Content Caching in Opportunistic Wireless Networks

Sanpetch Chupisanyrote

KTH Electrical Engineering

Degree project in
Electrical Engineering
Master of Science,
Stockholm, Sweden 2011

XR-EE-LCN 2011:008

KTH Electrical Engineering

Master Thesis Report

Content Caching in Opportunistic Wireless Networks

Sanpetch Chupisanyarote

Supervisor: Professor Gunnar Karlsson

Date: 14 June 2011

School of Electrical Engineering
Kungliga Tekniska Högskolan
Stockholm, Sweden

Abstract

Wireless networks have become popular in the last two decades and their use is since growing significantly. There is a concern that the existing resource of centralized networks may not sufficiently serve the enormous demand of customers. This thesis proposes one solution that has a potential to improve the network. We introduce decentralized networks particularly wireless ad-hoc networks, where users communicate and exchange information only with their neighbors. Thus, our main focus is to enhance the performance of data dissemination in wireless ad-hoc networks.

In this thesis, we first examine a content distribution concept, in which nodes only focus on downloading and sharing the contents that are of their own interest. We call it private content and it is stored in a private cache. Then, we design and implement a relay-request caching strategy, where a node will generously help to fetch contents that another node asks for, although the contents are not of its interest. The node is not interested in these contents but fetches them on behalf of others; they are considered public contents. These public contents are stored in a public cache. We also propose three public caching options for optimizing network resources: relay request on demand, hop-limit, and greedy relay request. The proposed strategies are implemented in the OMNeT++ simulator and evaluated on mobility traces from Legion Studio. We also campare our novel caching strategy with an optimal channel choice strategy. The results are analyzed and they show that the use of public cache in the relay request strategy can enhance the performance marginally while overhead increases significantly.

Table of Contents

List of Figures

List of Tables

1. Introduction

Wireless communication has become very popular in recent decades and has many benefits over the wired network. In wireless network, a device sends data through the air, which is seamless and always available while the wired network depends on physical links such as copper wires and optical fibers. The physical connection is one of the significant drawbacks in the wired network. All cables must be installed during the construction. The number of users is pre-defined and fixed, so it is difficult to expand the network infrastructure, in case, users' demand is increasing. These disadvantages are overcome by the wireless network. It allows relocation and expansion, and saves the cost of cables and installation. Moreover, the wireless communication introduces more convenience to people in terms of mobility. They do not need to stationarily stay in their places in order to communicate with other people via the wired network. When they use wireless communication, they can walk around or go outside their homes and workplaces without losing the connection because the wireless network support the users' mobility.

1.1. Wireless Infrastructure Networks

Since there is no physical cable in wireless networks, a wireless access point or a base station is introduced to support communication in the system. Figure 1 presents an example of an infrastructure based wireless network. Users connect to the network by using any type of mobile device such as smart phone, PDA, or tablet through a wireless access point (AP). Users do not communicate directly with one another. They need assistance from an AP, which is an entity associating with mobile devices and functions as a bridge or a relay point to provide access to a backbone network. The backbone network is the system that provides interconnections and routing for the exchange of data. This traditional wireless communication is considered as a centralized system [1].

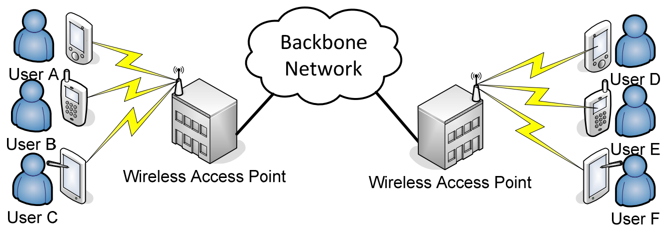

Figure 1. Wireless communication infrastructure.

The Inception of Social Networking

Nowadays, mobile devices are not only used for voice communications but also for data transmission. They enable us to watch videos, browse websites, etc. In particular, social networking data traffic namely Facebook, Twitter, or Flickr tends to surpass voice traffic. People can post and share their interests such as status messages, and photos from their mobiles anywhere and anytime. According to [2], during 2009, 400 million mobile subscriptions generate more data traffic than voice traffic from 4.6 billion mobile subscriptions. The forecast shows that this increase will double annually over the next five years. To satisfy users' demand, mobile operators need to invest more on expanding their network infrastructure. However, continuously expanding the network to support an insatiable need of customers is not an appropriate way to go in the long run because the operators have to purchase more network equipment to provide enough resources every year and the cost of the investment is expensive. Moreover, it takes the network operators a long time to expand the existing network. They have to forecast and plan how many devices or base stations that they have to purchase. Then, they make an order and wait until the devices are delivered. Then, the installation phase starts and this also consumes time until it is completely done. Besides, it also takes time to configure and test devices before launching them in the real network in order to assure that the newly-installed devices run stably and do not cause a problem to the real network. Therefore, in our perspective, the concept of *ad hoc network* which will be described in section 1.2 has potential for supporting this substantial growth.

1.2. Wireless Ad Hoc Networks

An ad hoc network is a wireless network with no centralized infrastructure such as access points or base stations. A wireless device in the ad hoc network establishes an one-to-one connection directly with another device that is in communication range. The ad hoc network takes advantages of decentralized server-base model. It enables users to establish a connection where the network is temporarily established or when it frequently changes. One example is a group meeting. Users have a meeting only for a short time, so the network needs to be set up momentarily. Furthermore, some users may leave the meeting early or some may come late; therefore, the number of users in the network may often change. Since there is no need for an infrastructure and a centralized server in P2P network, it evidently supports the users' mobility. Moreover, it avoids a single node failure and supports high reliability as well as high scalability [3]. More importantly, most models of the mobile devices available in the market nowadays can be implemented because they support various forms of radio communications in particular Bluetooth and IEEE 802.11.

Nevertheless, some types of ad hoc networks as mentioned in [4] and [5] are based on a network overlay which is considered as virtual links. The overlay must be set up and it requires a routing protocol such as Adaptive Distance Vector routing protocol (ADV), Ad hoc On-demand Distance Vector routing protocol (AODV), etc. Data can be transmitted only when a source node and a destination node are active at the same time. This restriction is not suitable for the scenario that users frequently and unpredictably join and leave the network. In this case, it may fail to update the overlay. As a result, a message might not be delivered. Thus, we propose the new notion which is appropriate for this type of network in the next section.

1.3. Opportunistic Wireless Networks

In this section, we introduce the novel concept called *opportunistic wireless networks*. The opportunistic wireless networks do not aim to establish an end-to-end path but they make use of the availability of individual devices [6], [7]. There is no assumption about a complete path from one node to another node. Nodes that are in contact range establish direct communication via radio interfaces [8]. After they finish establishing the connection, they start exchanging data. If a node finds that the content of its interest is available at its peer, it will send a message to request the content and download it from the peer. We define the term *"subscriber node"* for the node that requests content. The opportunistic wireless networks do not require the subscriber node and the node that has the content to be active at the same time. Both of them might never see each other at all. If the subscriber node is not currently available, a node that has the content will send it to another neighboring node. This function is called *"store-carry-forward"*. It will store and carry contents and may forward them if it meets a subscriber node in the future. It is also possible that the content may be disseminated to the subscriber node over multiple discontinuous wireless contacts but each one is only based on direct communication. Figure 2 shows an example of wireless opportunistic network. *Cindy* subscribes to the song *"symphony no. 5",* but this content is not available in her device now. *Bob* who works in the same office as Cindy knows that Cindy wants this song; therefore, Bob also helps Cindy search for this song. In the morning of the next day, while Bob is taking a bus to go to work, Bob's device discovers that *Alice*, who took in the same bus as him, is sharing many songs. One of them is *"symphony no. 5"*. When Bob found it, he sent a request to Alice for downloading this song from Alice and finally the song was stored in Bob's device. Then, when Bob arrived at his office, Bob met Cindy and he uploaded the song to Cindy. Eventually, Cindy got the song she wanted with Bob's assistance. This example clearly shows that the song is delivered from Alice to Cindy via Bob. Alice and Cindy never know each other and are not active at the same time. The song is transferred from Alice in the morning (7:55 A.M.) but Cindy gets it in the afternoon (12:31 P.M.). Nonetheless, there are also restrictions in the opportunistic wireless networks. Since the subscriber node and the node that contains content items can appear in the network at any time, it cannot support real-time communication. The application which adopts opportunistic data transmission should not be sensitive to the delay. One example of non-delay sensitive applications is file transfer [9]. File transfer is a mechanism for copying a file from one node to another node. It is not an interactive application or two-way communication. Even though it suffers from the delay during a transmission process, all data can be completely transferred. In addition to time delay, nodes also require capacity to store data that may be forwarded to the destination in the future. On the contrary, Voice over IP (VoIP) is a delay sensitive application. Users communicate in real-time. The delay causes an adverse impact on the conversation between them: They do not have smooth conversation. When the user on one side speaks, the user on the other side may not promptly hear what he has spoken because of the delay. This kind of service is considered unacceptable or has a bad quality of service (QoS). Hence, VoIP is an example of service that the opportunistic network concept is not appropriate to implement.

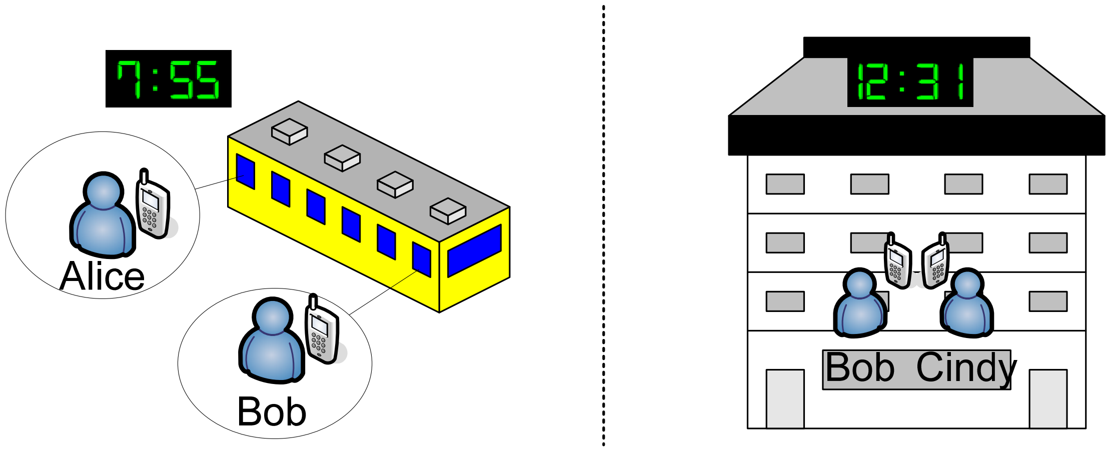

Figure 2. Wireless opportunistic network scenario.

1.4. The Podnet System

In this thesis, we study a special case of the wireless opportunistic ad hoc network called *Podnet* [10], [11]. Podnet is one of the data dissemination approaches designed for opportunistic content distribution. It works by means of wireless ad hoc network and *content-centric* networking. In typical communication, a node communicates with other nodes based on an address (i.e. IP address), but in content-centric networking, the node will communicate with others based on content items that exist in those nodes. This is because the contents are not stored in a fixed location or in a specific node. In other words, the node is not able to expect where the contents are found. It will know where the contents are stored only when it asks other nodes directly whether they carry these contents or not. If the content which is of a subscriber's interest is found in any node that is willing to share that content, the subscriber will solicit and download the content of interest from that node directly over ad hoc radio communication. Note that when a number of nodes participate in sharing contents, it is likely that there are more contents available in the network, and more storage capacities [12]. This means the efficiency of data dissemination in an opportunistic network increases with the number of nodes.

Figure 3 shows the system architecture and the content structure of podcasting [10], [13], [14], [15]. Applications access the system through an application programming interface (API), which defines the content structure for applications and allows them to publish contents or to subscribe to contents. The system design consists of several modules namely API, content solicitation, service discovery, and the solicitation protocol [14]. A convergence sub-layer serves for cross-layer interaction especially with the radio link. Content structure is divided into four levels: *feed*, *entry*, *enclosures*, and *chunk*. Contents are grouped into feeds. A feed is an unlimited storage for entries. An entry is the actual data of interest. An enclosure is a single file attachment such as a text file, audio or video. A chunk is the small part of an enclosure and it is a fixed-size data unit. The IDs of all content items in the system are also unique.

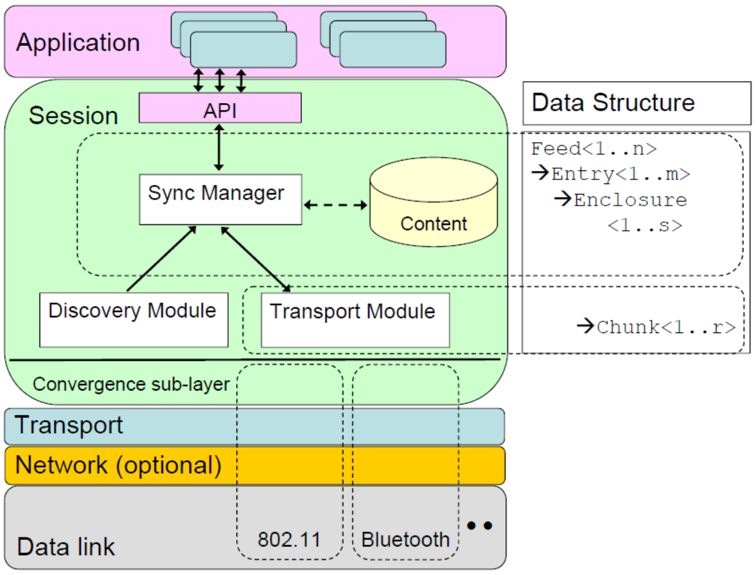

Figure 3. System architecture and content structures. [14]

The current system only implements private caching, and all contents that nodes store are those of their own interests. The idea of this strategy is that a node will download content items from neighboring devices (called neighbors) that are in communication range. The decision on which data to be downloaded is based on their pre-defined interests. If the node discovers that the neighbor stores data of its interest, it will ask that neighbor for downloading of the content items.

This thesis studies public caching. The concept is that the node will fetch and store not only private contents but also such contents that are not of its own interests. In order to support this system, two types of caching are introduced: private and public. The private cache contains only the contents that are of private interest to the user, while the public cache consists of data that can be of interest to neighbors. Contents are served from both caches in a mobile device. Our aim is to study different caching strategies and evaluate performance of data dissemination and overhead that the system experiences due to the public caching.

The remainder of the master thesis report is structured as follows. Section 2 contains a summary of related work. Section 3 provides the overview information about caching strategies. Section 4 explains the system design. Section 5 describes the implementation details. Section 6 introduces the experiment scenarios. Section 7 presents the results and the evaluation. Section 8 concludes the thesis and suggests some future studies.

2. Related Work

There are some studies about content distribution in wireless network. Each study has different concepts and designs but their purpose is to increase the performance of data dissemination in the network. We can categorize the related work into two major groups.

In the first group, authors in [16], [17], [18], [19] adopt an idea that *"Users that belong into the same community tend to share the same interest and have a strong social relationship"*. This means people who belong to the same community may have something in common. For example, students studying in a school of engineering are prone to be interested in robots, programming and technology issues while the students who study in a school of music tend to love in music, musical instruments, and arts. Besides, community does not refer only to the place that they stay together like in the same school or the same office but it also extends to the daily life of people. People who are staying in the same community may not know each other. For instance, some people go to work by taking the same route everyday. Although, they may get on the bus from different places and go to different destinations, these people stay together in the same bus for a while every day. Therefore, we consider that they belong to the same community because they meet one another very often. Even if it is only a short period of time, they can share or download contents from the same people very often. Papers [16], [17], [18], [19] adopt this concept of common strategies but they differ in the detail of design.

The authors in the second group believe in the idea that *"In the community, some people are more popular and they are likely to interact with more people than others"* [20], [21]. If contents are stored on popular devices, sometimes called *hubs*, there will be higher probability that data will be disseminated in a wider range. For example, celebrities are likely to visit more people than what office workers are. If we want our contents to spread over the wide area, it is a better idea to ask the celebrities to carry our contents. In [20], the popularity is computed by the changes in the degree of connectivity of a host while the authors in [21] employ a metric called *"betweenness"* and present a global ranking system called *"BUBBLE"*.

The details of all papers are described as follows.

The authors in [16] use the concept from the first group to form an overlay and implement publish/subscribe communication in delay tolerant networks (DTN). The overlay formation which is a set of logical links relies on community detection algorithms. There are two algorithms proposed in [16]: K-CLIQUE and SIMPLE. K-CLIQUE community is defined as complete sub-graphs of size k that can be reached from adjacent neighbors. Figure 4 shows an example of scientific community containing the networks of scientists who are working on biology, chemistry, mathematics, computer, physics, and astronomy. Each of the networks is called as sub-graph. There are six sub-graphs in this community and the maximal path for linking all sub-graphs is three as indicated with dotted lines. Therefore, it is considered as 3-

CLIQUE community. SIMPLE is a community detection algorithm based on the number of contacts and contact durations. The basic idea is that a mobile device will keep the information about nodes that it contacts and accumulates the contact duration of each other until the duration exceeds the defined threshold. Then, it promotes that node to its neighbor in the overlay.

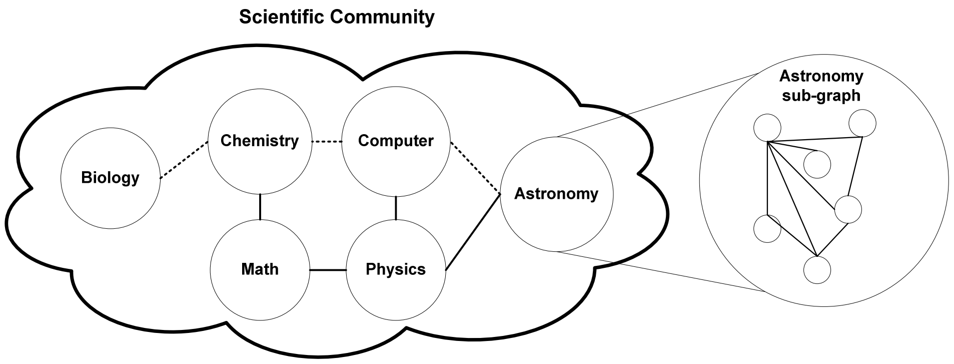

Figure 4. 3-CLIQUE community.

In both techniques, when a community is detected, the first step is to create an overlay by exchanging information between nodes. After, a broker node is selected whose function is to store and forward messages to all subscribers. Simulation results show that the overlay approach provides high reliability of content delivery when nodes belong to the same community. However, the authors assume that members, except the broker node, only store data they are subscribed to and do not consider public caching.

In [17], the design of the data dissemination is based on a utility function computed when a node contacts a peer. The node will fetch the content from the peer that gives the maximum value in the utility function (U).

$$U = \sum_i w_i u_i$$

where w_i is weighing value and u_i is utility components. The value of u_i is computed based on the size of the content, the probability that the content is available in the network and the probability that the node can get access to that content. See [17] for more calculation details. The value of w_i is based on four weighing policies proposed in [17]. In the most frequently visited (MFV) policy, the weight is calculated as the ratio of the time a user spends in one community to the total time a user spends in all communities. The most likely next (MLN) policy estimates the weight as equal to the probability that a user enters another community given that he/she is in the current community and the weight of current community is set to 0. The future (F) policy adopts the concept of MFV for future communities, but sets the weight of current community to 0. In the present (P) policy, the weight of the current community is equal to 1, but all other communities are 0. The last policy introduced in [17] is the uniform social (US) policy; in this case, all communities are equally weighed. The results reflect that the MLN and F policies give higher performance than the rest in terms of hit ratio, fairness and also in multi-hop social paths. Nevertheless, the authors in [17] assume that the cache space on a node accommodates exactly all messages belonging to an individual feed channel.

In [18], the authors present an idea of pre-fetching. A node determines whether to store contents in its cache based on a decision making technology such as analytic hierarchy process (AHP) and grey relational analysis (GRA). The paper evaluates AHP-GRA caching strategy (AGCS) on three aspects: expected path length, differential expected path length and priority of the content request. The concept of expected path length is that nodes store the content that has the minimum path cost (E_{path}) which is computed based on the probability that two nodes contact each other along the path.

$$E_{path} = min \sum \frac{1}{P_{i,j}}$$

where $P_{i,j}$ is the meeting probability between node i and node j. is the The differential expected path length (ΔE_{path}) indicates the relative delivery probability of the content to the destination node. It is calculated as the difference between E_{path} of two nodes and the nodes prefer the lower ΔE_{path} showing the closer relation between two of them. The simulation presents the result in AGCS, uniform and most solicited cases. The uniform strategy treats all contents equally. In the most solicited, it focuses on caching the most popular contents. The results show that AGCS outperforms the uniform policy and most solicited regarding request rate (the ratio of total contents delivered to total contents subscribed) and caching cost (the ratio of total contents cached to total contents delivered). Nonetheless, the proposed algorithm rely on ad-hoc routing computed by Dijkstra algorithm. Furthermore, the authors in [18] divided the network into serveral zones and each zone has different probabilities of being visited. For example, the area that tends to have higher probability of being visited is called *public zone* and each node will have one preferable zone named *home zone*. Regarding the content size, they only consider the small size of contents (128 kB) such as ringtones, wallpaper, etc.

The authors in [19] propose a feed selection strategy for enhancing the performance of data dissemination in the network. Nodes will select and carry feeds that are not of their interest and store them in their public cache. They propose three feed selection strategies: uniform (UNI), top popular (TOP), and optimize social welfare (OPT). In UNI, users randomly select a feed from the set of feeds that they do not subscribe to. In TOP, users prefer to pick up the most popular feed. OPT will not statically select the feed but it will dynamically change the feed that it carries according to a distributed algorithm. In OPT, when two nodes (node A and node B) communicate with each other, they will exchange information about contents that exist in their caches. If there are some contents that node A does not have but exist in node B, node A may drop its own public feed and replaces with the new feed from node B. Node A will make a decision as follows: (i) node A will select one of its feeds from its public cache uniformly; (ii) node A will uniformly select one of the feeds in node B that it does not have; (iii) node A computes probability *(q)* of exchanging these two feeds; (iv) node A draws a random number *(U)* uniformly in [0,1]; (v) if *U>min(1,q)*, then node A drops that selected feed and replaces with the new feed from node B. Note that for further information about how to compute *q*, see [19]. The simulation results show that OPT performs better in terms of lower dissemination time than the rest. This is beneficial in supporting data dissemination that has short contact duration; however, it does not guarantee that data can be completely sent, particularly in a long distance.

Paper [20] proposes *SocialCast*, a routing protocol supporting publish/subscribe communication, which exploits the prediction of co-location and mobility. The authors believe that if contents are stored in popular nodes, the contents will be disseminated better. The proposed routing protocol for popularity prediction consists of three phases. The first is *interest dissemination*. In this phase, each node broadcasts the list of its interested content to neighbors. Second, in *carrier selection*, the centralized system elects a node to be the forwarding node. The function of the forwarding node is to store and forward contents that it is carrying. The best forwarding node is calculated by comparing changes in the degree of connectivity which is the frequency that the node meets new neighbors. The last phase is *message dissemination*. The carrier node transmits messages to all neighbors that are interested in the contents. The results in [20] show that with the use of popularity prediction, SocialCast achieves better performance in terms of *message delivery* (the ratio between the actual number of messages sent to subscribers and the ideal one), and *network traffic* (showing the number of forwarded messages) compared to the no-popularity selection scenario. However, the buffer size is not taken into consideration by setting to the size to infinity.

The authors in [21] propose a novel algorithm called BUBBLE. BUBBLE tries to select popular nodes (high centrality) as relays by making a decision based on the knowledge of community structure. A source node forwards a message until it reaches the node which is staying in the same community as a destination node, and then this node delivers the message to the destination. The authors also consider the metric *betweenness*, which is the number of times that a node stays in the shortest path between two other nodes. This paper compares the results from BUBBLE with the following algorithms:

- **WAIT**—hold a message until the sender directly encounters the receiver.
- **FLOOD**—send messages to all nodes in the system.
- **MULTIPLE-COPY-MULTIPLE-HOP (MCP)**—a number of copies are transmitted subject to time-to-live (TTL).
- **LABEL**—a message is forwarded only to the nodes in the same community.
- **PROPHET**—a message is forwarded when it has higher delivery probability than the current node for a specific destination.

The results indicate that BUBBLE gives higher delivery success ratio (the ratio of sent out messages to the total created messages), and lower delivery cost (the total number of messages including duplicates) than other algorithms.

Here again, we meet the same limitation as in [16] and [17]: Nodes do not take into account the availability of a private cache and a public cache and the algorithms are dependent on a routing protocol, i.e. BUBBLE RAP has to establish paths between a source node and a destination node, so it depends on the routing protocol rather than a caching mechanism.

3. Caching Strategies

In general, users subscribe to all types of contents that they are interested in. It is possible that only some contents of interest are available in their device while some are missing. Therefore, the users (or nodes) will ask to download these missing contents from other nodes in the network. Nonetheless, all users do not have the same interests; their subscriptions vary. Therefore, when a node contacts another node that is in communication range and asks for the missing contents, there is no guarantee that the other node can upload the missing contents because the node that receives the request can share only the contents it carries, which are generally the private contents. Although the node that received the request cannot promptly provide those contents, it will try to fetch them from its neighbors to support the node that sends the request if they meet again in the future. Since these contents are not the contents of its interest, the node will store them in a public cache. Therefore, in this thesis, we propose public caching strategies for enhancing the performance of data dissemination.

To support the caching strategy, we introduce the terms as follows.

- **Available public list—**the list of IDs of contents available in the public cache.
- **Missing public list—** the list of IDs of contents missing in the public cache that a node is expected to download on behalf of other nodes.
- **Available private list—** the list of IDs of contents available in the private cache.
- **Missing private list—**the list of IDs of contents missing in the private cache that the node expects to download.
- **Available set —** the list of IDs of contents available in the node.
- **Interested list—**the list of IDs of contents expected to be downloaded that are missing from both the private and public caches.
- **Subscribed set** — the list of IDs of contents that the node subscribes to.
- **Waiting list** — the list of IDs of contents that the node expects to download from the peer that is now in contact.
- **Neighbor** — nodes that are in communication range.
- **Indirect neighbor** — nodes that are not in the communication range of subscriber nodes but provide the contents that are of interest to subscriber nodes.

Figure 5 shows an example of the caching model that we define according to the terminology above. The user initially subscribes to six contents which are 1, 3, 5, 7, 9, and 11. There are only five contents (1, 5, 9, 2, and 4) available in a device. The content 1, 5, and 9 are stored in the private cache while the content 2 and 4 are in the public cache. The missing contents, which are 3, 7, and 11, are added into an interested list.

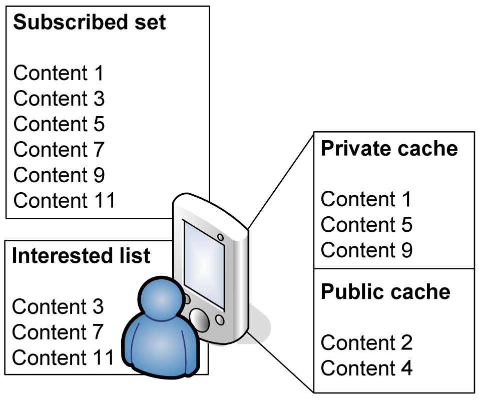

Figure 5. Caching model.

3.1. Relay Request

We propose a caching strategy called "relay request". The general idea behind this strategy is that neighbor nodes help a subscriber find missing contents from other nodes. The neighbor nodes will try to find those public contents, and to store, carry, and forward them to the subscriber node that previously sent the request.

First, a subscriber node sends its neighbors a request list specifying the contents that are missing. If the neighbor has one or more contents that match the request, it will inform the subscriber node that it has the contents and is willing to share them. Then, the data transfer process begins and finally ends when all contents that it can share are completely transferred. This is the end of the first step (private contents sharing). Then, if there are some contents that do not exist in the neighbor node 's cache, the caching process will begin. The neighbor node will try to search for these contents from its neighbors. These neighbors are perceived as indirect neighbors of the subscriber node. It will insert these public content IDs in an interested list, so the neighbor's broadcast message consists of the list of IDs of its own private contents that are missing and the public content IDs that it helps to solicit. If the indirect neighbor has the contents, the neighbor node will start downloading and will store them in its public cache so that it can forward them to the subscriber node when it meets that node again in the future. A process is shown in Figure 6.

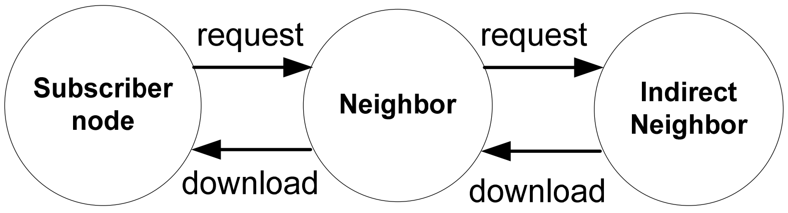

Figure 6. The concept of relay request.

Relay Request Process

The relay request protocol is displayed in Figure 7. Each node periodically broadcasts "broadcast REQ" messages containing an interested list. The process starts when User A begins to send the Broadcast REQ message. Once other nodes in communication range receive the Broadcast REQ message, they check whether they have these contents in their private or public caches. If the contents cannot be found, they will add these content IDs into its missing public list. However, if one or more contents are cached, they will send "unicast reply" messages back to the subscriber node. In Figure 7, User B sends a unicast reply message to User A. The unicast reply message contains the list of content IDs that User A asks for and which User B is able to share. When User A receives the unicast reply message, User A now knows what contents it can download from B and User A will store the content IDs in the waiting list, which is the list of the content IDs that the node expects to download from the neighbor. Then, it sends a "unicast REQ" message back to B to inform that it is ready to download those contents. User A will download the content items one by one. Hence, in the unicast REQ message, it contains only one content ID that User A wants User B to upload. When User B gets the unicast REQ, it starts a data transmission process by sending "DATA" messages to the subscriber node. In case a content item is larger than the maximum transfer unit (MTU), which is the maximum size allowed in data transmission, it fragments the item into smaller chunks. The sequence number and the total number of chunks will be added in the DATA message, setting the first chunk to number 1, the second to 2 and so on, so that the subscriber can reassemble all chunks in the correct sequence. The detail of fields in a message will be presented in section 4.2. On the User A side, when the DATA message arrives, User A will store the data either in its private cache or public cache depending on the type of the content. However, if the data needs reassembling, User A will temporarily store DATA messages until it successfully receives all of them from User B. If User A gets all data, it will put this content in its cache. Now this content is available and can be shared to other nodes. Unless all data arrives at User A, incompleted entries are discarded. After the first content has been successfully received, if there are more contents that User B can provide indicating in the waiting list, User A will send Unicast REQ for the next content item. The process is requesting one content item at a time because nodes have high mobility and short contact duration. Therefore, if User A sends the list of all contents that it wants in the first Unicast REQ message, User B will continuously send all of them. This probably causes User B to send a lot of unnecessary messages if two of them are later out of communication range. Many DATA messages that User B sends to User A might fail to be delivered.

Furthermore, the process takes into account the priority of the content. Nodes set the first priority to private data and the second priority to public data, which means a node will not start sending Unicast REQ to download public contents until it finishes downloading all private contents. The reason is that when a private content is completely transferred, it is certain that this downloading process is effective because the node gets contents that it desires. On the contrary, downloading a public content item will be useful only when it is forwarded to nodes that are interested in it. The relay node gains nothing in helping download public contents that are not of interest.

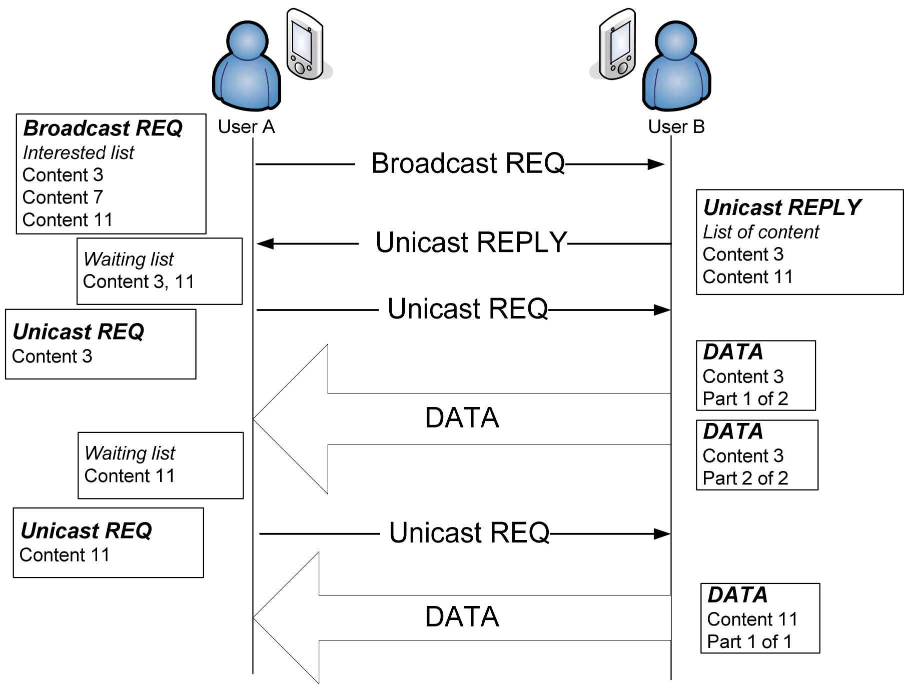

Figure 7. Relay request protocol.

The relay request process has a drawback. For example, when a subscriber node broadcasts the request messages, all nodes in communication range receive them. Nodes that have the content will respond back to inform the subscriber node that they are able to share this content, while the other nodes which do not have this content will try to fetch this content. However, the helper nodes do not know that the subscriber node does not need their assistance because it can download from a node that has the content. In some cases, a subscriber node sends the request to its neighbor and the neighbor completely fetches data and stores it in a public cache, but the content is not transferred because both nodes will never meet again in the future. Hence, we try to optimize the relay request process by introducing three public cache options. This optimization process will be explained in sections 3.2-3.4.

3.2. Relay on Demand

In a relay request process, when a node receives a broadcast request message containing a list of content IDs, it will check whether these contents are available in its own caches. If there is any content item not available in its caches, it will insert the content ID into its missing public list and will try to find the content from its neighbors. However, it may take long time until the public contents are discovered. The node that sends the request is probably out of communication range by then. Thus, if the node persists in searching the contents to help all nodes that it met in the past, the public list will be huge and it will be ineffective because it wastes a lot of time, capacity, energy and network resources to download contents which cannot be delivered to the node that requested them. Therefore, we introduce the function called "relay on demand" to optimize data dissemination. The concept of relay request on demand is the following: a node will request the public content only once. If other nodes in communication range are able to share the desired public contents, the node will fetch and store them in its public cache. However, if there is no response from any of neighboring nodes, it will not repeat broadcasting the same public contents request again.

Nevertheless, if the neighbor node receives the request for the same public contents from the same subscriber node again, it will help search these contents even though they are duplicated. This is because the request is up-to-date and it also implies that the subscriber node is still in communication range and the contents it needs have not been found yet. Therefore, the neighbor node will request the same contents for the second time and so on. In summary, this notion is that the neighbor node will relay the request for public contents whenever it receives a broadcastREQ message from other nodes and the neighbor node will send the request for those public contents one time per one request, which can be called relay on demand.

3.3. Two-hop Limit Relay Request

Multi-hop relay request is a strategy that allows contents to be downloaded from other nodes that are many hops away. Figure 8 presents an example of a multi-hop relay request. We assume that every node can communicate only with its adjacent nodes because of the limitation in communication range, which means that node 1 can communicate with node 2 only, while node 2 is able to reach both node 1 and 3. Node 3 can also contact two nodes which are node 2 and 4 but node 4 can only communicate with node 3. First, node 1 starts broadcasting a request message. Node 2 receives the broadcast message containing the list of the contents IDs that node 1 needs. However, the contents that node 1 requests are available neither in the private nor in the public cache of node 2; therefore, node 2 broadcasts a request to other nodes. Node 3 gets the request message from node 2; however, node 3 does not have these contents either. Thus, node 3 broadcasts the request to search for them. Fortunately, node 4 gets the broadcast message from node 3 and can share these contents. The download process is in a reverse direction from node 4 to node 3, to node 2 and finally to node 1. All download messages are sent unicast. Node 1 eventually gets the contents from node 4 by the help of two relay nodes (node 2 and node 3). Nevertheless, if the contents are not found in node 4, the process will carry on relaying the request to the next node. This is considered as an unlimited persistent relay request process.

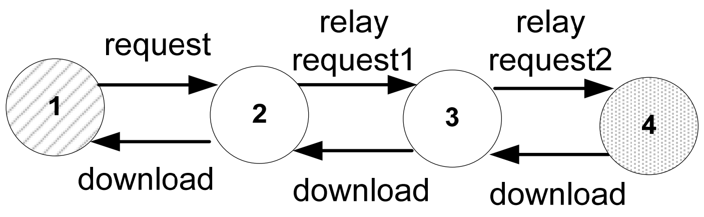

Figure 8. Multi-hop relay request.

This strategy is effective for a stationary network. In a network with high mobility, nodes are prone to move all the time. Thus, it is not appropriate to adopt multi-hop relay request for the following reasons. First, node 1 in Figure 8 may move out of communication range as shown in Figure 9. Node 2, 3 and 4 cannot contact node 1 again. As a result, it is futile for node 2 and node 3 to fetch the contents from node 4 in order to forward them to node 1. The second reason is that there is a possibility that node 1 moves closer to node 4 so that node 4 can directly upload the content to node 1 as shown in Figure 10. Thus, a process that node 2 and node 3 download the content for forwarding to node 1 is unnecessary. The two inefficient unlimited persistent relay request cases are displayed in Figure 9 and Figure 10. Therefore, we try to limit the request to be relayed only two hops. This is because during the time that other nodes are searching data for the subscriber node, the node does not stay still and does not wait for the help. It also moves around in the network to find the content it needs. If the content is delivered from many hops away, it is likely that the node has already gone out of communication range. Thus, we believe that when we limit the node to search public contents only two hops away (relay the request only one time), the time it takes to fetch the content is not so long that the node 1, node 2 and node 3 will move out of communication range. Moreover, less cache capacity and time for ineffective data transfer are consumed. This means that when node 1 broadcasts the request to node 2 and the contents which node 1 needs are not available in its caches, node 2 will broadcast the request for these public contents to node 3. If the contents that node 1 wants are not found in node 3, node 3 will not continue relaying the request for the contents that node 1 needs to node 4 as shown in Figure 11. The scope of searching public content will be restricted to 2-hop neighbors and there is maximally one relay node involved in the public data transmission process. This caching option can be extend from two-hop limit to 3-hop limit, which means the scope of searching public contents will limit to 3-hop neighbors.

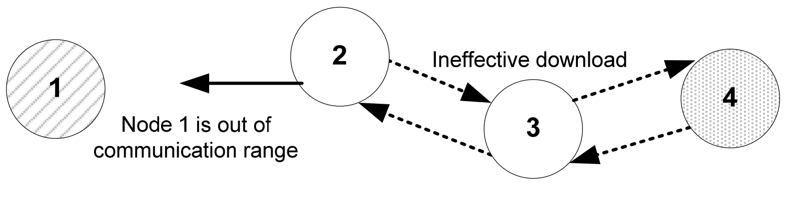

Figure 9. Node is out of communication range.

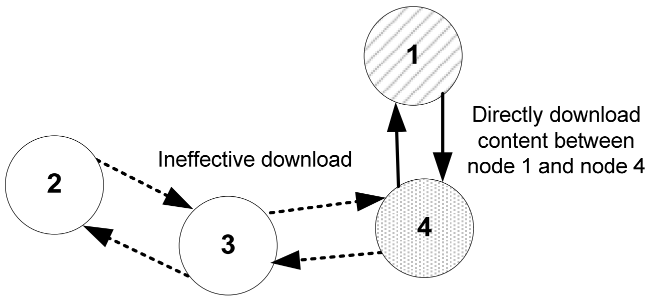

Figure 10. Node directly downloads a content.

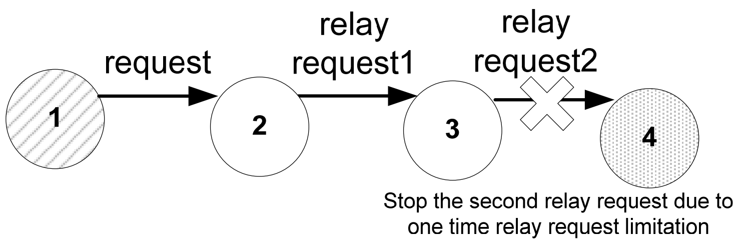
relay request1
relay request2
request
1
2
3
4
Stop the second relay request due to one time relay request limitation

Figure 11. One time relay request.

Furthermore, unlimited persistent relay request causes the traffic to grow exponentially. This results in a broadcast storm [22] which is the phenomenon that excessive traffic degrades network performance. An example is shown in Figure 12. We assume that each node has 3 neighboring nodes. Node in level 1 sends 3 broadcast request messages to 3 neighboring nodes. Neighboring nodes in level 2 receive the broadcast messages and then they relay the request messages to their neighbors which are the nodes in level 3; now there are 6 messages sent from nodes in level 2 to nodes in level 3. However, if we relay the request for second time from level 3 to level 4, 12 messages will be sent. This example shows that the number of messages increases exponentially from 3 messages to 6 messages and to 12 messages in each level. Hence, if we keep on relaying the request, the huge number of messages transmitted in the network will cause congestion in the network.

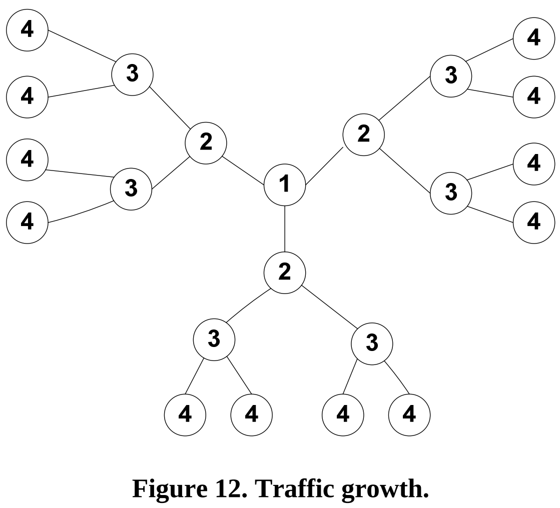

Figure 12. Traffic growth.

3.4. Greedy Relay Request

In this strategy, during the sending DATA process, a node does not download entire contents from a waiting list. It always listens to responses from its neighbors informing which contents they are able to share. As soon as it knows that the neighbors share its private contents and it is now downloading public contents, it will immediately update the waiting list, send the request for the new private content and ignore the old waiting list. The reason behind this idea is that, in the relay request approach, a node will download all contents in the waiting list that the neighbor node can provide no matter whether they are private contents or public contents. It will not start a downloading process with a new neighbor and will not update the waiting list until it finishes getting all contents in the waiting list from the first neighbor node. Therefore, the node may lose an opportunity to get new private contents from new neighbors during the time it does not finish downloading process with the old neighbor. For example, a node is downloading public contents from the first neighbor. A new neighbor also shares private contents. Unfortunately, the node cannot fetch the private contents from the new neighbor because it has not finished downloading the public contents from the first neighbor. This indicates that the node misses the opportunity to download the contents that it needs since it is downloading the public contents which are out of its own interest.

The details of greedy relay request process are described as follows. A local node periodically broadcasts its interested list. If other nodes are in communication range, the local node will get a unicast reply message from each of them. When the local node receives the unicast reply message, it responds with a unicast REQ to download contents and finally DATA messages will be sent until all contents are completely delivered. However, during the data download process, the node is still broadcasting the request list periodically (e.g. every one second); therefore, it receives unicast reply messages in response. If the current downloading data belongs to a public content, it will check new content IDs which are in unicast reply message whether there are private contents provided or not. In case, the list of content IDs in the unicast reply message contains a private content that it needs, it will not send the unicast REQ message for the next public content item to the neighbor it is now downloading from but will instead send the unicast REQ for a private content from the new neighbor. For instance, Figure 13 shows that User A subscribes to content 1 and 4, and helps download content 2 and 3. After User A broadcasts the message showing that he wants content 1, 2, 3 and 4, he gets a unicast reply message from User B showing that User B can provide content 1, 2 and 3. User A will insert the content ID 1, 2 and 3 in its waiting list. Then, User A starts downloading content 1, 2 and 3 respectively by sending the unicast REQ message for each content item. When content 1 is completely downloaded, User A sends the unicast REQ message for content 2. While User A is waiting to receive content 2, it receives the unicast reply message from User C in response of broadcast REQ, showing that User C is willing to share content 4. In this case, User A will not continue sending the unicast REQ for content 3 to User B but it sends the unicast REQ message for content 4 to User C instead. Note that User A still waits for content 2 to be completely transferred according to the previous unicast REQ that User A sent to User B because User A cannot cancel the unicast REQ message for content 2 that it has sent. Thus, B will send content 2 back to User A. When a DATA message of content 2 arrives at User A, it is not good to discard content 2: Since User B has spent time and resources to send content 2 and it now arrives at User A, User A should store the content that it previously requested in its public cache.

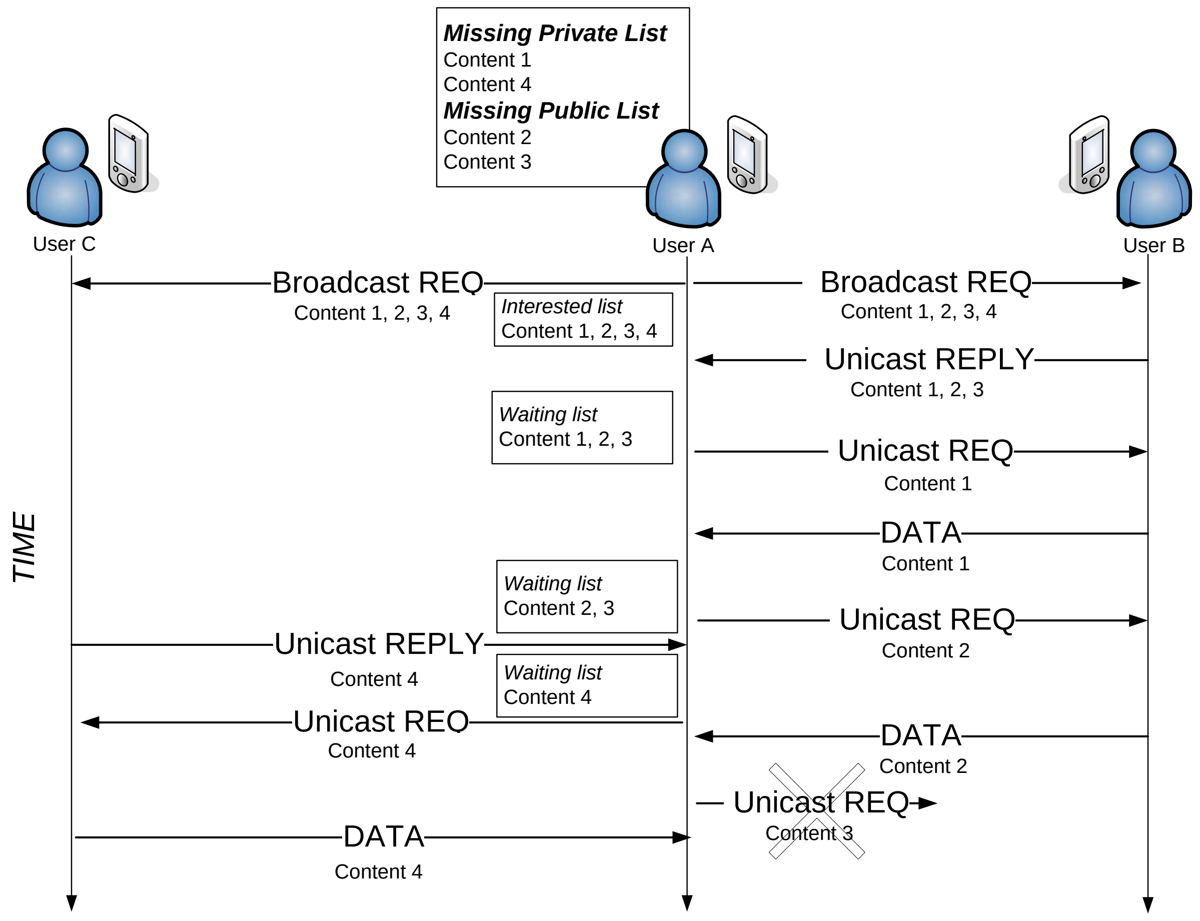

Figure 13. Greedy relay request process.

4. System Design

To conduct our study, we adopt the MiXiM [23] framework together with the OMNeT++ simulator. MiXiM offers the framework of simulation for mobile wireless and fixed wireless networks. It includes models of protocols, radio wave propagation, node components, and message delivery. MiXiM also provides a well-constructed API for writing an application on top of it to be run on OMNeT++ [24]. A user can make use of existing modules and extends them to a new application. The code in MiXiM is based on the C++ language. In this thesis, we extend the MiXiM framework for mobile wireless networks with MAC layer IEEE 802.11.

4.1. Node Components

A node refers to a mobile device in the system. It carries contents, shares contents and sends messages to other nodes. As shown in Figure 14, one node consists of six main modules, which are: interface card (NIC) module, network module, application module, mobility module, ARP module and utility module. The first three modules are connected together into layers and the rest of them assist in supporting an application function. In this section, we will describe each module in detail.

4.1.1. Network Interface Card Module

The network interface card (NIC) is the interface between a radio network and a node. It is divided into two layers which are physical layer and Media Access Control (MAC) layer. The physical layer is the lowest layer defining the means to transmit data through a medium which is air in the wireless network. It encodes data into radio signals. The second is MAC layer. It is the interface between the physical layer and the network layer. It encapsulates data messages received from the higher layer and sends them to the lower layer. In the same way, it decapsulates data messages received from the lower layer and sends them to the higher layer.

4.1.2. Network Module

The network layer is the interface between the NIC module and the application module. The function of this layer is end-to-end routing. It specifies source address and destination address, encapsulates data messages received from the application layer and then sends them to the lower layer. On the other side, it decapsulates data messages when they are received from the lower layer.

4.1.3. Application module

The application layer is the highest layer which works on a specific task defined by users. The function of an application is processed in this layer. In our simulation, the application layer manages the data dissemination process, caching strategy, and content management.

4.1.4. Mobility module

The mobility module gives the pattern of nodes' movement. First, the node is initialized a starting position. Then, when it begins to move, we can specify speed and direction to the node to move such as 1.6 m/s at 90 degrees. Instead of setting speed and direction, we can also set the position of the node at a specific time. For example, the position of the node is at co-ordinate (20, 40) in 0.6 second. The detailed explanation of the mobility is presented in section 5.2.

4.1.5. ARP module

Address resolution protocol (ARP) is a table mapping between network layer addresses and MAC layer addresses. This information is used for giving the address to a routing process particularly in sending unicast messages.

4.1.6. Utility module

The utility module mainly provides information exchange between different modules of a node to fully support a simulation.

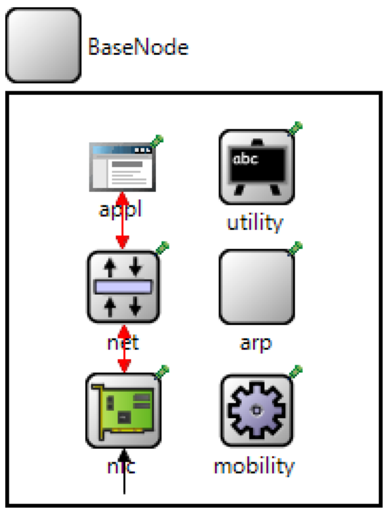

Figure 14. Node components.

4.2. Message

A message is the information which is sent from one node to other nodes for some purposes. It is first created in an application layer, the highest layer in the node. MiXiM offers a code for an application message which specifies a source address from which the message is created and a destination address where the message will be sent to.

MiXiM application messages

```
ApplPkt{
    Source address
    Destination address
}
```

To support our application, we need to send the list of content IDs and contents themselves as well. Therefore, our application message extends the MiXiM message and adds some parameters as follows.

Application messages

```
PodnetMsg extends ApplPkt{
    Sequence number
    Total parts
    Private list
    Public list
    Interested list
    Data
}
```

The following fields are parameters in a message specified at an application layer level.

- **Source address** — the address of source node in an application layer.
- **Destination address** — the address of destination node in an application layer.
- **Sequence number** — the field showing the chunk number of the fragmented content.
- **Total parts** — the field indicating the total chunks of the content after fragmentation.
- **Public list** — the list of IDs of public contents that a node is searching for.
- **Private list** — the list of IDs of private contents that a node is searching for.
- **Interested list** — the list of IDs of all contents that a node is searching for.
- **Data** — the actual content data.

When the message is sent down to the lower layers, it will be encapsulated by each layer and added more information until it is transmitted. The process is as follows: When the message is sent down to a network layer, the network addresses of a source and destination will be added. Then, the message will be encapsulated and sent down to the MAC layer. When the message arrives at the MAC layer, the MAC addresses of the source node and the destination node are inserted. The message is then encapsulated and sent down to the physical layer. At the physical layer, the message will be scheduled and sent out from a radio interface.

4.3. Content Popularity

We first create the pool of contents, which specifies the number of content items available in the network and the size of each content item. Each feed consists of one content item. Then, based on the Zipf distribution [8], the mobile device randomly selects feeds that it wants to subscribe to from the pool. After the device has assigned the subscribed feeds, only some of them will be assigned to be available in the device. Again, these available contents will be randomly selected based on the Zipf distribution. As a result, all contents that the device contains at the beginning are contents that it subscribes to, so all of them are stored in the private cache.

5. Implementation Details

5.1. Functions

The process shown in Figure 7 is implemented in the simulation program named OMNeT++ [25] a discrete event simulator. It provides a modular architecture for designing communication networks. It is programmed in C++ language. There are eight main functions as shown in Figure 15.

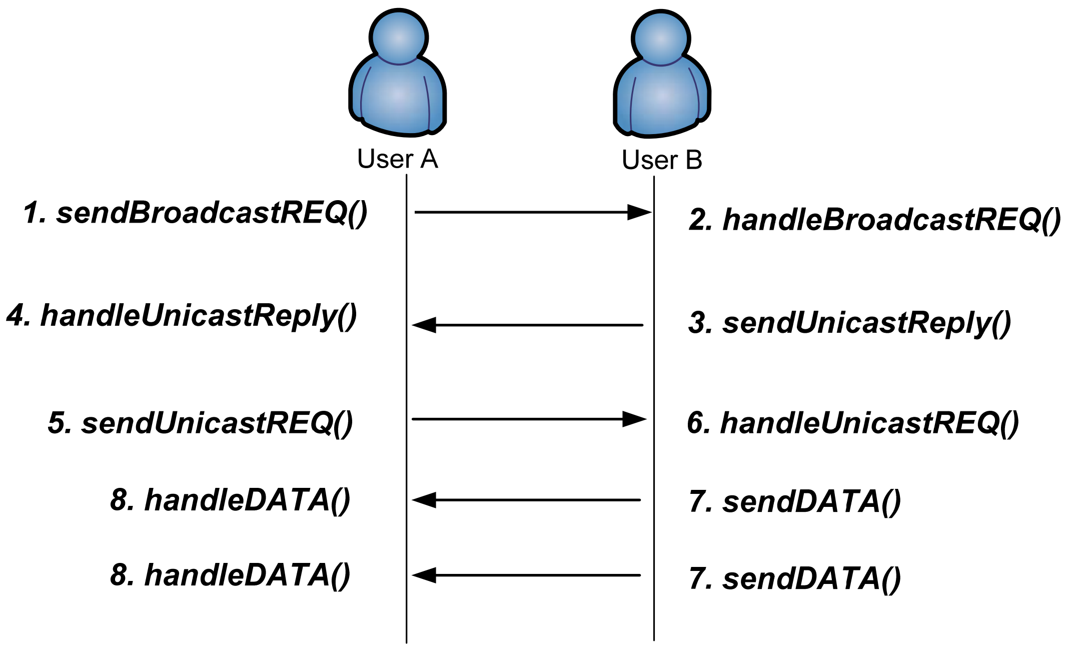

Figure 15. Functions.

5.1.1. Main Functions

Function 1. Node periodically broadcasts messages containing the list of contents that it needs.

```
sendBroadcastREQ (){
      create broadcast message
      message.insert(interested list)
      periodically send the message out every second.
}
```

Function 2. Node handles a broadcast request message, updates missing public list and interested list, and invokes sendUnicastReply function in case there is a content to download.

```
handleBroadcastREQ(message){
    incoming list = message.getList()
    peerAddress = message.getSource()
    For each content item ∈ incoming list do
        if (caches.find(content)==TRUE) then
            temporary list.insert(content ID)
        else missing public list.insert(content ID)
            interested list.insert(content ID)
        end if
    end for
    if (temporary list ≠ empty) then
        sendUnicastReply(temporary list, peerAddress)
    end if
}
```

Function 3. Node sends UnicastReply specifying the contents it can share

```
sendUnicastReply(temporary list, peerAddress){
    create unicastReply message
    message.insert(temporary list)
    send message (message, peerAddress)
}
```

Function 4. Node handles UnicastReply message, inserts the content items which are expected to be downloaded in waiting list, sets node state to BUSY, and invokes sendUnicastREQ function.

```
handleUnicastReply(message){
    incoming list = message.getlist()
    peerAddress = message.getSource()
    For each content item ∈ incoming list do
    If (content ID ∈ missing private list) then
        waiting list. append(content ID) in the beginning
    else if (content ID ∈ missing private list) then
        waiting list. append(content ID) in the end
    end if
    If (nodeState == IDLE) then
        nodeState = BUSY
        requested content = the first content in waiting list
        sendUnicastREQ(requested content, peerAddress)
        waiting list.delete(requested content)
    End if
}
```

Function 5. Node sends out a UnicastREQ message, and sets the timer for checking time out.

```
sendUnicastREQ(content ID, peerAddress){
    create UnicastREQ  message
    message.insert(content ID)
    schedule(timer, timeout period)
    send message (message, peerAddress)
}
```

Function 6. Node handles a UnicastREQ message, checks which content will be transferred and invokes sendDATA function.

```
handleUnicastREQ(message){
    incoming ID = message.getContentID()
    peerAddress = message.getSource()
    sendDATA (incoming ID, peerAddress)
}
```

Function 7. Node fragments a content if the size is larger than MTU, puts a sequence number and total chunks in the message, and sends messages out.

```
sendDATA(content ID, peerAddress){
    If (size of the content > Maximum Transfer Unit) then
        total part = (size of the content /Maximum Transfer Unit) +1
        sequence number = 0
        While (sequence number ≠ total part) do
            sequence number = sequence number + 1
            create Data message
            message.setSequence(sequence number)
            message.setPart(total part)
            send message (message, peerAddress)
        end while
    Else create Data message
        message.setSequence(1)
        message.setPart(1)
        send message (message, peerAddress)
    End if
}
```

Function 8. Node handles DATA message, cancels time out checking, temporarily stores data until all chunks are received, updates a missing public list, private list, public set, and private set. Additionally, if there is still content in waiting list, it will invoke sendUnicastREQ function again.

```
handleDATA(message){
    if (nodeState == BUSY and content ∉ caches) then
        CancelTimeOut(timer)
        Content ID = message.getContentID()
        peerAddress = message.getSource()
        sequence number = message.getSeq()
        total part = message.getPart()
        If (sequence number ≠ total part) then
            schedule(timer, timeout period)
        Else
            If (content ∈ missing private list) then
                private cache.insert(content)
                available private list.insert(content ID)
                missing private list.delete(content ID)
            Else public cache.insert(content)
                available public list.insert(content ID)
                missing public list.delete(content ID)
            End if
            interested list.delete(content ID)
            If (waiting list ≠ empty) then
                requested content = the first content in the waiting list
                sendUnicastREQ(requested content, peerAddress)
                waiting list.delete(requested content)
            else nodeState == IDLE
            End if
        End if
    End if
}
```

To support public cache options as mention in 3.2, 3.3, and 3.4, we modify some main functions as follows.

5.1.2. Relay on Demand Function

We modify the function sendBroadcastREQ() by adding a command in the code, as shown in italics. A node will clear a missing public list every time after it broadcasts a message but the missing public list is updated in handleBroadcast() function as it is.

Function 9. Modification in sendBroadcastREQ function

```
1:  sendBroadcastREQ(){
2:      create broadcast message
3:      message.insert(interested list)
4:      periodically send the message out every second.
5:      missing public list.clear()
6:  }
```

5.1.3. Hop-limit Function

In order to implement the hop-limit relay request, we add commands to sendBroadcastREQ() and handleBroadcastREQ() functions. Before a subscriber node sends broadcast messages, it will insert the list based on types of the content (private content or public content). When a neighbor node receives broadcastREQ message and the content is not available in its caches, it will check the type of contents in a waiting list whether they are considered as private contents or public contents from the perspective of the node that sends the message. If it is considered as a private content, the node will add the content ID in its missing public list. On the other hand, the node will discard the request for that content if it is public content and it will not help to search from other nodes. The following functions are a modification in handleBroadcastREQ() to support the hop-limit option.

Function 10. Modification in sendBroadcastREQ function

```
sendBroadcastREQ (){
    create broadcast message
    Message-> private(missing private list)
    Message-> public (missing public list)
    periodically send the message out every second.
}
```

Function 11. Modification in handleBroadcastREQ function

```
handleBroadcastREQ(message){
    incoming list = message.getList()
    peerAddress = message.getSource()
    For each content item ∈ incoming list do
        if (caches.find(content)==TRUE) then
            temporary list.insert(content ID)
        else
            if (content.type() == private) then
                missing public list.insert(content ID)
                interested list.insert(content ID)
            else discard it
            end if
        end if
    end for
    if (temporary list ≠ empty) then
        sendUnicastReply(temporary list, peerAddress);
    end if
}
```

5.1.4. Greedy Function

There are two functions that must be modified to support the greedy function. They are handleUnicastReply() and handleData().

Function 12. Modification in handleUnicastReply function

```
handleUnicastReply(message){
    incoming list = message.getlist()
    peerAddress = message.getSource()
    For each content item ∈ incoming list do
    If (content ID ∈ missing private list) then
        temporary list. append(content ID) in the beginning
    else if (content ID ∈ missing public list) then
        temporary list. append(content ID) in the end
    end if
    If (nodeState == IDLE) then
        nodeState = BUSY
        waiting list = temporary list
        requested content = the first content in waiting list
        sendUnicastREQ(requested content, peerAddress)
        waiting list.delete(requested content)
    Else if (requested content ∈ missing public list and the first item in temporary list ∈ missing
    private list) then
        waiting list. clear()
        waiting list = temporary list
        requested content = the first content in waiting list
        sendUnicast REQ(requested content, peerAddress)
    End if
}
```

Function 13. Modification in handleDATA function

```
handleDATA(){
if (nodeState == BUSY and content ∉ both caches) then
    If (data message is from a new node that shares a new private content) then
        CancelTimeOut(timer)
        Content ID = message.getContentID()
        peerAddress = message.getSource()
        sequence number = message.getSeq()
        total part = message.getPart()
        If (sequence number ≠ total part) then
            schedule(timer, timeout period)
        Else
            If (content ∈ missing private list) then
                private cache.insert(content)
                available private list.insert(content ID)
                missing private list.delete(content ID)
            Else public cache.insert(content)
                available public list.insert(content ID)
                missing public list.delete(content ID)
            End if
            interested list.delete(content ID)
            If (waiting list ≠ empty) then
                requested content = the first content in waiting list
                sendUnicastREQ(requested content, peerAddress)
                waiting list.delete(requested content)
            else nodeState == IDLE
            End if
        End if
    Else if (data message is from the node that we downloaded the last public content item) then
        CancelTimeOut(timer)
        Content ID = message.getContentID()
        peerAddress = message.getSource()
        sequence number = message.getSeq()
        total part = message.getPart()
        If (sequence number ≠ total part) then
            schedule(timer, timeout period)
        Else
            If (content ∈ missing private list) then
                private cache.insert(content)
                available private list.insert(content ID)
                missing private list.delete(content ID)
            Else public cache.insert(content)
                available public list.insert(content ID)
                missing public list.delete(content ID)
            End if
            interested list.delete(content ID)
        End if
    End if
End if
```

5.2. Mobility Model

Our mobility model is created by Legion Studio [26] which is the simulation software that simulates behavior of pedestrians in urban areas e.g. subway stations, shopping malls [27]. It records users' movement steps and reflects a real-life situation of the pedestrians. For instance, users will not walk too closely to each other to maintain personal space. Those who walk fast may have to decrease their speed when they are walking behind slow pedestrians where there is a bottleneck; however, if there is enough space for them, the fast people will speed up and pass the slower people. In the simulation, each pedestrian is displayed as a circular form with a size of approximately an actual pedestrian.

5.2.1. Subway Station Mobility Model

Our simulation uses a mobility model called Subway Station [27] as shown in Figure 16. It is a two-level indoor subway station that represents a densely populated network. The mobility is an open system, which means that users can come and leave the station at any time. People arrive or depart the station by entrance points on the top level floor or by trains from the platforms. The active area of the station is about 1921 m^2.

The details of the subway scenario are described as follows. People first come into the subway station from the entrance on the top floor of the station as shown in the right side of Figure 16. Before they can go to a subway platform, they must pass a checking point as shown in the bottom of Figure 16. At the checking point, they must first scan their valid metro card at the gate. If the card is valid, a door will open and they can walk down to the platform. The subway platform is located in the left part of Figure 16 and there are two tracks. When people arrive at this area, they will wait for the next train to come. After that, when the train arrives, people inside the train wagons will depart from the train in batches and walk out to the exit. The people who have been waiting for the train will enter the train wagons and finally the train leaves the station.

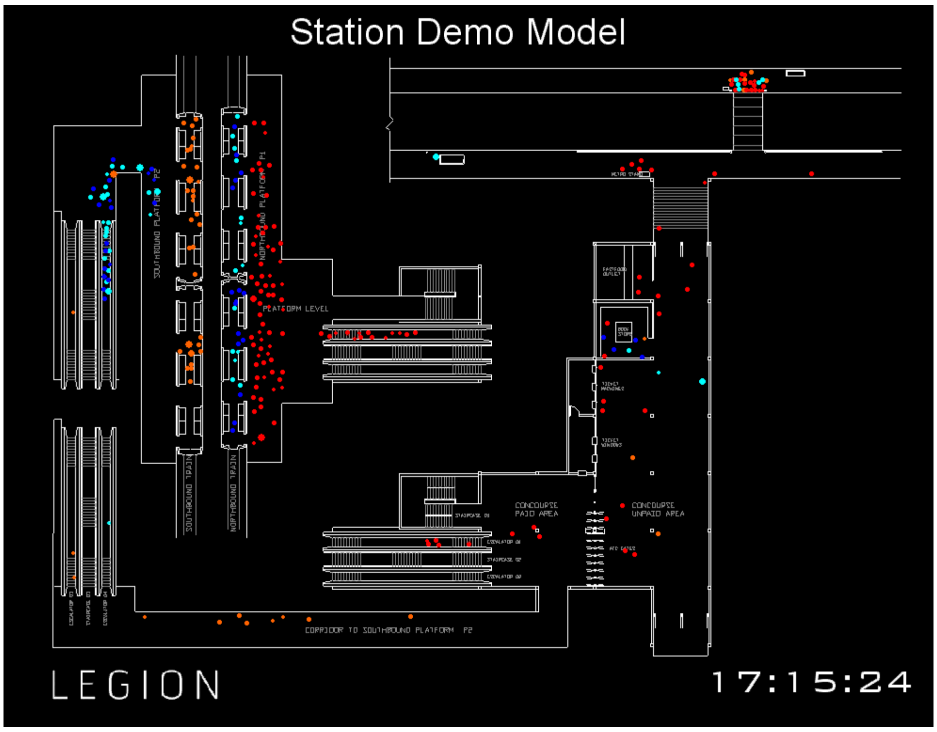

Figure 16. Subway station.

5.2.2. Östermalm Mobility Model

The mobility called Östermalm is presented in Figure 17. It is a downtown area in Stockholm consisting of a grid of intereconnected street [27]. The length of each street varies between 20 m and 200 m and the width is approximately 2 m. We assume that the arrival rates (λ) of all streets are equal and when the nodes arrive at an intersection, they will go straight on the same street with probability 0.5 or turn to other adjoining streets with equal probability. It is also an open system and the active area is approximately 5872 m^2.

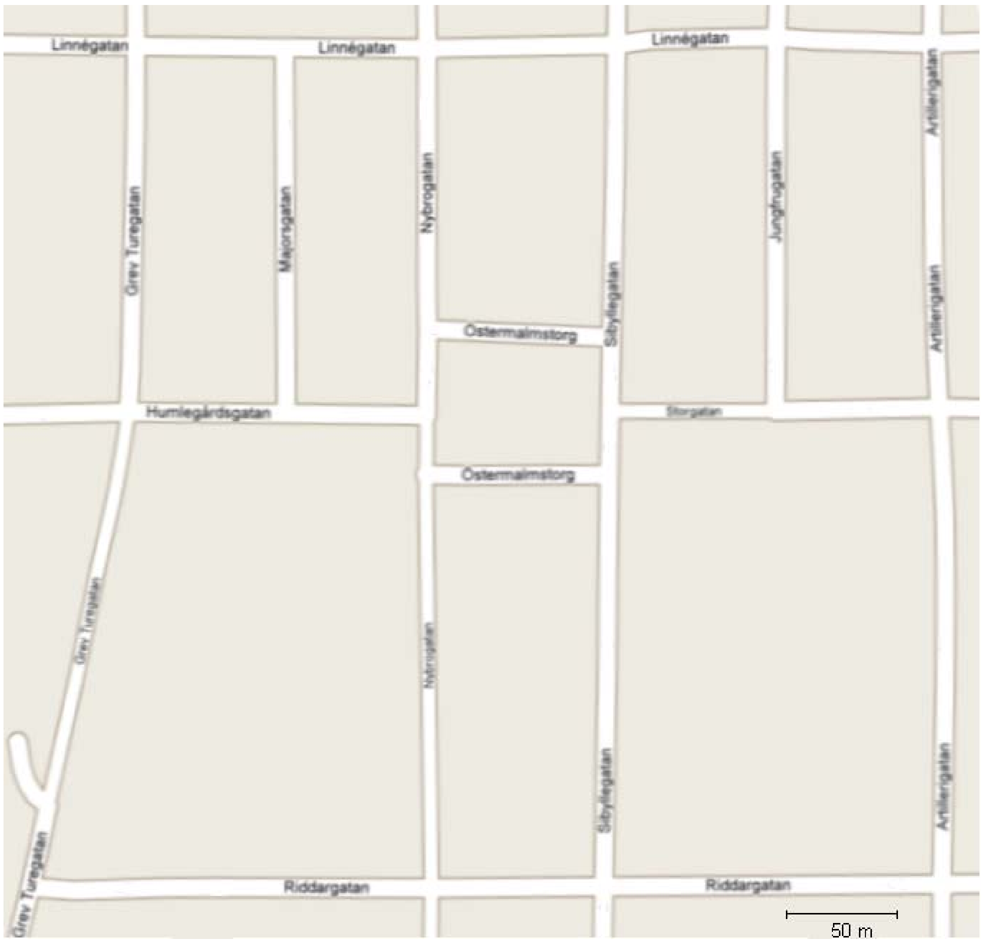

Figure 17. A part of downtown Stockholm.

6. Experiments

We hereafter describe the default setting of our experiment scenarios. We configure a node by setting its parameters according to Table 1, unless otherwise stated. The maximum communication range of each node is 10 meters. Each node periodically broadcasts a request list every second and it will terminate a connection with a current neighbor node if the neighbor node does not respond after 60 seconds. The node subscribes to contents on only 10 feeds from a total of 1000 feeds which are selected based on a Zipf distribution. Once again, based on Zipf, it selects 5 out of 10 subscribed contents to be private items that it contains at the beginning.

Simulation parameters	**Values**
Periodic broadcast of a request	every 1 sec
Timeout threshold	60 sec
Communication range	10 m
Total contents available in the network	1000 items
Numbers of contents per feed	1 content/feed
Numbers of subscribed feeds	10
Initial number of feeds	5
Feed popularity	Zipf(alpa=0.368)
Content size	Normal(3kB), stddev= 1 kB
Network	Subway Station
Speed	Truncated Normal(1.3m/s)
Simulation period	1 hour

Table 1. Simulation parameters.

Regarding the subway station mobility model, the speed of users is a truncated normal distribution with mean 1.3 m/s. The minimum speed and maximum speed are limited to 0.6 m/s and 2.0 m/s respectively. We run the simulation for 1 hour. As it is an open system, the number of nodes staying in the station varies according to the graph in Figure 18. During some period, the number of nodes increases suddenly. Sometimes it drops immediately. This is because when the train comes, people (or nodes) in the train who arrive at the station get out of the train immediately in batches. Thus, the number of nodes in this period significantly increases. After that they walk from a train platform, pass the gate and exit the station. Moreover, the nodes that are waiting for the train in the platform also take the train and leave the network. Thus, it can be seen from the graph that the number of users drops at a moment later.

Regarding Östermalm mobility model, we set arrival rate (λ) 0.05 to all nodes. The communication range is 20 m and simulation time is 2 hours. Other parameters are configured according to Table 1.

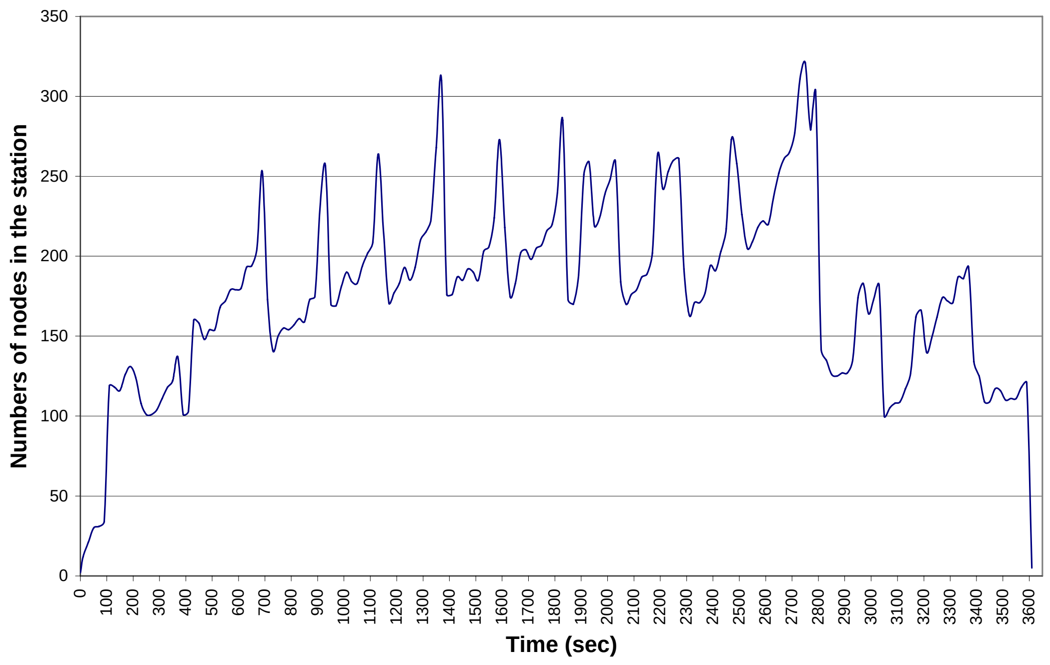

Figure 18. Number of nodes in the station.

6.1. Relay Request Strategy in Subway Station

Scenario 1: No caching

In this scenario, nodes use only a private cache to store and share contents. The parameters are configured as indicated in Table 1. The process is that a node finds and shares only the contents that are of its interest. If a node receives a request for a content item that is not of its interest, the node will ignore that request. In summary, the dissemination process is driven only by the users' interest.

Scenario 2: Public cache with one two-hop relay request

A node is configured to use both private and public caches. The private cache stores private contents whereas the public cache stores contents on behalf of others. When a node receives a request for content that is not available in its caches, it will help fetch the content from other nodes even though the content is not of its interest. In addition to the public cache function, we adopt the one-hop limit strategy for the search of public contents, which means a searching process for public contents is limited to two hops (one relay node) as mentioned in section 3.3. Other parameters are configured according to Table 1.

Scenario 3: Public cache with two-hop limit and greedy relay request

We adopt the concept of public cache with two-hop limit as a base scenario similar to scenario 2. In this scenario, we also add another option in this scenario which is the greedy relay request, which means that whenever a node can fetch private contents, it stops downloading the next public content. The detail of this concept is described in section 3.4.

Scenario 4: Public cache with three-hop limit

The design is similar to scenario 2; however, we change the hop limit from two hops to three hops.

Scenario 5: Public cache with three-hop limit and greedy relay request

We set the parameters in this scenario the same as scenario 3 except for the hop limit. We extend the maximum hop for searching public contents from two hops to three hops.

Table 2 presents the summary of public cache options for all scenarios.

scenario	Public Cache Options		
	Relay on demand	**Hop limit**	**Greedy**
scenario 1	-	-	-
scenario 2	Yes	2 hops	No
scenario 3	Yes	2 hops	Yes
scenario 4	Yes	3 hops	No
scenario 5	Yes	3 hops	Yes

Table 2. Public cache option configuration.

6.2. Optimal Channel Choice Strategy in a Subway Station

We adopt the idea about content caching proposed in [19]. Nodes periodically broadcast a list of contents that exist in their nodes. When neighbor nodes receive the broadcast message, they will get the list of contents that are shared. If the node finds contents of interest, it will send a request to get data. Moreover, the node will fetch contents that are not of its interest and will store them in public cache. The strategy for selecting public contents to store in the public cache is presented in algorithm 2 of paper [19]. The node will select a public content item from its public cache (*content i*) and a new public content item from its neighbor node (*content j*). Then, it computes the probability which is q parameter stated in [19]. If min(1,q) > Uniform [0,1], it will store the new public content and drop the old content that it has selected from its public cache. However, we have modified the calculation of q for our simulation since the authors in [19] adopt Gibbs distribution instead of Zipf distribution.

$$q = \frac{the\ total\ numbers\ of\ nodes\ in\ the\ system\ that\ store\ content\ \ i}{the\ total\ numbers\ of\ nodes\ in\ the\ system\ that\ store\ content\ \ j}$$

We run a simulation with the subway station mobility to evaluate this strategy and set the default parameters according to Table 1. We vary sizes of public cache from 0 to 5 and 10 contents. The results are presented in section 7.

6.3. Relay Request Strategy in Östermalm

In this experiment, we change the mobility model from a subway station to Östermalm. We run the simulation for 5 scenarios the same as 6.1: No caching, public cache with one two-hop relay request, public cache with one three-hop relay request, public cache with one two-hop relay request and greedy relay request, and public cache with one three-hop relay request and greedy relay request.

6.4. Optimal Channel Choice Strategy in Östermalm

This experiment is the same as 6.2 but the Östermalm mobility model is adopted instead of a subway station. We run the simulation in 7 scenarios and the sizes of public caches is 0, 1, 2, 3, 4, 5, and 10.

7. Results and Evaluation

In this section, we present the results from the simulation in section 6. We compare the results from scenarios in terms of goodput, which we define as follows:

$$Private\ Goodput\ (G_{pri}) = \frac{1}{Node\ life\ time\ in\ simulation} \times \sum_{\substack{all\ downloaded \\ private\ contents \\ in\ the\ node}} size\ of\ each\ content$$

$$Public\ Goodput\ (G_{pub}) = \frac{1}{Node\ life\ time\ in\ simulation} \times \sum_{\substack{all\ downloaded \\ public\ contents \\ in\ the\ node}} size\ of\ each\ content$$

$$Average\ Private\ Goodput\ (\overline{G_{pri}}) = average\ value\ of\ Private\ Goodput\ over\ all\ nodes$$

$$Average\ Public\ Goodput\ (\overline{G_{pub}}) = average\ value\ of\ Public\ Goodput\ over\ all\ nodes$$

$$Total\ Goodput\ = \overline{G_{pub}} + \overline{G_{pub}}$$

7.1. Results from Subway Station Mobility

A node subscribes to 10 feeds and it initially contains only 5 contents. As a result, it tries to download 5 more contents from other nodes in the network in an opportunistic way. First, we take into account the private contents that are downloaded. We compute the downloading rate of these private content items (G_{pri}). The average value, average private goodput ($\overline{G_{pri}}$), is calculated from all nodes in the network. The results with 95% confidential interval from relay request scenarios are shown in Figure 19. The graphs show that average private goodput ($\overline{G_{pri}}$) in scenario 1 (no caching) is 488 bit/sec. However, in scenario 2 and 3, $\overline{G_{pri}}$ drops to 458 bit/sec and 444 bit/sec respectively. This shows that when a node generously helps find public contents, it results in worse $\overline{G_{pri}}$ than no cahcing. This is due to the fact that a node is allowed to serve only one connection at a time. When the node receives a unicast reply in response to a broadcastREQ showing all content items that a peer node can share, the node will establish a connection and will download items from that peer until all items in a waiting list are completely downloaded or a time out occurs. However, the node loses the chance to contact other peer nodes even if the new peer can provide data for its private contents. An example in this case refers to Figure 13. This means User C sends unicast reply to User A while User A downloads public contents form User B. However, User A ignores to download its own private content from User C because all contents from User B indicating in the waiting list are not completely downloaded. This means downloading many public contents may block a private content transmission. In other words, when a node is very generous, it tries to download the contents that are not of its interest on behalf of others. As a result, it loses the opportunity to get the content which is of interest. Additionally, the other reason why $\overline{G_{pri}}$ becomes lower is that the public content which the node has completely fetched from its peer may not be shared to the node that requests it in the first place. It is probable that the subscriber node broadcasts a request and there are two neighbors in communication range. The first neighbor node has this content in its cache while the second neighbor node does not, so the node that does not contain the content will consider it as a public content item and will help the subscriber node find this content. Nonetheless, the subscriber node can get the content from the first peer node. Even if the second node can find the content and completely fetch it, the subscriber node will not ask for this content again because it has already got the content from the first node. Therefore, this is an unnecessary caching process because the node loses a chance to find the content of its own interest and the public content is not distributed as it should have been.

The results in scenario 4 and 5 show that the greedy relay request enhances the $\overline{G_{pri}}$, which means that more contents of interest are disseminated in the network. The greedy process allows nodes to download public contents only when it is free from downloading private contents. This process is similar to no caching because a node focuses on downloading private contents but, in this case, it also helps store public data if there is no more private content that can be downloaded. Furthermore, the graphs indicate that $\overline{G_{pri}}$ in 3-hop limit is higher than $\overline{G_{pri}}$ in 2-hop limit. The reason is the scope of finding the content is wider than only 2-hop, so more contents can be found. However, there is an impact of using caching strategy that the network experiences shown in Figure 20.

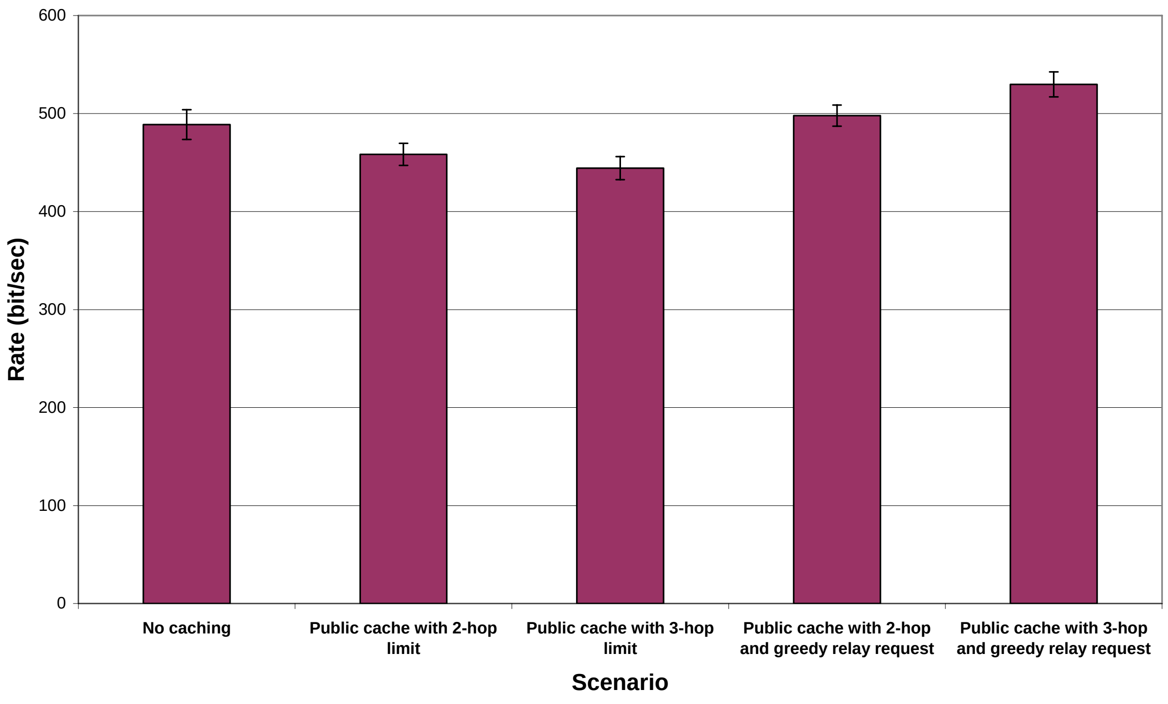

Figure 19. $\overline{G_{pri}}$ in relay request strategy (subway station).

It can be seen from Figure 20 that $\overline{G_{pub}}$ is considerably greater than $\overline{G_{pri}}$. This means there are a lot of public contents distributed in the network but they make quite small contributions to the content of interest. In other words, this can be considered as a traffic overhead, which is the unnecessary traffic that is sent in the network. The caching mechanism has an advantage in enhancing the performance of data dissemination but it greatly increases traffic in the network.

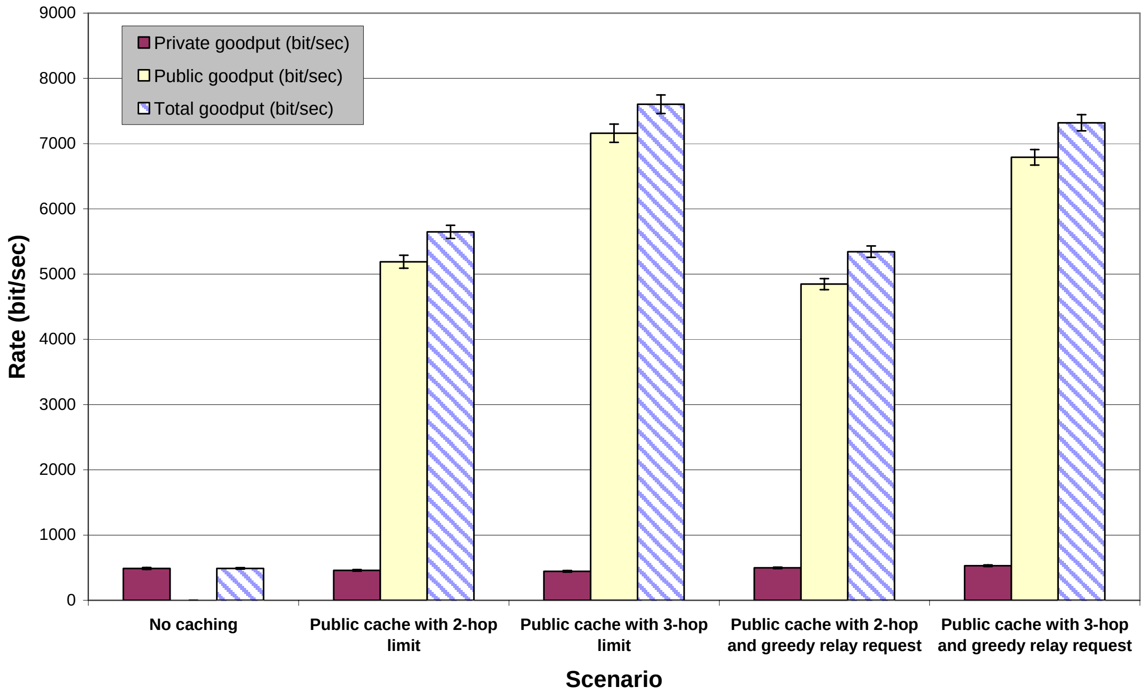

Figure 20. $\overline{G_{pri}}$, $\overline{G_{pub}}$ and total goodput in relay request strategy (subway station).

Data	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
Private goodput (bit/sec)	488	458	444	498	530
Public goodput (bit/sec)	0	5190	7159	4847	6790
Total goodput (bit/sec)	488	5648	8447	5344	8168
Public goodput/ Private goodput	0	11.33	16.12	9.73	12.81

Table 3. The detail of goodput in all relay request scenarios (subway station).

The results from the optimal channel choice strategy are presented in Figure 21 and Figure 22. It can be seen that when nodes increase the size of public cache, they get higher average private goodput. However, private goodput only slightly increases while average public goodput increases considerably, which is similar to the result from the relay request strategy.

Nevertheless, the public rate in optimal channel choice strategy is lower than the one in relay request strategy because of the limitation in the size of the public cache.

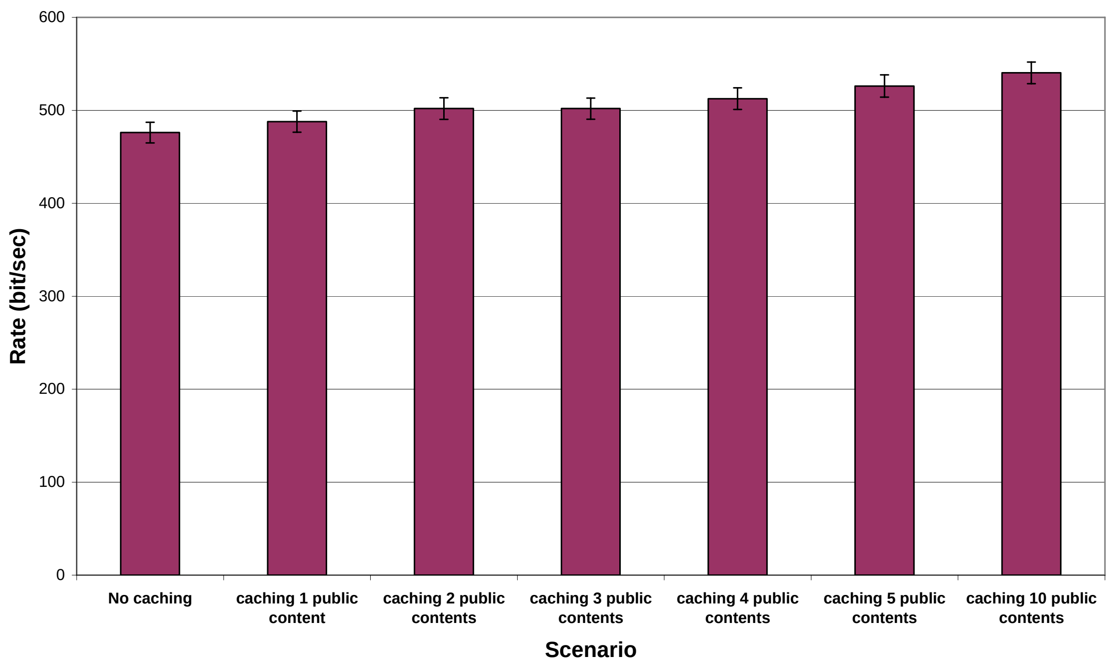

Figure 21. $\overline{G_{pri}}$ for the optimal channel choice strategy (subway station).

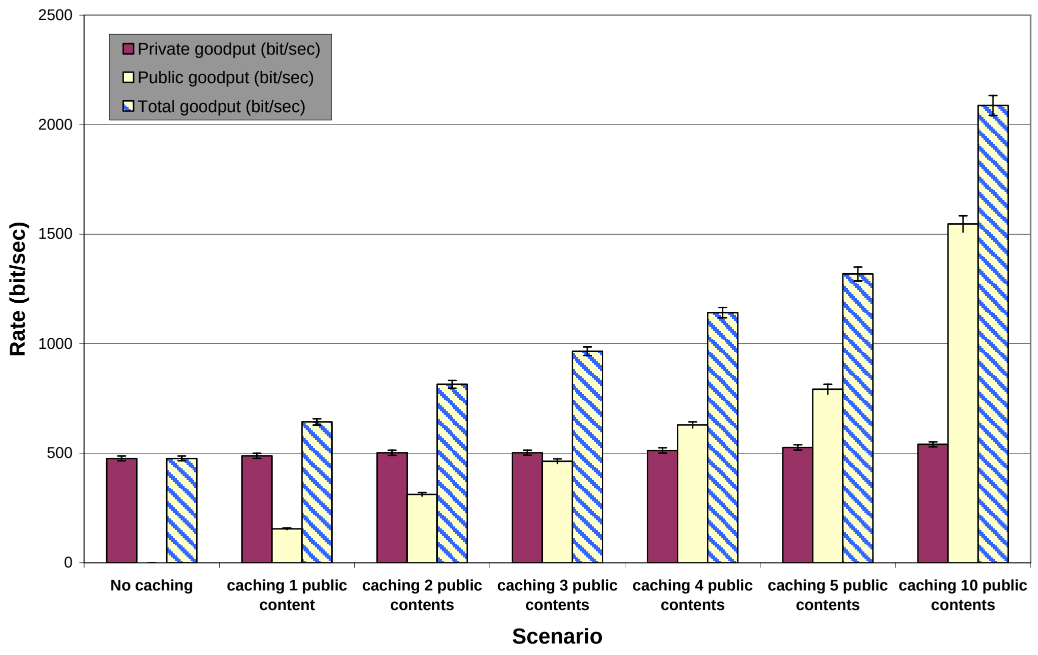

Figure 22. $\overline{G_{pri}}$, $\overline{G_{pub}}$and total goodput for the optimal channel choice strategy (subway station).

Data	No caching	1 public content	2 public contents	3 public contents	4 public contents	5 public contents	10 public contents
Private goodput (bit/sec)	476	488	502	501	512	526	540
Public goodput (bit/sec)	0	155	312	464	629	792	1547
Total goodput (bit/sec)	476	643	814	965	1141	1318	2087
Public goodput/ Private goodput	0	0.32	0.62	0.92	1.23	1.50	2.86

Table 4. The detail of goodput for the optimal channel choice strategy (subway station).

We evaluate advantages and disadvantages of both strategies. The advantage in relay request strategy is that neighbor nodes will fetch and store contents that a subscriber node requests, which means it wants to get those missing contents while in optimal channel choice, nodes store contents based on the prediction of q parameter. The content that they store may not disseminate at all since there is not any node requesting it. We compare the scenario that both of them use no caching as size of public cache is the same (0 public content caching). The results from Table 3 and Table 4 show that $\overline{G_{pri}}$ in relay request strategy is 488 bit/sec which $\overline{G_{pri}}$ in optimal channel choice strategy is only 476 bit/sec.

Nonetheless, nodes in relay request strategy excessively download the content. This can be seen from the ratio of $\overline{G_{pri}}$ and $\overline{G_{pub}}$ in Table 3 and Table 4 that in relay request strategy the maximum value is 16.12 while in optimal channel choice strategy is only 2.86. This means that nodes that adopt the relay request strategy have to waste more time, energy and resource to download contents that are not of their interest than the nodes that use optimal channel choice strategy.

7.2. Results from Östermalm Mobility

The results from the scenarios adopting Östermalm mobility are presented in Figure 23 - Figure 26. The graphs in Figure 23 and Figure 24 show that public caching with relay request scenario can slightly increase $\overline{G_{pri}}$ but results in significant growth of $\overline{G_{pub}}$. Public cache with three-hop limit with greedy relay request gives the highest $\overline{G_{pri}}$ amoung 5 scenarios, which is similar to the results from the subway station mobility. The results from optimal channel choice strategy in Figure 25 and Figure 26 also confirm that the bigger size of public cache results in the higher $\overline{G_{pri}}$. However, it increases only slightly while $\overline{G_{pri}}$ increases considerably. This is again similar to the result in 7.1.

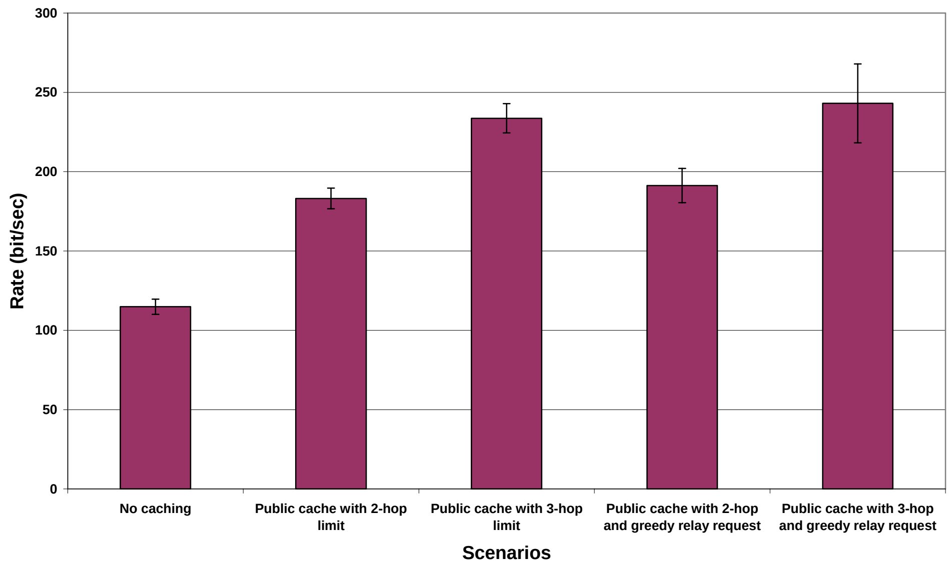

Figure 23. $\overline{G_{pri}}$ in relay request strategy (Östermalm).

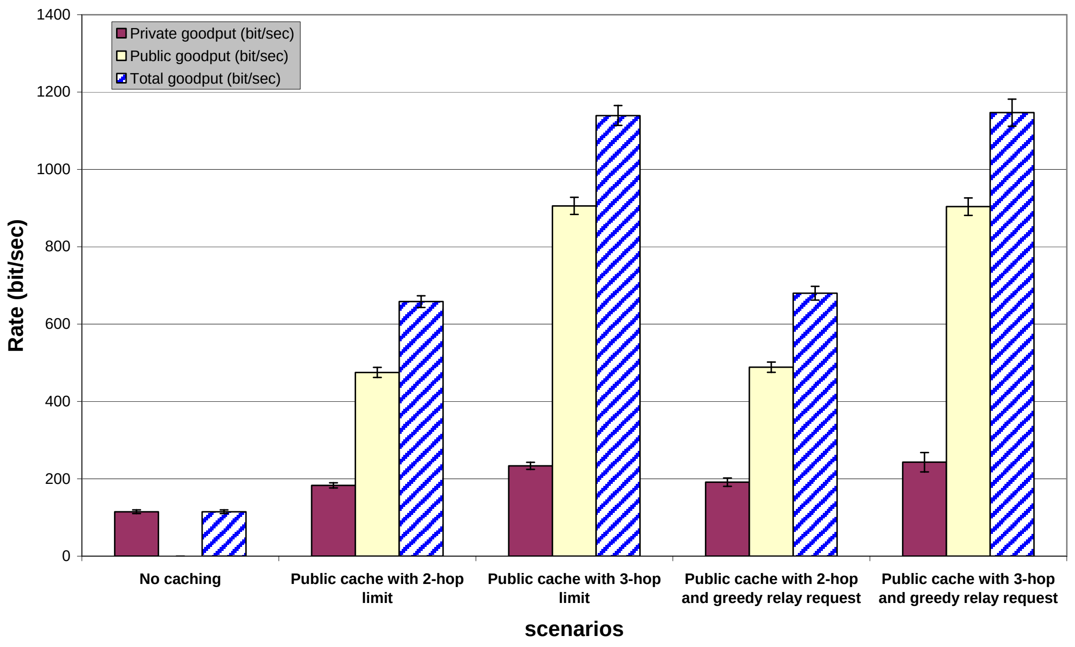

Figure 24. $\overline{G_{pri}}$, $\overline{G_{pub}}$ and total goodput in relay request strategy (Östermalm).

Data	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
Private goodput (bit/sec)	114	183	233	191	243
Public goodput (bit/sec)	0	475	905	488	903
Total goodput (bit/sec)	114	658	1139	679	1146
Public goodput/ Private goodput	0	2.59	3.88	2.55	3.72

Table 5. The detail of goodput in all scenarios (Östermalm).

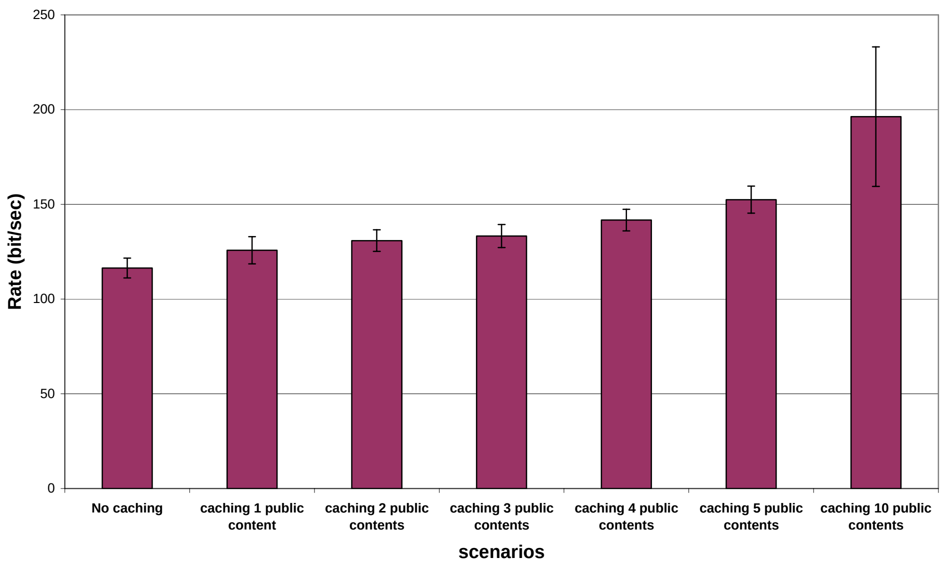

Figure 25. $\overline{G_{pri}}$ for the optimal channel choice strategy (Östermalm).

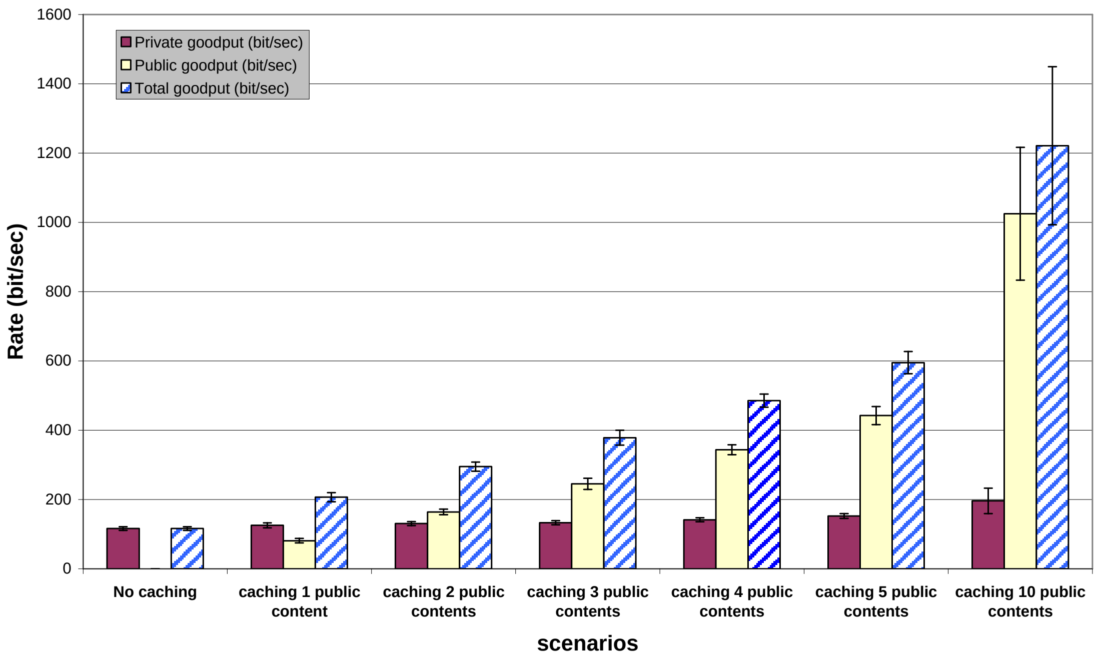

Figure 26. $\overline{G_{pri}}$, $\overline{G_{pub}}$ and total goodput for the optimal channel choice strategy (Östermalm).

Data	No caching	1 public content	2 public contents	3 public contents	4 public contents	5 public contents	10 public contents
Private goodput (bit/sec)	116	125	130	133	141	152	196
Public goodput (bit/sec)	0	81	164	245	343	442	1024
Total goodput (bit/sec)	116	206	295	378	485	595	1221
Public goodput/ Private goodput	0	0.65	1.26	1.84	2.42	2.90	5.22

Table 6. The detail of goodput for the optimal channel choice strategy (Östermalm).

8. Conclusions and Future work

In this thesis, we have studied content distribution in wireless peer-to-peer networks. Contents are delivered in an ad-hoc mode. Two nodes will exchange data only when they are in each other's communication range. Moreover, the contents will be disseminated in an opportunistic way.

We propose the concept of public cache for enhancing data dissemination performance. We believe that the huge capacity of mobile nodes could be beneficially exploited. Fortunately, thanks to a rapid pace of development in mobile devices, the storage capacity increases considerably. Current mobile devices have a huge storage size; for instance, a laptop has up to 1 TB, iPod has up to 160 GB, and a smart phone up to 64 GB [28]. As not all spaces are fully used, it is beneficial to utilize the left over capacity to store some more data in a public cache, which also does not exacerbate the performance of the device. Nodes will generously help store contents that are not of their own interest but on behalf of other nodes in the public cache. We propose a protocol for exchanging messages and three options for supporting the use of public cache which are memoryless, hop limit, and greedy relay request.

The results from the simulation in a subway station mobility and the Östermalm mobility show that this caching mechanism gives slightly better performance as can be seen from having higher private goodputs. However, the cost of this strategy is higher capacity consumption in mobile devices and higher overheads. The public goodput can be perceived as an overhead since they are extra loads but do not contribute to private interest of nodes. This indicates that the device consumes much resource for the contents that are not of its own interest.

It can be concluded that the public caching mechanisms increase the overhead significantly but can increase the performance only marginally.

Future work

Multi-chunks downloading

In a network where nodes have high mobility, contact duration may be very short. If the size of the content is big, it probably cannot finish downloading all chunks within a short contact duration. In this case, the node will declare that the content is unsuccessfully downloaded and it deletes the chunks that it has recently received. Thus, it is possible for the node to support a discontinuous content downloading process. The node may keep searching for the missing chunks of the content instead of starting to download again from scratch.

Dynamic Subscribers System

Many contents are shared in the network. A user may initially have groups of contents that are of his interest. However, he may get requests for the same content from many nodes in his vicinity. This may cause a significant effect of the user's interest. A person is interacting with other people, particularly with members in the same group. The group often points out and convinces all members in the group to go in the same direction. In this case, when many nodes want the same content and the user knows it, he probably changes his mind and becomes interested in and eventually subscribes to the same feeds as other nodes in the social group.

Recommendation System

It is quite difficult for the less popular contents to be distributed in the network. Probably only a couple of nodes subscribe to them. We may be able to disseminate them by a recommendation system. A user who carries this content may recommend other nodes to subscribe to this content. Some of the nodes that receive the recommendation may change their interest in this content and download it from the node that sends the recommendation. This may help less popular contents to become widespread in the network.

9. References

1 Stallings, William. *Wireless Communications and Networks*. Prentice Hall, 2005.

2 Ericsson. *Mobile Data Traffic Surpasses Voice*. Press releases. 2010, http://www.ericsson.com/thecompany/press/releases/2010/03/1396928.

3 Krishna Nadiminti, Marcos Dias de Assunção, and Rajkumar Buyya. *Distributed Systems and Recent Innovations: Challenges and Benefits*. Department of Computer Science and Software Engineering,The University of Melbourne, Australia, 2006.

4 Rajendra V. Boppana, and Satyadeva P Konduru. *An adaptive distance vector routing algorithm for mobile, ad hoc networks*. In Proceedings of IEEE INFOCOM, Anchorage, AK , USA, 2001.

5 Charles E. Perkins, and Elizabeth M. Royer. *Ad-hoc On-Demand Distance Vector Routing*. Second IEEE Workshop on Mobile Computer Systems and Applications, New Orleans, LA , USA , 1999.

6 Kaustubh S. Phanse, and Johan Nykvist. *Opportunistic wireless access networks*. In Proceedings of the 1st international conference on Access networks. 2006.

7 Bernhard Distl, Gergely Csucs, Sascha Trifunovic, Franck Legendre, and Carlos Anastasiades. *Extending the reach of online social networks to opportunistic networks with PodNet*. In Proceedings of the Second International Workshop on Mobile Opportunistic Networking. 2010.

8 Ólafur R. Helgason, *Opportunistic Content Distribution*. Licentiate Thesis in Telecommunications, KTH, Stockholm, 2010.

9 Behrouz A. Forouzan, *TCP/IP Protocol Suite*. McGraw-Hill. 2009.

10 Vincent Lenders, Gunnar Karlsson, and Martin May. *Wireless Ad Hoc Podcasting*. In Proceedings of IEEE SECON, San Diego, CA, June 2007.

11 Martin May, Vincent Lenders, Gunnar Karlsson, and Clemens Wacha. *Wireless opportunistic podcasting: implementation and design tradeoffs*. In Proceedings of the second ACM workshop on Challenged networks. 2007.

12 Matthias Grossglauser and David N. C. Tse. *Mobility increases the capacity of ad hoc wireless networks*. IEEE/ACM Transactions on Networking (TON). 2002.

13 PodNet. http://podnet.ee.ethz.ch/. Last visited [14 June 2011].

14 Ólafur R. Helgason, Emre A. Yavuz, Sylvia T. Kouyoumdjieva, Ljubica Pajevic, and Gunnar Karlsson. *A mobile peer-to-peer system for opportunistic content-centric networking*. In proceedings of the second ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds. 2010.

15 Martin May, Gunnar Karlsson, Ólafur R. Helgason, and Vincent Lenders. *A System Architecture for Delay-Tolerant Content Distribution*. In Proceedings of IEEE Conference on Wireless Rural and Emergency Communications (WreCom), Rome, Italy, October, 2007.

16 Eiko Yoneki, Pan Hui, ShuYan Chan, and Jon Crowcroft. *A Socio-Aware Overlay for Publish/Subscribe Communication in Delay Tolerant Networks*. In Proceedings of the 10th ACM Symposium on Modeling, analysis, and simulation of wireless and mobile systems, 2007.

17 Chiara Boldrini, Marco Conti, and Andrea Passarella. *ContentPlace: Social-aware Data Dissemination in Opportunistic Networks*. In Proceedings of the 11th international symposium on Modeling, analysis and simulation of wireless and mobile systems, 2008.

18 Yaozhou Ma, M. Rubaiyat Kibria, and Abbas Jamalipour. *Cache-based Content Delivery in Opportunistic Mobile Ad Hoc Networks*. Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE , New Orleans, LO , 2008.

19 Liang Hu, Jean-Yves Le Boudec, and Milan Vojnovic. *Optimal Channel Choice for Collaborative Ad-Hoc Dissemination*. In Proceedings of IEEE Infocom, San Diego, CA, 2010.

20 Paolo Costa, Cecilia Mascolo, Mirco Musolesi and Gian Pietro Picco. *Socially-aware Routing for Publish-Subscribe in Delay-tolerant Mobile Ad Hoc Networks*. IEEE Journal on Selected Areas in Communications. 2008.

21 Pan Hui, Jon Crowcroft, and Eiko Yoneki. *BUBBLE Rap: Social-based Forwarding in Delay Tolerant Networks*. In Proceedings of the 9th ACM international symposium on Mobile ad hoc networking and computing, 2008.

22 Cisco. *Traffic-Storm Control*. http://www.cisco.com/en/US/docs/switches/lan/catalyst 6500/ios/12.2SXF/native/configuration/guide/storm.html. Last visited [14 June 2011]

23 MiXiM. http://mixim.sourceforge.net/developers.html. Last visited [14 June 2011]

24 Ólafur Ragnar Helgason, and Kristján Valur Jónsson. *Opportunistic networking in OMNeT++*. In Proceedings of the 1st international conference on Simulation tools and

techniques for communications, networks and systems & workshops. 2008.

25 OMNET++. http://www.omnetpp.org/. Last visited [14 June 2011]

26 Legion. *http://www.legion.com/legion-studio*. Last visited [14 June 2011]

27 Ólafur R. Helgason, Sylvia T. Kouyoumdjieva and Gunnar Karlsson. *Does Mobility Matter?* In Proceedings of seventh International Conference on Wireless On-demand Network Systems and Services (WONS), 2010.

28 Apple. *iPod*. http://www.apple.com/ipod/. Last visited [14 June 2011]