node with the number of messages to expect. The node will transition back to the *idle* state after receiving the expected number of messages, where it will expect a *content* request or *announce* from the peer to continue the exchange.

Reception of the exact number of expected content messages as a result of a request is not guaranteed, since some messages may be lost in transit. A node must therefore expect a request or announce message while in receive mode. Such a message will immediately cancel the receive and trigger a transition to the provide mode or restart the receive.

The following timers guard the protocol state while in receive mode:

[1] Promiscuous mode and broadcast capabilities are dependent on the transport employed. Some solutions may not be able to support broadcast or to utilize overheard content.

1. A timeout for pending requests is employed to prevent stalling due to lost requests. The pending request timer is reset once content is received from the peer. A pending request is re-issued if the communicating peer is silent for the timeout period.
2. A timeout for pending content transfer is employed to detect stalling due to massive losses. The timer is reset each time a content message is received from the peer. A request is re-issued (possibly with modified parameters) when the content timer elapses.

A loss of several consecutive content messages can be interpreted as unfavorable conditions, e.g. a blockage, temporarily impeding communications. There is no danger of the protocol stalling, even in the event of prolonged losses, as the announce and request retry mechanisms will take effect as soon as the current sending turn is done. This will however potentially affect the fairness of the exchange since the intended recipient has now lost a considerable fraction of its sending turn. A timeout for content messages is employed to attempt to equalize content exchange in this event. A receiver node will re-issue its request, for the remainder of expected content in the turn, after a period of inactivity in the receive state. If the request goes through and reaches the sender, it will restart a sending turn and thus compensate for the losses incurred by its peer. The timeout should be greater than $2 \cdot RTT_{MAX}$ for the radio interface used, and long enough to allow for at least 2 sequential lost messages before triggering a request to prevent spurious requests due to isolated losses.

5.2.5.3 Provide

A node enters the *provide* mode after receiving a *request* message from a peer. It begins by examining the request and determining if it can be satisfied in full. If not, the node issues a *content advisory* message, notifying its peer that it is unable to fully or partially fulfill the request. The compact content descriptor of the request can be utilized to optimize the content transfer. If the descriptor is sufficiently detailed to include a partial description of the content carried by the requester, the node can immediately exclude those items. Note that the descriptor would only have to include the items of relevance to the present query in this case.

A node having some content to share then enters a state where up to msg_max *content* messages, each containing a piece of content, are prepared and sent. The number of messages sent depends on the amount of content that the node has to share. The content may be unicast to the particular peer or broadcast, as will be discussed later. The node

follows its sending turn with either 1) a request of its own if it wants content from the peer, or 2) an announce otherwise, inviting the peer to continue transfer.

A node transitions from the *provide* state immediately after transmitting the number of messages, so the time spent in provide can be assumed to be bounded by a known time determined by the processing and transmission delays, or

$$t_{provide} \leq msg\_max \cdot (t_{proc} + t_{trans})$$

A node unable to provide any content will send an *announce* or *request* directly after the *content advisory*, depending on whether the node wants to solicit content from its peer.

While in provide mode, a node cannot receive any content. The frequency band of the pair is occupied by the transmission, making listening on that channel pointless, if not impossible.

5.3 Pairwise Mode of Communications

The previous section assumed an ad-hoc communications mode, i.e. one where any node is free to solicit contents from any known node, and, similarly, respond to any received solicitation. An exclusive pairwise mode of communications, in which a node is restricted to communicating with a single peer at a time for a period of time is described by Karlsson et al. (2007) and Lenders et al. (2007). This section describes the extension of the model described in Section 5.2 necessary to enable exclusive pairwise operation.

A potential benefit of the pairwise approach, compared to the more disorganized ad-hoc approach discussed previously, is that a larger amount of content is expected to be received from each peer, assuming a common radio channel. The ad-hoc message passing approach is likely to provide a more diverse set of content from a larger set of simultaneous peers, but less content per peer, and hence quite probably less cohesive content.

The exclusive pairwise paradigm implies that at most one peer can be executing in the *idle* state at a time, in the context of the model of Figure 5.9. A number of modifications to the previous management and content exchange protocols are required to enable this mode of operation. The management state-machine is modified as shown in Figure 5.10, adding a session management functionality to the protocol. A protocol state-machine, shown in Figure 5.11 is spawned for each peer discovered, as before. An activation of the content exchange is, however, only attempted if the node is not currently involved in an exchange

with another peer. An activation attempt is in the form of a *pairing request*, which a node can only respond to in the *wait* state with a *pairing response*. If a peer does not respond to a pairing request, the node will try other known peers in order of last successful contact, or discovery time. If none respond, we can assume that all are engaged in ongoing pairings. The node will thus wait for a period of time before reattempting to pair with a peer. The *activateTimeout* triggers a new activation sequence for known peers. A disconnect event for the currently active peer, as a result of it moving out of range, will also trigger a series of activation attempts.

The solicitation protocol is modified as shown in Figure 5.11, adding an initial *wait* state. The content exchange protocol is unchanged from that depicted in Figure 5.9, apart from the added wait state, which is now the initial state. Figure 5.11 thus only shows the wait and idle states of the protocol and the transitions and actions which are added for the pairwise mode. The transition to and from the *idle* state is controlled by the management protocol. A node will not solicit or provide content to a peer, whose representative state-machine is executing in the wait state.

A soft bound, $t_{session}$, is placed on the length of time that two nodes can be in contact to promote fairness; the dissemination properties of a system would likely suffer if two nodes among a pool of potential ones were allowed to be in contact for an unlimited period. This bound ensures that contact with the current peer is broken and attempts are made to contact other potential peers at regular intervals. The content exchange protocol engine thus issues a notification to the management engine, while executing in the *idle* state, having executed for $\geq t_{session}$. The management protocol controls how this notification is handled: If only one known peer exists, the management protocol will choose to ignore the notification, thus continuing the present pairing for another $t_{session}$. If more than one peer is known, they are requested in increasing order of last completed contact.

Promiscuous utilization of broadcast or overheard content is allowed in the pairwise mode. A node executing in the wait state can store content of interest from a peer, although the content exchange is technically on hold.

5.4 Broadcasting of Contents

The wireless medium is in general an error-prone and scarce resource. It is therefore important to maximize its utilization. Unnecessary retransmissions, such as repeated point-to-point transmissions of the same data, should thus be minimized. Taking advantage of the broadcast nature of the medium can help in this regard. If multiple nodes are within

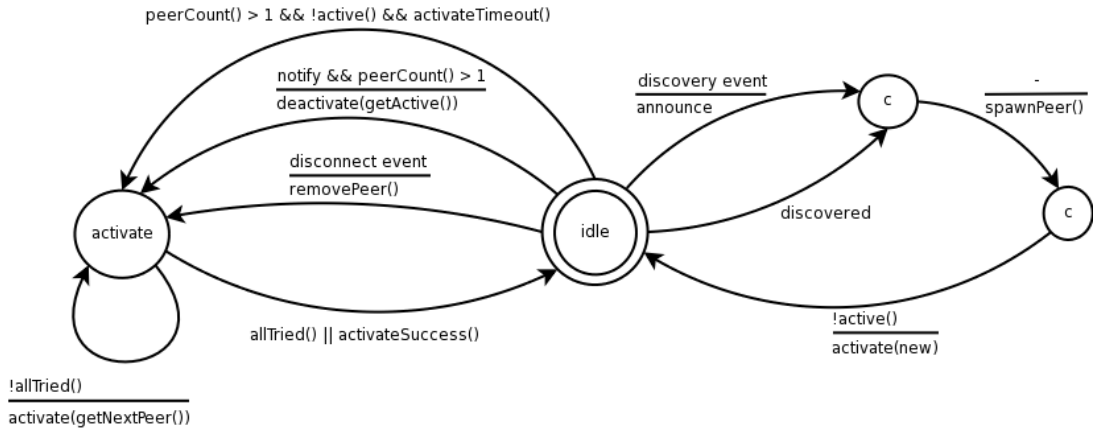


Figure 5.10: Management protocol state-machine for the pairwise message passing mode. This state-machine handles peer node discovery, creation and removal. A content exchange state-machine is spawned for each discovered peer and handles the actual content transfer.

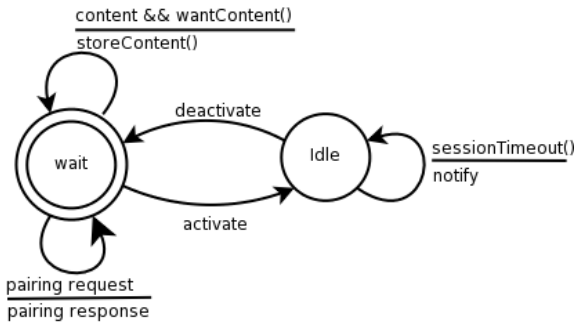


Figure 5.11: Content solicitation protocol state-machine for the pairwise mode. Only the wait and idle states are shown. Refer to Figure 5.9 for a fuller description of the protocol.

each others' range at the same time, it is an unnecessary waste of bandwidth to send the same content multiple times point-to-point. If broadcasting were to be used instead, all nodes interested in the content could receive it at the same time, providing optimal use of bandwidth by removing unnecessary transmissions, decreasing collision potential and thereby increasing available bandwidth.

Let us consider a case where three nodes are within each others' communications range, as shown in Figure 5.12. N_A has content that both N_B and N_C want. If N_A were to unicast the same requested content to both nodes then half of the communications capabilities is essentially wasted. When transferring from N_A to N_B, N_C is idle since the medium is occupied by the ongoing communications. N_C would overhear and ignore the content addressed to N_B. If the content were on the other hand to be broadcast, or the nodes in the system allowed to promiscuously utilize overheard content, then N_C would be

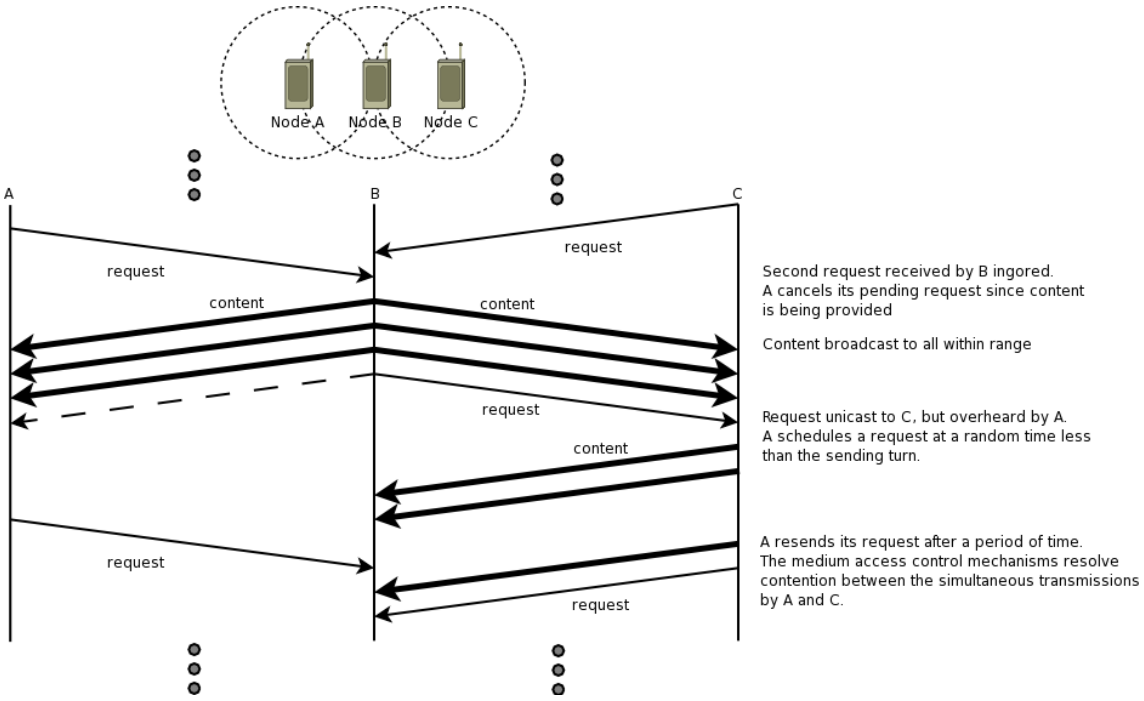


Figure 5.12: Broadcast of content messages.

able to utilize the communications between N_A and N_B. Broadcast of content can thus potentially increase the available bandwidth in a dense population of nodes.

5.5 Protocol Analysis

This section provides a brief analysis of the solicitation protocol as presented in this chapter. A deadlock is not possible in the protocol, since waiting is bounded in all states. Lost messages and superfluous control messages, however, pose a threat to protocol efficiency and are thus considered in particular. Excessive control message traffic can affect throughput and must therefore be discussed.

5.5.1 The Effect of Lost Messages

The service is assumed to be best effort, as previously discussed. Two separate cases of lost messages and their effects on the protocol must thus be considered.

1. Loss of content messages causes a corruption of a number of content pieces. This is certainly a disadvantage and may mean a corruption of a complete item, depending on the application consuming the content. However, such a loss does not break the

protocol itself, i.e. the remainder of the transfer does not stall or terminate as a result of such a loss. The sending turn by a node is always followed by a control message, signalling its end. The actual number of content messages received by a node is thus immaterial for protocol continuity. Increased efficiency in the case of several subsequent lost content messages is provided by the content timeout mechanism, which re-issues a content request for the remainder of content messages in a sending turn, after detecting no activity in the receive state for a period of time.

2. A lost control message has greater impact since they are few compared to content messages and essential to continuation of content transfer. The node emitting the lost control message has no way of detecting its loss. Similarly, the intended recipient has no way of knowing that a control message intended for it was lost. This is a potentially serious occurrence since the content exchange between a pair of nodes could stall if no precautions were taken. Two cases of lost control messages must be considered.

   (a) A discovering node sends an initial control message, request or announce, to bootstrap the content exchange. Let us assume this message is lost. The potential partner will most likely discover the node on its own, thus assuming the role of the discoverer. It is unlikely that the initial messages from both peers are lost, unless conditions are unfavourable for communications. A robust protocol must however consider this occurrence. Assuming that the underlying hardware only provides signals for acquired and lost nodes, both lost initial contact messages would mean a stalled protocol and thus a wasted contact opportunity. A discoverer will thus periodically retry its initial control message after a period of inactivity. The initial contact retries continue until the contact is deemed lost by the underlying service layer.

   (b) Lost announce or request following a content transfer is an equally serious occurrence. Such a loss would result in a stalled transfer if no precautions were to be taken, since the recipient node would have no way of knowing that the current sending turn by its partner node was done or what action to take. The protocol includes a timeout for outstanding requests to prevent this occurrence. The sending node re-issues the control message after a period of inactivity, thus restarting the interaction.

In summary, lost messages are shown to lead to inefficiencies in the protocol, but are mitigated by the use of timeout-retry mechanisms. The effects of lost messages can be minimized by careful determination of the timeout parameters. The solicitation protocol

is shown to be free from unrecoverable stalls, and thus from the inefficiency associated with a wasted contact opportunity.

5.5.2 Excessive Control Traffic

A potentially serious condition may arise where two peers continually bounce control messages back and forth. This can happen in two distinct cases for a naive protocol:

1. Neither node has any knowledge of the others' contents. In this case, the nodes would continually send a request, which would be answered by a reject message, followed by a request. This could potentially continue for the duration of the contact.

2. Both nodes know that they do not have any interest in their peer's contents. One would send an announce inviting the other to solicit content. The peer receiving the announce would respond with an announce of its own. This condition could continue for the duration of the contact.

The protocol handles both those cases. The approach taken here is to employ the content advisory message in the form of a reject message if a node is not able to service any of a peers requests. The requesting peer marks the node sending the rejection as uninteresting and will thus not solicit content from it for the time being. A request is not sent to a node marked as such, and thus limiting control traffic. A timeout is however employed to reset the interest flag periodically, at which time a node will retry soliciting content from its peer, which may have obtained new items from other peers hidden from the requesting node.

The overly polite problem of continual invitations is avoided by only allowing announces in response to announce messages, if the preceding message sent was not of this kind. An announce in response to a previously transmitted announce is thus ignored, stalling the exchange for the time being. A timeout is however employed to periodically restart the protocol and reattempt solicitation.

5.6 Summary

This chapter presents a light-weight solicitation protocol design, suitable for use in the PodNet system. It is independent of the underlying transport, providing maximum flexibility in the choice of implementation platforms. The protocol is presented in the setting

of the disorganized ad-hoc message passing mode, as well as the pairwise paradigm described by Karlsson et al. (2007); Lenders et al. (2007). The protocol was shown to be robust in case of message losses. This protocol has been implemented in part, as will be described next in Chapter 6.

Chapter 6

PodSim Simulation Model

This chapter describes the design and implementation of a simulator, *podsim*, for a PodNet wireless content distribution network. The primary purpose of the simulator in this dissertation is to verify the content solicitation protocol, described in Chapter 5. It is also an important contribution for the PodNet project and will be used as a basis for further development work and experimentation on the behaviour of PodNet communities.

A population of peers must be viewed as a whole: a dynamic system in which nodes are constantly entering, leaving and turning on and off. A likely scenario is a "world" of separate population islands, created around points of congregation like busy pedestrian streets, train platforms and squares. The performance of the system, i.e. its ability to provide content and fulfill user expectations, depends on several factors, many of which are outside the control of the system or its designers. The topology of the area in which the system is intended to function is immutable, as is the parameters of the radio technology used for communications. The mobility patterns within the area are the primary driving force of the dissemination, but they are largely random in nature and also completely out of the control of the systems designers. They are however usually somewhat predictable given sufficient studies, so the conditions at a given time of day can be approximated.

It is thus difficult to grasp the concept of deployment since the system is formed from an ad-hoc collection of independent nodes. The fixed platforms, gateways and caching nodes, add some degree of stability to an area or population island. A system designer can use a strategic placement of fixed nodes to maximize the performance of the system. Population studies, as suggested above, can give an indication of optimal fixed node (gateway and caching) placement. However, trials in a real environment are needed in each case to verify those estimates. Experiments with wireless hardware are time consuming, often expensive and repeatable results next to impossible to achieve. A simulation study, on the

other hand, allows for rapid trials of various options and parameters. Simulated scenarios typically run orders of magnitude faster than real-time, allowing multiple simulation runs to be done in a comparably short period of time.

The simulation model presented in this section is thus a logical step in prototyping of a PodNet system. It creates a sandbox for experimentation with the system, to evaluate protocols and algorithms, and to explore its potential for various applications. Simulation is obviously only an approximation of a real system. This technique is, however, a useful prelude to deployment, as the placement of fixed nodes and the system parameters can be explored before committing actual resources.

This chapter is organized as follows: The OMNeT++ simulation framework is introduced in Section 6.1. Section 6.2 briefly outlines the solicitation and communications model employed. The simulation model of podsim is then introduced in Section 6.3, which discusses the main modules of the simulation, gateways, roaming nodes, and various scenario level control modules. The constituent building blocks of the gateway and roaming node models are discussed further in Section 6.4. The chapter is then summarized in Section 6.5.

The full code for the *podsim* simulator, along with a user manual, is available for download at http://code.google.com/p/podsim.

6.1 The OMNeT++ Simulation Framework

The open source discrete event simulator OMNeT++ (http://www.omnetpp.org) (Varga, 2001) was chosen for this project. It supports modular development, where user-written modules in C++ are derived from a framework base class, *cSimpleModule*. Interface definition files, NED files, are created for each user defined module. Compound modules are created by assembling simple modules into a new NED file. The created modules, both simple and compound, can be reused in other compound modules, allowing for a very efficient reuse of code.

Data paths between modules are defined by input- and output gates and channels which connect them. The gates are the interface of a module with the outside world. Channels are defined to provide data paths between individual modules in a compound module. Modules can additionally send messages directly to a specified gate of another module, bypassing the defined channels structure.

Run-time parameters are defined in initialization files. Both standard environment and user-defined module parameters can be assigned values in the initialization files. The behaviour of the system can thus be extensively modified between runs. The initialization file can specify parameters for multiple runs, enabling running a batch of simulations with different parameters.

An OMNeT++ model can be compiled to run in console mode for maximum speed. Additionally, a GUI front-end can be added by compiling with a tcl/tk-based windowing library. The latter feature is useful for visualizing scenarios and debugging.

OMNeT++ is available for a variety of platforms. *podsim* is written for and tested on a Linux platform (Fedora core 6, 2.6 kernel). It has additionally been compiled on Windows XP. Since OMNeT++ simple modules are derived from a C++ base class, it is relatively simple to drop in production code for testing, i.e. code modules that can be reused on a real hardware platform. This is an important advantage, since a future goal of the PodNet project is to port the code created for this simulator to a real hardware platform. This multi-platform goal was observed in creating the simulator, both when writing the actual code and choosing supporting libraries.

Mobility support is provided by the Mobility Framework (MF) (http://mobility-fw.sourceforge.net) for OMNeT++ (Drytkiewicz, Sroka, Handziski, Köpke, & Karl, 2003). This library is widely used for mobility modeling in the OMNeT++ community. MF provides a base class for mobility modules, *BasicMobility*, which defines the necessary functions for mobility support. An attractive feature of this framework is the *Blackboard*, which supports exchange of location information between modules, using a publish-subscribe approach. Various standard and user contributed mobility modules are available, implementing a variety of mobility models, and can be used unmodified in *podsim*. Custom mobility modules, based on the MF model, were though created for this project as later described. MF also provides extensive support for radio channel emulation, such as fading models, although these features are not used in the *podsim* model. *podsim* requires nodes to be created dynamically at run-time, which the MF does not support in its present form. This requires a custom channel control implementation which utilizes the location awareness of the object factory (Section 6.3.1), together with the range calculation features of the WNIC module (Section 6.4.3).

6.2 Protocol and Communications Model

A very simple and idealized communications model is employed in the simulator. The nodes simply determine nodes in range and use the direct message passing feature of OMNeT++ to deliver messages. This is further described in Section 6.4.3. The message queue of the discrete event simulator is thus implicitly used to approximate medium access controls. This approach effectively abstracts away any specific link layer technology and medium access control methods. The peculiarities and complexity of any specific link layer protocol is thus removed for the time being, greatly simplifying the analysis of the protocol, which is the primary purpose of the simulator.

The solicitation protocol implemented in the podsim is a simplified version of the one presented in Chapter 5. The implementation is further discussed in Section 6.4.2. The *None*, *uniform* and *weighted solicitation* solicitation strategies, described in Chapters 3 and 4, are employed. A single channel is selected in each requesting turn, although the protocol described in Chapter 5 allows multiple channels to be solicited simultaneously. An entire sending turn is thus dedicated to a single channel. The sending turns are very short by default, which compensates for this limitation, at the expense of increased protocol overhead.

Content in the simulation model is identified by simple integer serial numbers; proper timestamps are not employed at the present time. Messages are thus generally requested by specifying a lower bound on serial numbers. This effectively simulates soliciting content stamped from time zero, where a piece is made available on the channel each unit time. A node ideally wants all content on its channels of interest (private channels), numbered from zero to the maximum provided during the simulation.

6.3 Simulation Model

The simulation model consists of a set of NED files, simple and compound, assembled into an OMNeT++ *network*. A network is simply a NED file, which connects the collection of modules into a coherent communications model.

An example for a square simulation is shown in Figure 6.3. This scenario includes a single gateway, located at the center of the square. Several mobile nodes were dynamically generated and are in this case navigated using a random waypoint mobility model. The global *object factory*, *observer* and *channel control* modules are always present in a scenario. The node factory is described in Section 6.3.1 and observer in Section 6.3.2. The

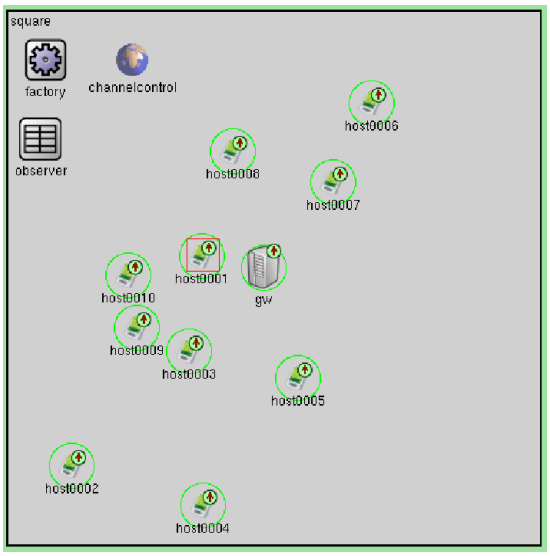


Figure 6.1: A typical simulation scenario. A single gateway is placed in the center of a square region. A number of nodes is created and then move autonomously within the region. The random waypoint mobility model is used for the simulation depicted.

channel control module is unused at the present time, but required for mobility support, as described in Section 6.4.4. A number of *fixed gateway nodes* is also always present. The scenario is thus partly defined at compile time.

The gateway and mobile node objects use an identical node model, described in Section 6.3.3. Their constituent modules are described in Section 6.4.

podsim can be compiled as either console or GUI based. Full visual mobility and message passing between nodes is supported in the GUI mode, enabling very powerful visualization and debugging. The console mode is however more appropriate for batch runs.

The model described here supports extensive control of the simulation through an initialization file, including the node types and number in the scenario, mobility model and parameters and protocol settings. Mobile nodes are dynamically generated at run-time according to the settings for the scenario. The most important scenario level parameters are listed below:

sim-time-limit sets the simulation duration in seconds.

output-scalar-file specifies the name of the scalar log file, used to log various parameters of the modules at the end of their run.

scenarioSizeX sets the width of the simulation scenario.

scenarioSizeY sets the height of the simulation scenario.

numGateways sets the number of gateways in a scenario. Gateways are by default placed at $(scenarioSizeX/2, scenarioSizeY/2)$. The placement should be overridden when placing multiple gateways.

gw[*].x places gateway node * at a specified x location. The default is $scenarioSizeX/2$.

gw[*].y places gateway node * at a specified y location. The default is $scenarioSizeY/2$.

In addition, the modules have associated initialization parameters, which will be discussed in the following sections.

6.3.1 Node Factory

A *node factory* object is used to dynamically create nodes during a simulation. The factory supports two modes of operation:

1. A fixed number of nodes can be initialized at simulation startup.
2. An externally generated mobility script can be utilized to generate nodes dynamically during the run, allowing nodes to enter and depart the scenario. This provides considerable flexibility in the mobility modeling and effectively abstracts it out of the protocol simulator.

The object factory forms a part of the *trace mobility model*, reported by Helgason and Jónsson (2008). Note, however, that the factory described in this dissertation is an older version than the one described in the paper and utilizes a flat text tracefile, rather than an xml one.

6.3.1.1 Initialization parameters

The following parameters can be specified in the initialization file to customize the behaviour of the node factory:

scenarioSizeX is the width of the scenario in meters.

scenarioSizeY is the height of the scenario in meters.

moduleClass is the class name of generated modules, i.e. a class name of a OMNeT++ compound node object. This must match a compiled object existing in the simulation path.

moduleType defines the type of generated nodes. It can take the values *gateway*, *node* or *caching*. This is used to initialize a created node object to its chosen role.

moduleName is the name prefix of generated nodes. A serial number is appended to the moduleName label to create the actual name of the created node.

mobilityModel is the name of a mobility model applied to generated nodes. This must correspond to a NED file name of a module derived from *BasicMobility*. The trace mobility model requires use of the *TraceMobility* module.

iconName Name of the icon (with path). This allows customization of the appearance of the node when running OMNeT++ in the GUI mode.

initCreated is the number of players generated at startup. Enabled if the *mobilityModel* is not *TraceMobility*.

traceFile File with mobility trace used with *TraceMobility* object. This option is further described in the following sections. The *traceFile* can be omitted if not utilizing the trace mobility model.

6.3.1.2 Trace mobility

The trace mobility model uses scripts to manage mobile node location and lifetime during the simulation run. The mobility script format utilized in podsim is described below.

One event is specified per line in the input file, whose format is as follows:

```
{command} {time} {node} [details]
```

`command` is *create*, *destroy* or *waypoint*. `time` is OMNeT++ decimal time from the beginning of the scenario. `node` is an integer uniquely identifying the node. `details` are defined for the *create* and *waypoint* events as described below.

```
create {time} {node} {x y z} {type}
```

`create` specifies creation of a mobile node at a specified time and location. *Type* is an optional parameter, specifying the type of node to be created. This string must correspond to an existing OMNeT++ module in the simulation.

```
destroy {time} {node}
```

`destroy` specifies the destruction of a node. Time is optional; a negative time will simply destroy the node after the last leg of its journey is travelled and its final pause is done. A destroy event with a specified time will destroy the node at that exact time, regardless of any remaining waypoint events.

```
waypoint {time} {node} {x y z} {velocity} {pause}
```

`waypoint` specifies the next location of the node, its velocity and pause time at the destination. Normally distributed variations of velocity and pause times are supported. Time is optional as with the *destroy* command; if less than zero, then the time of travel is deduced from the distance and velocity. If however the next consecutive event specifies a time, the velocity is deduced from the distance and travel time.

Nodes are thus created dynamically during the course of the simulation run. This can be contrasted to the method currently employed by most protocol simulators, where the entire population of nodes is created at startup and nodes "flown" into the scenario on creation times. Although, the results can be identical, this method has the disadvantage that the simulator has to manage a potentially huge collection of mobile nodes which are outside the current boundaries of the simulation. This places a computational burden on the hosting platform. In contrast, the trace mobility model guarantees that the protocol simulator only has to manage the collection of nodes which have the potential to communicate at any given time.

A mobility script can be generated using the *UrbanMobility* tool, described by Helgason and Jónsson (2008). It takes a input a grid map, a set of generators and routing probabilities, and generates a script consisting of create, destroy and waypoint events. An example scenario is shown in Figure 6.2. The Östermalm region of central Stockholm is here used as a grid plan. A set of generators, denoted by λ, are configured and introduce nodes with an exponentially distributed arrival rate into the grid. Each intersection has associated routing probabilities, determining which street a node will enter next. A node is destroyed when it reaches the grid boundary by taking one of the exit streets. A proof-of-concept *random waypoint* script generator, *rwpy*, is also described by Helgason and Jónsson (2008). It generates a mobility script based on the random waypoint mobility model and supplied parameters.

Other mobility generators, e.g. *UDel* (http://udelmodels.eecis.udel.edu) (Bohacek, Sridhara, Singh, & Ilic, 2004; Kim, Sridhara, & Bohacek, n.d.; Sridhara & Bohacek, n.d.) can also be used to create tracefiles. A simple script is then used to convert the UDel formatted file to the one used by the node factory. UDel is a sophisticated tool for mobility support. Studies of mobility patterns have been compiled into a very comprehensive model, which is then employed to generate traces for a street map. UDel is also capable of generating detailed propagation traces which take into account blockage and multipath effects generated by the modeled street scenario. Building this kind of mobility simulation sophistication into a protocol simulator is certainly possible, but would clearly add to its complexity and increase running times. Offline mobility generation, combined with a

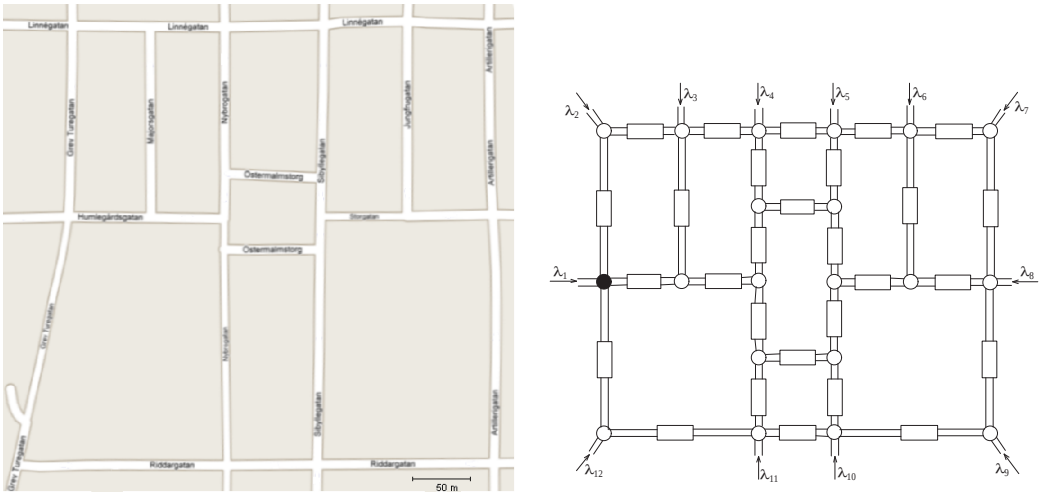


Figure 6.2: The Östermalm region of central Stockholm Stockholm and the corresponding network of street segments

mobility model capable of running such traces, is thus a viable solution for sophisticated mobility modeling.

6.3.1.3 Location services

The node factory discovers an existing scenario at startup time and stores found modules, equipped with a WNIC interface, in a static modules list. Static modules are created by the simulator itself and must therefore also be destroyed by it. The only static modules in the current version of podsim are the gateway nodes.

The dynamic objects list contains the objects instantiated by the node factory, which must therefore also manage their destruction. The factory has therefore global knowledge of the scenario at any given time. It is thus logical to utilize it for providing location information. This feature is utilized by the WNIC modules, which periodically query the factory for nodes in range. This is further described in Section 6.4.3.

6.3.2 Global Observer

The purpose of the global observer is to log state information during the run. A global logging object simplifies log file management, as this functionality would otherwise be handled individually by each simulated node, leading to potential problems with concurrent file access and added node complexity. The observer provides a set of methods, which log location and cache usage statistics. Nodes log this information periodically, at minimum at their destruction. The information collected is utilized to create usage and

performance statistics. The frequency of the logging is controlled by the initialization settings in effect for the nodes.

6.3.2.1 Log file format

The global observer takes a single initialization parameter, which is the name of a log file. A single log file can be utilized for multiple simulation runs and is thus never overwritten by the observer. Each instantiation of the observer module stamps its creation time into the file:

```
[GLOBAL_OBSERVER:SESSION_START T={time} RUN={run number}]
```

time is here the world time in Unix format and *run number* is the one specified in the initialization file currently executing. Multiple runs can in this way be easily aggregated by processing a single file, each run being clearly defined.

A run log consists of multiple lines of the form:

```
T=[world time];OBJ=nodename;CREATETIME=t;SIMTIME=t;LOC={x,y};{ch};
```

T is the world time of the entry in Unix format.

OBJ is the node name which recorded the entry.

CREATETIME is the creation time of the node.

SIMTIME is the simulation time (starting from zero in each run).

LOC is the {x,y} coordinates of the node.

ch is a list of channels.

The channel list entry is on the form

```
CH={ID=number,NAME=name,TYPE=type,[statistics]};
```

Id, *name* and *type* for the channel are mandatory, but other entries are not specified, giving added flexibility for logging. Multi-channel systems concatenate entries for all their channels forming a $CH = \{c_1\}; \ldots; CH = \{c_n\}$ chain.

6.3.3 PodNet Node

A single node model is employed for both the gateway and roaming node simulation. This is consistent with the concept of the multi-role node design discussed in Chapter 4. The node is a OMNeT++ compound module, assembled from a number of building blocks, further described in Section 6.4.

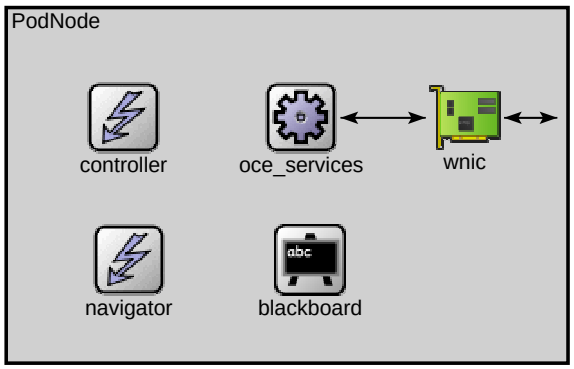


Figure 6.3: PodNet node simulation model. The figure shows the compound model, PodNode, which is composed of a number of software modules. Each of the submodules is an object derived from the base class of the OMNeT++ simulation framework.

The simulation model for the PodNet node is shown in Figure 6.3.

controller is a simple controller for the node, and contains the *content cache* object. Its purpose is to read a channels initialization script and initialize the cache.

oce_services is the *opportunistic networking services*, the module which contains the solicitation protocol implementation.

wnic is a simple implementation of a wireless network interface. The direct message passing feature of OMNeT++ is utilized to exchange messages, using the location services of the node factory module.

navigator handles node movement. This must be a BasicMobility derived module, but is defined at runtime. Mobility modules are specified in the initialization file, as discussed in Section 6.3.3.1. Modules supported in the current version are the *RandomWaypointMobility* and *TraceMobility* modules.

blackboard is used for compatibility with the *Mobility Framework (MF)*. It is employed to transfer location information between modules, using a publish-subscribe model. The blackboard will not be further described here, but refer to the MF documentation for details.

The OMNeT++ scalar file feature is used to log a variety of scalar information on the node termination. The exit report can be disabled for individual nodes and components. The scalar file name is a scenario parameter, as discussed earlier.

A subscriptions system is not modeled in this version of *podsim*. Rather, the cache of selected nodes can be set as infinite, essentially generating content on demand. This simplifies the simulation framework considerably, since external providers and associated

infrastructure can be omitted from the model. This feature is used for the gateway nodes modeled in the simulations described in Chapter 7.

6.3.3.1 Initialization parameters

The initialization parameters of the PodNet node model are as follows:

x, y, z initial location of the node.

moduleType defines the type of the nodes. It can take the values *gateway*, *node* or *caching*.

mobilityModel is the mobility model applied to the node. This must correspond to a NED file name of a mobility module, derived from *BasicMobility*. The trace mobility model requires use of the *TraceMobility* module.

exitReport is a boolean defining whether the node shall log an exit report to the scenario scalar file at termination.

6.4 Node Building Blocks

This section describes the building blocks of the node model, described in Section 6.3.3. Each submodule is derived from the *cSimpleModule* base class of the OMNeT++ framework.

6.4.1 Module Controller

The module controller provides initialization and logging services for the PodNet node model, in addition to serving as a container for the content cache object. It read the channels initialization script on startup and initialize the cache. Future versions of the simulator can use the controller for other initialization functions, e.g setting up application layer consumers. The controller logs periodic or final state snapshots to the global observer, described in Section 6.3.2, if enabled in the initialization file.

6.4.1.1 Initialization parameters

The initialization parameters of the controller module are as follows:

moduleType defines the type of the nodes. It can take the values *gateway*, *node* or *caching*.

configFile specifies the node xml configuration file. This is presently used to initialize the channels cache. Refer to Section 6.4.1.2. An empty string initializes a content cache without any channels.

updateInterval sets the update interval for the controller. The channels cache time dependent parameters must be periodically updated. The display is also updated periodically if running in GUI mode.

infiniteCache is a boolean parameter specifying whether the contents of the channels cache are infinite. Refer to Section 6.4.1.3.

stateSnapshotEnabled is a boolean specifying whether periodic snapshots of the node state should be logged by the global observer, described in Section 6.3.2

finalSnapshotEnabled is a boolean specifying whether a final state snapshot should be logged by the global observer.

snapshotInterval specifies the interval at which state snapshots are logged to the global observer. This parameter is only effective if stateSnapshotEnabled=true.

exitReport is a boolean defining whether the node shall log an exit report to the scenario scalar file at termination.

6.4.1.2 Channels initialization file

The channels initialization file specifies the channels known by the node at startup. Note that nodes can discover channels from peers during the simulation run. Nodes in the simulation may thus have several different initialization files. A node with unspecified channels initialization file will initialize an empty channels cache and must therefore discover channels dynamically from peers.

The initialization file is in XML format. The libxml2 (http://xmlsoft.org/index.html) library was chosen for XML parsing support. It is written in C and is distributed as part of the *GNOME* project. It is also highly portable and has been employed on a variety of platforms.

An example configuration file is shown in Figure 6.4. Five channels, four feed and one update, are initialized in this example.

```
<?xml version="1.0" encoding="UTF-8"?>
<podsim-config>
  <channels auto-subscribe="true" subscribe-count="1" >
    <channel name="News" id="1" type="public" content="feed" priority="normal" />
   <channel name="Weather" id="2" type="public" content="feed" priority="normal" />
   <channel name="Music" id="3" type="public" content="feed" priority="normal" />
   <channel name="Traffic" id="4" type="public" content="feed" priority="normal" />
   <channel name="City map" id="5" type="private" content="update" priority="high" />
  </channels>
</podsim-config>
```

Figure 6.4: A sample channels configuration file

6.4.1.3 Content cache

The content cache of the podsim is based on STL collection classes, *map* and *vector* template objects. This results in a compact in-memory cache, sufficient for the purposes of the simulation. Such a solution, combined with a off-line XML storage for content, could also be a practical approach to implementing a cache for a small hand-held device. The relational database approach, described in Chapter 8, represents a more resource intensive but robust design. A UML diagram of the content cache classes is shown in Figure 6.5.

The content cache also models the subscription system in the podsim implementation in a rather brute-force way. The cache can optionally be specified as infinite; a node with infinite cache will generate and add requested items on demand, essentially modeling a provider with a very large cache or procurement times that are very small on the timescale of the ad-hoc domain. This is useful for modeling a gateway node.

6.4.2 Opportunistic Networking Services Module

The *opportunistic networking services (ONS)* module encapsulates the ad-hoc solicitation protocol, described in Chapter 5, and its associated services. The protocol implementation is further described in Section 6.4.2.2.

The ONS module is notified of peer discovery by the WNIC module, described in Section 6.4.3, which in this setting functions as the abstraction layer, described in Chapter 4.2. Similarly, the WNIC notifies the ONS of lost contacts. Peers are added to the *peers* collection on discovery and marked as unreachable upon loss notification. Peers in the collection are not removed at loss notification, but rather time out after a period of time. This feature enables use of past history of lost nodes in the protocol. This is for example useful when dealing with intermittent connectivity situations at the extreme radio range, when nodes can be lost and reacquired repeatedly.

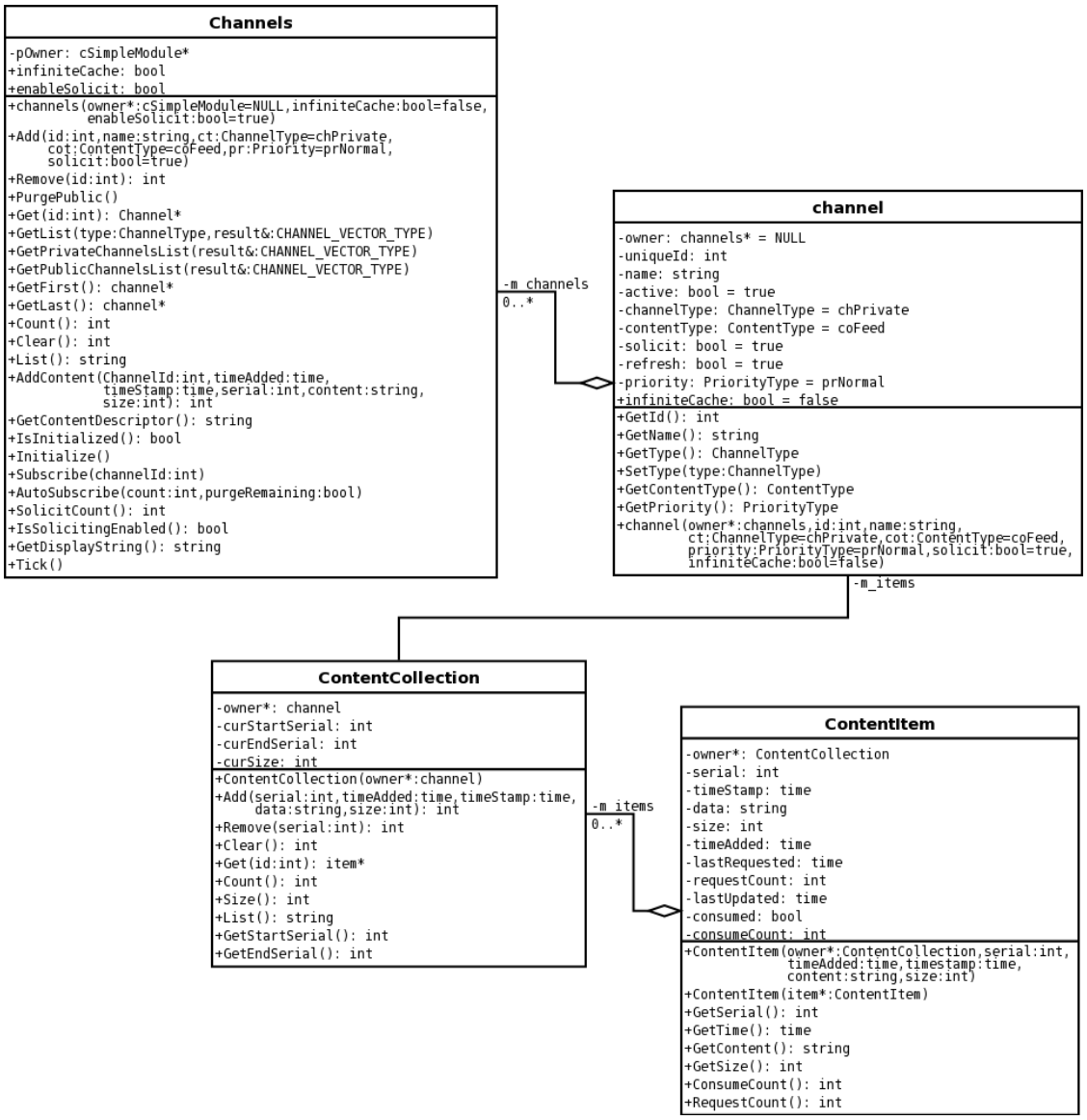


Figure 6.5: UML diagram for content cache implementation of simulation model

A two-way communications channel is defined between the ONS and WNIC modules by means of input/output gates and zero-delay links. OMNeT++ messages, simulating link layer frames, are sent to and received from the WNIC. The ONS contains the logic to pack, unpack and interpret the encapsulated protocol messages. The pairwise limitation on peer contact is not employed; rather, the ONS protocol implementation utilizes the ad-hoc message passing mode, described in Chapter 5. This simplifies its implementation considerably. This is a reasonable simplification, as the trials conducted with *podsim* in this dissertation focus on very sparse scenarios, in which nodes encounter on average a single peer at a time.

6.4.2.1 Initialization Parameters

The initialization parameters of the ONS module are as follows:

moduleType defines the type of the nodes. It can take the values *gateway*, *node* or *caching*. This is used in the protocol implementation to determine whether the node should be strictly a provider (gateway) or if it is allowed to solicit content.

updateInterval sets the update interval for the module. The peers datastructure is the only aspect of the protocol module which must be periodically updated.

solicitationStrategy sets the solicitation strategy in effect. *none*, *uniform* and *weighted* are currently supported.

privateCacheWeight sets the weight of the private cache [0..1]. This setting is only effective if the *weighted* solicitation strategy is selected.

messagesPerTurn sets the maximum number of content messages to be transmitted in each turn.

broadcastContent is a boolean selecting whether content messages should be broadcast or unicast. Unicast addresses the link layer message to the MAC address of the intended recipient, while the broadcast option causes content messages to be addressed to `FF.FF.FF.FF.FF.FF`.

peerTimeout sets the timeout for unresponsive peers. A peer record is removed from the peers collection after an inactivity period of *peerTimeout*.

timeoutRetriesEnabled is a boolean determining whether pending content and control message timeouts are enabled.

pendingContentTimeout sets the timeout for pending content. This parameter is only effective if *timeoutRetriesEnabled* is true. A request is retriggered after a period of inactivity when expecting a content message.

pendingControlTimeout sets the timeout for pending content. This parameter is only effective if *timeoutRetriesEnabled* is true. A request is retriggered after a period of inactivity after sending a control message. The pending control message is retransmitted when the timer elapses.

channelStatusTimeout sets the period for refreshing the channels. The node will mark a channel of a peer as done if content is not received for it. The done status of all channels is reset periodically, thus retriggering a solicitation sequence.

controlMessageSize sets the control message size in bytes. A single size is assumed for all control messages. This parameter is needed for proper modeling of transmission delay of control messages.

contentDescriptorEnabled is a boolean determining whether the compact content descriptor of control messages sent by a peer is to be examined. Setting this parameter to false ignores the content descriptor, reducing the amount of information that the node has about the peer.

channelDoneMarkingEnabled is a boolean determining whether a peers channel should be marked as done when unable to receive content on it.

exitReport is a boolean defining whether the node shall log an exit report to the scenario scalar file at termination.

6.4.2.2 Protocol Implementation

The ONS module contains the implementation of the solicitation protocol, described in Chapter 5. The protocol implementation is though not a direct implementation of the state machine approach. Rather, a collection of peer datastructures contains state for discovered peers. A single thread of execution drives the protocol for all the peers simultaneously. The peers datastructure is shown in the UML diagram depicted in Figure 6.6.

A node receiving a notification of a new peer from the WNIC module creates a new peer record and adds it to the peers collection. An *announce* message, containing a summary of the nodes channels is sent to the discovered peer. This *announce* is periodically retriggered if the peer is unresponsive, indicating a lost message. The content transfer is however not started until a response is received from the peer. A node receiving an *announce* from an unknown peer will similarly create a new peer record. The response for a roaming node is to send a solicitation *request* message, while a gateway will send an *announce* to invite solicitations. This starts the content transfer sequence which remains active while the peers are in contact. A notification from the WNIC of a lost contact will stop the content transfer attempts and mark the peer as out of contact. The peer record is however not removed until after a timeout period.

A soliciting node receiving an *announce* will attempt to solicit content from the sender by sending a *request* message. The channels to be solicited are determined by the solicitation strategy in effect. The exception are non-soliciting nodes, e.g. those with the defined role of a gateway, which always respond with an *announce* to invite the sending peer to solicit content. A soliciting node will only send an *announce* in response to an

Figure 6.6: UML diagram for the peers data structure

announce if the last message received or sent was of another type. This measure prevents continual bouncing of *announce* messages between two soliciting peers if neither wants content from the other. An *announce* message contains a descriptor field, which carries a summary of the channels of the sending peer. This description can be analyzed by the receiver to determine if the node will attempt to solicit content from the peer. A node which determines a peer to be uninteresting will not attempt to solicit content, but rather responds with an *announce* to invite solicitation. The format of the channel summary is as follows:

Update channel:

```
CH:{channel id};CHN:{channel name};CHT:UPDATE;SET:{T/F};VER:{version};TIME:{timestamp}
```

Feed channel:

```
CH:{channel id};CHN:{channel name};CHT:FEED;SS:{start serial};ES:{end serial}
```

CH is the channel identifier. Integer in this simulation.

CHN is the name of the channel as a string.

CHT is the channel type and can take the values FEED or UPDATE.

SET is used for update type channels, and indicates whether the content has been received. Can take the values T (true) or F (false).

VER indicates the current version of the content which the node carries. Applicable to *update* type channels.

TIME indicates the current timestamp of the content which the node carries. Applicable to *update* type channels.

SS indicates the current lowest content serial on the channel. Applicable to *feed* type channels.

ES indicates the current highest content serial on the channel. Applicable to *feed* type channels.

A node receiving a *request* message checks if it can fulfill the request, at least in part. If not, a *content refuse* message is sent. This message is essentially the content advisory discussed in Chapter 5. A more elaborate advisory is however not needed in this implementation, since a single channel is solicited in each turn. A content *refuse* message must always be followed by an *announce* or *request*. If the receiver of the *request* is able to provide some content, it prepares a number of *content* messages, up to the maximum allowed in a turn, and transmits to the solicitor. The content can be unicast or broadcast, depending on the settings, as discussed in Section 6.4.2.1. The sequence of *content* messages is followed by either a *request* or *announce* message, depending on whether the sender wishes to solicit content from its peer. The *request* message carries the same channels summary as the announce message, which can be analyzed by the recipient to make intelligent choices for future solicitation.

The receiver of a *content refuse* message marks the channel indicated as done for the peer in question. It will thus not try to solicit content for that channel in the near future. Note that this status times out, after which the node will reattempt solicitation on all channels. A receiver of a *content refuse* does not respond to the message in any way, since a following message is expected from the peer, as discussed above.

The receiver of a content message adds the embedded content to its cache, but does not respond to the message in any way, since a request or announce is expected to signal the end of the senders turn.

Losses can be enabled, as discussed in Section 6.4.3. A node can thus loose messages with some probability, meaning that the message is marked as corrupt. The protocol module will not handle corrupt messages. This can cause content or control messages to be lost. A loss of content message causes reduced throughput, but is otherwise not serious, as discussed in Section 5.5. The loss of control message will however potentially stall the protocol, as they are essential for proper turntaking to take place. The protocol implementation of the ONS module can optionally utilize timeouts to handle lost control messages. The parameters are listed in Section 6.4.2.1. If timeout retries are enabled, the node will re-attempt to send the control message in question after a period of inactivity by its peer, thus preventing protocol stalls. A content timeout is employed to retransmit a request for remaining pending content messages after a period of inactivity. This measure is used to attempt to recover quicker from temporary unfavourable conditions which may cause several content messages in a sequence to be lost. An absolute lower bound on the control message timeout is the RTT of an exchange of a control message followed by a content message. The expected RTT is $\frac{L_{control}+L_{content}}{R}$ if the propagation and processing delays are disregarded. This works out to $20.10\ ms$ for a control message size of $512\ bytes$, data message size of $2000\ bytes$ and a link data rate of $1\ Mb/s$. The lower bound on content message timeout is simply $L_{content}/R$, which is $16.0\ ms$ using the parameters stated above. Note however that the timeouts should in reality be considerably longer to account for medium contention.

Content messages in this simulation are serially numbered from one. A node keeps track of the highest serial received on each of its channels. It will solicit content on a channel, starting from the next serial to the highest one received. A node does not attempt to fill gaps in the channel contents caused by lost messages. The simplified implementation does not consider any assembly of content pieces into chunks or complete items. The content pieces are only used for statistical evaluation of reception.

6.4.2.3 Concurrent Contacts

The average concurrent contacts are tracked using an incremental average function of the form

$$\bar{c}_n = \frac{n-1}{n} \cdot \bar{c}_{n-1} + \frac{1}{n} \cdot c_c$$

where *n* is the number of samples (contact events), $\bar{c}_n$ is the concurrent contacts at contact event n and c_c is the current number of known nodes. The average of concurrent contacts is sampled whenever nodes are discovered. Time is not considered, nor is the indicator updated when nodes leave. This method thus only gives an indication of the average concurrent contacts; the estimate is however due to be higher in this case than if time or nodes leaving were considered. The maximum and minimum concurrent contacts are also maintained and logged at the end of each run.

6.4.3 Wireless Network Interface

The *WNIC* module simulates a simple wireless network interface, operating on a single radio channel. The module listens to messages received from the simulated wireless medium on their *medium_in* port. No defined communications channels exist between *WNIC*s in the simulation; the direct message passing facilities of OMNeT++ are employed for transmissions, sending messages directly from a provider to the *medium_in* port of a recipient. The module will automatically reject all received messages on channels other than its own. It can optionally reject received messages with a MAC other than its own defined one, or the broadcast MAC address.

Direct message passing is employed to deliver messages to peers in range (see Section 6.4.3.3), since this method does not require communications channels to be set up between the nodes. The alternative would be to define channels between each node, which is rather unwieldy for a large scenario, especially one with a dynamic node population. The direct message passing is also felt to be closer to the physical reality of the wireless medium.

The WNIC model is extremely simple and essentially ignores all of the MAC layer details of an actual wireless link solution. A conscious decision was made in this implementation to simplify the link layer down to the bare minimum and focus on the protocol itself instead. Modeling an actual link layer, e.g. IEEE 802.11 or Bluetooth, would have added considerable complexity, both to the simulator and analysis of its results. A more sophisticated WNIC model, e.g. IEEE 802.11 from the INET framework for OMNeT++, can be used with a minimum of modifications at a later time, when a more realistic modeling is desired.

The WNIC subscribes, through the *blackboard*, to location updates posted by the *navigator* object. This is required in order to perform validation of received messages and to determine peers in range when broadcasting messages to the wireless medium.

6.4.3.1 Initialization Parameters

MAC sets the address of the interface. A MAC address is generated and assigned if this parameter is not set.

bitrate of the interface in bits.

transmitPower of the interface in mW.

detectionThreshold of the interface in dBm.

radioChannel specified as an integer.

probBitError is the probability of a bit error. Only effective if *enableErrors* is set. Calculated for received messages.

probCollision is the probability of a message loss due to collisions. Only effective if *enableErrors* is set. Calculated for received messages.

enableErrors is a boolean determining whether the interface can loose messages.

associationDelayMean sets the mean association delay in seconds. A minimum delay of 0.5 seconds is required. The notification of a discovered peer to upper layers is delayed by the association delay to simulate the various delays involved in peer discovery. Used with *associationDelaySd* to calculate the actual association delay using a truncated normal distribution.

associationDelaySd sets the standard deviation to use when calculating the association delay, as discussed for *associationDelayMean*

associationLimit is used to simulate hardware limitations by disabling polling for new contacts once the number of known peers has reached this level. This feature is disabled if set to is zero.

exitReport is a boolean defining whether the node shall log an exit report to the scenario scalar file at termination.

enabled is a boolean determining whether the interface is active. If false, the interface will destroy all received messages, both on the host and medium ports. The peer will thus be unreachable over the simulated wireless medium.

6.4.3.2 Broadcast Simulation

The inherent broadcast nature of the wireless medium is simulated by sequentially sending a message to several nodes. Sending a message to all entities in a simulation will however quickly lead to scaling problems. The global object location awareness of the *object factory* is thus used to send a message only to those nodes deemed to be in range at any given time.

Upon receiving a message for transmission from the upper layers, the WNIC queries the factory (see Section 6.3.1) for nodes in range, giving its hosts present location as parameter. The factory responds with a list of pointers to node objects. The WNIC will then sequentially send duplicated messages to the *medium_in* ports of those nodes, using the send direct approach. This measure reduces the number of messages which the simulator has to handle, while capturing the essence of the broadcast nature of the wireless medium.

6.4.3.3 Validation of Received Messages

Sending range calculations and verification of received messages, is determined by the simple point source formula:

$$|I| = \frac{P}{4 \cdot \pi \cdot r^2}$$

where intensity *I* is related to the power *P* by the square of the distance between sender and receiver. Random fluctuation in range, along with dropouts and bit errors are presently simulated with simple boolean probability functions, giving the following expression for the validity of a received message:

$$\Phi_{valid} = (I > I_{min}) \wedge \Phi_{range} \wedge \Phi_{dropout} \wedge \Phi_{error}$$

The probability of a bit error is calculated using $P_{no\,error} = (1 - ber)^L$, where *L* is the message length in bits and *ber* is the bit error rate. This method is used in the OMNeT++ library for determining bit errors on its communications channels. A message containing one or more bit errors is considered lost. Dropouts are determined by a simple boolean probability function. Rough range fluctuations are simulated by applying the random loss potential Φ_{range} to messages whose sending distance is greater than γ_{safe}, determined as a

percentage of the nominal maximum range. More sophisticated models, e.g. for multipath fading, can easily be added at a later time.

6.4.3.4 Transmission Delay Simulation

Channel data rate is simulated by delaying a message queued for transmission by $T_d = \frac{L}{R}$, where L is the message size in bits and R is the modeled channel data rate.

6.4.3.5 Peer Discovery Services

The WNIC provides peer discovery services by periodically polling the factory (see Section 6.3.1) for nodes in range, giving its hosts present location as parameter. The WNIC subscribes to location updates posted to the *Blackboard* by the navigator. The factory responds with a list of pointers to node objects deemed to be within range. Unknown peers are added to a list maintained by the WNIC and a notification provided to upper layers (the ONS module) after a delay period.

The association latency of the wireless interface is modeled by a truncated normal distribution. The mean and standard deviation of the probability distribution are input into the module via the initialization file. The delay represents the period from first opportunity of discovery until the protocol layer becomes aware of the new peer.

Peers in the WNICs collection time out relatively quickly if not refreshed during the periodic polls. Known peers which are not reported by the factory in the next update period can be assumed to be out of range. They are removed from the WNICs peer collection after the timeout period and a notification of lost contact provided to upper layers.

6.4.4 Mobility Modules

The mobility modules handle location updates of mobile nodes for the duration of the simulation. *Random waypoint (RWP)* and *trace mobility* modules, based on the Mobility Framework (MF), were created for the *podsim* mobility support. The modules are derived from the *BasicMobility* base class of MF and utilize the *Blackboard* to communicate location updates to subscribing modules.

6.4.4.1 Random Waypoint Module

The *RandomWaypointMobility* module employs the *random waypoint (RWP)* (Johnson, Maltz, & Broch, 2001) mobility model. Although the realism of this model has been called into question (Yoon, Liu, & Noble, 2003; Navidi & Camp, 2004; Bettstetter, Resta, & Santi, 2003) it is nevertheless an established reference widely used in mobility research. The guidelines of (Navidi & Camp, 2004) were followed to initialize the simulation in the steady-state distribution. In addition, velocity and pause times were chosen from a truncated normal distribution, with a non-zero lower bound.

The mobility parameters of the navigator are set at initialization time, after which the node moves autonomously until it is destroyed. The parameters are:

x, y and z coordinates of the initial location. Negative initial location causes the module to choose initial x and y location from a uniform distribution, constrained by the scenario size.

updateInterval at which the node location is recalculated. A one second interval is default. Direction and distance of travel at each update is calculated from the node specified node speed and interpolation of the present and next waypoints.

velocity and velocitySd specifies the mean velocity in m/s and the standard deviation. A truncated normal distribution with a min of 0.5 m/s is used for randomly choosing velocity at each waypoint. The velocity is constant until the next waypoint is reached.

pauseTimeMean and pauseTimeSd specify the mean pause time and standard deviation employed to calculate pause times at each waypoint from a truncated mean distribution.

6.4.4.2 TraceMobility Module

The *TraceMobility* module is designed for use in the *trace mobility model*, described in Section 6.3.1.

The node factory initializes the navigator at creation time with the waypoints and associated transit velocities for the entire run, using the *setMoveEvents* method of the *TraceMobility* module. The navigator runs autonomously there after, until the node is finally destroyed at the end of its scripted run. Location update events are handled in sequence, interpolating the nodes position between waypoints. The node will optionally pause for a period of time when the waypoint is reached.

The initialization parameters of the *TraceMobility* module are:

x, y and z initial location of the node, set by the object factory on its creation.

updateInterval at which the node location is recalculated. A one second interval is default. Direction and distance of travel at each update is calculated from the node specified node speed and interpolation of the present and next waypoints.

6.5 Summary

A simple simulation model, *podsim*, for a PodNet community, has been presented. The primary purpose of the simulator is to present an implementation of the solicitation protocol of Chapter 5. Simulations of complex systems, comprised of several independent gateways and a number of dynamically generated roaming nodes, are supported. A random waypoint model is implemented, in which a number of nodes (fixed for the duration of the simulation) mingle within a predetermined region and exchange content. A second mobility model uses externally generated trace files to dynamically create and navigate a collection of nodes. This provides great flexibility in simulation of opportunistic scenarios in which nodes are entering and leaving the area of interest during the simulation. Chapter 7 presents results of simulations, performed with *podsim*, for the purposes of verifying the content solicitation protocol and some initial explorations into the behaviour of PodNet communities.

Chapter 7

Evaluation

This chapter presents evaluation, via the *podsim* simulator described in Chapter 6, of the content solicitation protocol, presented in Chapter 5. Basic verification tests of the protocol implementation of Chapter 6, is presented in Section 7.1. Analysis of content transfer in a multi-channel system is then presented in Section 7.2.

7.1 Protocol Verification

An implementation of the solicitation protocol design of Chapter 5 is presented in the simulation model, described in Chapter 6.

In this section, two key aspects of the solicitation protocol, are be verified through simulation and analysis of the results. This section considers the following factors:

- The effects of using message history for reduction of control traffic.
- The effects of lost content and control messages.

7.1.1 Experimental Setup

A static gateway is placed in the center of a $1000 \times 1000\ m$ square. A number of mobile nodes are dynamically created and controlled by a trace navigation script. Ten feed type channels are specified for the system. The gateway has infinite content available on all ten channels. The mobile nodes select one channel at random at startup as their private channel, while the other nine are carried as public. A rather large coverage of approximately $\Delta = 100\ m$ was chosen for this test. A truncated normal distribution is used for

the association latency, with $\mu = 3\ s$ and $\sigma = 1\ s$. A 1 Mb/s link data rate is used. A content exchange turn is composed of 10 data messages, each 2000 bytes, followed by two control messages of 512 bytes. The *none*, *uniform* and *weighted* solicitation strategies were used in these trials. A private channel weight of 0.5 was used in the weighted solicitation cases.

7.1.2 Optimization of Control Message Traffic

The protocol includes measures for minimizing extraneous control message traffic. The peers data structure of *podsim* is used to keep track of the channels which the node has interest in and which of those have already been tried. A node can thus intelligently solicit its peer and thereby reduce the message traffic. This state is periodically reset, at which time the nodes will retry solicitation on all their channels. The announce and request messages carry a summary of the channels which the sending node carries, the content descriptor. This list can be used by the recipient to make intelligent solicitation choices for the peer in question, further reducing control message traffic.

The purpose of this trial is to evaluate the effectiveness of these measures. The simulation is first run with both the content descriptor and channel history marking disabled. A second run is then done with the optimizations enabled, and the results compared.

The experimental setup of Section 7.1.1 was used for this trial with the mobility scenario shown in Figure 7.1. Node 1 travels on a rectangular path, as shown, while node 2 is static for the duration of the simulation. Node 1 first moves through the coverage of the gateway, where it downloads contents at full capacity for the duration of the contact. It then encounters node 2, which will solicit content in accordance with its strategy. This represents the simplest possible scenario which demonstrates both the interaction of a gateway with a mobile node and the content carrying capabilities of a mobile to an isolated node. Losses are disregarded in these cases for simpler analysis.

7.1.2.1 Optimizations Disabled

Let us first consider the very naive version of the protocol which has no memory of past exchanges or knowledge of its peers contents. The parameters *contentDescriptorEnabled* and *channelDoneMarkingEnabled* are set to false. Refer to Section 6.4.2.1 for further discussion on these parameters. This translates to continuous blind solicitation by all nodes, regardless of past exchanges or content descriptor.

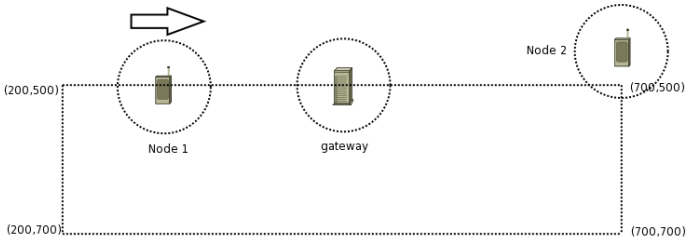


Figure 7.1: Basic verification scenario with two mobile nodes and a single gateway. Node 1 travels around a rectangular path with a constant velocity. Its path takes it through the coverage of the gateway, which has infinite content, and into the range of the static node Node 2. A radio range of approximately 100 m is indicated by a dotted circle.

The results are shown in in Table 7.1. *Sent* is the number of content messages sent by the node. *Received* is the private and public content received by a node. *Associated time* is the total duration of associations. *Associated time - sending* is the duration of time that a node is not receiving, and is used for throughput calculations. This can be calculated for node 1, given that the gateway is strictly a provider. Node 2 is strictly a receiver in this case, and will thus not spend any time sending. Control messages, sent, received and total are finally shown. This includes both request and announces.

Node 1 solicits and receives content from the gateway while within its coverage. The efficiency can be seen to be $\approx 0.95Mb/s$ for a $1Mb/s$ link. This is as expected for the protocol behaviour of a turn composed of 10 data messages, followed by two control messages. Node 1 is seen to receive slightly fewer messages than the number sent by the gateway; the discrepancy is explained by the fact that some messages sent by the gateway are lost due to the movement of node 1 out of its coverage. All the content received by node 1 is on its single private channel in the none and uniform cases, while the content received is distributed between the public and private caches in the weighted case by about the 0.5 ratio specified as the private channel weight. Node 2 receives no content at all in the none and uniform cases. The "none" strategy always requests content on the private channels, disregarding all public channels. The uniform strategy degenerates to none in this case since the nodes maintain no knowledge of their peer's status. Node 2 receives some content in the weighted solicitation case, but the private v.s. public content received is characterized by the available content at the provider; node 2 will ask for content on its private channel roughly $50\ \%$ of the time while node 1 will not have extensive contents for the channel, given that it is a public channel in its case. Control message traffic is seen to be quite extensive. As the nodes maintain no knowledge of their peers status, the interactions between nodes 1 and 2 are characterized by a large number of request/reject exchanges. This can easily be optimized as discussed in the next section.

Strategy	Node	Sent	Received		Associated time		Throughput	Control messages		
			Private	Public	Total	Sending		Send	Receive	Total
None	gw	17020	0	0	287.8	287.8	946.3	1704	1703	3407
	node 1	0	17008	0	452.6	164.8	0.0	21859	21853	43712
	node 2	0	0	0	166.3	0.0	0.0	20154	20151	40305
Uniform	gw	17020	0	0	287.8	287.8	946.3	1704	1703	3407
	node 1	0	17008	0	452.6	164.8	0.0	21859	21853	43712
	node 2	0	0	0	166.3	0.0	0.0	20154	20151	40305
Weighted	gw	17010	0	0	287.6	287.6	946.3	1703	1702	3405
	node 1	8880	8487	8511	453.4	165.8	856.8	4218	5099	9317
	node 2	0	830	8039	167.3	0.0	0.0	3400	2510	5910

Table 7.1: Protocol verification tests. Optimizations disabled

Strategy	Node	Sent	Received		Associated time		Throughput	Control messages		
			Private	Public	Total	Sending		Send	Receive	Total
None	gw	17150	0	0	289.9	289.9	946.6	1719	1716	3435
	node 1	0	17134	0	453.3	163.4	0.0	1738	1735	3473
	node 2	0	0	0	164.9	0.0	0.0	22	18	40
Uniform	gw	17070	0	0	288.6	288.6	946.4	1711	1707	3418
	node 1	9730	17057	0	451.9	163.3	953.2	2722	2700	5422
	node 2	0	0	9713	164.8	0.0	0.0	995	1009	2004
Weighted	gw	17120	0	0	289.4	289.4	946.4	1716	1713	3429
	node 1	9703	8370	8737	452.3	162.9	953.2	2720	2700	5420
	node 2	0	780	8908	164.3	0.0	0.0	990	1001	1991

Table 7.2: Protocol verification tests. Optimizations enabled

7.1.2.2 Optimizations Enabled

The previous trial was repeated, while enabling optimizations based on prior knowledge of peer status. The parameters *contentDescriptorEnabled* and *channelDoneMarkingEnabled* were set to true, resulting in a more intelligent solicitation, as previously discussed. A channel status timeout of $10\ s$ was used, after which the nodes will retry solicitation of contents on all channels from the peers. The results are shown in Table 7.2. Refer to the previous section for further description of the individual columns.

The number of control messages is seen to be reduced considerably for the two mobile, when compared to the previous results, shown in Table 7.1. This is significant improvement, since excessive control traffic will congest the medium in more complex scenarios and thus unavoidably lead to reduced overall throughput. The optimizations enabled in this simulation run can thus potentially increase system efficiency.

Throughput is essentially equal to the previous cases, since the excessive control messages are mostly a problem when content is not available; rich content scenarios will thus not be affected in the simple case with two nodes. This is demonstrated by the control message traffic sent and received by the gateway node in the previous test, shown in Table 7.1.

7.1.2.3 Summary

The control message traffic can clearly be seen to be reduced considerably, when the optimizing measures of examining the content descriptor and keeping track of past exchanges, are enabled. Excessive control traffic is undesirable, since it will decrease the availability of the wireless medium. These measures are thus clearly of some benefit.

The optimizations are though not seen to be effective in rich content scenarios, e.g. when a node encounters a gateway with infinite content. This is to be expected, since the excessive control traffic is only a problem in cases where content is scarce.

Node 2 is seen to receive some private content in all cases when the weighted solicitation strategy is used, while only public content is received in the uniform cases. This means that the content desired by the owner of node 2 is not being delivered in the uniform solicitation cases; all its received content is for the future benefit of other users. Weighted solicitation is thus clearly of some benefit, at least in this simple scenario.

7.1.3 The Effects of Losses

Losses were disregarded in the previous trials, which is hardly realistic in a wireless scenario. The protocol employs timeout mechanisms to trigger retries, as described in Chapter 5 and Section 6.4.2.2. A timer is set when control messages are sent. The timer is decremented until a response is received from the peer. The control message is retriggered if the timer elapses without any response from the peer, indicating a lost message. Similarly, a timer guards received content messages. The timer is set each time a content message is received. We can assume that several sequential content messages have been lost if this timer elapses.

The uniform and weighted solicitation cases described in the previous sections were rerun, but now enabling losses due to bit errors and collisions. The experimental setup was otherwise unmodified. The error probabilities used were $p_{bit.error} = 10 \cdot 10^{-6}$ and $p_{collision} = 10^{-6}$.

7.1.3.1 Timeout Retries Disabled

All measures to handle the errors were initially disabled by setting *timeoutRetriesEnabled* false. Refer to Section 6.4.2.1 for further discussion on this parameter. This disables all handling of losses.

Strategy	Node	Sent	Received		Associated time		Throughput	Control messages		
			Private	Public	Total	Sending		Send	Receive	Total
Uniform	gw	1215	0	0	144.4	144.4	134.6	218	215	433
	node 1	470	1029	0	309.0	164.5	45.7	430	353	783
	node 2	0	0	389	165.4	21.0	0.0	150	194	344
Weighted	gw	1238	0	0	144.4	144.4	137.2	223	220	443
	node 1	579	504	545	309.0	164.6	56.3	380	337	717
	node 2	0	36	469	165.4	21.0	0.0	129	141	270

Table 7.3: Protocol verification tests. Lossy scenario with loss handling disabled

The results are shown in Table 7.3. The throughput for both the gateway and node 1 are seen to drop dramatically. The primary reason for the drop is stalling of the protocol due to loss of control messages as discussed in Section 5.5.

7.1.3.2 Timeout Retries Enabled

Retries were now enabled by setting *timeoutRetriesEnabled* true. This enables handling of losses by retransmissions upon elapsed control and content message timers.

The effect of enabling retries due to lost control messages was explored by varying the timeout interval for (a) control messages, (b) content messages and (c) both combined from 5000 ms down to 100 ms in several steps. A single simulation run was made for each timeout value, using the weighted solicitation strategy.

Figure 7.2 shows the ratio of timeouts to content messages for varying timeout intervals. The timeout interval in ms is plotted on the x-axis, and the number of timeouts to content message ratio on the y-axis. A bit error rate of 10^{-6} gives a probability of ≈ 0.002 of corrupt content messages and ≈ 0.0005 of corrupt control messages. It is therefore to be expected that the retry to content message ratio lies somewhere between those two values, as can indeed be seen from Figure 7.2.

Figure 7.3(a) shows how the efficiency of the protocol increases as the timeouts are shortened. The throughput of the lossless case, 946.5 Kb/s, is approached as the timeouts are shortened.

Figure 7.3(b) demonstrates the ratio of control messages to the number of sent content messages. The lossless case has a mean ratio of 5.94 content message to every control message. This ratio can be seen to be considerably lower in the lossy case, or about 3.5 content messages to every control message. The ratio can however be considered to be constant for all reasonable timeout values, as is expected due to the constant bit error rate.

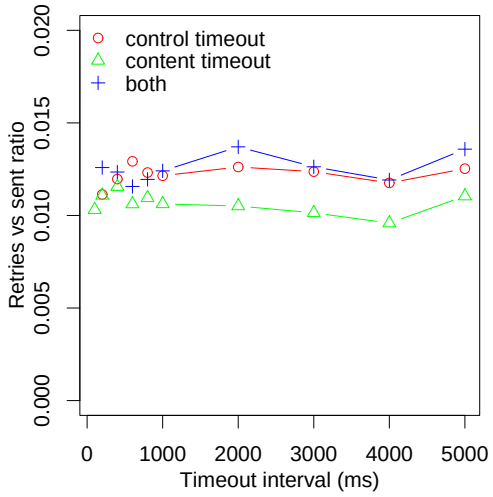


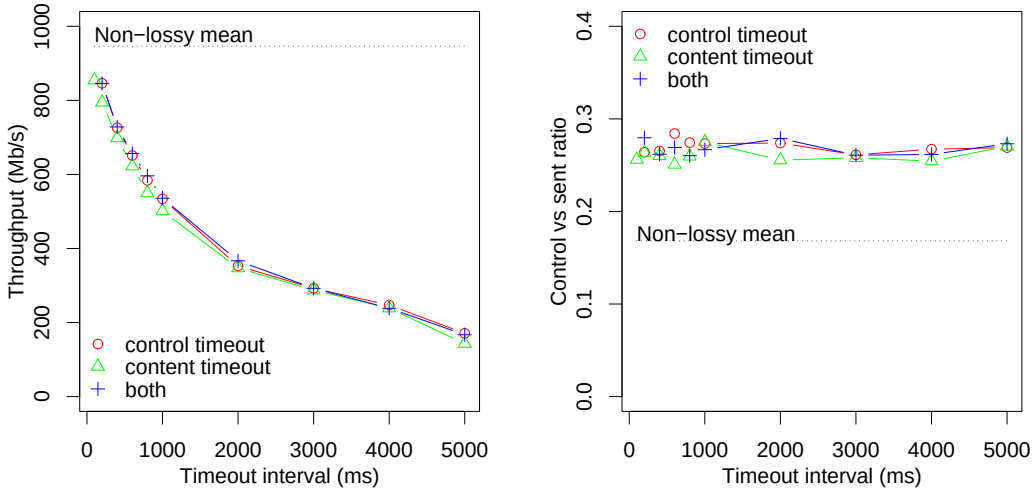
Figure 7.2: Ratio of sent content versus timeout events for timeout intervals of various lengths.

7.1.3.3 Summary

In summary, the simple timeout mechanisms of the protocol can clearly be seen to increase efficiency. The available bandwidth will, however, be more limited in a more complex scenario. It is thus unlikely that rates like those found in this experiment are realistic and hence that the timeout values have to be more conservative. Indeed, the simulation model runs into problems when using timeouts shorter than 100 ms, as the simulated RTTs can be longer, giving rise to frequent and spurious retries. The unnecessary retry events have the effect of causing the protocol module to produce considerably more content messages than the bandwidth of the channel allows to be transmitted. Further exploration of realistic timeout values and more optimal strategies to ensure efficiency are however reserved for future work.

7.2 Multi-Channel Square System

The main purpose of the simulator is to investigate the behaviour of a multi-channel PodNet community. This section reports several experiments and their analysis on the behaviour of such communities. Contact metrics are first presented in Section 7.2.2, followed by an analysis of content transfer characteristics in Section 7.2.3. Finally, the content distribution characteristics of a single small content item to a population of nodes are discussed in Section 7.2.4.



(a) (b)

Figure 7.3: Effects of loss events and timeouts on throughput and efficiency. (a) throughput vs timeout interval. (b) Ratio of control messages to content vs timeout interval.

7.2.1 Experimental Setup

A simple square scenario was created, in which a single gateway node resides at the center and a number of mobile nodes mingle within a $1000 \times 1000\ m$ rectangular region, as shown in Figure 7.4. The node density, that is the ratio of maximum aggregate radio coverage to the square area, is the important factor in this experiment. It is thus the scenario area, rather than its shape, which is of interest in this experiment. Preliminary runs have shown statistically identical results for rectangles with equal areas, given that the shorter side is at least $\geq 2 \cdot radio\ range$.

All trials focused on relatively sparse scenarios of up to 50 nodes. Sparse scenarios can be considered more of a challenge for mobility driven content dissemination, while conversely, the performance can be assumed to be quite good in dense scenarios with many contact opportunities. The number of nodes in each trial were fixed for the duration of the simulation, which was $7200\ s$, unless otherwise noted.

A radio range of approximately $\Delta = 10\ m$ was chosen for all tests, analogous to that of Class II Bluetooth devices. A truncated normal distribution was used for the association latency, with $\mu = 3\ s$ and $\sigma = 1\ s$.

The *random waypoint* mobility model was used for all tests (see Section 6.4.4.1). The mobile nodes were placed according to the guidelines set forth by Navidi and Camp (2004) to initialize the population in the steady-state distribution. The speed of the nodes was picked at random from a truncated normal distribution with $\mu = 1.5\ m/s$ and $\sigma = 0.5\ m/s$, with

a minimum speed of 0.5 m/s, thus avoiding the zero speed problem of RWP. The choice of the mean walking velocity was inspired by Wiseman (2007). A truncated normal distribution was also used for the pause times, using $\mu = 20\ s$ and $\sigma = 10\ s$. Although not based on a real measurement, we feel that this number is not unreasonable for the behaviour of users in an open area. A further treatment of the effects of pause times is given by Karlsson et al. (2007).

The *ad-hoc communications mode* was used for all simulations, in which nodes are free to communicate with any node within range at any time. The pairwise restriction of contacts is thus lifted.

Errors and losses were disabled in these runs for ease of analysis. Enabling losses can be assumed to give reduced throughput as discussed in the previous section.

Each data point in the following is a mean aggregate of 50 runs, which is commonly assumed to be sufficient for the error to be approximately normally distributed. Each of the runs was initialized with a randomly chosen seed, giving a fair degree of randomness to the process. 95% confidence intervals are employed, where applicable.

1, 2, 4 and 6 channel systems were used in the runs. Each mobile node picks one of the available channels at startup as its private channel, while carrying the other ones as public. *Uniform* and *weighted* solicitation strategies were employed in the simulations. The public/private weight for the weighted case was set to 0.5, giving $p_{pr} = 0.5$ and $p_{pu,i} = 0.5 \cdot \frac{1}{N}$. A random request order of public channels was used in the uniform case, with the probability of requesting content on public channel *i* $p_{pu,i} = \frac{1}{N}$, where N is the number of public channels.

7.2.2 Contact Metrics

This section investigates the contact statistics obtained by running the simulator on the square RWP scenario, described in Section 7.2.1. The contact metrics considered are: number and duration of contacts, average association latency and concurrent contacts. The results are important for further analysis of experiments, and serve as further verification of the simulator implementation of Chapter 6.

7.2.2.1 Number and Duration of Contacts

Simple relations between node density and the number of contacts and contact duration can be established for a RWP mobility model in a closed system. It is relatively simple

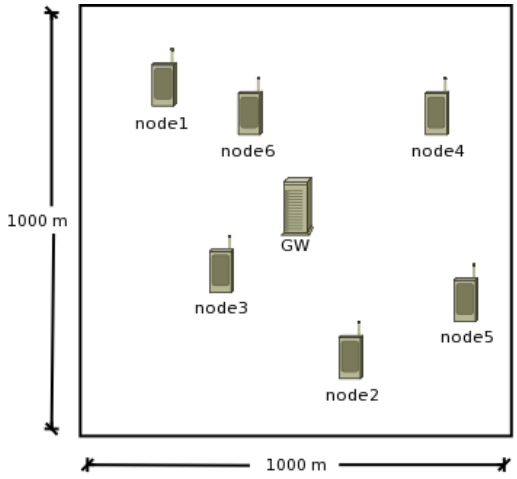
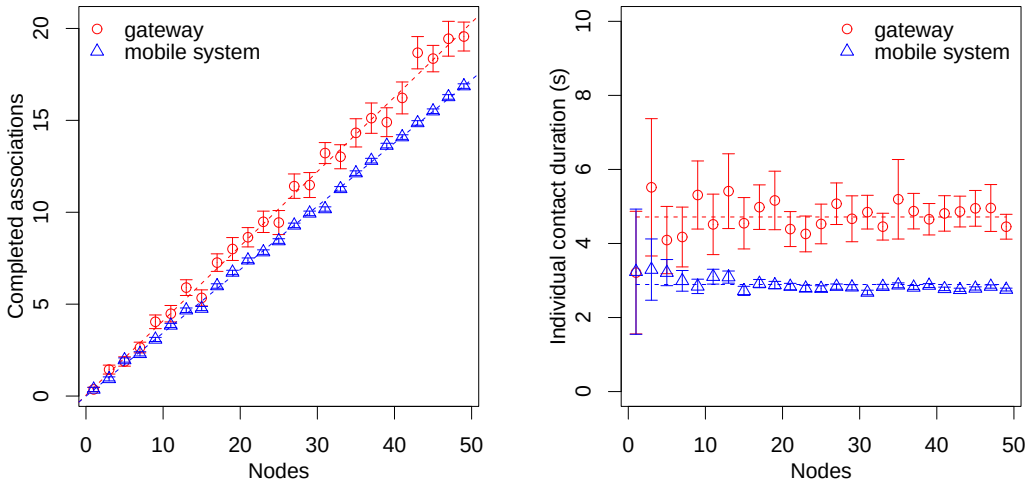


Figure 7.4: Experimental setup of square simulations. Square with a single gateway and a number of mobile nodes, which move autonomously according to the random waypoint mobility model.

to see that we can expect (1) the average number of contacts made in the system scales quadratically with the number of nodes in the system and 2) the contact duration is on average constant.

Let us consider a simple example. A contact is established when nodes cross paths. For a two node system n_1 and n_2, the number of contacts is on average some constant c. Adding a node n_3 to the system produces an expected number of contacts $c' \approx c_{n1,n2} + c_{n1,n3} + c_{n2,n1} + c_{n2,n3} + c_{n3,n1} + c_{n3,n2} \approx 6 \cdot c$. Adding a node n_4 produces $c'' \approx c_{n1,n2} + c_{n1,n3} + c_{n1,n4} + c_{n2,n1} + c_{n2,n3} + c_{n2,n4} + c_{n3,n1} + c_{n3,n2} + c_{n3,n4} + c_{n4,n1} + c_{n4,n2} + c_{n4,n2} \approx 12 \cdot c$. In general, $c \cdot n \cdot (n-1)$ contacts are made in a n node system, assuming a constant node density and mobility parameters. The number of contacts that a particular node n_n makes can thus be assumed to scale linearly with the number of nodes, giving $c \cdot (n-1)$ contacts on average. This approximation is however only valid for a sparse scenario where the ratio of the aggregate radio range of the nodes is small compared to the size of the simulation environment.

The number of contacts for a gateway node are shown in Figure 7.5(a) for a run with a single channel and *uniform* solicitation strategy. It is seen to scale linearly with the number of nodes in the system as expected. An expression for the expected number of contacts for a gateway in a $1000 \times 1000\ m$ square is $n_{c,gw} \approx 0.41 \cdot (n_{nodes} - 1)$, while the expected number is $n_{c,mh} \approx 0.35 \cdot (n_{nodes} - 1)$ for a mobile host. The expected number of contacts for a gateway is seen to be higher, most likely an artifact of the RWP behaviour. The gateway is placed at the center of the square, where the steady-state distribution of RWP is known to be highest; it is therefore logical that a static node in the most densely



(a) Total number of contacts (b) Contact duration

Figure 7.5: Total number and duration of contacts for a single channel system.

populated area would encounter the most peers. Nearly identical results were obtained for 2, 4 and 6 channel systems and the weighted solicitation strategy. The number of contacts is thus a function of the node density and mobility parameters, rather than the solicitation strategy, as would be expected.

The duration of contacts, shown in Figure 7.5(b), is seen to be independent of the node density, as would be expected. The average contact duration for the gateway in the single channel case is $\bar{d_g} = 4.72\ s$. The average individual contact duration for the entire mobile system is however $\bar{d_m} = 2.89\ s$. This is to be expected since the relative velocity of two mobile nodes is on average $2 \cdot \mu_v$, while the static gateway sees a mean relative velocity of μ_v. A contact duration of $d_m \approx \frac{1}{2} \cdot d_g$ is therefore to be expected. The number of samples for the mobile system is here far greater than for the gateway and d_m can thus be considered more reliable. The numbers given above are for a run with a single channel. Identical runs with 2, 4 and 6 channels give comparable results. The results are similarly independent of the solicitation strategy as expected.

A best case average contact with a gateway node occurs when the mobile node is on a path that coincides with its location. For a radio range of $\Delta \approx 10\ m$ and a mean velocity of $\bar{\nu} = 1.5$, the expected duration would be $\hat{d} = 13.3\ s$. The path of the mobile node will however more likely be on a trajectory that traces a chord though the gateways circle of coverage, giving a shorter contact. The discovery and association latencies must further be subtracted from the potential contact time. The results obtained from the simulator can thus be assumed to be reasonable, although a derivation of the expected contact duration for the scenario has not been done.

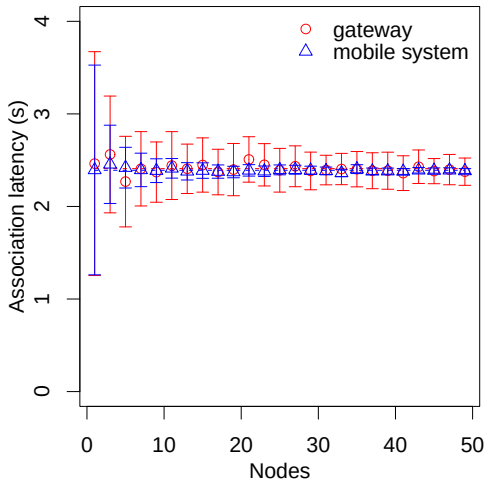


Figure 7.6: Average association latency for a gateway node

7.2.2.2 Association Latency

Association latency includes factors such as node discovery, notification and negotiation. It is of great importance in a system, such as PodNet, where the performance depends on taking maximum advantage of the scarce contact opportunities. Latency of several seconds is common (Karlsson et al., 2007; Wacha, 2007) and must be accurately accounted for in the simulation. The purpose of the trial reported in this section is to verify this aspect of the simulation.

The simulation model uses a truncated normal distribution to model the various combined factors involved in the association latency. The runs done here use a truncated normal distribution with $\mu = 3\ s$, $\sigma = 1.0\ s$ and a minimum of $0.5\ s$ to model the association latency. In addition, an additional average delay of $0.5\ s$ is expected since the *WNIC* model polls its environment for new nodes at $1\ s$ intervals.

The association latency is shown in Figure 7.6. An average latency of 3 seconds is expected from the simulation parameters. The latency can be seen to be somewhat lower, which is explained by the fact that the node which associates quicker notifies its partner, at which point its association period is cut short. Thus, an average association period, which is somewhat shorter than the μ used for the distribution, is to be expected. Nearly identical results were observed for both uniform and weighted solicitation strategies, and 1 ,2, 4 and 6 channel systems. The association latency is thus confirmed to be independent of other system parameters and reasonable with respects to the probability distribution used for its simulation.

7.2.2.3 Concurrent Contacts

Concurrent contacts are of interest, for example to determine the probability of interference between nodes. It is also of interest to measure the potential benefit from using broadcast dissemination versus unicast for content dissemination. The contact metrics are collected by the protocol module of the simulator, as discussed in Section 6.4.2.3. Let us now consider the number of concurrent contacts for the experimental setup described in Section 7.2.1.

The number of concurrent contacts is shown in Figures 7.7 for a single channel run using uniform solicitation strategy. The average concurrent contacts is seen to be close to 1 for both the gateway and as an aggregate of the mobile system. The maximum concurrent contacts were also observed and are shown here, both as an average across the population and the absolute maximum observed. The average of the maximum is seen to be approaching the maximum concurrent contacts for the gateway, indicating increased utilization with increased node densities.

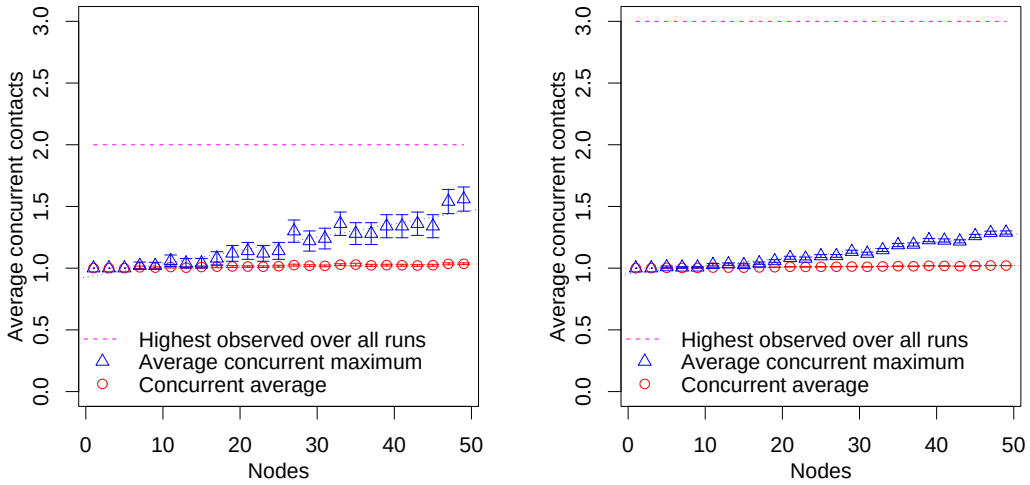
Identical runs were made for 1, 2, 4 and channels, as well as both uniform and weighted solicitation strategies. The greatest number of concurrent contacts seen across all runs was 4 peers, while the average maximum was 1.1 peers for the mobile system and 1.2 for the gateway.

The concurrent contacts for sparse scenarios of up to 50 nodes are thus seen to be rather low, indicating that the contacts can be considered to be essentially pairwise. This implies that the ad-hoc communications mode is equivalent to the pairwise one for low node densities, as would indeed be expected.

7.2.3 Content Transfer Characteristics

7.2.3.1 Content Supplied by the Gateway

The total amount of content supplied by a gateway node is shown in Figure 7.8(a). It can be seen to scale linearly with the number of nodes, which follows from the previous finding that the number of contacts scales linearly with the number of nodes while each contact can be considered to be of constant duration. It can also be considered to be independent of the number of channels in the system and the solicitation strategy. The gateway is here modeled with infinite content, and is thus able to satisfy any request. For a lossless scenario, this means that its utilization is at maximum during each contact opportunity, regardless of the number of channels or solicitation strategy. Figure 7.8(a) shows



(a) Gateway (b) Mobile system

Figure 7.7: Concurrent contacts

the average content delivered per node in the mobile system. It can be approximated to be constant at $\approx 535\ Kb$ in this topology for all node and channel numbers and for both solicitation strategies. This approximation is validated by considering the 95% confidence intervals shown in Figure 7.9 for a 6-channel system using uniform solicitation. Comparable results were found for the other runs.

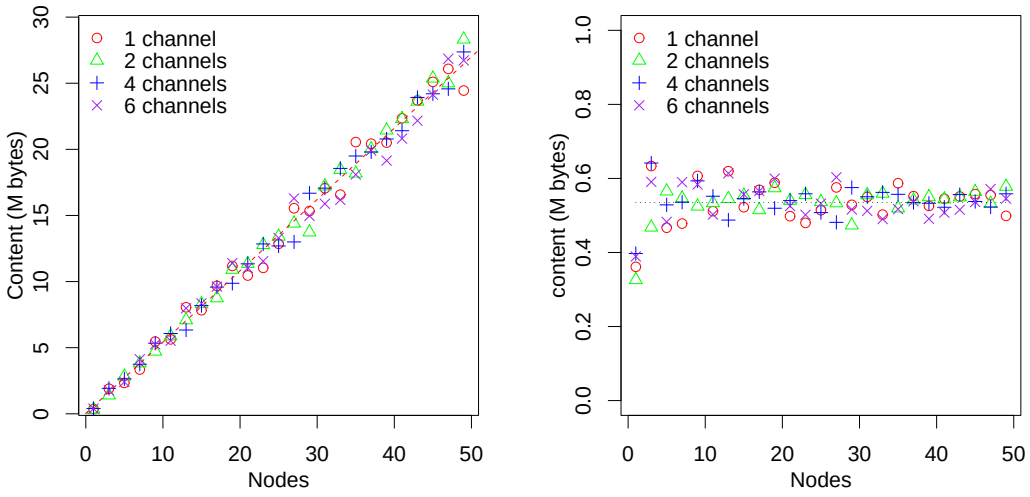
Figure 7.10 shows the throughput of the gateway for runs using (a) uniform and (b) weighted solicitation strategies. It is seen to be slightly decreasing as the number of nodes in the system increases, most likely caused by increased control message traffic as the probability of multiple concurrent contacts increases.

7.2.3.2 Mobile System Content Transfer

The performance of a system comprised of multiple nodes each carrying a subset of the total available channels is of particular interest for assessing the functionality of a content distribution system. The performance will depend on several factors, most importantly the node densities or arrival rates and the topology in which it is deployed.

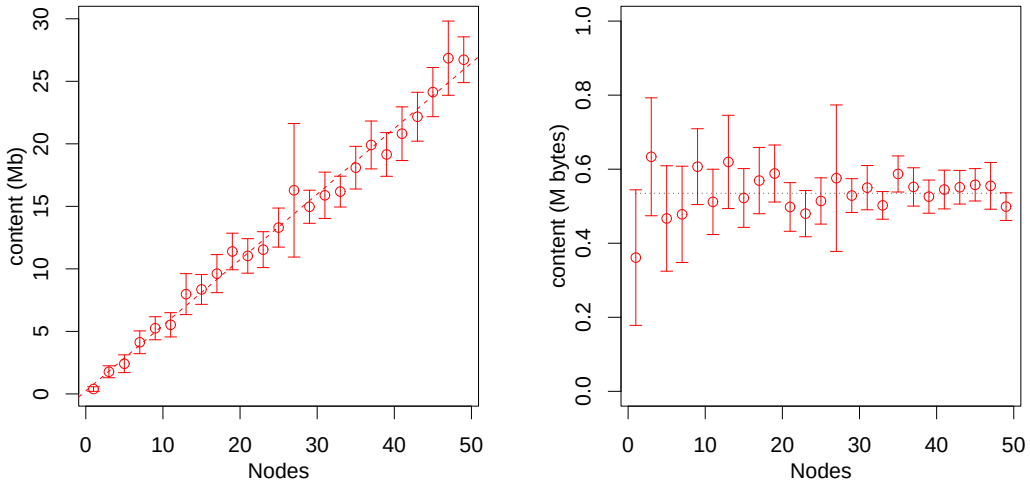
Multichannel tests were conducted in which a varying number of channels were specified for the system. The channels were in all cases defined globally to the system, giving each node initial knowledge of all channels. Each node chose one channel at random as private (subscribed), while the others were carried as public channels.

The total content exchange within the closed mobile system is shown in 7.11. The contribution of the gateway is shown as a reference by a fitted line, using the results from



(a) Total (b) Per node

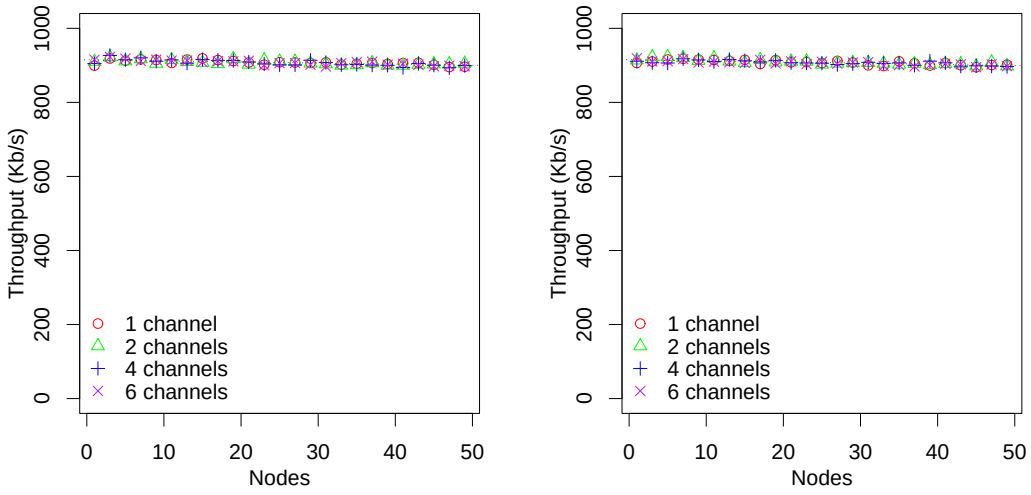
Figure 7.8: Content supplied by a gateway node. Uniform solicitation strategy in 1, 2, 4 and 6 channel systems.



(a) Total (b) Per node

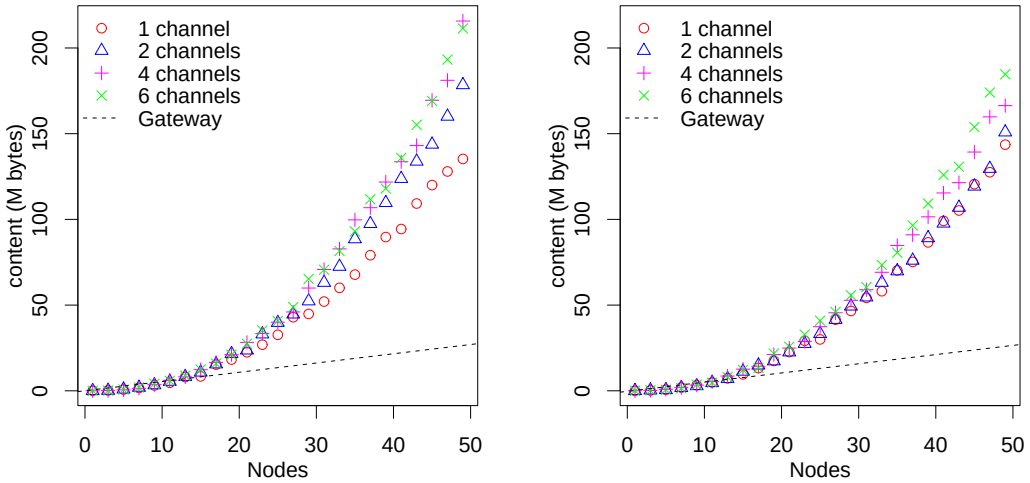
Figure 7.9: Content supplied by a gateway node. Uniform solicitation in a 6-channel system.

the previous sections. The mean contribution of the gateway is however subtracted from the mean content received by the mobile system in this case. It is apparent that the content exchange of the mobile system exceeds the gateways contribution at $n \approx 12$ nodes for both solicitation strategies, showing the effect of the peer-to-peer content exchange in the mobile system. Note however that this is the total amount exchanged, so there will be considerable amount of duplicate content supplied. Figure 7.12 shows the mean content transfer per node, rather than the whole system. The content transfer per mobile



(a) Uniform solicitation (b) Weighted solicitation

Figure 7.10: Gateway throughput. Uniform and weighted solicitation strategies.



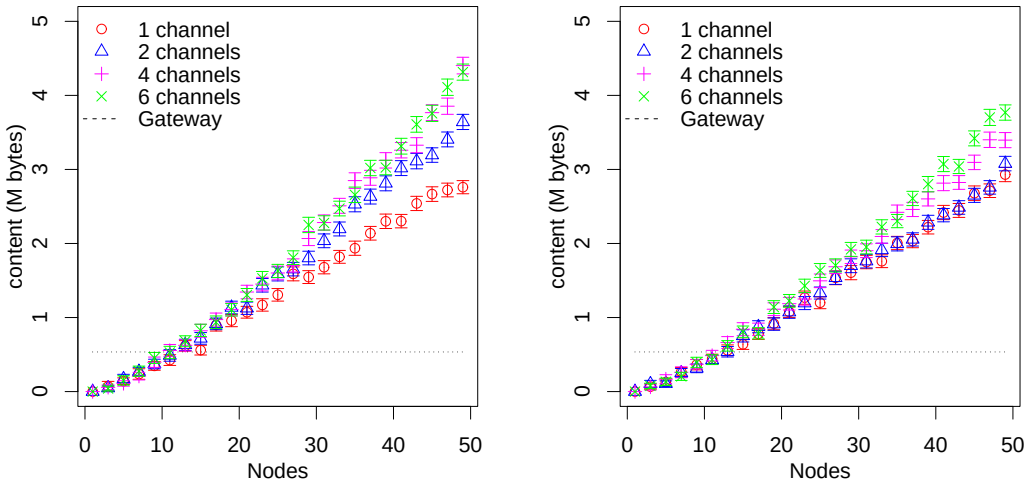
(a) Uniform solicitation (b) Weighted solicitation

Figure 7.11: Total content transfer in the mobile system.

node is seen to increase approximately linearly with the number of nodes in the system as expected from the results of the preceding sections.

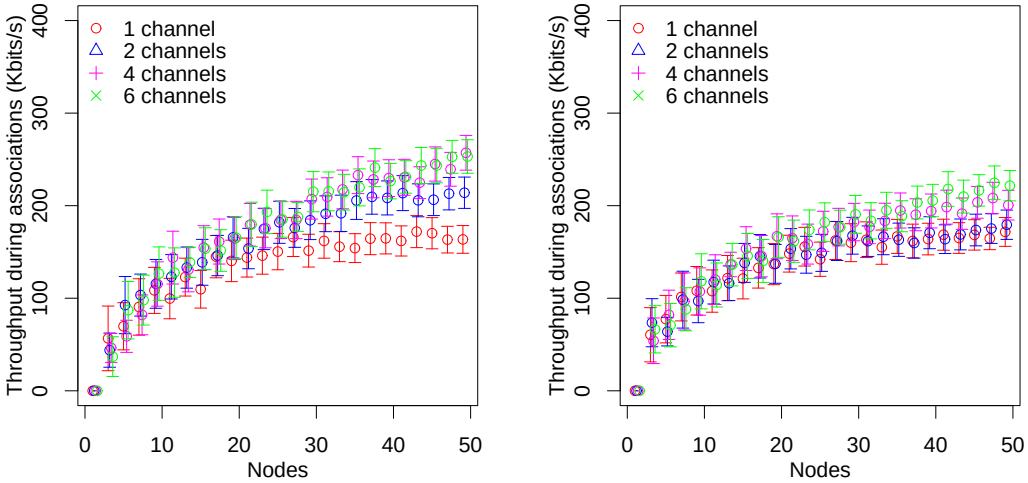
Figure 7.13 shows the throughput during association periods. The throughput appears to approach a maximum value, which is a function of the number of channels, although runs with larger systems are required to establish that.

It is apparent that the amount of total content exchanged increases with the number of channels in the runs discussed above. A reasonable explanation is that there is greater aggregate amount of content of interest as the number of channels increases, and thus more contacts result in transfer. This will however have to be established in future work.



(a) Uniform solicitation (b) Weighted solicitation

Figure 7.12: Mean content transfer per node in the mobile system.

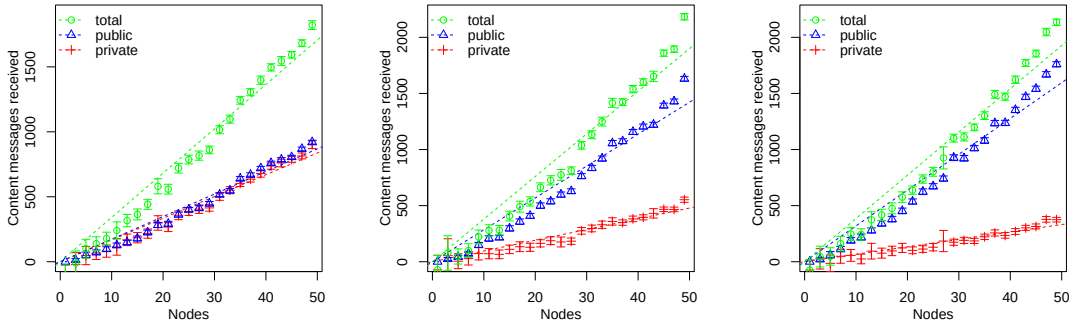


(a) Uniform solicitation (b) Weighted solicitation

Figure 7.13: Mean throughput in mobile system during associations.

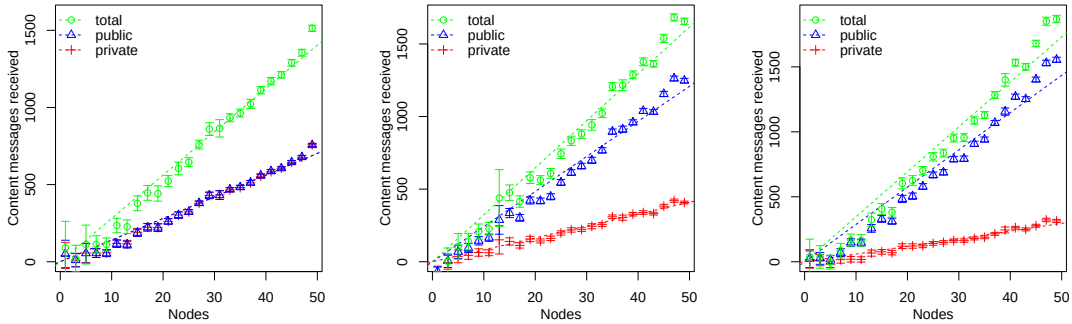
7.2.3.3 Private and Public Content Received in the Mobile System

Figures 7.14 and 7.15 show the private, public and total content received by a mobile node. The average contribution of the gateway is subtracted from the private and total content in each case, giving the approximate content received from mobile peers. A linear model was fitted to each dataset. The content received can be considered to scale linearly with the system size, above a minimum system size of ≈ 12 nodes. The public content received in the uniform case, depicted in Figure 7.14, can be seen to be about $n_{channels} - 1$ times the private content received. Very similar characteristics are seen in Figure 7.15 for the weighted solicitation runs. We can thus tentatively conclude that



(a) 2 content channels (b) 4 content channels (c) 6 content channels

Figure 7.14: Private and public content received on average by a mobile node using uniform solicitation.

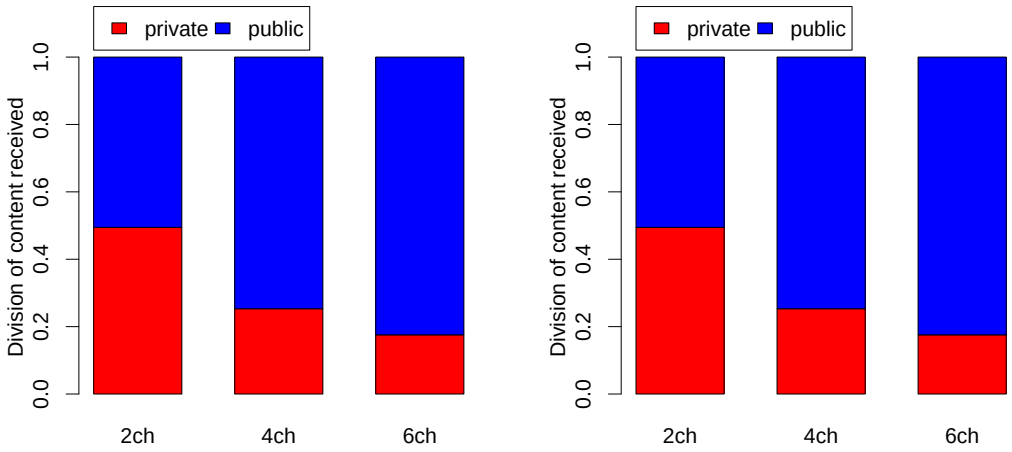


(a) 2 content channels (b) 4 content channels (c) 6 content channels

Figure 7.15: Private and public content received on average by a mobile node using weighted solicitation.

the performance of weighted and uniform solicitation are similar with regards to average content transfer.

Figure 7.16 shows comparisons of the ratio of private and public content received for both the uniform and weighted solicitation strategies. The results can be considered identical, supporting the previous conclusion of the equivalence of the uniform and weighted solicitation strategies. The most likely cause is that the aggregate content received over multiple contacts is such that on average a content ratio of $total \cdot \frac{1}{n_{channels}-1}$ is received on private channels, for the configuration used. The weighted solicitation strategy is, however, more likely to yield favourable private content in cases of few intermediary nodes, as shown in Section 7.1.



(a) Uniform solicitation (b) Weighted solicitation

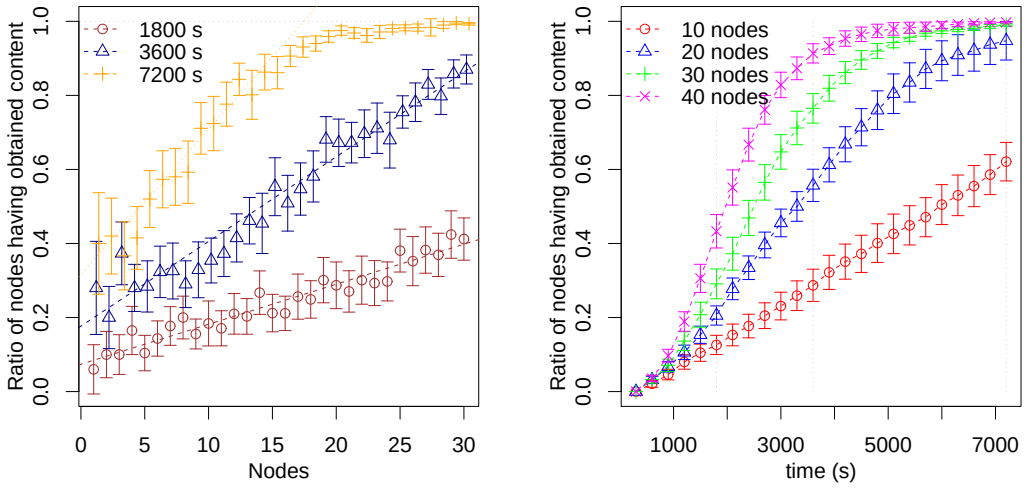
Figure 7.16: Ratio of private and public content received by a mobile node, using weighted solicitation. Measurements for a 50-node system shown.

7.2.4 Content Distribution Characteristics

The content distribution in a closed system can be analyzed using the distribution of small, atomic data units; a node either has the content or it does not. The following trials use a update channel type and a small data packet of 1000 bytes. 50 runs were done for each number of nodes in the system, each with a randomly chosen initial seed.

Figure 7.17(a) shows the reception ratio plotted versus the number of nodes in the system. Reception ratio is here the fraction of nodes that have obtained the content at the end of the run. Runs of 1800, 3600 and 7200 seconds were performed with varying number of nodes. The 1800 and 3600 second data sets can be approximated by a linear model as shown. The 7200 second dataset shows a knee where the reception ratio approaches 1. The linear model drawn was fitted to the data up to the visible start of the knee. The probability of alarm acquisition can thus be approximated as linearly increasing with the number of nodes in the system for a given period of time.

The content acquisition trend is further shown in Figure 7.17(b). A varying number of nodes were placed in the square and the content sampled at 300 second intervals. The reception ratio can be seen to follow an s-shaped curve, more pronounced for the higher node densities. The methodology of this series of runs was similar to the one previously discussed, except that 100 runs were done with different random seeds for each number of nodes and the time series averaged to give the final result.



(a) Reception ratio vs. number of nodes (b) Reception ratio vs. time

Figure 7.17: Reception ratio of a single piece of content. 1000x1000 m square.

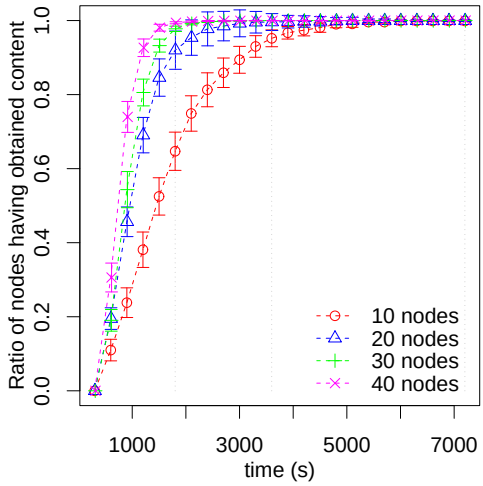


Figure 7.18: Reception ratio of a single piece of content. 500x500 m square.

We can conclude, not surprisingly, that the ratio of nodes with a particular content item increases with time after its introduction into the system. Node density is also seen to be critical in the the spread of the content. We observe that for a density of 40 nodes in a $1000 \times 1000\ m$ square, a ratio of about 80% distribution of that particular content item is achieved at about $3000\ s$. This may appear to be a rather long timescale for content spread. Consider, however, that the target deployment of a PodNet system would rather be a more confined area, like a network of streets, where a considerably higher node density would be expected. An example of content spread in a more densely populated area is shown in Figure 7.18, and demonstrates faster dissemination with the higher node density.

Chapter 8

PodNet Node Prototype

This chapter describes the implementation of a prototype PodNet node software, based on the design presented in Chapter 4.

The solicitation protocol and modeling of the content exchange in the wireless ad-hoc domain was addressed in Chapter 6. There are, however, several aspects of the design which are difficult to represent in a simulation model. The simulator implements a very minimal content cache to conserve memory and the subscription services are abstracted out of the implementation by allowing nodes to have infinite content available. The interactions of content consumers and the cache are also difficult to demonstrate adequately in the simulation. The prototype described in this chapter compliments the simulation model by focusing implementation of those parts. Together with the simulation model, it presents a proof-of-concept implementation of the design presented in Chapter 4.

This chapter is organized as follows: The architecture of the node prototype is introduced in Section 8.1. The content cache and subscriptions subsystems are then described in Sections 8.2 and 8.3. The management command set of the node is described in Section 8.4. Application layer demonstration consumers are discussed in Section 8.5. The chapter is then concluded in Section 8.6.

The full code for the prototype, along with a user manual, is available for download at http://code.google.com/p/podnode.

8.1 Node Prototype Architecture

The architecture of the node prototype is shown schematically in Figure 8.1. It is based on the design presented in Chapter 4. Note that the ad-hoc networking layer and solicitation protocol are not implemented here.

A single node design is envisioned, as discussed in Chapter 4. This prototype does thus not make a distinction between the roles of a gateway and a general roaming consumer. The role of the node can be partly shaped at implementation time, by choosing to include or exclude the subscriptions system and related parts of the content cache. This would result in a node unable to procure content directly from the Internet, which may be appropriate for a simple roaming node. The solution chosen for the meta-data storage can also be influenced by the intended role of the node. The final role of the node should be shaped at deployment and run times, by making the appropriate policy choices.

The main components of the node prototype are

Cache manager encapsulates the content cache and associated services. It utilizes a relational database for its meta-data storage and a normal directory in the file system of the host OS for its actual content store. Cache manager exposes a number of well-defined interfaces, which the other node sub-systems utilize. The cache design is described in Section 8.2.

Subscriptions manager handles subscription of contents from Internet sources. The content procured is added to the content cache. The subscriptions manager and associated modules are described in Section 8.3.

Management command set is a set of scripts for management of the cache and subscriptions systems. This is described in Section 8.4.

Application layer consumers are not strictly a part of the node implementation. Sample consumers, described in Section 8.5, were though implemented for demonstration purposes.

Ad-hoc network services represents the network and protocol layers for the ad-hoc domain. This module is excluded from the prototype.

The node prototype presented in this chapter is written in Python (http://www.python.org), which is a modern scripting language suitable for rapid prototyping. Its strong list and collections support, along with extensive third party libraries, makes it a very attractive option for this application. Although Python is typically classified as a scripting language, it is also a sophisticated and modern object-oriented language.

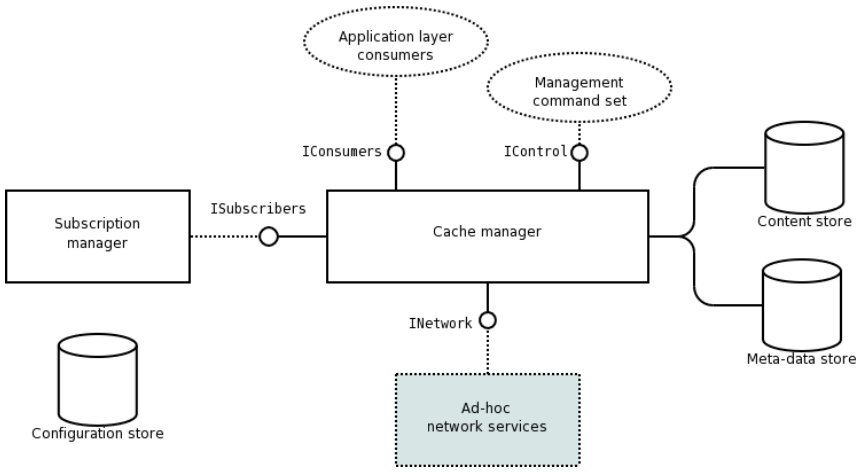


Figure 8.1: Gateway node subsystems

8.2 Content Cache

The content cache stores all content and associated meta-data which a node carries. The content is organized into public and private channels, as described in Chapters 3 and 4.

The content cache encapsulates all storage and management operations and provides well defined interfaces for use by subsystems when accessing its contents. The interfaces are shown in Figures 8.1 and 8.3

Appropriate meta-data storage options vary amongst potential platforms and node purposes. A fixed platform, like a gateway, can be assumed to have rather large capacity, and may thus benefit from implementing the content cache as a relational database. Mobile solutions would perhaps employ a light-weight solution, e.g. in form of a STL-based collection, as done in the simulator described in Chapter 6. The cache interfaces towards the other subsystems should however remain consistent, regardless of the storage solution used. The meta-data storage solution chosen for the prototype is a relational database, specifically MySQL (http://www.mysql.org) v 5.0. Other SQL solutions can, however, be employed with minor changes. An UML diagram of the table structure is shown in Figure 8.2. The actual content items are simply stored in a dedicated directory on a drive mounted on the hosting computer, linked to the content meta-data in a database field.

The configuration for channels and feed subscriptions is stored in the content cache database, along with the actual content. Channels configuration defines the channels which are exposed to peers in the ad-hoc domain. The subscriptions configuration is currently limited to rather simple syndication news feeds. The feeds table thus configures the links to the relevant feeds on the Internet and their update frequency.

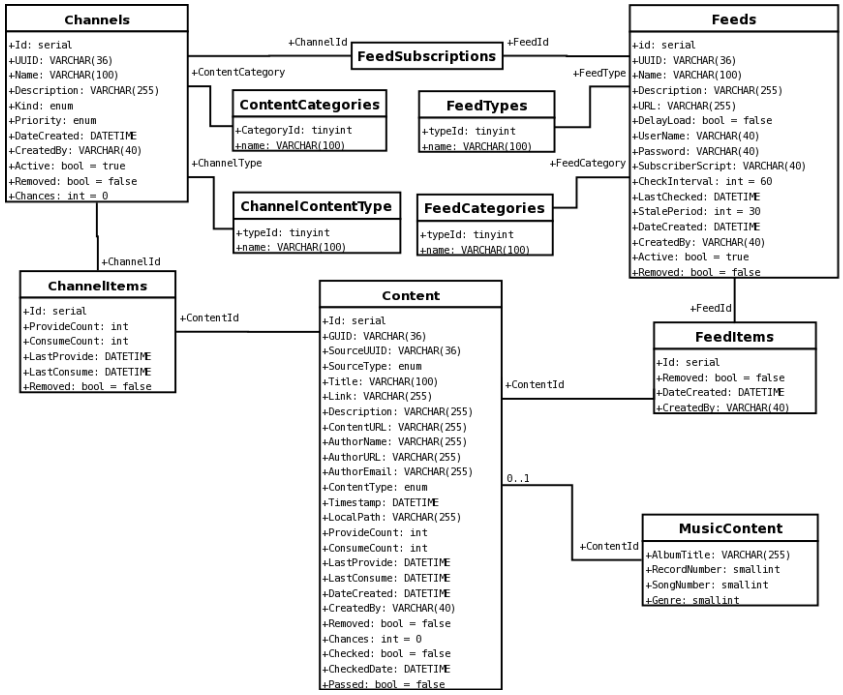


Figure 8.2: UML diagram of the relational database structure of the content cache

The types of content stored by the cache in this prototype are:

- Simple *RSS* and *Atom* syndication objects in the form of images or audio items.
- Blog entries entered locally on the host platform.
- Music content in the form of mp3 files entered locally on the host platform.

The id3 tags of locally entered music content in mp3 format is parsed using the *id3-py* parser library (http://id3-py.sourceforge.net).

The actual content meta-data is stored in the *Content* table. Its structure is based on the Atom syndication standard, although somewhat generalized and simplified. Each record represents a *content item*. Its form is flexible enough to accommodate a wide variety of content. More specialized meta-data storage solutions may however have to be added, as demonstrated by the *MusicContent* table. It links to a particular feed item, further specifying the entry. The fetching and preparation of meta-data reports from entries of this or similar forms must be handled internally by the cache manager and transparent to objects which use its services. Content can optionally be marked upon consumption, thus indicating to the cache management utility (see Section 8.2.3) that the content can safely be removed.

The *Channels* table stores the channels known by the device. Channels can be created on the device, thereby adding them to the channel pool of the population. The *ChannelItems*

table stores references to the content items added to the cache through peer-to-peer interactions on a particular channel. Content can also be added locally by user action, as described in Section 8.4.

The *Feeds* table stores definitions for feed subscriptions, as will be further discussed in Chapter 8.3. Each record stores definition on a particular source, which in this prototype are assumed to be RSS or Atom syndication feeds. References to items procured on a particular feed are stored in the *FeedItems* table. Feeds are mapped to one or more channels, through the *FeedSubscriptions* table. There is thus a one-to-many relationship between the Internet sources and the content channels of the cache. This provides considerable flexibility in the configuration of content channels.

8.2.1 Cache Library

The cache manager prototype consists of a set of Python classes. Abstract classes are defined for content, channels and feeds, which are designed to wrap the three main datastructures of the cache. Derived classes further define the database specific functionality, which in this case depends on the MySQL relational database. This is shown schematically in the simple UML diagram of Figure 8.3.

A setup text file in the startup directory is used to set all system parameters, such as implementation specific definitions. Module level class factories are called to create the proper object type, allowing scripts which utilize the library to be independent of the particular implementation in use. An example call, creating a *Feeds* object instance is shown below:

```
# Get a cache object from the library module. The module level class
# factory GetCacheObj returns a proper instantiation of the cache
# in accordance to globally defined system parameters.
cache = GetCacheObj();
# Get items from the cache as tuples of content
content = cache.Get(all=all);
```

The interfaces described in the next chapter provide an additional abstraction layer. Direct invocation of the factory objects is thus as a rule not necessary.

Note that an implementation of a simple roaming node which does not require feeds can omit the *Feeds* class implementation in its entirety, along with the feeds table in the database.

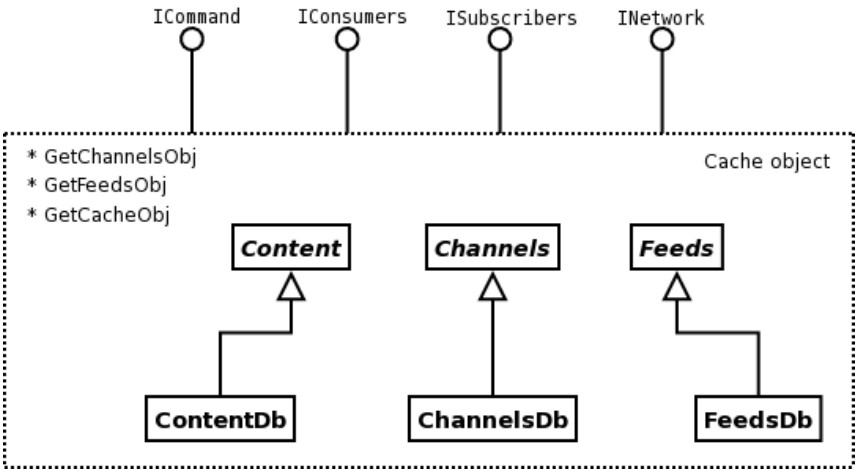


Figure 8.3: UML diagram of the Python classes of the content cache. Module level factory functions are indicated by *.

8.2.2 Cache Interfaces

The main cache objects, *Feeds*, *Channels* and *Content*, wrap the cache datastructures. Since Python does not include the notion of data hiding, the objects can be instantiated and used directly to access the cache, though this use would likely cause the programs that utilize the cache to be too dependent on its structure. Providing interfaces for access to the cache is certainly a better design choice. Thus, a number of well-defined interfaces, shown schematically in Figure 8.3, provide an abstraction layer between the cache objects and the modules of the system which utilize them:

IConsumers defines an interface towards the application level consumers, e.g. news-readers and music players, which consume contents from the cache. It provides methods to query the cache, retrieve contents and manage channel subscriptions.

ISubscribers defines an interface towards the subscriptions subsystem. It provides methods to add content to the cache and manage feeds. This interface can be omitted when implementing a simple roaming node which does not require feeds management.

IControl defines an interface utilized for control of the cache. This includes methods for managing channels and contents.

INetwork defines an interface towards the ad-hoc content solicitation protocol module. It includes methods for creating public channels on demand, as well as to query, solicit and add contents to the cache. This interface is though not implemented at the present time.

8.2.3 Cache Updater Utility

The cache size is a concern on a handheld device, although the size of commonly available memory and other storage is ever expanding. The cache is as a rule large on a fixed system, but can by no means be considered unlimited. Cache management strategies must therefore be considered, to keep the size of stored content within manageable limits. This is especially true of the public channels, which can be expected to be numerous in any PodNet population.

A cache management algorithm for use in a PodNet node is discussed in Section 4.2.2.2. This algorithm is implemented in the *cache_updater* management script. The script is run by an explicit user action, but automatic scheduling should be added at a later time. *cache_updater* takes the following arguments:

```
cache_updater
  --prlimit {Mbytes} --pulimit {Mbytes}
  [--remove {Kbytes}] [--chances {number}] [options]
```

prlimit is the maximum size of the private cache in M bytes.

pulimit is the maximum size of the public cache in M bytes.

remove is the amount of space (in K bytes) to free, once the limit is reached. Default is 10 K bytes.

chances are the number of updates after removal of a content item, which should be performed, before the associated meta-data is removed. Default is 3 chances

options specifies how the items should be selected:

lp marks the least popular items for removal. This is the default option.

lrs marks the least recently requested items for removal.

Items marked as consumed or removed on *private* channels are purged, once the private cache has reached the *prlimit*. Warnings are otherwise issued for private channels when the limit is exceeded. Items marked for removal on *public* channels are purged. In addition, a number of items are selected for removal, using either the least popular or least recently solicited criteria. The two level strategy of removing content items, while retaining meta-data for a number of updates, is employed.

8.3 Subscriptions System

The purpose of the subscriptions system is to pull content from the Internet. It consists of a subscription update scheduler, described in Section 8.3.1, which periodically triggers readers appropriate for the content source in question. An example is the RSS/Atom feed reader, described in Section 8.3.2.

8.3.1 Subscription Update Scheduler

Although complex scheduling involving adaptive prefetching based on usage statistics is conceivable, the prototype described here uses simple predefined scheduling of updates. Later enhancements using more advanced content retrieval scheduling can be based on the work presented here.

The subscriptions updater in the prototype is in the form of a Python script, *feed_updater*, described in Section 8.3.2. Each *Feeds* object defined in the content cache includes a definition for a updater to utilize and the update check interval. The *Feeds* table is queried, utilizing the *ISubscribers* interface, at startup and a thread is spawned for each active feed. The threads run autonomously for the duration of *feed_updater*s run, instantiating and executing the specified reader at the predetermined intervals. The instantiation parameters of the reader in question are stored in the thread datastructure when it is created and they are assumed to be invariant for the duration of the run.

Content added by the subscriptions subsystem is considered to be untrusted by default. It is marked as new and unverified in the meta-data store. A consistency check should be run on this content before it can be considered as trusted. This is the purpose of the *verify* script of the command set described in Section 8.4.

dock_updater is in essence a one-shot version of the *feed_updater*. The *dock_updater* is intended to be executed by specific user action during periods of Internet connectivity. It is thus more appropriate for a roaming node rather than a gateway. The predefined subscriber is instantiated and executed for each active feed, although each runs consecutively, rather than in its own thread of execution as in the *feed_updater*.

8.3.2 Atom/RSS Feed Reader

A number of Internet based-content sources are conceivable for a PodNet node. These include:

- Atom and RSS syndication feed subscriptions
- Podcast subscriptions
- HTTP and FTP downloads
- Multicast subscriptions

This prototype only considers Atom and RSS syndication feeds. Further, the types of sources supported are in this prototype limited to sites with simple content, i.e. feeds distributing content as a single file, rather than richer feeds requiring multiple objects to be combined into an HTML page. The test content for the prototype is a number of sites serving daily comics in the form of a single small image file.

A subscription reader for Atom/RSS feeds, *feed_reader_atom*, was created which parses a feed specified by a name. This is a Python script utilizing the *ISubsribers* interface (Section 8.2.2) to access the feed definition in the content cache, as well as to add content to the cache. The reader described uses the *Universal Feed Parser* (v 4.1) library for Python (http://feedparser.org). This library is one of the most advanced open-source feed reader libraries currently available and handles both RSS and Atom feeds of all versions transparently and in a robust manner.

The reader can be executed on demand from the command line, but it is normally used in conjunction with the *feed_updater* scheduler, described previously.

8.4 Management Command Set

The node is controlled by a command set comprised of a collection of Python scripts, listed in Table 8.1, and are intended to be executed by the user in a terminal window. All content and settings are contained in the content cache and the command scripts utilize its *ICommand* interface. Refer to the online documentation for further implementation details and command line parameters. Note that a simple roaming node with no needs to access the Internet directly, could potentially employ only a subset of the commands, since the feed related functionality could be excluded.

8.5 Content Consumers

Three Python scripts, *wiew*, *play* and *read*, were developed to demonstrate the interaction of the content cache with an application level consumer, utilizing the *IConsumer* interface.

Name	Description
mkfeed	Create a feed
mdfeed	Modify a feed
rmfeed	Mark a feed for deletion. The record is not removed until purge is called
lsfeed	List feed information
lscontent	List content
subscribe	Assign a channel to an existing feed
unsubscribe	Remove a channel subscription from a feed
mkchannel	Create a channel
mdchannel	Modify a channel
rmchannel	Mark a channel for deletion. The record is not removed until purge is called.
lschannel	List channels and channel contents
addcontent	Add content to a channel. This is the method used to manually add feed items locally at the node, e.g. blogs, traffic reports or own music collection
rmcontent	Remove content from the cache.
integrity	Cache integrity check
orphans	Check for orphaned content items, i.e. content items in the directory structure without corresponding meta-data entries.
verify	Verify new content in the cache. This script can be expanded to call a virus scanner or other content consistency checker. The current implementation simply sets all content as passed.
top	List channels and content items in order of popularity or ordered by most recent requests
purge	Purge items marked for deletion from cache

Table 8.1: Management command set

These are not strictly a part of the PodNet node design, but rather demonstrate the concept of consuming content from the cache. More thorough consumer design or an interface library for existing consumers is left as future work.

view displays a image file from the cache.

play plays a music file from the cache

read displays a text file, e.g. a blog, from the cache

All the scripts use the IConsumers interface, described in Section 8.2.2, to access the content cache. They all share a common command line parameters set of the form:

channel specifies a channel name or numeric identifier.

name specifies the title of the content.

bottom specifies the lower limit to be applied to a content timestamp.

top specifies the upper limit to be applied to a content timestamp.

mark remove specifies whether a content item should be marked for removal after consumption.

list all reports content items which fit the criteria, but does not consume.

All parameters are optional; omitting all simply consumes a random item of the appropriate type from the cache. Specifying a channel limits the search to one channel. Further specialization on maximum and minimum age can be employed, while the most restrictive search specifies a content item name. A content item can be marked for removal upon consume.

8.6 Summary

This chapter presented a working proof-of-concept prototype of a node in the PodNet system. This implementation fills some conceptual gaps in the simulation model, presented in Chapter 6, which did not address the subscriptions system design and only included a rudimentary content cache. The node prototype has been running on a Linux platform for several months and has proven successful in procuring content in the form of simple gif and jpeg encoded comics from syndication news feeds on the Internet. A set of content consumer scripts was developed for demonstrating their interactions with the cache. This implementation is an acceptable proof-of-concept for the cache design and subscribers of a general PodNet node. It further supports the feasibility of a single design for all roles of PodNet nodes. Future work on this prototype involves implementation of the solicitation protocol and wireless network services.

Chapter 9

Conclusions and Future Work

This dissertation is first and foremost a contribution to the ongoing PodNet project. It is a stepping stone in the path towards the comprehensive design envisaged, and will hopefully prove to be useful as such.

Contribution

A large fraction of the project effort was dedicated to the development of the light-weight solicitation protocol, expanding on the work presented in (Lenders et al., 2007). The protocol is designed to be highly efficient, reducing the overhead to the bare minimum, allowing nodes to potentially take full advantage of the brief and likely relatively infrequent contact intervals to exchange content. The basic protocol design does not assume any specific transport and can potentially be implemented to run at the link or transport layers.

A two-pronged approach was taken to the development of the proof-of-concept node prototype, where a Python prototype demonstrates the content cache and subscription service, and a simulation model implements the solicitation protocol. Together, these two models form a somewhat disjoint implementation of a PodNet node, which can be configured as a gateway or general roaming client.

The simulation model is based on the OMNeT++ framework and implements the content solicitation protocol, along with considerable data capture code for analysis purposes. The simulator is capable of running scenarios with a number of static gateways and dynamically created roaming nodes. OMNeT++ is a highly modular framework, allowing relatively straight-forward expansion of the simulation model for use in more complex

scenarios. The mobility models employed are the de facto standard RWP and a trace mobility module, using externally generated mobility scripts. The scripted mobility approach allows use of sophisticated mobility models without adding complexity or processing load to the simulation model. The modular nature of the simulation framework, however, allows other mobility models to be substituted for those used in this dissertation. The simulation model abstracts away from any specific radio or link layer technology by implementing direct message passing between nodes, using a simple fading model to determine potential recipients within range. This simplification was done for ease of development as well as simpler analysis of results. The simulator has to date been employed for basic protocol verification tests and for some initial investigations into the nature of PodNet systems.

The gateway node prototype was implementation in Python, demonstrating the content cache and subscription services. The ad-hoc protocol and network communications are not represented in this prototype. A relational database is used for meta-data storage, while the content itself is stored in a dedicated directory. RSS and Atom syndication feeds are supported in the demonstration subscribers. In addition, locally entered text and music files are supported. The meta-data datastructure is based on the Atom syndication standard. The gateway prototype has been running for several months, pulling content from selected RSS syndication feeds, providing simple content in the form of gif or jpeg images. The datastructure presented is thus sufficient for this kind of content, as well as locally entered blogs and music content. Other types of content and content sources have to be investigated and added at a later time.

Experience and Lessons Learned

The work progressed somewhat differently from the original ideas in many aspects. An implementation of the solicitation protocol was necessary for creation of the simulation model, which was thought to be a more practical proving ground than an actual hardware deployment. The protocol and associated simulation work was in fact more time consuming than originally thought and proved to be a considerable fraction of the total project effort.

The gateway concept and design also changed considerably during the course of the project. The original concept was one of a clear distinction between a fixed, large capacity gateway node and small lower capacity mobile peers. The distinction between the two types of nodes is however somewhat blurred in reality. Consider that a common

handheld device, like common mobile phones have a multitude of network interfaces, including 3G and Wi-Fi, as well as up to several gigabytes of memory. Such a device can easily function as a mobile gateway, a gateway in your pocket essentially, although its availability is most likely lower than that of a fixed, dedicated platform. Therefore, the design presented here is equally applicable to a mobile node and a gateway. A device with a highly available network connection simply adds the subscription services to become a functioning gateway.

Overall, the project proved to be a very valuable learning opportunity for the author and will hopefully be equally useful for future work on the PodNet project.

Future Work

Gateway Node Prototype Development

The gateway prototype presented can be considered to be a initial stepping stone for further development. Its purpose in this project was to provide a testing ground for the subscriptions system, content cache and cache management functions. Considerable further work is required before an actual prototype can be deployed.

The content solicitation protocol needs to be implemented and integrated into the prototype. A Python protocol module running over UDP/IP, based on the one present in the simulation model, is certainly feasible as a first step.

The prototype stores rather limited types of content. The meta-data structure may well have to be expanded to allow other types of content. Similarly, subscribers for other content sources than the Atom/RSS/Podcast type feeds have to be considered. Some feasible content sources are certainly multicast streams and FTP/web servers. More complex syndication feeds, like those provided by major news organizations, also need to be considered. For example, the feed presented by BBC online services contains a list of current news in the form of links to web pages. These would need to be downloaded for caching in a gateway, most likely needing a custom subscriber. A further issue with such complicated content items is size and presentation on small handheld devices. A conceivable method is to compress the entire page into a single file which is then fragmented and disseminated in the system. An equally valid method is to extract the body text and possibly images associated with it and compress, filtering out advertisements and links to other stories which those complicated items contain.

The present subscription manager is a simple periodic updater, much like common pod-catchers or news syndication clients. A more flexible adaptive strategy can however be employed, using usage statistics. A relatively unpopular channel can thus be updated on demand or relatively infrequently, thereby conserving bandwidth. A higher demand channel would however most likely benefit from more frequent updates. Simple channel popularity metrics can easily be employed for this purpose. Temporal knowledge of usage peaks can also be used to prefetch content beforehand, thus preventing cache misses; the intervals of contact between a mobile node and gateway are likely to be short so cache miss based fetching may not be acceptable.

Simulation Model Development

The simulation model has proven to be a useful tool for experimenting with the solicitation protocol and obtain insights into the behaviour of a content exchange system. There is however considerable work to be done.

The present work essentially ignores link layer issues. The next logical step in its development is to employ a real wireless linklayer model, e.g. the IEEE 802.11 models supplied with the INET framework for OMNeT++. Protocol and implementation issues due to an actual link layer technology are likely to arise in such work.

The content cache model of the simulator is very simple. A more complex model, like the one implemented in the gateway node prototype can conceivably be implemented, mostly for development and evaluation of cache management algorithms. Such a model will however add considerable load to the simulator, which may prove to be unacceptable for runs of a large system.

The protocol implemented is a somewhat simplified version of the one described in Chapter 5. A fuller implementation should ultimately be done and integrated in the simulator.

The original goal of this project was to develop code deployable both on a simulator and on an actual hardware platform. This goal was not fully realized due to time constraints. The datastructures and protocol module of the simulator are written in C++, using multi-platform libraries, and are thus fairly portable. There is however a lot of OMNeT++ specific code that will have to be removed for a deployable version. A parallel development of the simulator and a deployable version is suggested: OMNeT++ specific wrappers can be created around the deployable code, abstracting the simulation specific features away

from the main code, while at the same time maintaining a common code base for the simulator and deployable version.

The pairwise connection paradigm was not implemented in this version. It should definitely be implemented in future versions, since many link layer transports impose this constraint.

Analysis Work

Comparative analysis of the disorganized ad-hoc content exchange model employed in this simulator and the pairwise paradigm presented in Chapter 5 should be done. The pairwise approach may well be more practical, both for reasons of hardware restrictions and throughput characteristics. The more disorganized ad-hoc mode was used in the work presented in this dissertation, mainly for simpler implementation.

Fuller comparison studies of the uniform and weighted solicitation strategies should definitely be performed. The weighted solicitation may well prove to be equivalent to the uniform case, as noted in Chapter 7, for considerable number of contacts.

Inter-Gateway Network and Protocol

An inter-gateway protocol was initially proposed as part of this project. It was however not implemented due to time constraints in this project. Interconnecting gateway nodes on the wired side into an overlay network is a worthy continuation of this project. A likely implementation is a self-organizing peer-to-peer network, perhaps using a number of well-known servers to provide lookup services for fellow gateways. Another approach is to obtain IP addresses of gateways from mobile peers which come into the gateway range, contacting those gateways and obtaining their known peers. Such a network would grow in an organic manner, maintaining known contacts with periodic hellos.

An applications for the inter-gateway network is for example a distributed cache or content delivery network could be implemented if gateways are enabled to exchange content descriptions. There may well be benefits in obtaining content from a gateway peer in close proximity, rather than fetching it from the original source. Proximity information is not unreasonable to obtain if the gateways are fixed computing platforms. Alternatively, a hop count and/or RTT metrics could be maintained by each node for its peers.

Another application for an inter-gateway network is sharing of usage statistics. Such information can potentially be utilized by a gateway to dynamically manage its update

schedule or prefetch content to anticipate usage peaks. Lets consider a scenario: A distribution node is located on a train, while inter-connected gateways are located at each platform. A group of cultivated persons enters the train, wishing to listen to the unpopular classical channel. The local gateway is only likely to be able to deliver a small amount of content to this group. Sharing this information with the upstream gateway however enables it to prefetch a considerable amount of content for the group, ready for transfer when the train enters the next station.

Multi-Channel Issues

The present work assumes a simple broadcast radio channel which is shared by all nodes. This model is similar to the one used by IEEE 802.11 devices, but other link layer technologies may use more sophisticated models. A possible scenario is sharing of a common narrow band control channel, while dividing the remaining spectrum among several data channels. Adaption of the model presented in this dissertation to such technology essentially involves implementing the pairwise model, adding a channel negotiation phase at the beginning.

Protocol Verification

Formal verification of the protocol presented in this dissertation is an ongoing side project, in collaboration with the *Icelandic Centre of Excellence in Theoretical Computer Science (ICE-TCS)* (http://www.icetcs.ru.is). The protocol can easily be proven to be free of deadlocks, since it employs soft state of peers. The project however aims to maximize the efficiency of the protocol, at least under given operational conditions, by employing formal verification methods and tools to study the effects and interactions of various timeout/retry strategies. A tool for this kind of work is not currently known to exist, but a possible basis is the tool *UppAal* (http://www.uppaal.com) (Larsen, Pettersson, & Yi, 1997).

Multiple Radio Channels and Port Abstractions

This dissertation work assumed a simple model where a single common radio channel was used for all nodes in the system. There are some benefits, e.g. being able to utilize broadcast of content and promiscuously overheard control messages. There are however considerable drawbacks in the form of congestion of the channel in the presence of two or more communicating pairs within transmit range of each others.

Multiple radio channels can easily be utilized in the system by employing a common low bandwidth discovery and control channel with a number of high bandwidth data channels. The pairwise paradigm of section 5.3 can then be extended to allow multiple simultaneous pairwise connections by a single node, each using a dedicated radio channel. Nodes would send their control messages on a single common control channel, enabling negotiation of data channel frequency. Promiscuous utilization of overheard control traffic could also be enabled. It is however unlikely that broadcast of content would be as effective in this setting as in the common channel one. Further work on this aspect is however reserved for future work.

Transport layer multiplexing of TCP or UDP traffic can be considered a higher level incarnation of the same principles. A number of simultaneous pairwise connections can be hosted by a single node, given that each partner uses a dynamically allocated pair of ports. A well-known discovery port and a corresponding daemon service could be used to enable nodes to discover each other and to negotiate the communications ports.

Protocol Optimization

The protocol is of primary importance for the performance of the system as a whole. The most important performance metric is the actual content throughput, excluding all control message traffic and header overhead. All protocol overhead wastes the content transfer potential and should thus be avoided.

Control message traffic is minimal in the protocol and thus difficult to reduce. Their size can be reduced by optimization of the header and reduction or elimination of the compact content descriptor. The latter measure is however of dubious benefit since it may as well cause inefficiencies due to an increase in unsatisfiable requests. The options bits in the type A header (see Appendix A) could however be utilized to indicate removal of either of the attachment length indicators if unused, thus removing a number of bytes from the header.

Requests and announces following a content provision turn can also be removed, if some of the control bits in the header were dedicated to this purpose. A control bit set in the content message would thus indicate what action the sender would like the receiver to take, in fact it would function identically to the announces or requests following a sending turn as discussed above. Utilizing the ability of a request message to carry multiple content specifiers also helps to conserve bandwidth, as sufficient material for an entire sending turn may potentially be specified by just a single request. Following turntaking

requests might thus be empty, or the previously mentioned approach of using control bits in the content messages to signal turntaking might be used.

The most effective optimization of a link layer frame is to eliminate the network and transport layer overhead and pack the protocol messages directly into it. A link layer implementation is thus the most efficient solution for the protocol, although technically more challenging than a transport layer solution.

Content messages are likely to take up most of the bandwidth utilized by the protocol. They are both numerous and large in size. It is important to decrease the size of the content message overhead if at all possible.

- A message can contain meta-data of considerable size. Sending meta-data once is risky unless a reliable transport is utilized; the single message carrying the meta-data may well be lost, thereby loosing the information. A simple optimization is to include meta-data with every n-th content piece, thereby increasing the usable space for content in other messages. Choosing n carefully should ensure a significant probability of receiving the meta-data for a content item, without resorting to any retries. This would be suitable for content which is resistant to dropouts; the loss of a fraction of the pieces making up the content item could be accepted. Another possibility is to transmit meta-data separately from the content itself, which would entail a partially "reliable" protocol. Meta-data and other essential traffic would then be transmitted by reliable means, using acks, timeouts and retries, much like is done in TCP.
- The channel and content identifiers in the type B header are rather long, 128 bit GUIDs. Reducing these would be preferable. No reliable method has been found so far, although using a smaller locally unique identifier could be acceptable in some cases. A random number of say 32-bits may well be sufficiently unique in a localized setting to allow reconciliation of channel/content items for a single transfer. The full GUID of the content would however have to be carried in the meta-data associated with the item for future transfers.

Promiscuous utilization of overheard content messages may well enhance performance and improve bandwidth utilization as discussed in Section 5.4. This measure is however of limited utility in the case of multiple radio channels. There is however likely a single common radio channel for control messages, even in a multi-channel scenario. Utilizing overheard channel updates can for example allow a node to build an image of potential peers without having to issue any requests. Promiscuously utilizing control messages intended for other nodes may help a node to deduce its state in relation to other nodes in

the vicinity and thereby optimizing message traffic. A pending request can for example be cancelled if a node receives overheard content, as shown in Figure 5.12, thereby conserving bandwidth wasted by continually re-issuing the request.

The PodNet systems efficiency is obviously greater as the number of participants grows. Power is the main limitation of current handheld devices, which in essence encourages selfish behaviour by turning the device off, or disabling wireless interfaces to conserve power, having received content of interest. The peer-to-peer content exchange is thus bound to suffer. Energy efficiency of the content exchange protocol and peer discovery must thus be considered. Work on energy efficient peer discovery is e.g. presented by Wang et al. (2007).

Chapter 10

References

Aiyer, A. S., Alvisi, L., Clement, A., Dahlin, M., Martin, J.-P., & Porth, C. (2005, October). BAR fault tolerance for cooperative services. In *SOSP'05 20th ACM Symposium on Operating Systems Principles* (p. 45-58). Brighton, United Kingdom.

Ari, I., Hong, B., Miller, E. L., Brandt, S. A., & Long, D. D. E. (2003, October). Managing flash crowds on the internet. In *11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 03)* (pp. 246–249). Orlando, FL.

Ari, I., Hong, B., Miller, E. L., Brandt, S. A., & Long, D. D. E. (2004). *Modeling, analysis and simulation of flash crowds on the internet* (Tech. Rep. No. UCSC-CRL-03-15). UCSC.

Bai, F., & Helmy, A. (2005). *Impact of mobility on mobility assisted information diffusion (MAID) protocols* (Tech. Rep.). Dept. Electrical Eng., University of Southern California.

Berners-Lee, T., Fielding, R., & Masinter, L. (1998). *RFC2396: Uniform Resource Identifiers (URI): Generic Syntax.* United States: RFC Editor.

Bettstetter, C., Resta, G., & Santi, P. (2003, July-Sept). The node distribution of the random waypoint mobility model for wireless ad hoc networks. *IEEE Transactions on Mobile Computing*, *2*(3), 257-269.

Bianchi, G., Tinnirello, I., & Scalia, L. (2003, October). Handover across heterogeneous wireless systems: A platform-independent control logic design. In *Proc. IEEE WPMC.*

Bittorrent. (n.d.). [online] http://www.bittorrent.org/.

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, *13*(7), 422-426.

Blue Project.* (n.d.). [online] http://www.csg.ethz.ch/research/projects/Blue_star.

Bohacek, S., Sridhara, V., Singh, G., & Ilic, A. (2004). *The UDel Models - MANET Mobility and Path Loss in an Urban/Suburban Environment* (Tech. Rep.). University of Delaware.

Broder, A., & Mitzenmacher, M. (2003). Network applications of bloom filters: A survey. *Internet Math*, *1*(4), 485-509.

Burgess, J., Gallagher, B., Jensen, D., & Levine, B. N. (2006, April). Maxprop: Routing for vehicle-based disruption-tolerant networks. In *Proc. ieee infocom.*

Burleigh, S., Hooke, A., Torgerson, L., Fall, K., Cerf, V., Durst, B., et al. (2003, June). Delay-tolerant networking: an approach to interplanetary internet. *Communications Magazine, IEEE*, *41*(6), 128-136.

Buttyan, L., Dora, L., Felegyhazi, M., & Vajda, I. (2006). Barter-based cooperation in delay-tolerant personal wireless networks. In *In chants. 06: Proceedings of the 2006 sigcomm workshop on challenged networks* (p. 197-204). New. York, NY, USA.

Chaintreau, A., Hui, P., Crowcroft, J., Diot, C., Gass, R., & Scott, J. (2006, April). Impact of human mobility on the design of opportunistic forwarding algorithms. In *Infocom 2006. 25th ieee international conference on computer communications. proceedings* (p. 1-13). Barcelona, Spain.

Cheverst, K., Davies, N., Mitchell, K., Friday, A., & Efstratiou, C. (2000). Developing a context-aware electronic tourist guide: some issues and experiences. In *Chi '00: Proceedings of the sigchi conference on human factors in computing systems* (pp. 17–24). New York, NY, USA: ACM Press.

Chinchilla, F., Lindsey, M., & Papadopouli, M. (2004). Analysis of wireless information locality and association patterns in a campus. *IEEE INFOCOM*, *2*.

Crawdad. (n.d.). *Crawdad project.* [online] http://crawdad.cs.dartmouth.edu.

Das, S., Nandan, A., Pau, G., Sanadidi, M., & Gerla, M. (2004). *Spawn: A swarming protocol for vehicular ad-hoc wireless networks.*

DataMan. (n.d.). *Dataman.* [online] http://www.cs.rutgers.edu/dataman/.

Delay tolerant network research group. (n.d.). [online] http://www.dtnrg.org.

Drytkiewicz, W., Sroka, S., Handziski, V., Köpke, A., & Karl, H. (2003, Jan). *A mobility framework for omnet++.* 3rd International OMNeT++ Workshop. Department of Telecommunications, Budapest, Hungary.

Fall, K. (2003). A delay-tolerant network architecture for challenged internets. In *Sigcomm '03: Proceedings of the 2003 conference on applications, technologies, architectures, and protocols for computer communications* (pp. 27–34). New York, NY, USA: ACM Press.

Fawal, A. E., Boudec, J.-Y. L., & Salamatian, K. (2006). *Self-limiting epidemic forwarding* (Tech. Rep. No. LCA-REPORT-2006-126). EPFL, I&C.

Feedparser Library. (n.d.). [online] http://feedparser.org.

Felber, P., & Biersack, E. (2004). Self-scaling networks for content distribution. In *Proceedings of the international workshop on self-* properties in complex information systems.*

Frenkiel, R., Badrinath, B., Borres, J., & Yates, R. (2000, april). The infostations challenge: balancing cost and ubiquity in delivering wireless data. *IEEE Personal Communications Magazine*, *7*(2), 66-71.

Gerber, A., Houle, J., Nguyen, H., Roughan, M., & Sen, S. (2003). P2P, the gorilla in the cable. In *National Cable & Telecommunications Association (NCTA).*

Gerla, M., Tang, K., & Bagrodia, R. (1999, February). Tcp performance in wireless multi-hop networks. In *Proceedings of IEEE WMCSA'99.* New Orleans, LA.

Goodman, D., Borras, J., Mandayam, N., & Yates, R. (1997, May). INFOSTATIONS: a new system model for data and messaging services. In *IEEE 47th Vehicular Technology Conference* (Vol. 2, p. 969-973). Phoenix, AZ, USA.

Grossglauser, M., & Tse, D. (2002, August). Mobility increases the capacity of ad hoc wireless networks. *IEEE/ACM Transactions on Networking*, *10*(4), 477-486.

Haartsen, J., Naghshineh, M., Inouye, J., Joeressen, O. J., & Allen, W. (1998). Bluetooth: Vision, goals, and architecture. *Mobile Computing and Communications Review*, *2*(4), 38-45.

Haggle project. (n.d.). *Haggle project.* [online] http://www.haggleproject.org.

Helgason, O. R., & Jónsson, K. V. (2008). Opportunistic networking in OMNeT++. In *First international conference on simulation tools and techniques for communications, networks and systems (SIMUTOOLS 2008), OMNeT++ workshop (submitted).*

Helgason, O. R., & Karlsson, G. (2008, January 23-25). On the effect of cooperation in wireless content distribution. In *The fifth annual conference on wireless on demand network systems and services (WONS).* Garmisch-Partenkirchen, Germany.

Holland, G., & Vaidya, N. (2002, March). Analysis of TCP performance over mobile ad hoc networks. *Wireless Networks*, *8*(2-3), 275-288.

IEEE. (2003). *ANSI/IEEE Std 802.11, 1999 Edition (R2003) - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.*

Izal, M., Urvoy-Keller, G., Biersack, E., Felber, P., Hamra, A., & Garces-Erice, L. (2004). *Dissecting BitTorrent: Five months in a torrent's lifetime.*

Johnson, D., Maltz, D., & Broch, J. (2001). DSR: The dynamic source routing protocol for multihop wireless ad hoc networks. In C. Perkins (Ed.), (pp. 139–172). Addison-Wesley.

Juang, P., Oki, H., Wang, Y., Martonosi, M., Peh, L. S., & Rubenstein, D. (2002). Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. *SIGPLAN Not.*, *37*(10), 96–107.

Jung, S., Lee, U., Chang, A., Cho, D.-K., & Gerla, M. (2007). Bluetorrent: Cooperative content sharing for bluetooth users. In *Percom '07: Proceedings of the fifth ieee international conference on pervasive computing and communications* (pp. 47–56). Washington, DC, USA: IEEE Computer Society.

Karlsson, G., Lenders, V., & May, M. (2007, April). Delay-tolerant broadcasting. *IEEE Transactions on Broadcasting - Special Issue on Mobile Multimedia Broadcasting*, *53 no 1*, 369-381.

Kent, C. A., & Mogul, J. C. (1995). Fragmentation considered harmful. *ACM SIGCOMM Computer Communication Review*, *25*(1), 75-87.

Kim, J., Sridhara, V., & Bohacek, S. (n.d.). *Realistic simulation of urban mesh networks - part i: Urban mobility* (Tech. Rep.). University of Delaware.

Kurose, J. F., & Ross, K. W. (2005). *Computer networking. a top-down approach featuring the internet* (Third edition ed.). Pearson Education, Inc / Addison-Wesley.

Larsen, K. G., Pettersson, P., & Yi, W. (1997). UppAal in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT). Special section on timed and hybrid systems*, *1*(1-2), 134-152.

Leach, P., Mealling, M., & Salz, R. (2005, July). *A universally unique identifier (UUID) URN namespace.*

Lenders, V., Karlsson, G., & May, M. (2007, June). Wireless ad hoc podcasting. In *Proceedings of SECON 2007, IEEE.*

libxml2 XML Parsing Library. (n.d.). [online] http://xmlsoft.org/index.html.

May, M., Karlsson, G., Helgason, O., & Lenders, V. (2007, October). A system architecture for delay-tolerant content distribution. In *WreCom 2007.*

May, M., Wacha, C., Lenders, V., & Karlsson, G. (2007, September 14). Wireless opportunistic podcasting: Implementation and design tradeoffs. In *Chants 07.*

Mobility Framework. (n.d.). [online] http://mobility-fw.sourceforge.net.

Motani, M., Srinivasan, V., & Nuggehalli, P. S. (2005). Peoplenet: engineering a wireless virtual social network. In *Mobicom '05: Proceedings of the 11th annual international conference on mobile computing and networking* (pp. 243–257). New York, NY, USA: ACM Press.

MySQL. (n.d.). [online] http://www.mysql.com/.

Nandan, A., Das, S., Pau, G., Gerla, M., & Sanadidi, M. (2004, October). *Co-operative downloading in vehicular ad-hoc wireless networks* (Tech. Rep. No. 040035). UCLA CS.

Nandan, A., Tewari, S., Das, S., Gerla, M., & Kleinrock, L. (2006, Jan). AdTorrent: Delivering location cognizant advertisements to car networks. In *WONS 2006 : Third Annual Conference on Wireless On-demand Network Systems and Services* (p. 203-212). Les Ménuires, France.

Natarajan, A., Motani, M., & Srinivasan, V. (2007, April). Understanding urban interactions from bluetooth phone contact traces. In *8th passive and active measurement conference, pam* (p. pages 115-124). Louvain-la-neuve, Belgium.

Navidi, W., & Camp, T. (2004, Jan). Stationary distributions for the random waypoint mobility model. *IEEE Transactions on Mobile Computing*, *3*(1), 99-108.

Nielson, S. J., Crosby, S. A., & Wallach, D. S. (2005, February). A taxonomy of rational attacks. In *4th. IPTPS.*

Nottingham, M., & Sayre, R. (2005). *RFC 4287: The Atom Syndication Format.*

OMNeT++ Discrete Event Simulator. (n.d.). [online] http://www.omnetpp.org.

Ott, J., & Kutscher, D. (2004, June). Why seamless? towards exploiting wlan-based intermittent connectivity on the road. In *Proceedings of the TERENA Networking Conference, TNC.*

Ott, J., & Kutscher, D. (2005). A disconnection-tolerant transport for drive-thru internet environments. *IEEE INFOCOM*, *Vol 3.*

Panagakis, A. V. A., & Stavrakakis, I. (2007, January 7-12). On the effects of cooperation in dtns. In *In proc. of the second ieee/createnet/icst international conference on communication system software and middleware (comsware).*

Papadopouli, M., & Schulzrinne, H. (2000). Seven degrees of separation in mobile ad hoc networks. In *Global telecommunications conference, 2000. globecom '00. ieee* (p. 1707-1711 vol.3). San Francisco, CA, USA.

Papadopouli, M., & Schulzrinne, H. (2001a). Design and implementation of a peer-to-peer data dissemination and prefetching tool for mobile users. In *Proceedings of the first ny metro area networking workshop.*

Papadopouli, M., & Schulzrinne, H. (2001b). Effects of power conservation, wireless coverage and cooperation on data dissemination among mobile devices. In *Mobihoc '01: Proceedings of the 2nd acm international symposium on mobile ad hoc networking & computing* (pp. 117–127). New York, NY, USA: ACM Press.

Pasick, A. (2002, November 4). *LIVEWIRE - file-sharing network thrives beneath the radar.* [online] http://in.tech.yahoo.com/041103/137/2ho4i.html.

Pietilainen, A.-K., & Diot, C. (2007, December). Experimenting with Real-life Opportunistic Communications using Windows Mobile Devices. In *Proc. conext student workshop.*

PodNet Project. (n.d.). *PodNet Project.* [online] http://podnet.ee.ethz.ch.

Python. (n.d.). [online] http://www.python.org/.

Royer, E., & Toh, C. (1999, april). A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications*, 46-55.

Savage, S., Cardwell, N., Wetherall, D., & Anderson, T. (1999, October). TCP congestion control with a misbehaving receiver. *ACM SIGCOMM Computer Communication Review*, *29*(5), 71-78.

Scott, J., Hui, P., Crowcroft, J., & Diot, C. (2006). Haggle: A networking architecture designed around mobile users. In *Third IFIP wireless on demand network systems conf.*

Shah, R., Roy, S., Jain, S., & Brunette, W. (2003). Data mules: modeling a three-tier architecture for sparse sensor networks. In *Proceedings of the first ieee. 2003 ieee international workshop on sensor network protocols and applications, 2003.* (p. 30-41).

Sollazzo, G., Musolesi, M., & Mascolo, C. (2007, June). Taco-dtn: A time-aware content-based dissemination system for delay tolerant networks. In *Proceedings of the first international workshop on mobile opportunistic networking (mobiopp).* Puerto Rico.

Sridhara, V., & Bohacek, S. (n.d.). *Realistic propagation simulation of urban mesh networks* (Tech. Rep.). University of Delaware.

Stallings, W. (2005). *Wireless communications & networks* (2 ed.). Pearson Prentice Hall.

Stevens, W. R. (1994). *TCP/IP Illustrated, Volume 1, The Protocols*. Addison-Wesley.

Stutzbach, D., Zappala, D., & Rejaie, R. (2004). *Swarming: Scalable content delivery for the masses* (Tech. Rep. No. CIS-TR-2004-1). UNIVERSITY OF OREGON,.

Stutzbach, D., Zappala, D., & Rejaie, R. (2005). The scalability of swarming peer-to-peer content delivery. *Lecture Notes in Computer Science*, *3462/2005*, 15-26.

UDel Models. (n.d.). [online] http://udelmodels.eecis.udel.edu.

UppAal. (n.d.). [online] http://www.uppaal.com.

Vahdat, A., & Becker, D. (2000, April). *Epidemic routing for partially-connected ad hoc networks* (Tech. Rep. No. CS-200006). Duke University.

Varga, A. (2001, June). The OMNeT++ discrete event simulation system. In *European Simulation Multiconference (ESM'2001).*

Wacha, C. (2007). *Wireless ad-hoc podcasting with handhelds*. Unpublished master's thesis, Swiss Federal Institute of Technology Zurich (ETH).

Wang, W., Srinivasan, V., & Motani, M. (2007). Adaptive contact probing mechanisms for delay tolerant applications. In *Proceedings of the 13th annual ACM international conference on Mobile computing and networking, MobiCom '07* (p. 230-241). Montreal, Quebec, Canada: ACM Press.

Wiseman, R. (2007). *The pace of life.* [online] www.britishcouncil.org/paceoflife.pdf.

Ye, T., Jacobsen, H.-A., & Katz, R. (1998). Mobile awareness in a wide area wireless network of info-stations. In *Mobicom '98: Proceedings of the 4th annual acm/ieee international conference on mobile computing and networking* (pp. 109–120). New York, NY, USA: ACM Press.

Yeo, J., Kotz, D., & Henderson, T. (2006). CRAWDAD: a community resource for archiving wireless data at Dartmouth. *SIGCOMM Comput. Commun. Rev.*, *36*(2), 21–22.

Yoon, J., Liu, M., & Noble, B. (2003). Random waypoint considered harmful. In *Proceedings of INFOCOM. IEEE.*

Yuen, W., & Yates, R. (2003, Dec). Optimum transmit range and capacity of mobile infostation networks. In *IEEE Global Telecommunications Conference, 2003. GLOBECOM '03.* (Vol. 2, p. 1130- 1135).

Yuen, W., Yates, R., & Mau, S.-C. (2003, March). Exploiting data diversity and multiuser diversity in noncooperative mobile infostation networks. In *IEEE INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies.* (Vol. 3, p. 2218- 2228).

Yuen, W. H., Yates, R., & Mau, S.-C. (2003, March). Noncooperative content distribution in mobile infostation networks. In *IEEE Wireless Communications and Networking, 2003. WCNC 2003.* (Vol. 2, p. 1344- 1349).

Yuen, W. H., Yates, R. D., & Sung, C. W. (2003). Effect of node mobility on highway mobile infostation networks. In *MSWIM '03: Proceedings of the 6th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems* (pp. 82–91). New York, NY, USA: ACM Press.

Zhao, W., Ammar, M., & Zegura, E. (2004). A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *MobiHoc '04: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing* (pp. 187–198). New York, NY, USA: ACM Press.

Appendix A

Solicitation Protocol Messages

A.1 Protocol Header Format

This section describes the format of the content solicitation protocol messages. Figure A.1 shows the solicitation protocol message frame in relation to the transport frames, here a IEEE 802 MAC frame (Stallings, 2005, pp. 424-425). A transport layer protocol engine, utilizing UDP/IP, is assumed in this example, although an equally valid approach is to short-circuit the transport and network layers and utilize the link layer transport directly.

Two general message types are used in the protocol, each with a number of subtypes. The format of the messages is shown in Figure A.2. A further description of the fields is given in Tables A.1 and A.2. Two general header types, A and B, are defined. Type A header is used for all control messages, while A and B headers are required for content messages.

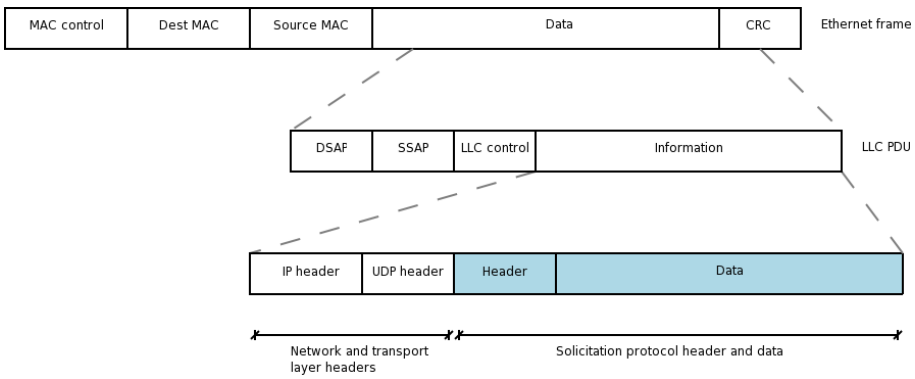


Figure A.1: Solicitation protocol message relative to a link frame.

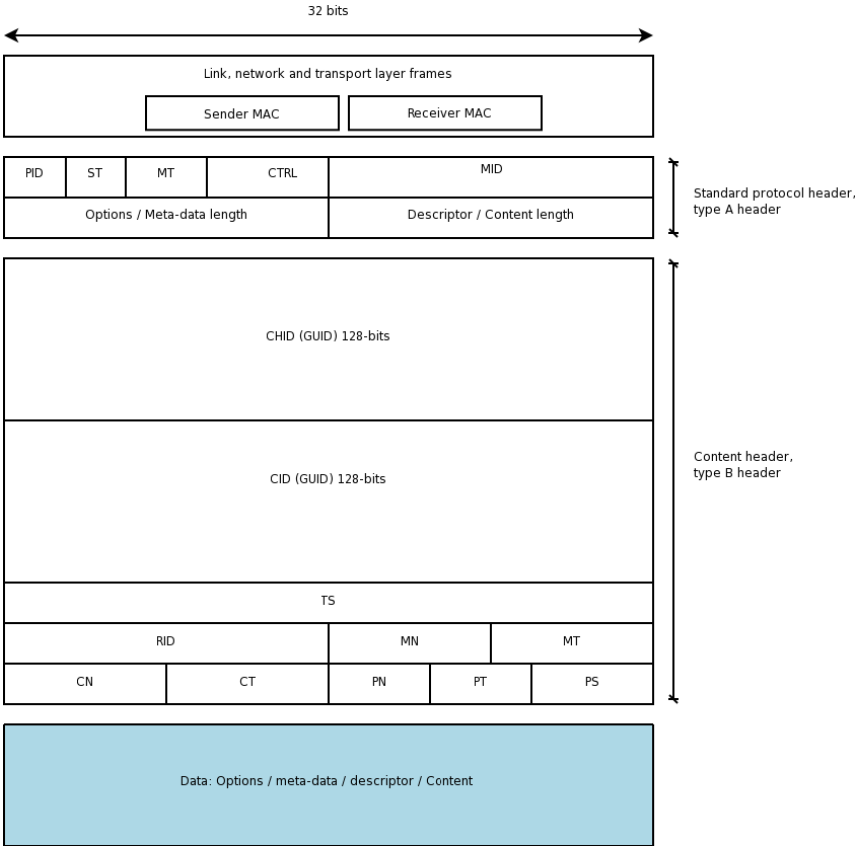


Figure A.2: Solicitation protocol message format. The relevant fields from the encapsulating layers are shown, others are omitted.

Abbreviation	Name	Size (bits)
PID	Protocol version id	3
ST	Sender Type	3
MT	Message Type	4
CTRL	Control	6
MID	Message ID	16
OPTLEN	Options length	16
MDLEN	Meta-data length	16
DESCRLEN	Descriptor length	16
DATALEN	Content chunk length	16

Table A.1: Type A header fields. Standard message header. Note that the fields describing the size of attached data are at most 32 bits in length since options/meta-data and descriptor/content occupy the same location in the header.

A.1.1 Type A Header

PID Protocol version identifier. Zero by default, indicating the version specified in this work. Later versions of the protocol should be identified by version IDs to allow backwards compatibility of interpretation of messages and protocol rules.

Abbreviation	Name	Size (bits)
CHID	Channel ID (GUID)	128
CID	Content ID (GUID)	128
TS	Timestamp	32
RID	Request ID	16
MN	Message number in turn	16
MT	Total messages in turn	16
CN	Chunk number within content item	16
CT	Total chunks in the content item	16
PN	Piece number within chunk	5
PT	Total number of pieces within the chunk	5
PS	Piece size in bytes	6

Table A.2: Type B header fields. Content message header. These fields are required in addition to the type A header to uniquely identify a chunk of content.

ST Sender type. Numeric identifier of originator node type. Used as diagnostics only in the current version of the protocol, but later optimizations may require differentiation of node roles.

0 Unknown (default)

1 Roaming node

2 Gateway

3 Caching node

MT Message type. Numeric identifier of message type. **Required**.

0 Unknown (error condition)

1 Channel request

2 Channel response

5 Content request

6 Content message

7 Content advisory message

8 Content rejection message

11 Pairing request message

12 Pairing response message

15 Announce

CTRL Control bits. Currently not used, but reserved for future optimization purposes in later versions of the protocol.

MID Message ID. 16-bit identifier of messages. Serially numbered by originator. The message id of a request is echoed back in content messages associated with it as RID (request id), enabling the requester to identify the request which triggered the content transfer.

OPTLEN/MDLEN Options or meta-data length in bytes. Options are filters and additional specifications sent with request messages, while meta-data length is specific to content messages. The utilization is implied by the MT field.

DESCRLEN/DATALEN Length of content descriptor or content in bytes. The utilization is implied by the MT field.

A.1.2 Type B Header

CHID Channel ID. 128 bit GUID, uniquely identifying the channel. Assigned by the channel creator.

CID Content ID. 128 bit GUID, uniquely identifying the content. Assigned by the content provided. Feed contents downloaded from the Internet are assigned a GUID by the retriever node, which can either be a gateway with an Internet connection or a mobile node with a highly available connection like 3G.

TS Timestamp. A UNIX time_t format is expected.

RID Request id. The request id is included in the type B content header to allow a receiver to identify the request that triggered the transfer. The message id (MID) of a request is echoed back in content messages as RID.

MN Message number in turn. Used to monitor lost content messages, but otherwise not utilized in the protocol.

MT Total number of messages in the turn. See MN. Not utilized in the current version of the protocol.

CN Chunk number, serially numbered from zero, for the content item identified by CID. CN is required to enable a receiver to reassemble a received content item from a number of chunks.

CT Total number of chunks in the content item. This is required for proper reassembly of the content.

Message	Description
Announce	Announce a nodes presence and capabilities to a peer node
Pairing request	Requests a pairing with a discovered peer node
Pairing response	Response to a pairing request
Content request	Request content from a peer node
Content advisory	Advises a requesting node that all or part of a request cannot be fulfilled
Content	Encapsulates a single piece of content or part of a channels/content listing

Table A.3: Protocol messages

PN Piece number within the chunk specified by CN. Needed to allow for fragmentation of chunks larger than payload size of a transport frame. 5 bit field gives a piece number from 0 to 31.

PT Total number of pieces in the chunk. This is required for proper reassembly of chunks. 5 bit field gives a maximum of 31 pieces in a chunk. Along with PS, this implies that a single chunk can at most be 31 kB in size. The chunk size should in reality be considerably smaller.

PS Piece size. Can be variable sized pieces within a chunk, so the size must be specified for each one. The size is in bytes/16, enabling 16 to 1024 bytes to be encoded for each piece within 6 bits. A single piece along with all headers should fit into a transport frame, which in the common case of a Ethernet or IEEE 802.11 link layer protocols has a payload of at most 1500 bytes.

A.2 Protocol Messages

A relatively sparse set of messages is employed in order to make the protocol light-weight and simple to understand and implement. The message set is described in this section. A summary of the messages is presented in Table A.3.

A.2.1 Announce

Type A header and payload (See A.1.1).

A node announces its existence and contents by unicasting an *announce* message to a peer. The announce message carries a summary of the content carried by the sending node, preferably in the form of a compact descriptor like a Bloom filter (Bloom, 1970)

in the *Descriptor* field whose length is specified in DESCRLEN. Announces can also be broadcast, serving as a beacon to announce a nodes presence.

Announces can be unicast to a specific peer or broadcast. Nodes discovering new peers send unicast an announce to bootstrap content exchange by inviting solicitations. If a compact descriptor of the nodes contents is employed, the newly discovered peer immediately has a rough idea of the nodes contents. The peers can thus make intelligent choices on whether to solicit content.

The present version of the protocol does not include any negotiation of parameters. The announce can, however, serve that purpose in future versions.

A.2.2 Pairing Request

Type A header and payload (See A.1.1).

A *pairing request* is comparable to an announce message, but is used to request an exclusive pairing with a potential peer node. A node can only respond to a pairing request if currently unpaired. The response is in the form of a *pairing response* message. Both pairing requests and responses can carry compact content descriptors and thus serve a comparable purpose to that of an announce in updating peers of the content carried by the node.

A.2.3 Pairing Response

Type A header and payload (See A.1.1).

See *pairing request*. A node can only respond to an *pairing request* if currently unpaired by sending a *pairing response*.

A.2.4 Content Request

Type A header and payload (See A.1.1).

A node sends a *request* to a peer to solicit content. The request can be either for content or a channel update, differentiated by the message type (MT) field. A receiver of a request then supplies the content to the best of its ability, optionally using a *content advisory* message to notify the requester of unavailable content.

A request can optionally carry a compact descriptor in the *Descriptor* field whose length is specified in DESCRLEN. This is analogous to the *announce* message.

The request can carry an *Options* field, which is a list of the form:

$$[number\,of\,requested\,messages]\,(\{channel; weight; filter; priority\})$$

A single request can thus carry a list of content, allowing a requester to solicit most or all of the content it wants in a single request.

The number of requested messages is optional. If omitted, the system default should be used. An optional list of tuples specifying the content requested can be specified. An empty *options* field signifies that either the requester wants any random content from its peer or it is requesting continuation of a previous request; a list of request tuples is unlikely to be satisfiable in a single turn and a receiver of a request would thus cache it for the given peer and utilize to schedule its content provision for several turns.

- *Channel* is a channel GUID. Can be omitted, in which case the filter field must specify the content to receive, e.g. by a meta-data search filter.
- *Weight* is the assigned weight for a private channel or cumulative popularity index for public channels. Unused if *channel* is omitted.
- Filter is an optional field that can further specialize a request.

  NONE No filtering. The tuple will be ignored if part of a list, unless a channel is specified. If the only entry in the options field, and both channel and filter are unspecified, the request will be treated as empty

  TIME A filter of the form TIME{begin[,end]} specializes a request in time by restricting it to a maximum age. A end time can optionally be specified, further narrowing the request.

  ITEM A filter of the form ITEM{guid} specializes a request to a single item GUID.

  CHUNK A filter of the form CHUNK{GUID,{(chunklist)|{chunkstart:number}}} specializes a request to a certain list or range of chunks within a specified content item.

  PIECE The PIECE{GUID,chunk,{(piecelist)|{piecestart:number}}} filter specializes a request to a certain list or range of pieces within a single chunk of a specified content item.

MSEARCH A meta-data search filter can specify any meta-data field and a value, optionally using the wildcard *. A meta-data search for any content by Eric Clapton would for example be: MSEARCH{author="*clapton"}. A node receiving a meta-data search criteria tries to satisfy it from any channel if a channel specifier is omitted in the query.

EXCLUDE The EXCLUDE{channel:(GUID list)|channel:[compact descriptor]} exclusion filter. A GUID list is large in size and thus impractical if a significant items should be excluded. A Bloom filter generated on the item IDs may however be of more manageable size.

- Priority takes the values {high, normal, low} and overrides the weight when a provider evaluates the request.

A node receiving a request is not expected to order or evaluate request tuples by specialization. Tuples of higher specialization can thus be masked by less restrictive tuples and thus rendered useless. It is up to the requester to ensure that such masking does not take place.

A request should be constrained to a single transport frame. The solicitation strategy of the requester shall thus select items to request in such a manner that the resulting query fits within the transport frame payload constraints.

A.2.5 Content Advisory

Type A header and payload (See A.1.1).

A node can optionally send a *content advisory* response message if it is unable to fully or partially provide requested content. Information on the request tuple that triggered the response is encoded within the *Options* field. It shall include the following:

- The MID of the request that triggered the response.
- A list of the request query tuples that the receiver is unable to provide.

The content advisory is an optional message and non-essential for the basic function of the protocol. It informs the requester that further queries on the subject are likely to be wasted. An advisory message should trigger no response in an recipient, apart from registering that the requested content is unavailable. It can thus be employed for optimization of the requester's solicitation strategy. The sender of the rejection shall follow it up by either a query of its own (thus reversing the provider/receiver roles) or by those content messages it is able to provide.

A node can optionally include a timestamp with its response to indicate approximate time of arrival for the requested content. This functionality is only meaningful for a gateway or any other node with an semi-permanent Internet connection. The retry timestamp is encoded within the *Options* field of the message.

A *content rejection* is a special form of *content advisory*, identified by its own MT, indicating that the receiver of a request cannot fulfill any part of a request. A node receiving a *content rejection* can mark the sender as uninteresting or try a different query.

A.2.6 Content

Type AB header and payload (See A.1.1 and A.1.2).

A single piece of content is packed into a content message and sent to a requesting peer. A node will send several content messages in a sequence, followed by either an *announce* or *request*, depending on the action it wants its peer to take. A sequence of content messages followed by a control message constitutes a *sending turn*.

Two distinct interpretations of content messages are used by the protocol:

Content messages contain a single content *piece* packed into a content message. The current chunk and piece number are encoded in the header, enabling the receiver to assemble the actual content item. The total number of messages and the serial number of the current message within a sending turn are also included. The content is stored in the Content field, whose length is specified in DATALEN. The content metadata can optionally be returned in addition to the actual content, and should be stored in the meta-data field, whose length is specified in MDLEN. Chunks/pieces of a content are not required to be transmitted in sequential content messages. Nor can a receiver assume that sequential content messages contain the same content item. A receiver should always use all available data, content GUID along with chunk and piece numbers to reconcile a content item.

Channel update messages contain meta-data for a channel encoded in a XML format. Chunk and piece numbers are in this case not used and the message number and total number of messages (MN and MT fields) are used to handle transport fragmentation and subsequent assembly. A channel update message is assumed to be transmitted in sequence and the sending turn utilized much shorter than a normal content transfer turn.

REYKJAVÍK UNIVERSITY

HÁSKÓLINN Í REYKJAVÍK

School of Computer Science
Reykjavík University
Kringlan 1, IS-103 Reykjavík, Iceland
Tel: +354 599 6200
Fax: +354 599 6301
http://www.ru.is