AN AGENT-BASED CONTENT-CENTRIC NETWORKING APPLICATION FOR DATA RETRIEVAL

Master Thesis

presented by

Wafaa El Maudni El Alami
University of Neuchatel
2013

Supervisor:
Professor Dr. Torsten Braun
Faculty of Science
University of Bern

Contents

List of Figures

List of Tables

Acknowledgment

First of all I thanks my advisor Carlos Anastasiades for sacrifice his valuable time, for his guidance, his detailed and patient explanations and for their comments that helped me to improve this thesis. Prof. Dr. Torsten Braun for giving me the opportunity to realize this thesis in his *Computer Networks and Distributed Systems* group at the University of Bern and for his helpful comments.

Then, my biggest thanks go to my family for supporting me in many different ways. Finally, I would like to thank all my friends that did not stop to encourage me.

Abstract

Content-Centric Networks (CCN) is a new networking paradigm, where messages are routed based on names instead of host identifiers. Therefore, CCN is advantageous in mobile networks because communication can be performed independently of changing communication partners. However, in opportunistic networks, content retrieval is challenging because contacts between users and their mobile devices are intermittent and not predictable. If the requester is not connected to the content source, the requester cannot retrieve the content.

In this thesis an application for agent-based content centric data retrieval was implemented. This approach gives to the requester the opportunity to delegate the content retrieval to other nodes, i.e., agents. Mobile agents store the retrieved content locally in their mobile repository, which is synchronized with their home repository (continuously connected to the Internet). Agents that retrieved the content can notify the original requesters. Then, the requester can retrieve the content via unicast communication from the mobile repository or from the home repository, if it has no direct connection to the agent. This approach helps also to increase the content density because the mobile agents replicate the content source by storing the retrieved content in their local repository.

The implemented mechanism is evaluated on Android smartphones. The evaluations showed that for large files the agent-based content retrieval is more efficient than the regular retrieval by forwarding the Interests over two hops (first hop in unicast communication and multicast communication in the second hop). We evaluated the probe interval, i.e. the time the requester waits before asking for a notification, in different scenarios with one and two agents. Evaluation results show that the number of notification requests with a probe interval of 30 seconds is around 80% smaller than with probe interval of 1 second. Evaluations with multiple agents showed that multiple agents would probably decrease the content retrieval time. The more agents are looking for the content independently, the higher is the probability of finding the content. However, if the agents are looking for the content simultaneously, then the content retrieval takes more time because they need to share the bandwidth of the content source.

Chapter 1

Introduction

Nowadays, as the Internet has become a cloud for services, knowing the location of the source is irrelevant: the user is more interested in the content than in the source. In addition, proliferation of mobile devices is increasing. As a result, the host-to-host communication model is not convenient. Thus, the need of another flexible and suitable architecture arises. For this reason, a new approach, Content Centric Networking (CCN), has been developed.

Content Centric Networking is an alternative approach to host based communication as in traditional computer networks. The main principle of CCN is that the user may look for the data by content names rather than by host identifiers, and any node that has the corresponding data replies. This reduces the traffic, improves network performance and increases network reliability.

1.1 Motivation

In Content Centric Networking, a simple scenario is where the content source is in transmission range of the requester. Then CCN works in a best way: the request of the needed content is forwarded to the content source, which replies by transmitting the content. However, in delay tolerant networks, where connections between nodes are intermittent, or in small-world topologies, where not every node sees the content source, it makes sense to delegate the content retrieval to agent nodes, which may meet the content source. The more agents look for the content, the higher is the probability that the content can be found or retrieved assuming it is available.

1.2 Task Formulation

In order to increase the probability of finding the content, the requester can delegate the content retrieval to other nodes, i.e. agents, and receives a notification from an agent as soon as the content has been found. Such delegation requires an agreement between the requester and agent about the way of the delegation and notification.

The goal of this master thesis is to develop an Android Agent-Based Content-Centric Networking Application for Content Retrieval. The detailed tasks are as follows:

* Implement a handshake mechanism between requester and agent to delegate content retrieval to agents, based on CCNx code running on Android smart phones.

* Implement/configure a home repository (connected to the Internet), which synchronizes with mobile agent's repository on the smart phone based on the CCNx synchronization mechanism for repositories.

* Implement a notification mechanism to notify the requester as soon as the content is found

* Evaluate the whole mechanism on Android smart phones in different scenarios.

1.3 Contributions

An overview of the contributions of this thesis is provided in the following:

* We implemented an approach of delegating the content retrieval to others nodes, to enable the requester to receive content from content sources that are never directly met. In addition, we implemented the handler of a synchronization mechanism between the agent node and the agent proxy, which facilitates the retrieval by the requester if the agent is not directly reachable anymore by the requester.

* Our experiments showed that the implemented approach has higher throughput to retrieve large files than the retrieval using the standard ccngetfile application of CCNx over two hops of communication. Moreover, the implemented mechanism was evaluated with multiple agents. The results proved that an appropriate value of the agent Interval, i.e. time to delegate the content retrieval to a new agent, depends on the file size and on the intercontact time between the agents and the content source.

1.4 Outline

The thesis is structured as follows. In Chapter 2, the concepts of content-centric networking that are related to this work are presented. The architecture of Agent-Based Content Retrieval is described in Chapter 3. It explains in detail how the mechanism works at different sides: requester and agent. In addition, Chapter 4 presents the implementation of the entire mechanism. Then, in Chapter 5 the results of the evaluation in different scenarios are presented. Finally, Chapter 6 discusses the presented mechanism and concludes the thesis; it presents also possible future work.

Chapter 2

Related Work

2.1 Content Centric Networking

Content-Centric Networking was introduced by Van Jacobson[1]. The CCNx [2] project provides an open source implementation of the content-centric networking approach, developed at the Palo Alto Research Center (PARC).

2.1.1 Interest and Data

CCN communication is based on two message types: Interest and Data. Figure 2.1 shows the Interest packet and the Data packet. An Interest packet is used to request data. It contains a

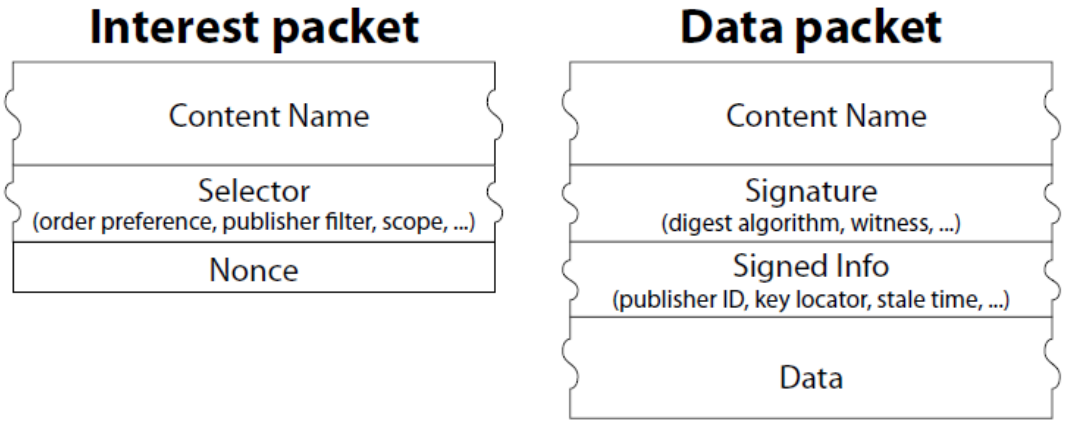


Figure 2.1: CCN packets types

prefix of a content name and some additional fields to limit the number of responders. The most important fields used in this thesis, which will be expained later in this section, are: AnswerOriginKind, Scope, Exclude and Interest lifetime.

A Data packet contains a ContentObject and is transmitted in response to the Interest message. It contains the ContentName, the Signature and the data. Every Interest can only retrieve or consume at most one Data packet.
A CCN node requests a content based on the longest-prefix match and the additional limiting Interest fields and forwards the Interest to an available face. A face is a generalization of the concept of interface. It may be a connection to a network or directly to an application party. The forwarding and the processing of CCN messages are performed by the CCN daemon (CCND).

2.1.2 CCN Forwarding Model

The forwarding model is based on three elements in CCN node: the content store (CS) as a cache, Forwarding Information Base (FIB), which contains a table of the outbound faces for specific Interest prefixes, and the PIT, which is a table of sources of unsatisfied Interests. When an Interest arrives, the lookup is performed on CS first, if a matching ContentObject satisfies all the specifications in the Interest, it will be transmitted as response to the requester. Otherwise, if there is match in the PIT, the arrival face will be added to the list of unsatisfied Interest and the Interest will be discarded. Otherwise, the FIB provides the face where to forward the Interest. If no prefix is found in the FIB, the Interest will be discarded. InterestLifetime indicates the time remaining before the Interest times out [4], the default value is 4 seconds. The Interest can be removed also from the PIT if the InterestLifetime is over.
The propagation of an Interest can be limited by the Interest field called Scope, e.g. scope 0 limits the propagation to the local ccnd, scope 1 to the application on the local host. The accepted response to a given Interest can be specified by two fields: AnswerOriginKind, e.g value 0 rejects any responses from the content store, value 1 uses responses from the cache. The field Exclude is an element that embodies a description of name components that should not appear as a continuation of the Name prefix in the response to the Interest [4].

2.1.3 Data Storage

There are two ways of storing data: in Content Store and in Repository:

* Content Store : is used for temporarily caching arrived Data packets. The Content Object field freshnessSecond is used to define the lifetime of the ContentObject in the Content store relative to the reception time.

* Repository: is used for persistent storage of CCN Content Objects. There are different ways to write data into the repository:

    1. Using a standard CCNx application ccnputfile, this is used to store local files into the repository.
    2. A specially formed start-write Interest that triggers the repository to retrieve and store the Content, using special command marker [5].

2.1.4 Synchronization

Synchronization is a CCNx facility that allows defining Collections of data in Repositories that are to be automatically kept in synchronization with identically defined Collections in neighboring repositories [6]. It is based on the following elements:

* Collection: is a set of contents under the same prefix. The definition of prefix is known as a slice

* Sync Agent: is responsible for keeping information about local Collections up to date as content is added to the Repository, and detecting new Content Objects of the Collection in the remote repository that are not in the local repository. It is also responsible to reply to any requests from the remote repository for information about the local Collection. A Sync tree is the representation of all the content in the Collection and is built and used by the Sync Agent. As the files are added to the local repository, the Sync tree is updated.

* A Root Advise Interest: is an Interest used by the Sync Agent to determine if there are any names in a remote Collection that are not in the local Collection.

2.2 CCN in Opportunistic and Mobile Networks

Previous works in [7] proposed two content discovery algorithms for opportunistic content-centric networks, Enumeration Request Discovery based on name enumeration requests and the Regular Interest Discovery based on regular Interests. Content discovery is important in mobile CCN to have information about the available content. The authors conclude that content-centric networking is advantageous in opportunistic one-hop communication because of multicast discovery, which enables quick discovery in dynamic environments.

In [8], an extension of CCN file transfers is implemented that enables mobile devices to download complete data files even if a transfer is intermittent. In case of disrupted file transfers, the proposed mechanism persistently stores already received content objects locally and resumes file transfers when the connectivity is regained. The approach proposed in this thesis to resume file transfers is different from this approach as explained in Section 4.4 . The approach proposed in [8] stores already received segments locally without need to provide it to the others and then the CCN header and the signature are not stored. However, in our approach a possibly untrusted agent that retrieves content for a requester needs to ensure the authenticity and integrity of the retrieval content. Therefore, signatures and CCN header information are required to be stored with the content.

In [9], an approach based on overhearing multicast traffic is proposed to configure the forwarding table (FIB) of the CCNx framework. The proposed approach enables mobile devices, where connectivity patterns may change often, to dynamically configure the FIB with temporary entries to forward Interests towards a potential content source. In this work, the FIB is updated also dynamically whenever a new delegation is accepted (content retrieval in phase II) and whenever a notification is received in phase III.

Moreover, in [9] the multicast and unicast transfers are compared in terms of throughput and transmitted messages. The authors observed that unicast communication results in much higher throughput because of MAC-acknowledgment and faster retransmission. In unicast communication, the retransmission of packets, in case of collisions, is done on the MAC-layer and the higher layered CCNx does not recognize the packets lost. In multicast communication, a requester re-expresses an Interest after an Interest timeout. Thus, the retransmission is done by CCNx on the application layer which takes more time compared to the MAC layer retransmission. In this thesis, we therefore use unicast communication as often as possible. As soon as a notification is received, the requester can address the agent to retrieve the content directly via unicast. The IP address is not available at the CCN application level. Therefore, the agent includes its own IP address in the payload field of the Data packet transmitted in phase III (notification message). Upon the reception of a notification message, the requester can temporarily add a new unicast face and address the agent directly via unicast to retrieve the content.
The idea of the agent-based approach is to support content-centric opportunistic communication in mobile networks. Existing work in mobile CCN communication targets mainly architectures with a static core network to which mobile nodes, i.e. content sources or consumers, connect. There are basically two approaches to handle content source mobility: First, to use a locator-name split. Second, to use redirection points in the network. When using a locator-name split, e.g., [10],[11],[12], location dependent identifiers, i.e., locators, are prepended to the content name depending on their point-of-attachment. These locators are then used for routing and forwarding only. To avoid resigning all content objects when moving, the locator needs to be excluded from the signature and cache comparisons need to be performed on the content name only. When using redirection points or rendezvous points, e.g.,[12], [13],[14],[15],[16],[17], mobile devices can register their position to these nodes similar to mobile communication architectures such as cellular networks or mobile IP. A requester would then first contact the rendezvous point to get the current location of the mobile content source. Requests can then be sent directly to the new location or are redirected via the rendezvous point. In case of mobile consumer, e.g. [14], the mobile requester sends the content query packet to its rendezvous point. The rendezvous point tries to discover the content itself. After that, it delivers the content Data packets to the mobile requester. When detecting the imminent of the handover event, the rendezvous point stops delivering content Data packets and only stores the content data in its local repository. When the mobile requester acquires a new IP address, it notifies its rendezvous point. Then, the rendezvous point transmits the stored content data packets toward the new location of mobile requester node. In this approach, a small communication delay is introduced because the communication needs to be redirected via the rendezvous point independent of the position of the mobile device.

Other works have investigated the routing of Interests in mobile ad-hoc networks, e.g., [18], [19], [20], [21] based on different criterias and algorithms such as e.g., the distances between the nodes or structured routing based on DHTs. However, these approaches do not consider delays and connectivity dynamics that can be introduced in mobile networks.

In the Haggle project [22] an opportunistic and delay-tolerant communication system was developed. Nodes request and exchange content upon their encounters by supporting the separation of application logic from transport bindings. This is currently not possible in CCN because

Interests have a lifetime and are re-expressed if it expires. The agent-based approach developed within the scope of this thesis targets a similar approach as in Haggle but uses content-centric primitives. Nodes do not need to exchange connectivity information and users can explore their vicinity quicker by transmitting multicast requests instead of connecting to each neighbor node subsequently.

The home repository used in this thesis is optional and can assist nodes to notify requesters or retrieve content from mobile agents. The concept is similar to custodian-based information sharing [23] but users do not only synchronize their own personal content but also content retrieved for others. This supports communication in disruptive and delay-tolerant networks where agents can synchronize content with their home repository if connected to the Internet. Requesters can then obtain the content from home repositories if they have no direct connection to the agent even if they are not connected to the Internet at the same time as the agent.

In [24], another Content-Centric Opportunistic architecture inspired by CCN is developed. It proposes a new multiplatform information-centric framework that can be used in a dynamic environment to assure seamless operation.

Chapter 3

Agent-Based Content Retrieval

3.1 Problem Description

In Content Centric Networking, nodes within transmission range can exchange data through ad-hoc communication without being connected to the Internet or knowing the identifiers of any nodes. A node can receive what it needs by broadcasting an Interest to the network asking for the desired content. Then, any node in its vicinity that has the corresponding data can reply. In such a situation, CCN communication works well as the needed content is available at least in one node within the transmission range of the requester.

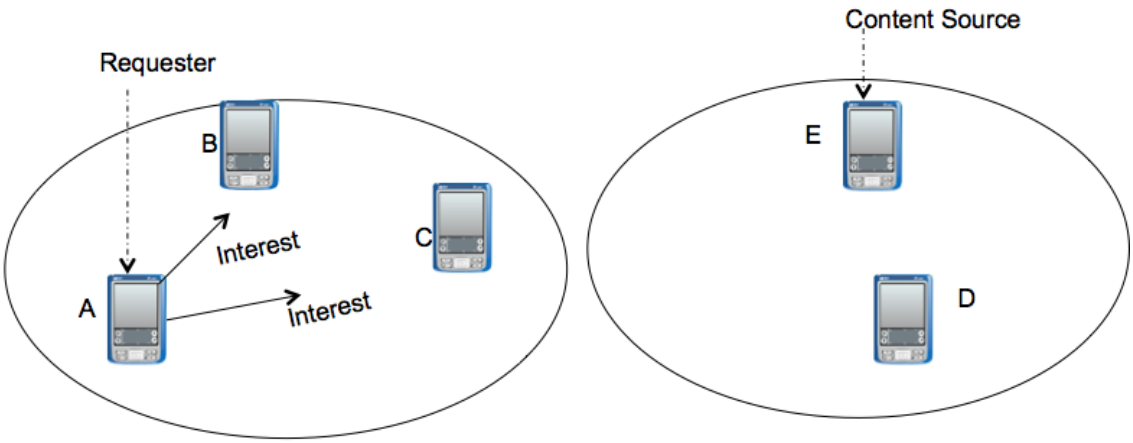


Figure 3.1: Requester and content source in disjoint transmission ranges

Figure 3.1 shows a network with requester and content source, the ellipses show the transmission and reception range of the requester and content source, e.g. node A can reach node B and C, but can not reach node E and D which are far away. In such situation, since the transmission ranges are disjoint, traditional CCN communication does not work: if node A needs the content that is not contained in any nodes within its transmission range, but it resides only in node E, node A will never receive the content, until it meets node E.

3.2 Agent-Based Content Retrieval

To be able to reach content not available in any neighbour nodes, the requester can delegate the content retrieval to other nodes called agents (subsection 3.2.3). These agents are mobile and can move closer to a content source than the original requester and get the desired content. Such delegation should be performed in a controlled way based on agreements between requester and agent to avoid denial-of-service attacks. The agent will then retrieve the content for the requesters (more details in subsection 3.2.4) and notify the original requester that the content has been found and where it can be retrieved so that the original requester can abort the search (subsection 3.2.5). We give a more detailed overview in subsection 3.2.2.

3.2.1 Roles

During the delegation process, we can distinguish between three roles: requester node, agent node and agent proxy.

- Requester node: starts the process of delegation; it is responsible to select the agent and to delegate the content retrieval to him or her.

- Agent node: is responsible to reply to delegation requests, to look for desired content and to notify the requester node if the content is found.

- Agent proxy: is the agent's home repository which is permanently connected through the Internet.

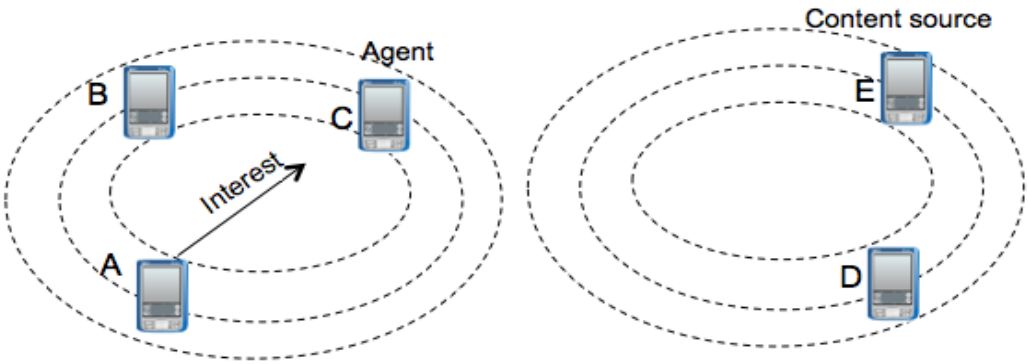
3.2.2 Overview

Delegation process is divided into three main phases as described in Figure 3.2:

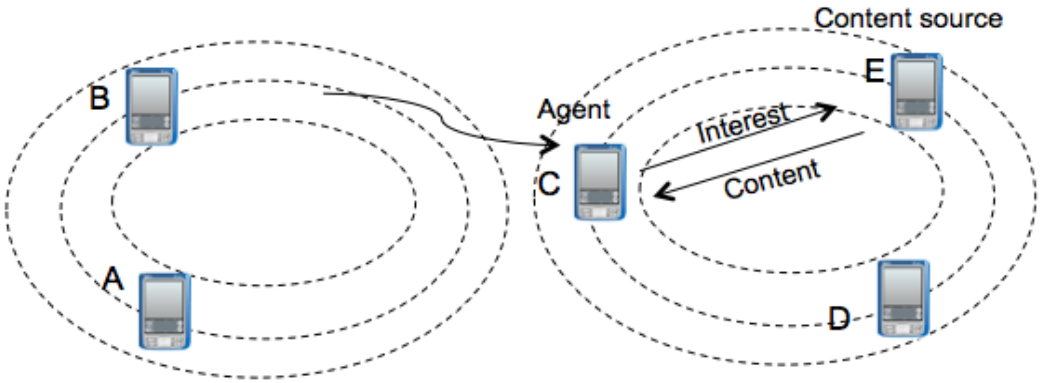
* Phase I, Agent Delegation: Fig 3.2 a) shows a network with a requester node (node A), an agent node (node C) and a content source (node E) which is far away, not in the transmission range of the requester node. If requester A tries to find the content in its immediate neighborhoods but does not receive any response, it will start the agent delegation and transmit a multicast request looking for available agents. Then, it selects one of them and delegates the task of finding the content to the selected agent (e.g. node C). More details are given in Subsection 3.2.3.

* Phase II, Content Retrieval: at this phase, the selected agent node (node C) starts the process of searching the desired data. Thus, as shown in (Fig 3.2 b): when agent node C goes to the area where it can reach node E that has the content, agent node C can ask for this content and stores it in the repository of its mobile device. More details are given in subsection 3.2.4.

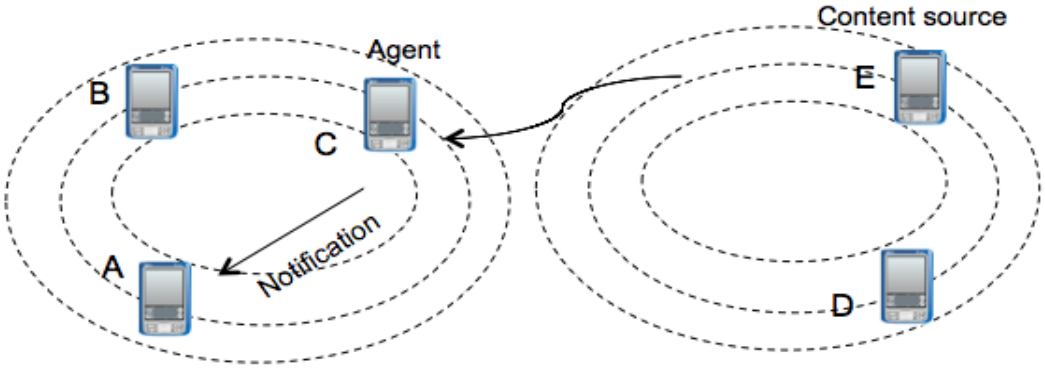
* Phase III, Notification: Later as shown in Fig 3.2 c), when agent node C can reach again requester node A, it will send a notification to the requester to inform it that the content has been found. We give more details about this phase in subsection 3.2.5.



(a) Phase 1: Agent Delegation



(b) Phase 2: Content Retrieval



(c) Phase 3: Notification

Figure 3.2: Agent-Based Content Retrieval phases

3.2.3 Phase 1: Agent Delegation

This section describes the first phase of the handshake mechanism. Below, we demonstrate a sample scenario where the requester node tries to find a suitable agent from the neighbours to delegate the content retrieval to it.

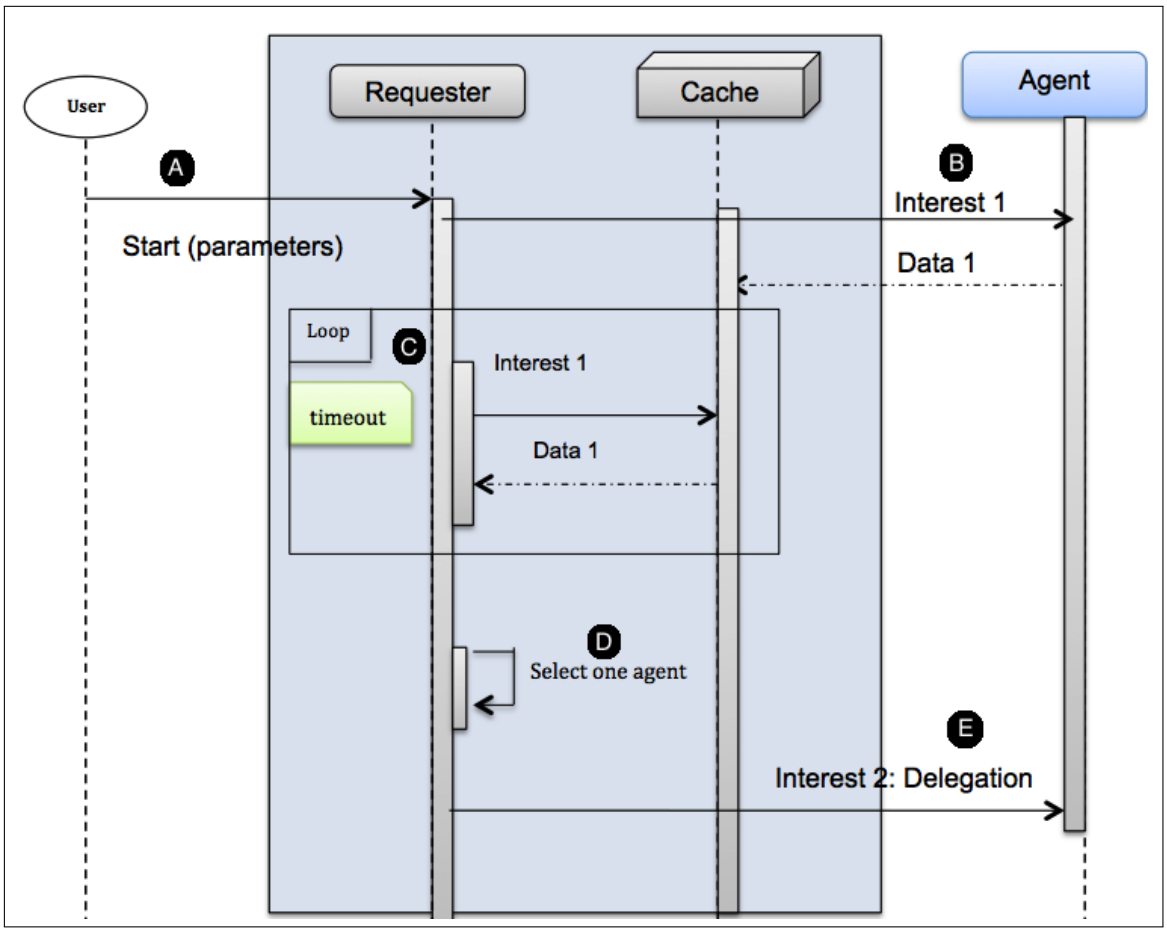


Figure 3.3: Agent Delegation

Figure 3.3 illustrates a scenario of agent delegation. It contains the requester node, composed of the requester application and the cache, and an agent node. When the user starts the delegation process **[A]**, the requester node sends a multicast request looking for potential available agents. The request is an Interest comprising of the prefix */ferrying*, the content name and the parameters as listed in Table 3.1. It has the following structure:
/ferrying/%C1.<namespace>~ <parameters>

At the agent node **[B]**: when the neighbour agent receives the Interest, which matches the prefix */ferrying*, it replies with a Data message containing its node identity. The Data has the following name structure: /ferrying/%C1.<namespace>~ <parameters>/nodeID

All the responses from available agents are stored in the cache of the requester node. As

/ferrying	Prefix of delegation
namespace	Name of the content needed
parameters	Optional parameters, e.g., GPS coordinates of a search area.

Table 3.1: Interest1 Components

a normal process in CCN, every Interest can only retrieve at most one Data packet (1 Interest <->1 Content), so only the first response is sent to the requester application. In order to receive other responses, the requester needs to ask the cache for other responses as illustrated in step **[C]**. The requester sends the same Interest excluding the nodeID of already received responses, so that the cache will send only the new ones. The requester may ask the cache until no new answer is received within a defined time (timeout) or the number of received responses reaches a given maximum number specified by the user.

After receiving multiple answers from available agents **[D]**, the requester generates an agent list and selects one of them. The requester uses the nodeID of the selected agent to delegate content retrieval to it **[E]**. The delegation is performed again with an Interest as we will see later in subsection 4.1.4.1. This Interest must include a */groupID*, which is a random number and must be transmitted to the agent, because the requester will use it for the Notification in phase three, as explained later in Section 3.2.5).

3.2.4 Phase 2: Content Retrieval

During this phase, the selected agent triggers the process of searching the content desired by the requester. If the agent would first request the content, store it locally and then include a copy in the repository, the content would be resigned by the agent and therefore not trustworthy for the original requester. To avoid that, we can ask the repository to get the content. Therefore, the agent's repository will store the entire content including the signature.

Figure 3.4 illustrates a simple scenario of the Content Retrieval phase: First, the agent extracts the name of the desired content from delegation Interest received at the end of Phase I. Then, **[A]** it periodically sends a multicast Interest to all current neighbor nodes asking for the content, until receiving a response. Second, **[B]** if the agent receives a response from a content source including the full name of the content, it asks the repository to retrieve all content objects including all signatures **[C]**. Finally, the agent needs to check that the repository has completely received the content before notifying the requester. More details on this can be found in subsection 4.3.2.

3.2.5 Phase 3: Notification

This phase is started by the agent once it has verified that the content is successfully retrieved by the local repository.

In CCN, if the Interest lifetime is over, the Interest is deleted from the PIT in the CCN deamon running on the nodes, i.e., in agent and requester nodes. There are two possibilities to send notifications to the requester as shown in Figure 3.5, in step **[A]**. First, if the content is

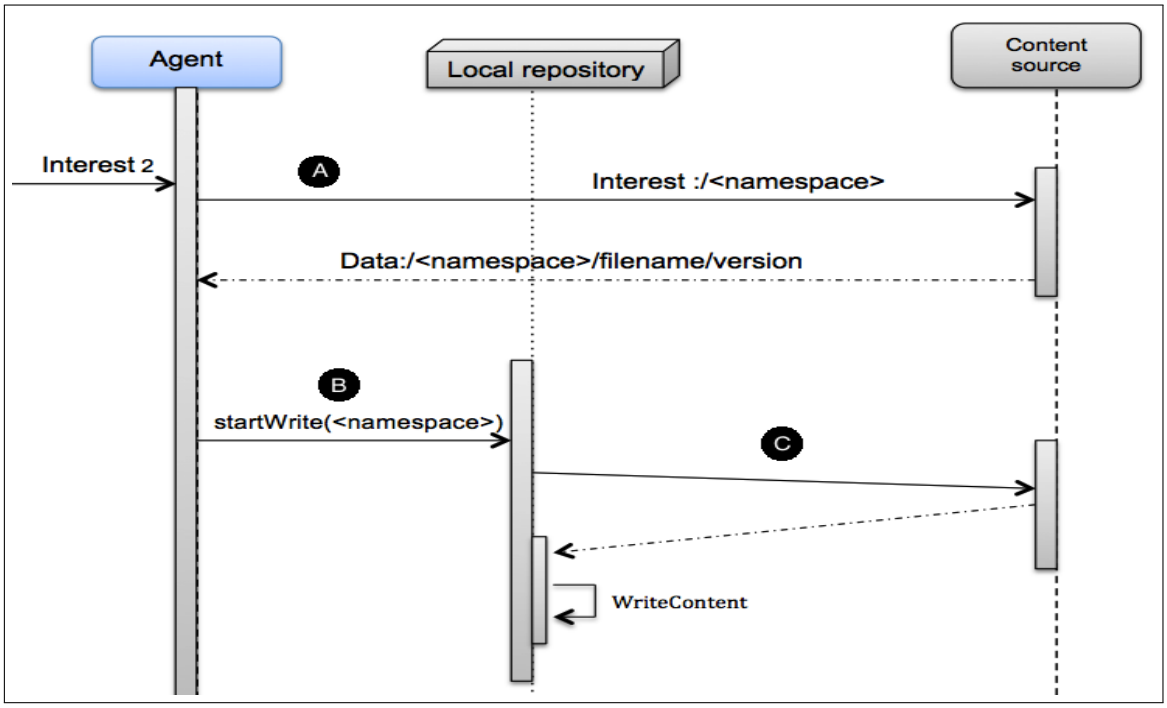


Figure 3.4: Content Retrieval

received within the Interest lifetime of the delegation Interest, the agent may send a notification as reply to the delegation Interest. Second, if the Interest lifetime is over **[B]**, it needs to receive a notification Interest from the requester in order to send the notification. This Interest is composed of the prefix */groupID*. With this prefix, the requester can address all agents to which he has delegated the content retrieval at the same time without addressing them individually. Therefore, if the requester node does not receive any immediate notifications, it can check if any other agent has already retrieved the content with the prefix */groupID*. If no existing agent has retrieved the content, it may delegate the retrieval to another agent repeating the agent selection phase in subsection 3.2.3. The requester should transmit the same *groupID* that has been transmitted to the previous agents.

Notification message

The main aim of the notification message is to inform the requester that the requested content has been found and has been successfully stored in the agent's mobile and home repositories. However, the requester needs the identity of the agent to ask it directly. Therefore, the agent includes its current IP address as well as the IP address of the agent proxy, i.e., home repository, in the notification message.

Getting the content

After receiving the notification, the requester tries to retrieve the content from the agent. It has two possibilities of getting the content, depending on whether the agent is still in transmission

range of the requester or not. As shown in Figure 3.5, first in step **[C]** an entry associated to IP address of the mobile agent must be added temporarily to the FIB of the requester. The requester can try to ask directly the agent node for the content. Second, if it does not receive any reply when directly asking the mobile agent, it can add the agent's home repository IP address to the FIB **[D]** and request the content from there.

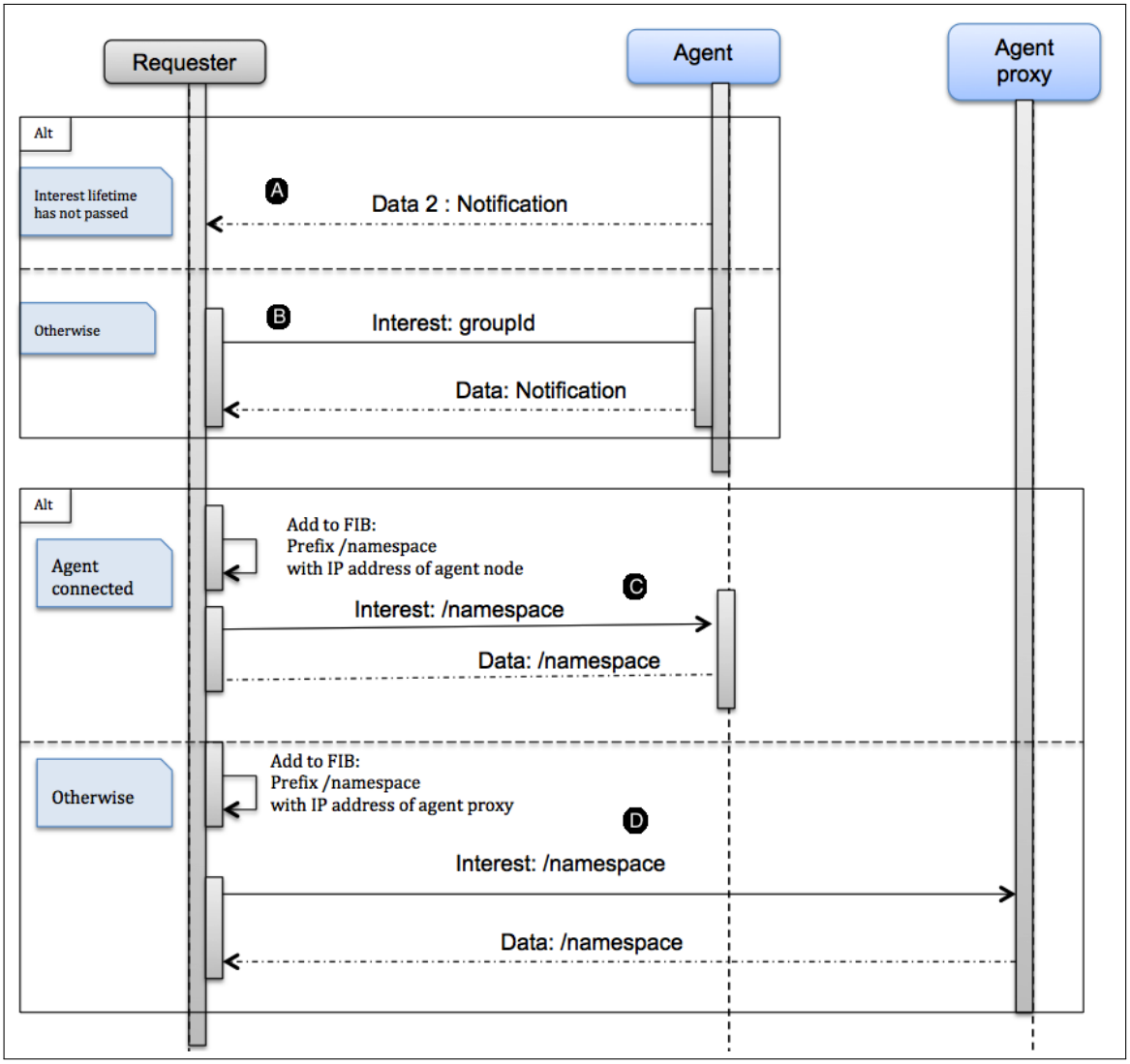


Figure 3.5: Notification phase

3.2.6 Synchronization with Agent Proxy

The agent proxy is the home repository of the agent node and is permanently connected to the Internet. Therefore, every content object in the local repository, i.e., the mobile repository,

should be stored in the home repository as well. The aim of the agent proxy is that the requester can retrieve the content even if the mobile repository is out of transmission range.

Permanent Synchronization

The local repository should keep the files needed by the requesters synchronized with the home repository. For this reason, synchronization on a specific collection, i.e., set of content objects whose names share a common prefix, must be established between the two repositories. The collection should cover all the content files that the agent has stored in the local repository to serve all requests from the requester nodes.

Figure 3.6 shows the flow chart of the synchronization mechanism. Synchronization is maintained by periodically exchanging Root Advise Interests **[A]**, i.e. an Interest that is used to detect if there are any names in a remote collection that are not in the local collection. When they find that they have not the same collection, they try to compare and update the content to obtain identical collections**[B]**. After that, they start to send periodically the Root Advice Interest **[C]**, until the collection is updated at one of the repositories.

However, in case of mobile clients that are regularly disconnected from the home repository, the agent does not need to permanently check if the collection is updated and transmitting Root Advice Interest would not result in any response. Therefore, the agent needs to synchronize the mobile repository with the home repository only when it is connected to the Internet.

Temporary Synchronization

To resolve the lack of permanent synchronization, the agent needs to handle synchronization with the agent proxy: it needs to control the beginning and the end of the synchronization mechanism. As a result, the synchronization will be established temporarily when the agent is connected to the Internet. Figure 3.7 illustrates a simple scenario of temporary synchronization: when the user starts synchronization manually, mobile and home repositories define a collection and start to exchange Root Advise Interests. When they detect that there is new content in the mobile agent's repository, the home repository will ask for it.

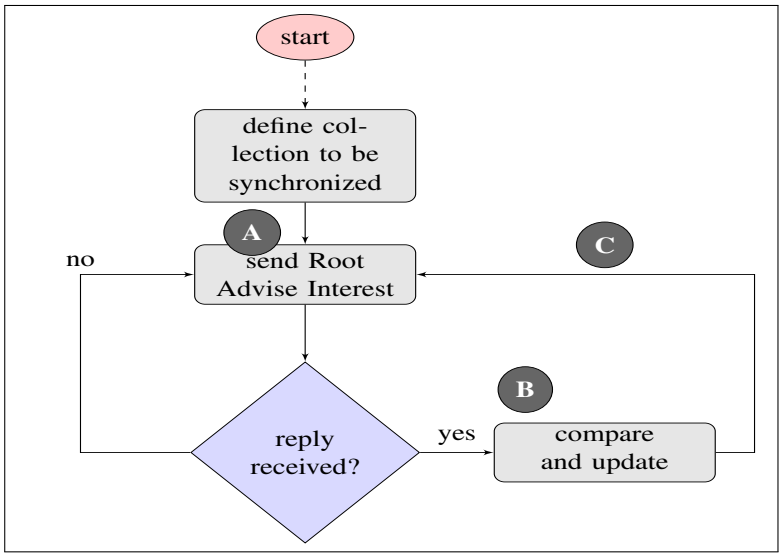


Figure 3.6: Flowchart of permanent synchronization

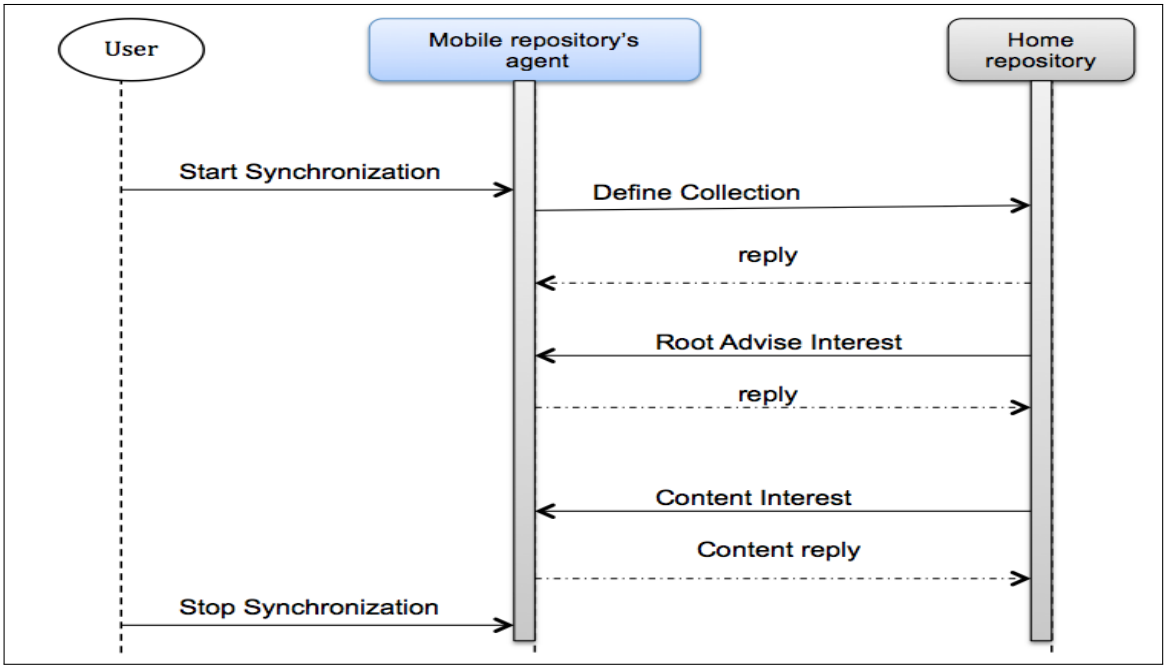


Figure 3.7: Temporary synchronization

Chapter 4

Implementation

In this chapter, the implementation is described step-by-step and is explained by works flows. Three applications have been developed to implement the Agent-Based Content Retrieval mechanism:

* *CCNRequester* application: is provided to the requester node and it performs the delegating procedure.
* *CCNAgent* application: is provided to the agent node and it performs the content retrieval and notification procedures.
* *CCNSynchronization* application: is provided to the agent node and it handles the synchronization process.

4.1 Delegation Procedure on CCNRequester

Figure 4.1 illustrates a flow chart of the *CCNRequester* application, which is provided to the requester node. First, an Interest (*initialInterest*), composed of the prefix */ferrying* and content name will be sent to any nodes in range as a multicast request **[A]**. To send agent requests to neighbours, a multicast face needs to be configured, because the neighbors are not known. The */ferrying* prefix needs then to be registered to this multicast face. By that, *initialInterest* will be transmitted to all neighbour nodes. The rest of the functionalities illustrated in flow chart 4.1 will be explained in subsection 4.1.1 and 4.1.2.

4.1.1 Processing of Incoming Agent Responses

Agent responses are processed within the *handleAgentResponses* function. This function checks if a reply has been received and triggers the delegation process. It consists of the following functionalities:

1. Extraction of the identity of the responders from the agent's Data message.
2. Looking for other responses by calling *getResponsesFromCache* function. This function transmits similar Interests as *initialInterest* but with two limitations: First, it should get

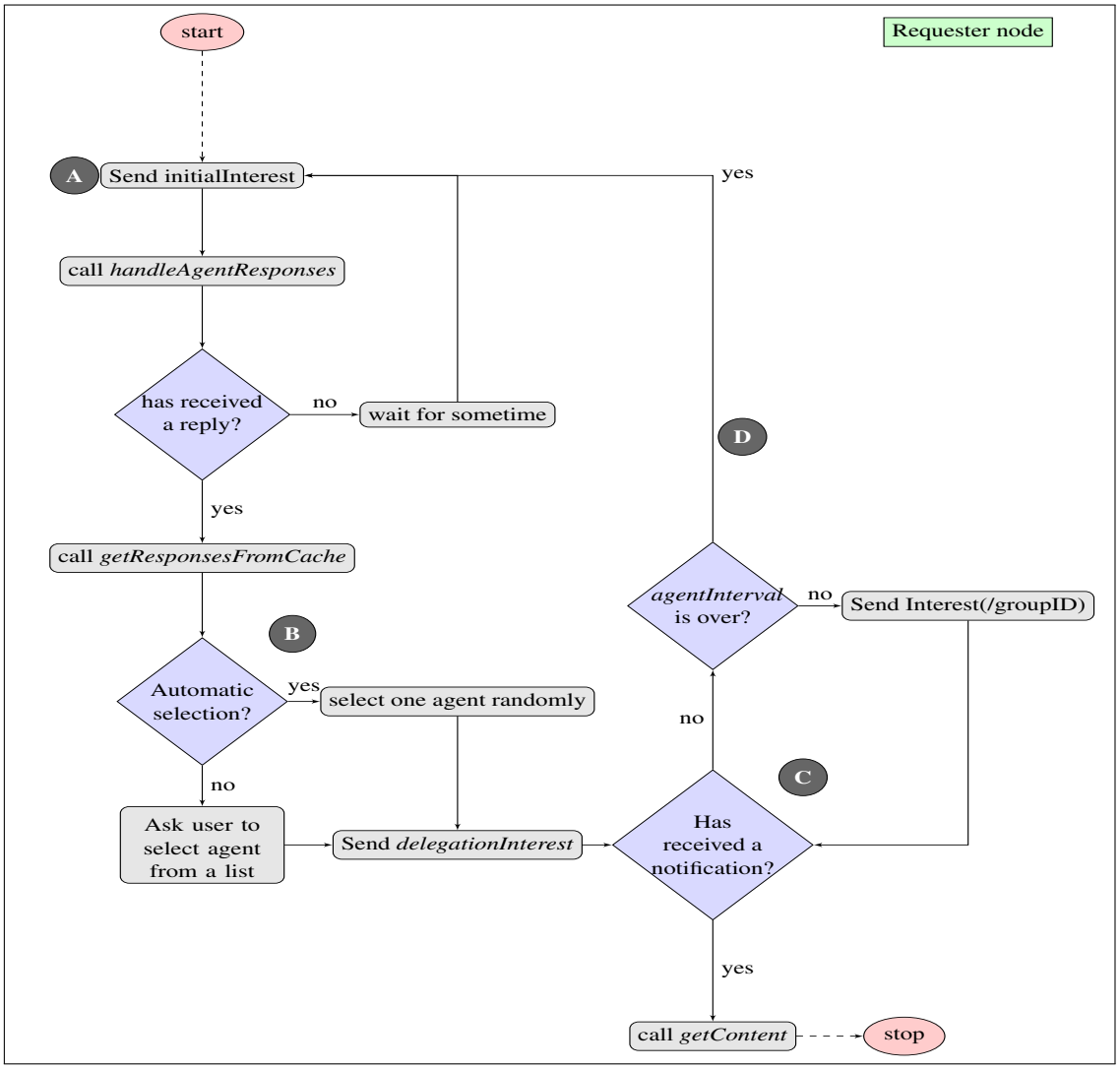


Figure 4.1: Flow chart of the delegation process in the requester node

only the new responses. Then, the identity of the previous responders needs to be included in the exclude list of *initialInterest*. Second, the Interest must be forwarded only to the cache and not to neighbour nodes. Therefore, Interest scope is set to value ”0”.

The *getResponsesFromCache* function transmits the modified *initialInterest* and adds the reply to the agent list which is provided to the *CCNRequester* application. This will be repeated periodically until the number of received responses reaches *maxNumberAgents*, which is specified by the user, or the configurable parameter *TIMEOUT_CACHE* is reached.

After receiving the list of available agents, only one agent should be selected at a time. The selection is performed manually by the user or randomly if automatic selection is enabled **[B]**. After the agent is selected, the *delegationInterest*, i.e., an Interest containing the agent identity, is

sent to the corresponding agent. The structure of this Interest is provided in subsection 4.1.4.1.

4.1.2 Receiving a Notification

After sending the *delegationInterest*, as shown in Figure 4.1 step **[C]**, the requester checks first if it has already received any notifications that the content has been completely received by the agent so that the requester can start the retrieval. Otherwise, the prefix *groupID* needs to be registered to the multicast face. The requester waits for some time, i.e. *probeInterval*, and then sends an Interest including the *groupID* asking all existing agents for a notification.
The requester transmits the notification request in *groupID* periodically at every *probeInterval* until receiving a notification or until the *agentInterval* has passed. The *agentInterval* is the time that the requester waits until asking a new agent **[D]**. After every *agentInterval*, the requester goes back to step **[A]** and repeats the whole process, excluding the previously selected agents from the next list of responders. This process will be repeated at each *agentInterval* until the requester receives a notification or the *timeoutRequest* has been reached.

To avoid that Interests from previous agent requests are retrieved, the *initialInterest* has the field *AnswerOriginKind* set to 0 to avoid any responses from the cache. This means that Interests are not answered from the content store. Also, to avoid the retrieval of previous answers from the cache when creating the agent list, all the responses to agent delegations have a *FreshnessSeconds* value much lower than the *agentInterval*. This means that if a new list is requested, the previous answers are already stale. In our implementation, we set the *FreshnessSeconds* to 3 seconds and set the *AnswerOriginKind* field to 0, i.e. meaning no stale answer from content store or application.

4.1.3 Retrieving the Content from the Agent

After receiving a notification, the requester can request the content as shown in Figure 4.2. The *CCNRequester* retrieves the IP addresses of mobile and home repositories from the payload of the received notification. Before asking for the content, a unicast face with the IP address of the mobile repository needs to be created and the prefix */namespace* has to be registered to this Face **[A]**. If the agent node is still in the transmission range of the requester node, the content will be successfully retrieved via the unicast face. Otherwise, if an error has been raised during the content retrieval **[B]**, another unicast face with the IP address of the agent's home repository will be created **[C]**. Whenever connected to the Internet, the requester can request the content from there. Afterwards, when the retrieving process has been completed, a notification is sent to the user to inform him or her that the requested content has been successfully received.

4.1.4 Data Structure

4.1.4.1 Delegation Interest Structure

The Delegation Interest is used to trigger the content retrieval at the agent node. It should contain two components: "remainingTime" and "groupID", which will be used by the selected

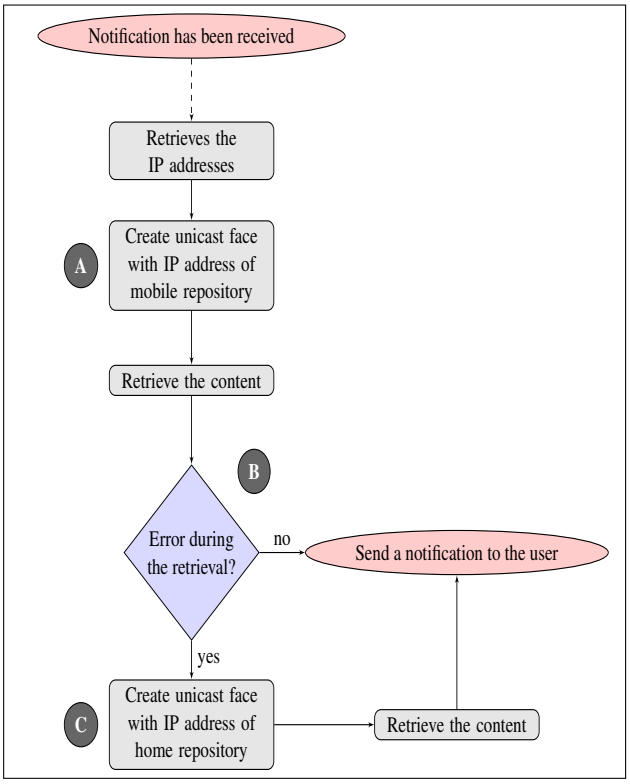


Figure 4.2: Flow chart of the last part of the content retrieval process at the Requester node

agent.
The *delegationInterest* has the following structure:

```
/ferrying/nodeID/%C1.<namespace>~<parameters>/remainingTime/groupID
```

* *nodeID*: is a short string representation of publisher public key digest, which identifies the identity of the agent's CCND.
* *remainingTime*: It specifies the remaining time the requester is looking for the content. When this time is over the requester stops the delegation process. This time is transmitted to the agent so that the agent does not need to look for the content anymore if this time is over.
* *groupID*: is a hash of random number and is used to identify all agents that are used to find a given content.

It is required to put the *remainingTime* as component and not as parameter, because the agent needs to know where it can find the *remainingTime* from the received Interest. The

remainingTime is a mandatory component and should exist in every delegation Interest. The <parameters>component is for optional information that can be related to the desired content such as e.g., coordinates where the content can be found.

There are two possible *delegationInterest* structures:

```
1- /nodeID/ferrying/%C1.<namespace>~<parameters>/remainingTime/groupID
2- /ferrying/nodeID/%C1.<namespace>~<parameters>/remainingTime/groupID
```

In the first possibility, the requests would have been directed directly to */nodeID*. However, all requesters would need to register */nodeID* within their FIB for a short time at each delegation to forward these Interests to the wireless medium. In the second possibility, there is no need to register any additional prefix within the FIB besides the /ferrying prefix. However in the second option, all the neighbour agents will receive *delegationInterest*, but only the one, which has the same identity (*nodeID*) responds.

4.1.4.2 Summary of the variables

Table 4.1 gives an overview of the variables mentioned in this section and their meanings.

probeInterval	is the time the requester waits before asking for a notification
agentInterval	is the time the requester waits to delegate the content retrieval to a new agent
timeoutRequest	is the lifetime of the current request
maxNumberAgents	is the maximum number of agents that the requester can delegate the content retrieval to.
TIMEOUT_CACHE	is the time after which the *CCNRequester* application stops to send the *initialInterest* to the cache asking for another response.

Table 4.1: Summary of the variables

4.2 Content Retrieval on CCNAgent

4.2.1 Processing of Incoming Interests

Figure 4.3 shows the tasks performed by the *CCNAgent* application. After starting the application, an Interest filter with prefix */ferrying* needs to be registered, so that any Interests whose prefix matches /ferrying will be delivered to the handler of *CCNAgent* application. In addition, a multicast face needs to be created and the prefix */ferrying* has to be registered to it **[A]**.

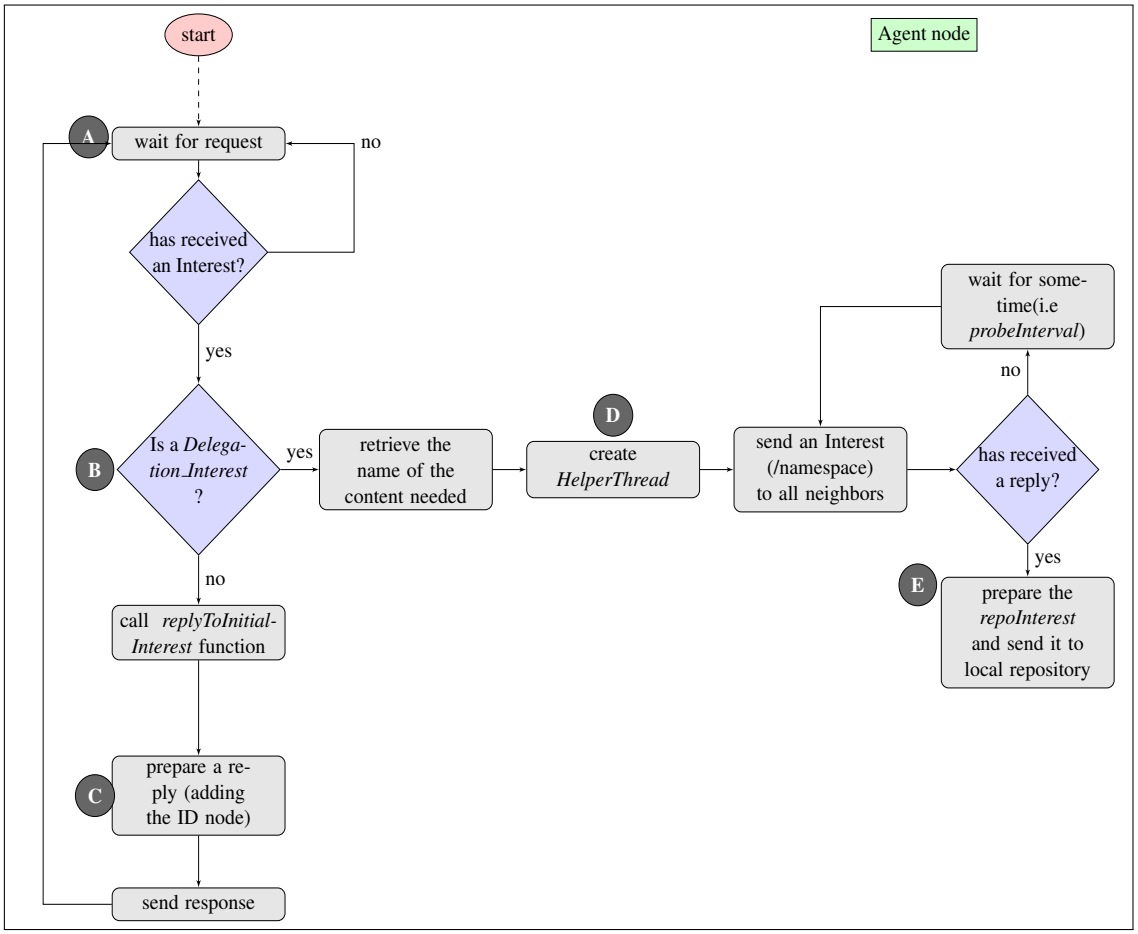


Figure 4.3: Flow chart of *CCNAgent* application tasks

When the agent receives an Interest, a verification must be performed to detect if the received Interest is an *initialInterest* or a *delegationInterest* **[B]**. First, if the current Interest matches only the prefix */ferrying*, the application calls the function *replyToInitialInterest*, which prepares a reply message by appending */nodeID*, and sends it to the requester **[C]**. Second, if the agent receives the *delegationInterest* that matches the prefix */ferrying/nodeID*, the application retrieves the requested content name and creates another thread, i.e. the *HelperThread*, **[D]** that will take charge of searching and getting the desired content. First, the prefix /$<$*namespace* $>$ needs to be registered to the multicast face so that the agent node can ask neighbor nodes for the content. Then, the HelperThread will transmit an Interest in /$<$*namespace* $>$ asking for the full name of the content. Second, after receiving a reply from a content source **[E]**, the *HelperThread* sends an Interest (i.e. *repoInterest)* to the local repository asking it to get the content.

4.2.2 Repository Interest Structure

The *repoInterest* has the following structure:

```
ccnx:/<fullname>/%C1.R.sw/<NONCE>
```

First, the Interest should instruct the local repository to retrieve and store the corresponding content. This is indicated by the start write command marker %C1.R.sw. Second, the *repoInterest* should not be forwarded to any neighbour node and should be transmitted only to the local repository. Therefore, the scope of *repoInterest* needs to be set to the value "1", i.e., only targeting applications on the originating host.

The<NONCE>component is used for marking random nonce values designed to make *repoInterest* names unique. It is used to assume that the start write Interest, e.g. *repoInterest*, will be received by the repository and might not be answered only from the cache.

4.3 Notification

Figure 4.4 shows the flow chart of the notification process: First, before the end of *delegationInterest* lifetime, the *CCNAgent* application checks if the content has been completely retrieved by the repository. So, it reads a Boolean variable. This variable can be set to true only by the HelperThread when it has completely retrieved the desired content. If the Boolean variable is true, the notification is included in a *delegationInterest* Data packet. Otherwise, it waits until the *HelperThread* reports that the content has been successfully written to the local repository **[C]**, as explained later in subsection 4.3.2, and then it will start listening to any Interest whose prefix matches */groupID*. Therefore, it needs to register a filter with prefix */groupID* so that any Interest, whose prefix matches */groupID*, will be delivered to the *CCNAgent* application.

When the agent receives the Interest */groupID*, it replies with the notification as a Data message and unregisters the filter associated to prefix */groupID*.

4.3.1 Notification Structure

The CCNAgent includes the current IP address of the agent node, which is automatically retrieved from the system, and the IP address of the home repository, which is configured by the user at startup, in the payload of the notification Data packet.

The payload Data is structured as follows:

```
/IPM/<IP Address of Mobile Repo>/IPH/<IP Address of Home Repo>
```

IPM indicates that the following IP address is the IP address of the agent's mobile repository. IPH means that the following IP address is the IP address of the home repository.

4.3.2 Verification Mechanism of Content Retrieval at Agent Node

Since the repository does not give any notifications when a content retrieval is finished, a verification process must be performed to ensure that the content is completely retrieved. Only if the content retrieval is complete, the notification message can be transmitted. There were three proposals to achieve that:

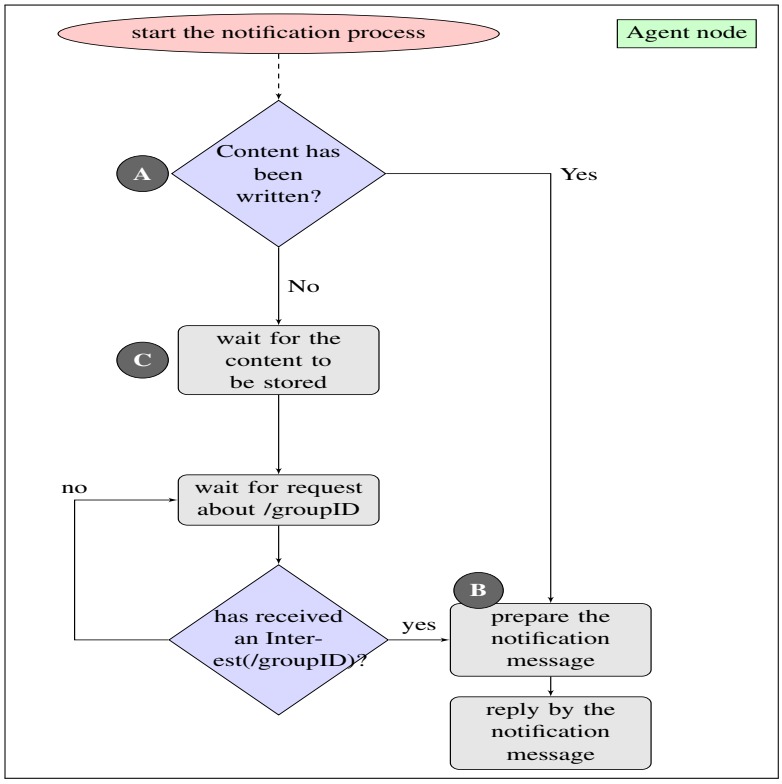


Figure 4.4: Flow chart of the notification process

* Trying to get the last segment. However, there's no guarantee that the blocks are written in order and all blocks prior to the last segment have been received.

* Using the "checked write" command. However, checked write only guarantees that the requested block has been written and it used with the first block to get a general idea that the repository has begun writing the file.

* Trying to read the file: is the approach applied in this work, and explained below.

Figure 4.5 illustrates the flow chart of the verification process. The verification process starts by reading the content as normal sequential reading in CCN **[A]**. The reading must be performed from the local repository and not from the neighbors. To achieve this, the scope of every Interest during the reading procedure must be set to value "1".
If the reading process has successfully passed, the *HelperThread* can report to the *CCNAgent* application that the content has been written **[B]**. Otherwise, if the reading procedure raised any errors when getting a given segment, the *HelperThread* remembers the number of the failed segment and waits for a short *readInterval* until trying to read the failed segment again **[A]**. In case of large files, the file transfer may not be finished yet and the read process may have reached the last received segment. If this is the case, then a second read attempt after the readInterval may be successful. If the reading fails twice at the same segment **[C]**, it will wait for *writeInterval*,

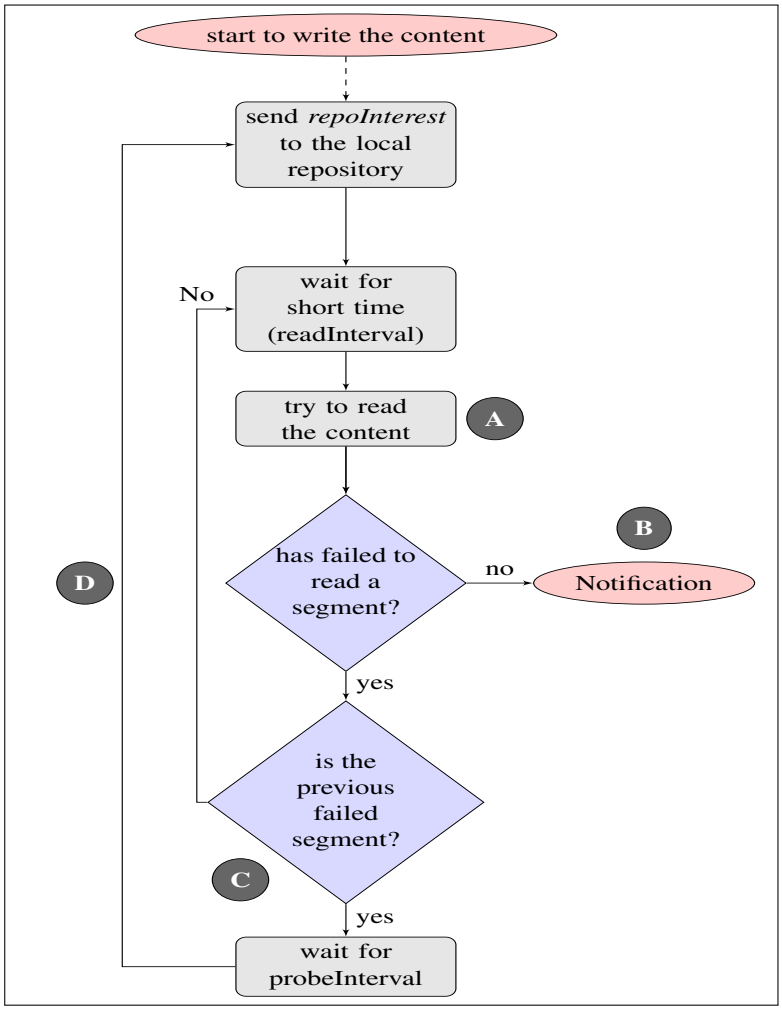


Figure 4.5: Flow chart of the verification mechanism in the agent node

and then it will send once again the *repoInterest* (see subsection 4.2.2 for its structure) to the local repository **[D]** trying to let it retrieve the complete file. At this time, the *HelperThread* tries to read the file from the previously failed segment. The *HelperThread* repeats this process until the reading is completed successfully or the *remainingTime* is over.

Table 4.2 gives an overview of the variables mentioned in this subsection and their meanings.

4.4 Resuming Disrupted Transfers

If the content source gets disconnected and the repository cannot retrieve a specific segment during the content retrieval, the repository will abort the content retrieval after some time. At this point, the reading process will obviously fail to check if a file transfer is complete. As

writeInterval	is the time the agent waits before asking the repository again to get the content, if the repository has failed to retrieve the content at the first time
readInterval	is the time the agent waits to check if the missed segment has been stored in the repository

Table 4.2: Summary of the variables

explained in the previous subsection, the agent will wait for *writeInterval* before asking the repository to retrieve the content again. Even if the previously failed segments are already stored in the repository, the repository does not remember already received segments and requests these segments again. The Interests are forwarded to the CCND and since the CCND receives these Interests from the repository's internal face, it cannot forward it back on the same face. As a result, the CCND will forward these Interests via the multicast face to the neighbors if the content is not available in the cache anymore. To avoid new requests in already received content objects, the already stored segments need to be loaded in the content store so that the repository can get the previously stored segments from the content store and does not need to forward Interests to its neighbors. Loading segments from the repository to the content store needs to be done in parallel a little bit before the repository asks for the content so that all requests can be satisfied just from the cache and do not need to be forwarded to neighbors.
If the repository asks for a segment that is not yet loaded in the content store, the CCND will forward the Interest to the neighbors and a duplicate content object will be received, because until the segment is received from the neighbors it is also loaded from the repository.
There are two approaches to avoid the forwarding of such Interests resulting in duplicates:

1 - Forcing the CCND to not forward any Interest from the repository when loading the already received segments. Then, the repository will get the previously stored segment only from the content store. This can be enforced by unregistering the prefix /namespace from the multicast face and register it again after the loading process finishes. After registering the prefix again, the remaining segments can be forwarded to neighbors. However, the main drawback is that the Interests are still created by the repository but are just dropped at the ccnd. Also, no concurrent file transfer under the prefix /namespace is possible.

2 - Starting the loading process a bit before than the repository's retrieval process. Thus, the repository's retrieval will be started with a delay. Then, after this delay, the content may already be in the content store and the Interest may not be forwarded. However, it is difficult to define the sufficient delay. It should not be too short so that there is enough time to load all segments from the repository and no Interests need to be forwarded, which may generate received duplicate ContentObject. In addition, it should not too long so that the first few content segments are not already replaced by higher segment numbers due to the limited cache size. We tested delays starting from 200 ms to 2 seconds, but in all cases at some point of the repository's retrieval process, some Interests are still forwarded to the neighbors.
We evaluated both solutions as shown later in Section 5.1. The evaluations show that approach 2 results in 41% more received duplicate content objects than approach 1. For this reason, we integrated the approach 1 to the CCNAgent application.

Another solution to resume file transfers is proposed in [8]. It uses the meta information, which contains the name and the version of the content object, the next segment to receive and the position of the partial data file. When the transfer starts, partial data is stored to the file system. If an Interest timeout, the meta information will be also stored to the file system. When the connectivity is regained and the transfer is restarted, the stored meta information is used to provide information about the next needed segment. Then, the retrieval process starts to load the file only from the previous failed segment. However, this mechanism allows storing already received segments only locally without providing it to others. This means signatures and CCN header information are not stored. However, in the agent-based content retrieval as explained in 3.2.4, the agent needs to retrieve the content with the original signature so that it will be trustworthy for the original requester.

4.5 Synchronization with the Home Repository

The synchronization between mobile and home repository can be performed whenever the mobile repository is connected to the Internet. For simplicity, in this work, the synchronization is started and stopped manually by the user on demand. The *CCNSynchronization* application needs some information to establish the synchronization between the mobile and home repositories:

* IP Address of the home repository: Before starting the synchronization, the agent node needs to establish a connection with the home repository so that the Root Advise Interests can be forwarded from the agent node to the home repository. Therefore, the *CCNSynchronization* application should first create a unicast face with the IP address of the home repository.

* Collection prefix: Collection names need to be defined in the mobile repository to be automatically synchronized with identical collections in the home repository. In this work, the collection prefix must be provided by the user when starting the *CCNSynchronization* application.

  As a result, the agent node can synchronize all content under the specified collection prefix with the home repository. There are two possibilities to define collections and keep all data in the mobile repository synchronized with the home repository:

  1. Mobile and home repositories need to define a special meta collection that will contain only the list of names and file versions that need to be stored in the home repository. The home repository will retrieve new files and versions from this collection and later ask the mobile repository for the content. As a result, all the data in the mobile repository will be synchronized with the home repository.
  2. The agent has to send to the home repository the prefix of the new content, so that the home repository can define a collection with the received prefix and establishes synchronization with the new collection prefix.

In the agent proxy, i.e. home repository, the synchronization is started manually by the user: the FIB entry and the definition of the collection are configured manually.

The *CCNSynchronization* application is based on the CCNx synchronization library, First, *CCNSynchronization* registers a unicast face in the FIB with the static IP address of the home repository and defines a collection using the prefix name provided by the user. Then, the synchronization will be started. Second, the application needs to know when the repositories are completely synchronized. To achieve that, Root Advise Interest will be transmitted periodically to check if the repositories have the same hash of the sync tree for the collection. The Root Advise Interest contains the combined hash of the sync tree for the collection. When the home repository receives the Root Advise Interest, it will compare the root hash in the received Interest with its own root hash. If they do not match, it will respond with its own root hash and the root node of its collection and then the *CCNSynchronization* application will report to the user that the repositories need to be synchronized. If the root hashes are the same, no response will be sent and then *CCNSynchronization* application reports to the user that the repositories are synchronized. When the user manually stops the synchronization, the previously created slice will be deleted together with the created unicast face.

Since the IP address of the mobile repository may change depending on the network it is attached to, the home repository needs to receive the current IP address from the agent. To achieve this, the mobile repository needs to connect to the home repository to inform it about its current IP address. The home repository needs to listen to such information messages from the mobile repository and configure the FIB with the current IP address of the agent.

4.6 Graphical User Interface

4.6.1 CCNRequester application GUI

Fig 4.6 a) shows the screen shot of the graphical user interface (GUI) of the CCNRequester application. The user needs to enter the following parameters:

- Namespace: is the name of the desired Content
- Parameters: other parameters that may be used in order to find the content such as e.g., coordinates where the content can be found.
- Lifetime: corresponds to *timeoutRequest* as described in Section 4.1.2. It is the lifetime of the request.
- Filename: the name of local file where the data save, i.e., how locally stored files are named.
- Agent interval: corresponds to the *agentInterval* variable as described in Section 4.1.2. This is the time the requester waits before looking for a new agent.
- Number of agents: is the maximum number of agents that the requester can delegate the content retrieval to. It corresponds to the variable *maxNumberAgents* in Section 4.1.1

- Probe interval: corresponds to the *probeInterval* variable as described in Section 4.1.2. It is the time the requester waits before asking for a notification.
- Selection of agent: the user needs to select the agent from the agent list either manually or let it be selected automatically, as explained in Section 4.1.1.

Afterwards the user can start the application.

If the user selects manual agent selection, the application receives a list of available agents and a pop-up window opens with the list of all available agentId as shown in Figure 4.6 b). Then, the user can select one agent to delegate the content retrieval to. Additionally, another pop-up window will open at the end of the content retrieval to inform the user that the data has been successfully stored (as shown in Figure 4.6 c)).

4.6.2 CCNAgent application GUI

CCNAgent and CCNSynchronization applications are handled in the same GUI, shown in Fig 4.7. The user needs to enter the IP address of the home repository as well as the collection prefix for the synchronization. Also, he or she needs to enter the Write Interval and Read Interval before starting the agent retrieval process. These values correspond to the *writeInterval*, i.e. the time the agent waits before asking the repository to retrieve the content, and the *readInterval*, i.e. the time the agent waits before it checks if the missed segment is already stored in the repository. More details on these intervals can be found in Section 4.3.2.

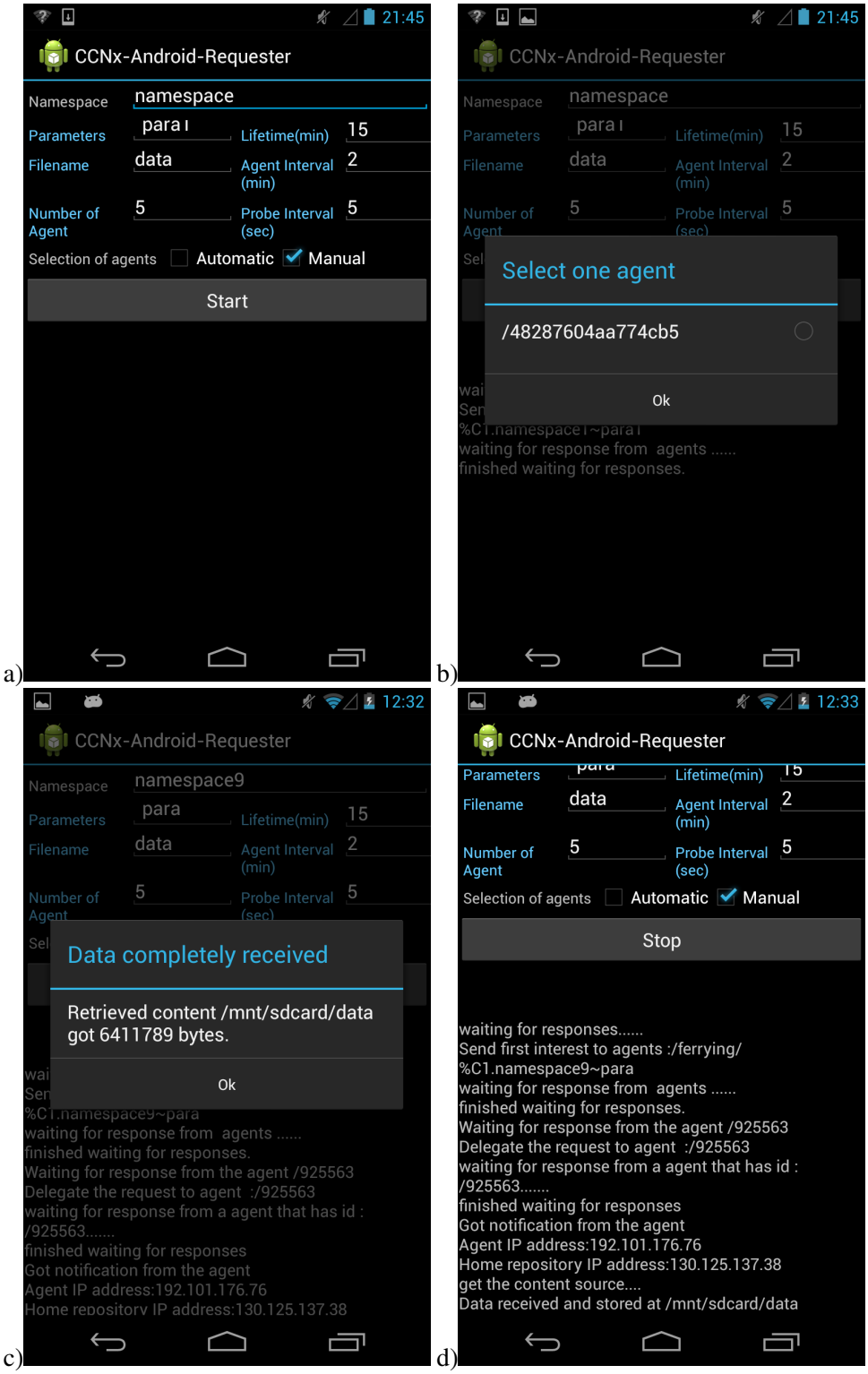


Figure 4.6: CCNRequester application GUI

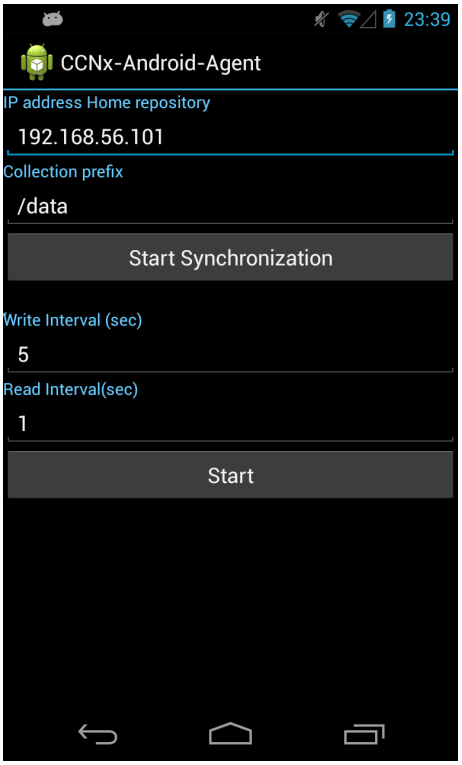


Figure 4.7: CCNAgent and CCNSynchronization applications GUI

4.7 Deployment on Smart Phone: Ad-hoc Networking Support for Android

Unfortunately, Android does not support ad-hoc networking. With Android devices as delivered in smart phones, is neither possible to connect to an ad-hoc network nor create a new one. It was therefore required to manually root Google Nexus 4 devices, which are the devices used during this thesis, to get root access on the underlying Linux system and try to bypass that restriction.

With Android's Linux tools, it is possible to overcome the lack of support for ad-hoc networking by manually configuring the network.
To create a new ad hoc network, the iwconfig program from the Wireless Tools for Linux package was therefore added to the phone. Afterwards, the interface can be put into ad-hoc mode by configuring the ad-hoc mode using the appropriate sequences of iwconfig commands as described in Appendix 7.1.1). By that, it is possible to create an ad hoc network and connect to this network from any laptop or non-Android device. However, the Google Nexus 4 devices cannot connect to an existing ad-hoc network. The following techniques were tried as suggested online but without any success. These techniques require a rooted devices and su (super-user) priviledges which need to be added to the device as explained in 7.1.5.

* **Manually configure the wpa_supplicant to connect to an ad hoc network**
  This method uses the wpa supplication settings file to force the driver to go to ad-hoc mode. So wpa_cli utility is used for interacting with wpa_supplicant and needs to be added to the phone. In Appendix 7.1.2, description of how to apply this method is given. We applied this method in the Google Nexus 4 devices: the wpa_supplicant attempts to connect to the network but never succeeds.

* **Apply "Enable Wifi Ad Hoc mode" application**
  It enables WiFi Ad Hoc mode on supported Android devices. This application replaces wpa_supplicant with another one, which supports ad hoc network. However, the current application is not compatible with the Google Nexus 4 device. So the wifi cannot be enabled with the new wpa_supplicant.

* **Edit the wpa supplicant.conf**
  This method modifies directly the configuration file for the wpa_supplicant as described in Appendix 7.1.3, the result of applying this method in Google Nexus 4 devices is the same result found in the first method: the wpa_supplicant tries to connect but it never succeeds to establish the connection.

* **Modify the android source code to enable ad hoc wifi**
  Edit the source code of the wpa_supplicant as well as the wireless settings. Afterwards, the new code source can be built and the new image system can be flashed to the phone, see Appendix 7.1.4 for more details. After flashing the new image system to the Google Nexus 4, it was able to see the ad hoc network in the Wi-Fi settings, but could not connect to it.

All these methods were applied in Google Nexus 4 devices, but none of them fixes the issue of not being able to join existing ad hoc networks. According to the posts in forums [26] and [27], Nexus 4 uses a different Wi-Fi chipset, and this could be the reason why these methods do not work. The forums [26], [27] and [28] have already started to discuss about the problem of ad hoc network in Google Nexus 4 devices, but it has not been solved yet. We have also introduced our issue of not being able to join existing ad hoc networks with Google Nexus 4 devices in forums Android Enthusiasts [29] and Android Central [30], but unfortunately the proposed methods from the members of these forums did not fix it. So, this question is still open.

Chapter 5

Evaluation

In this chapter we measure the performance of the developed CCNAgent, CCNRequester and CCNSynchronization applications. In addition, we investigate:

* The throughput difference between a normal content retrieval, e.g. using the ccngetfile application, and the content retrieval through an agent.

* The influence of the parameters: Probe Interval, Agent Interval.

5.1 Preliminary Results

In this Section, we present the evaluation results of the two approaches to resume disrupted transfers. As explained in Section 4.4, there were two possibilities to avoid or reduce the received duplicates when reading content from the repository for a resume operation:

- Approach 1: Forcing the CCND to not forward any Interest from the repository when loading the already received segments.
- Approach 2: Starting the loading process a bit before than the repository's retrieval process.

We evaluated both approaches with a file size of 4MB, and a variable delay, which equals the number of already stored segments multiplied by 10ms. The results are shown in Figure 5.1. The y-axis shows the number of received duplicate content objects.

Figure 5.1 shows that approach 2 results in 41% more received duplicate content objects than approach 1. Therefore, in the following evaluations we use approach 1. As we can see in Figure 5.1, even with approach 1, when the prefix is unregistered from the multicast face and no Interests can be forwarded to neighbors during the loading process, still some duplicate content objects are received. In approach 1, the duplicate content objects are not generated during the loading process but received during the regular retrieval of segments from the content source. If content objects get slightly delayed until they are returned to the repository of the agent, the Interest timeout may be reached and Interests are reexpressed resulting in content duplicates.

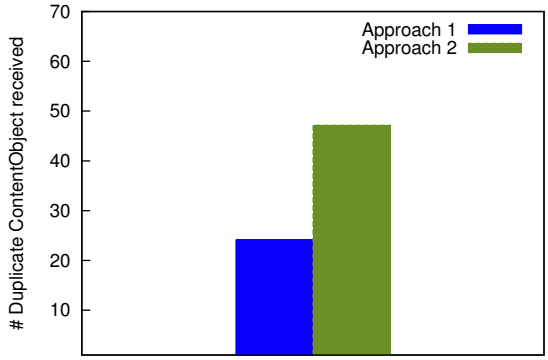


Figure 5.1: Number of received duplicate content objects for a file size of 4MB.

5.2 Topology

The evaluation is performed on Android smart phones, i.e. Google Nexus 4 devices, and a laptop, i.e. an Apple MacBook Pro. The smart phones were running Android 4.2.2. Three Nexus 4 devices were used, two of them as agent nodes and the third one as requester. Since the Nexus 4 devices do not yet support ad hoc networking, the evaluation is performed in infrastructure mode. The evaluations are performed in three scenarios:

1. Scenario 1: Figure 5.2 a), illustrates the topology of the scenario. Furthermore, it shows the assigned frequencies and the used IP address configuration. It shows a network with three CCN nodes: two smart phones and a laptop.

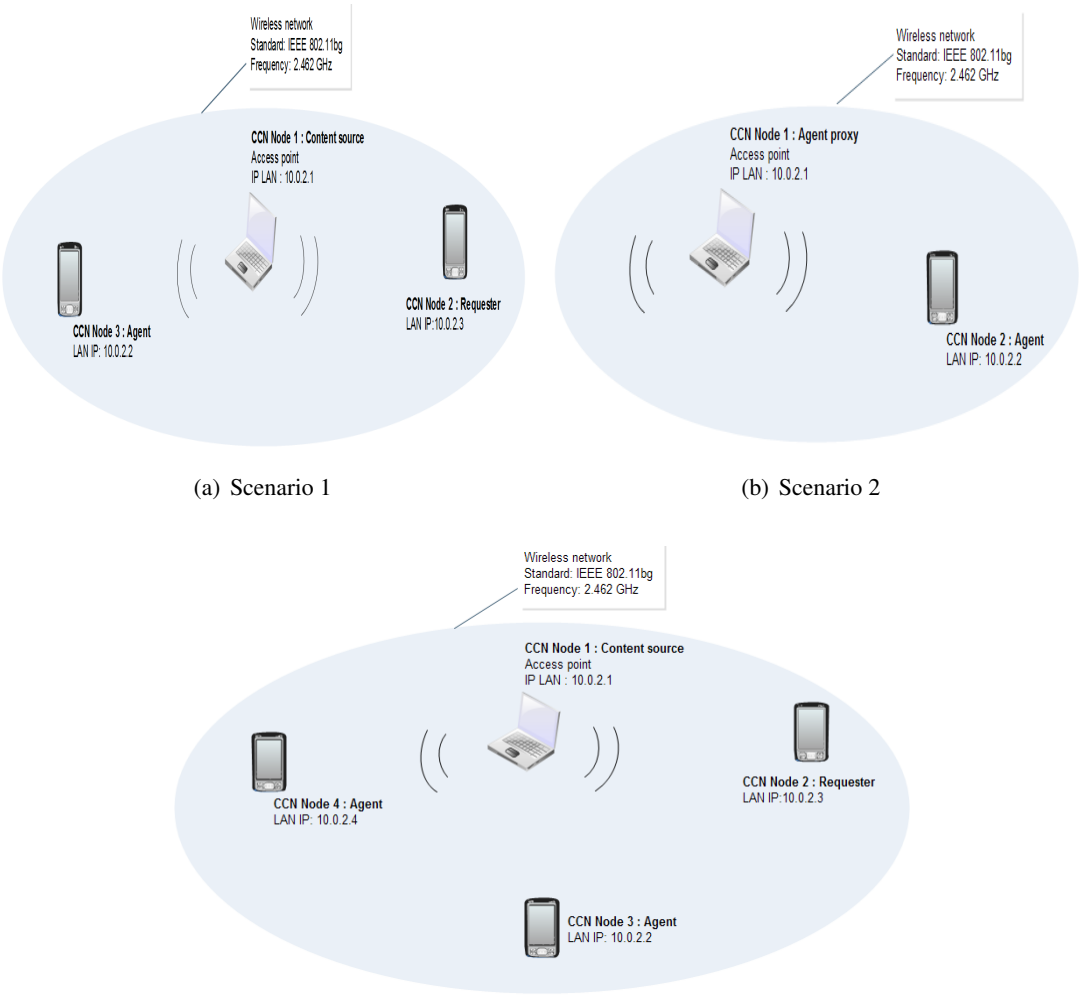
    * Laptop (CCN node 1) acts as a content source
    * Smart phone 1 (CCN node 2) acts as requester node
    * Smart phone 2 (CCN node 3) acts as agent node

2. Scenario 2: Figure 5.2 b), illustrates the topology of the scenario. It shows a network with two CCN nodes: a smart phone and a laptop.

    * Laptop (CCN node 1) acts as agent proxy, i.e. home repository.
    * Smart phone (CCN node 2) acts as agent node
      In this scenario, we evaluate the synchronization between the agent proxy and the home repository.

3. Scenario 3: Figure 5.2 c), illustrates the topology of the scenario. It shows a network with three CCN nodes: two smart phones and a laptop:

    * Laptop (CCN node 1) acts as a content source
    * One smart phone (CCN node 2) acts as requester node
    * Two smart phones (CCN node 3 and 4) act as agent nodes



(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

Figure 5.2: Testing architectures

5.3 CCNx configuration and Network Setup

The CCNx-Android-Services application, a wrapper for the ccnd daemon and the repository, is installed on every Android device. CCNx-0.7.1 is installed on the laptop and the smart phones. The configuration of CCNx, such as the definition of forwarding entries, is automatically performed by the CCNAgent, CCNRequester and CCNSynchronization applications on the Android devices. At start up of CCNAgent application, a multicast face is created with port 59695 and the prefix /ferrying is registered to it. At each delegation, the CCNAgent application registers the content name prefix to the multicast face and unregisters it after sending a notification or when the request lifetime has passed. More details can be found in Section 4.1. At start up of

CCNRequester application, the multicast face is also created with the same port and the prefix /ferrying is registered to it as well as the prefix /groupID. After receiving a notification containing the IP address of the agent's mobile repository, the CCNRequester application creates a unicast face and registers the content name prefix to this face. More details are given in Section 4.2. The FIB on the laptop needs to be configured manually. The following configurations are used:

* In scenario 1 and 3, when the laptop acts as content source, we have to manually configure the FIB to support multicast overhearing. Thus, a multicast face needs to be configured in the laptop with the following command:

  ```
  ccndc add / udp 224.0.23.170 59695
  ```

  The prefix "/" matches all possible prefixes and means that everything is forwarded to the multicast face.

* In scenario 2: when the laptop acts as agent proxy, at start up it is required to manually define the collection that needs to be synchronized using the following command:

  ```
  ccnsyncslice -v create ccnx:/topo ccnx:/data
  ```

  The prefix */topo* is the topological prefix used by Sync Agents to exchange information about the collection such as Root Advise Interests.

  The prefix */data* is the collection prefix, i.e. the common prefix for all names in the collection to be synchronized with the agent's mobile repository. All content objects inside the collection are synchronized between home and mobile repository.

  Therefore, the same */data* prefix needs to be used by the agent node at start up of the synchronization to define the collection to be synchronized with the home repository. Additionally, because synchronization may only be performed when connected to the Internet, the agent proxy needs to add a unicast face on the FIB with the IP address of the agent node using the following command:

  ```
  ccndc add ccnx:/ udp 10.0.2.2
  ```

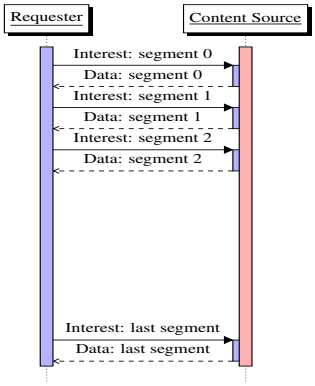
  The IP address 10.0.2.2 belongs to the mobile agent node.

  In the evaluation of scenario 1 and 3, a simple modification in the CCNRequester application is required: at content retrieval phase, when the agent retrieves the content from the content source over the multicast face, the requester will also receive the content because it is in the same infrastructure network (cf. fig. 5.2 a) and 5.2 c)). Therefore, the requester may hold the content in its cache already prior to receiving the notification. To avoid retrieving the content from the cache, emulating the situation where the content source and the requester are not in the same network, the Interest messages set the AnswerOriginKind field to 0 indicating that no answer from the content store is accepted. This modification is required only in the evaluation setup of this thesis because all the nodes are in the same network and the transmission is performed over multicast.

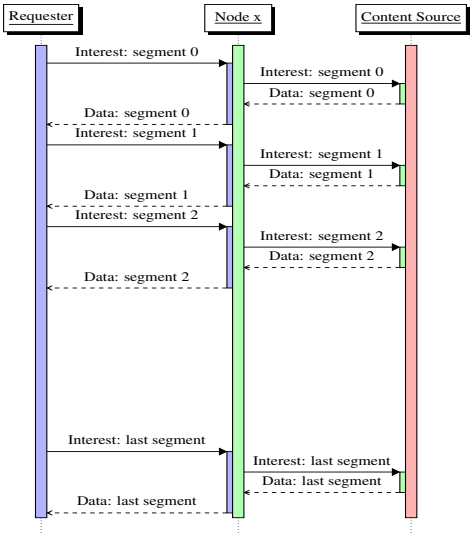
5.4 Evaluation Results

5.4.1 Scenario 1: Content Retrieval via an Agent

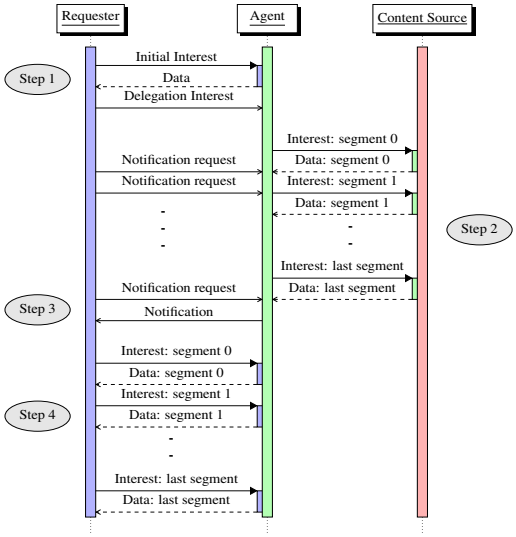
In scenario 1, we evaluated the complete agent-based content retrieval. Figure 5.3 c) shows the message sequence between all nodes used in this scenario. The evaluated mechanism includes agent delegation (step 1), content retrieval (step 2) and notification (step 3) until the requester receives the entire content (step 4). The used segment size is 4096 Bytes.



(a) Normal content retrieval using ccngetfile application over one hop



(b) Normal content retrieval using ccngetfile application over two hops



(c) Agent Retrieval

Figure 5.3: Content retrieval using ccngetfile application over one-two hops and through an agent

Parameters of CCNRequester application	
Probe Interval	5s
Agent Interval	2min
Number of agents	2
Parameters of CCNAgent application	
Reading Interval	1s
Writing Interval	30s

Table 5.1: Evaluation settings of scenario 1

5.4.1.1 Unicast vs Multicast Communication

In the default configuration of the implemented mechanism, the transmission of messages, between requester and agent in step 1 and between agent and content source in step 2, is performed using multicast communication. However, between requester and agent in step 4, the message transmission is performed using unicast communication, because the agent selects a single agent and uses the IP address received in the notification Data. In this evaluation, we intend to evaluate the throughput difference of the implemented mechanism using the following communication modes between the agent and the content source:

1. First, we used the default configuration of the implemented mechanism, where the transmission between the agent and the content source, the transmission of Initial and Delegation Interests at step 1 and the transmission of notification requests at step 3, are performed via multicast communication. The content retrieval by the requester in step 3 is performed using unicast communication.

2. Second, we kept the same configuration as in the previous test except the transmission between the content source and the agent at step 2, it is performed via unicast communication for this test. Therefore, a unicast FIB entry with the IP address of the content source is registered manually on the agent's FIB.

We measured the throughput for file sizes of 200KB, 1MB, 4MB, 10MB, and 20MB. For each file size 50 test runs were performed. The evaluation parameters are shown in Table 5.1. A short description of the parameter meanings can be found in Tables 4.1 and 4.2.

	File size				
	200KB	1MB	4MB	10MB	20MB
Multicast	66	137	175	197	195
Unicast	84	301	719	931	1020

Table 5.2: Medium values of the throughput in kbps.

Figure 5.4 shows the throughput when transferring files of different sizes until the requester retrieves the content. The y-axis shows the throughput in kilobits per second (kbps). The x-axis displays the transferred file sizes. Table 5.2 shows the median values of the throughput.

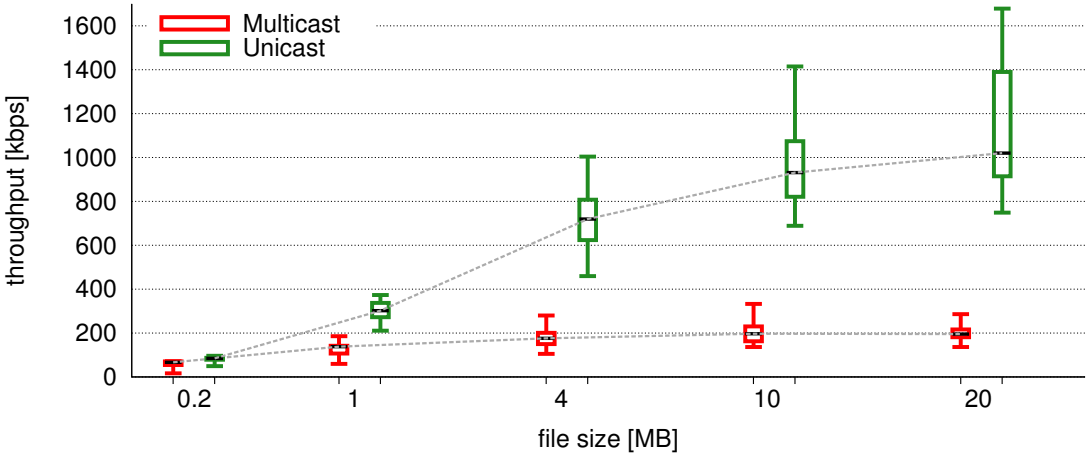


Figure 5.4: Throughput of the implemented mechanism tested with unicast and multicast communication between the agent and the content source.

As one can see, the throughput of the implemented mechanism in multicast communication is at around of 200 kbps for files larger than 1MB. However, depending on the file size the throughput in unicast communication is between 0.5 and 5 times higher for file sizes of 1MB and 20MB. The throughput is higher in unicast communication because of a faster transfer rate due to rate adaption and faster retransmission in case of collisions. For small file sizes of 200KB, there is no difference between unicast and multicast communication. This is because, the message overhead during step 1 and step 4, which is performed in multicast for the both tests, is larger compared to the data transmission during step 2.

5.4.1.2 Agent Retrieval vs Regular Retrieval over two hops

We compare the content retrieval through an agent with a regular content retrieval, e.g. using the ccngetfile application. In the first test, we performed the forwarding of Interests via unicast on the first hop and multicast in the second hop. In the second test, we performed the forwarding of Interest using unicast communication in both hops.

5.4.1.2.1 Multicast and Unicast communication

In order to compare the agent-based content retrieval with Interest forwarding using ccngetfile, we performed the following three tests:

1. We measured the throughput of the content retrieval using the ccngetfile application over one hop between the requester and a content source via multicast. The requester gets the content directly from the content source as shown in Figure 5.3 a). This measurement is only performed as reference.

2. We measured the throughput of the content retrieval using the ccngetfile application over two hops communication as shown in Figure 5.3 b). The requester asks node x for the

content and node x forwards the Interests to the content source. Figure 5.5 illustrates the topology used for this test: First, the requester has a unicast entry in the FIB with the IP address of node x. Then, the requester sends Interests out over the unicast face, i.e. face2. Interests received by unicast will cause node x to automatically create a unicast face with the IP address of the sender and the replies are automatically transmitted via this face. Second, a multicast face, i.e. face1, is configured at node x to forward Interests via multicast to a (unknown) content source. We used unicast communication in the first hop between the requester and node x and multicast communication in the second hop between node x and the content source. The scenario is configured like that to generate a scenario similar to the agent-based content retrieval where content retrieval between agent and content source (step 2 in Figure 5.3 c)) is performed via multicast and between requester and agent (step 4 in Figure 5.3 c)) via unicast (see also subsection 5.4.1.1 for more information).

3. We measured the throughput of content retrieval via agent as configured in Figure 5.2 a) transmitting all messages via multicast, including step 1, 2 and 3 in Figure 5.3 c). Step 4 is performed in unicast communication.

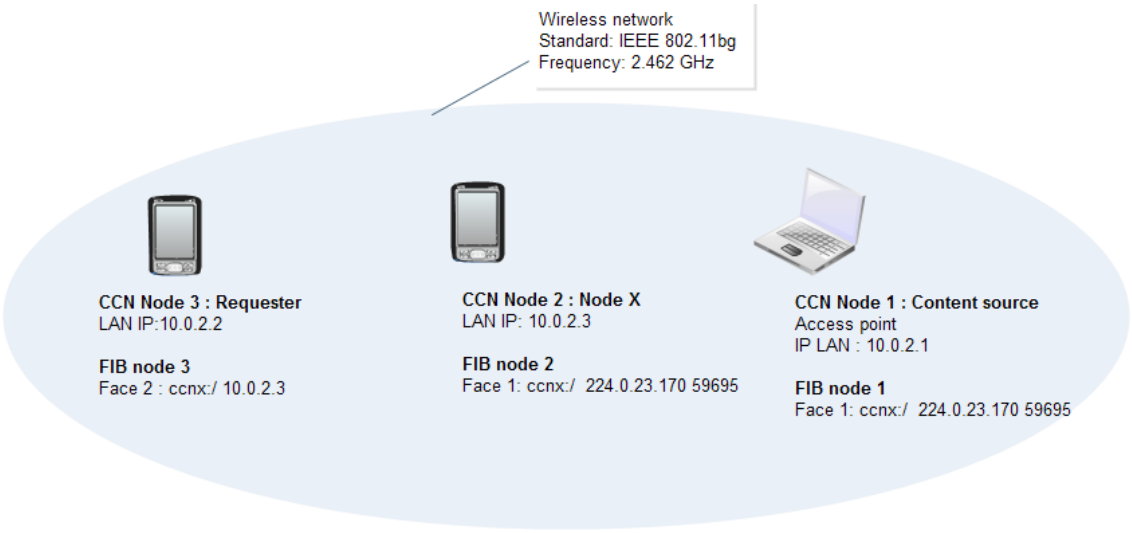


Figure 5.5: Topology used to evaluate ccngetfile application over two hops.

Figure 5.8 shows the throughputs of ccngetfile via one or two hops as described above and the implemented agent-based content retrieval. Table 5.3 shows the median values of the throughput. Obviously, the throughput of ccngetfile over two hops is approximately halved compared to the direct transmission via one hop because every Interest and Data message is transmitted twice due to intermediate forwarding. However in ccngetfile via one hop, Interests from the requester are received directly by the content source and then Data messages are transmitted directly from the content source to the requester. When comparing the content retrieval

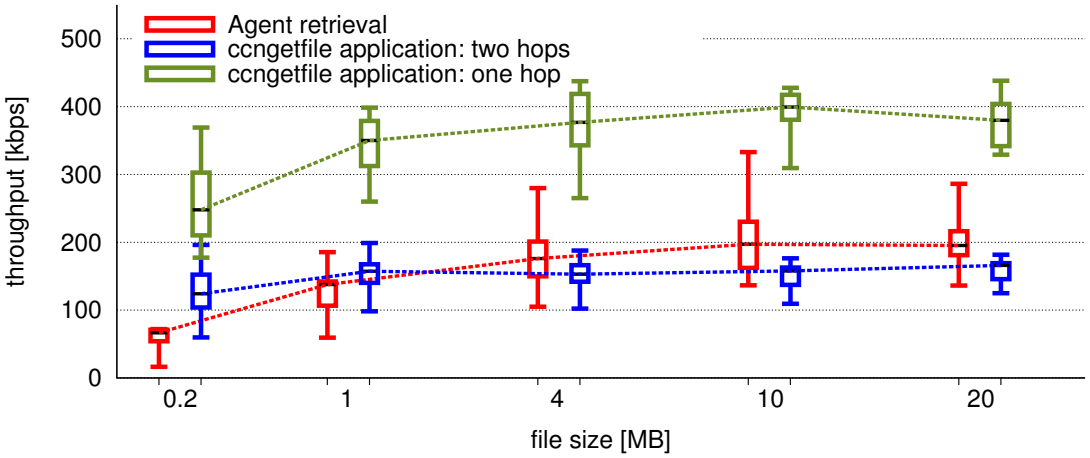


Figure 5.6: Throughput of the implemented mechanism compared with ccngetfile content downloads. The communication is performed via unicast and multicast according to the test setups.

	File size				
	200KB	1MB	4MB	10MB	20MB
Agent retrieval	66	137	175	197	195
ccngetfile : two hops	124	157	152	157	165
ccngetfile : one hops	247	350	376	399	379

Table 5.3: Medium values of the throughput in kbps.

via an agent and the normal retrieval over two hops, as we can see in Figure 5.8, with ccngetfile over two hops, the agent-based retrieval results in approximately 20% higher throughput for files larger than 4MB. However, for file sizes smaller than 1MB, ccngetfile results in higher throughputs than the agent-based content retrieval. This behavior can be explained with the help of the message sequence of ccngetfile in Figure 5.3 b) and the agent-based retrieval shown in Figure 5.3 c). In the agent-based content retrieval, the two hops file transfer is performed subsequently. First, the agent retrieves the content from the content source via the multicast communication (step 2) and after the file is finished and the requester has received the notification, it can retrieve the content from the agent via unicast (step 4). Therefore, the transfer rate on each hop can reach the maximum capacity. With ccngetfile over two hops in Figure 5.3 b), the file transfer via node x is performed in parallel and not subsequently as with the agent retrieval. Node x receives an Interest from the requester via the unicast face and forwards it to the content source via the multicast face. Then, the transmission is limited by the ”weakest link” (the smallest transfer rate, which is the multicast transfer rate between agent and content source). For large files this behavior is problematic because unicast requests are only satisfied if the forwarded multicast requests are satisfied as well. Therefore, complete file transfer is limited by the multicast transfer speed and unicast Interests may timeout and need to be reexpressed decreasing the throughput. For smaller file transmissions below 1MB, the relative overhead for agent delegation in step 1 and

notification in step 3 is higher and therefore the agent-based retrieval results in a lower throughput than with ccngetfile where no delegation and notification is performed. The throughput has been decreased for file size 1 MB by 20% and for 200 KB by 80%.

5.4.1.2.2 Unicast Communication in two hops

In this part, we compare the agent-based content retrieval with the regular retrieval using the ccngetfile application if all data communication on both hops (except agent delegation and notifications in the agent-based approach) is performed via unicast. We performed the two following tests:

* First, we measured the throughput of the content retrieval using the ccngetfile application over two hops communication as shown in Figure 5.3 b). However in this test, the transmission between requester and node x and between node x and the content source is performed using unicast communication. Figure 5.7 illustrates the topology used for this test. It shows also the FIB configuration of each node: The requester has a unicast entry in the FIB with the IP address of node x. Then, the requester sends Interests out over the unicast face, i.e. face2. At node x a unicast face to the content source, i.e., face 1 is configured.

* We measured the throughput of the agent-based content retrieval as configured in Figure 5.2 a). The FIB is automatically configured by the applications. The agent delegation and content notification are performed via multicast. After receiving a notification, the requester performs the content retrieval via unicast. The content retrieval between agent and content source (step 2 in Figure 5.3 c) is also performed via unicast. This test was only performed to compare agent-based content retrieval with ccngetfile. In practice, an agent would not be able to forward requests to a content source via unicast because the IP addresses of available content sources are not known.

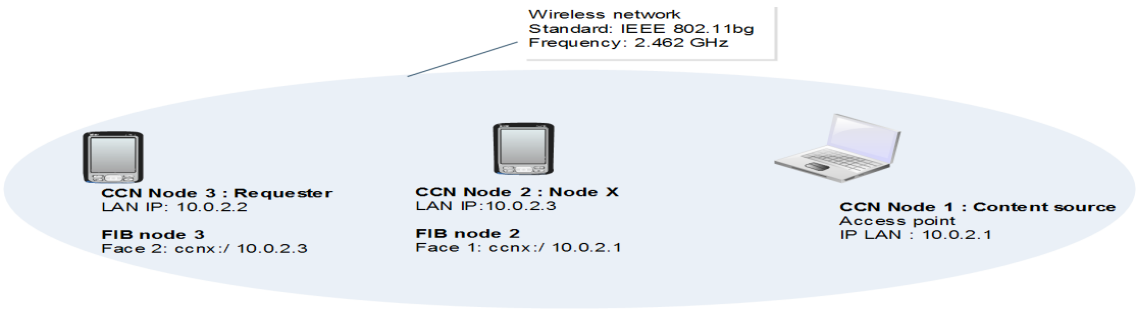


Figure 5.7: Topology used to evaluate ccngetfile application over two hops unicast communication.

Figure 5.8 shows the throughputs of ccngetfile via two hops as described above and the agent-based content retrieval. Table 5.4 shows the median values of the throughput. When using unicast communication over both hops, the throughput of ccngetfile is by a factor of 2.5-3 higher

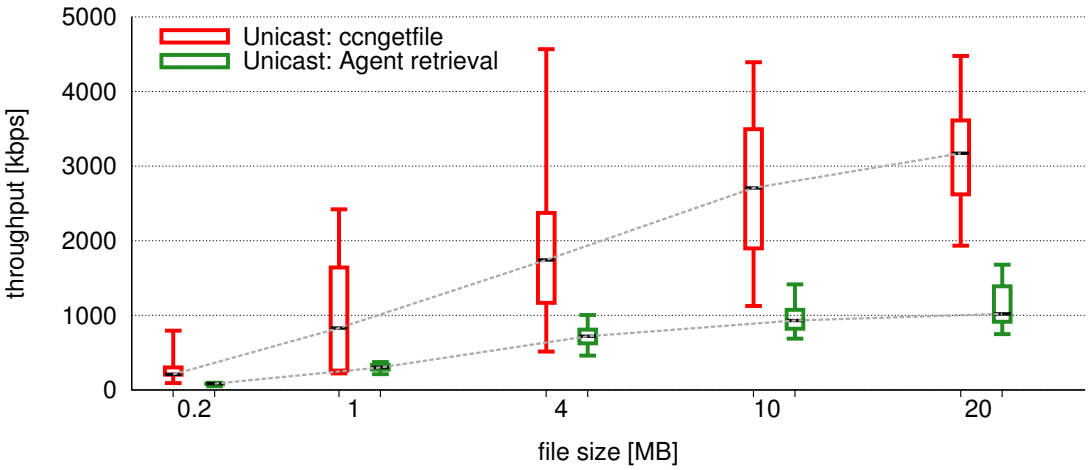


Figure 5.8: Throughput of the implemented mechanism compared with ccngetfile content downloads. The communication is performed via unicast according to the test setups.

	File size				
	200KB	1MB	4MB	10MB	20MB
Agent retrieval	84	301	719	931	1020
ccngetfile : two hops	208	828	1742	2707	3171

Table 5.4: Medium values of the throughput in kbps.

than the agent-based content retrieval. This is because in the agent based content retrieval, the agent's repository needs to store each received segments, which is on secondary storage, while everything can be forwarded via the cache with ccngetfile. Storing files in a repository is an extra effort, which is performed in addition to the cache. If file transfers are performed subsequently as with the agent-based approach, some segments may not be in the cache of the agent anymore due to cache replacements and need to be retrieved from the repository, particularly, if transmitting large files. Because repository storage is slower than the content store, the overall throughput is slower. When using the ccngetfile application, node x can forward content directly via the cache. For small files of size 200KB or 1MB, the agent may also forward content to the requester via its cache. However, for small files, the overhead of agent delegation and notification is larger compared to the effectively transmitted data. Therefore, ccngetfile is also more efficient for small files. However, in dense networks where many data objects are requested from many content sources, cache replacements would be performed even faster. If the content is stored additionally on secondary storage as in the agent-based approach, similar requests from other requesters do not need to be forwarded to the original content source and can be directly satisfied by the agent's repository. This would result in a higher throughput compared to ccngetfile.

5.4.1.3 Message Exchange

In scenario 1 as configured in Figure 5.2 a), we measure also the transmitted Interests and received ContentObjects at the agent node. Figure 5.9 illustrates the transmitted and received messages at the agent. It comprises the Interests sent from the agent to the content source, the received Interests from the requester, the received ContentObject from the content source, the ContentObject sent to the requester and the number of received duplicate ContentObjects at the agent. We performed 50 test runs to transfer a 200KB, 1MB, 4MB, 10MB and 20MB file. The values for the probe interval, agent interval, reading interval and writing interval are the same as in Table 5.1. The y-axis shows the number of messages. The x-axis shows the transferred file sizes. Table 5.5 shows the medium values of the exchanged messages.

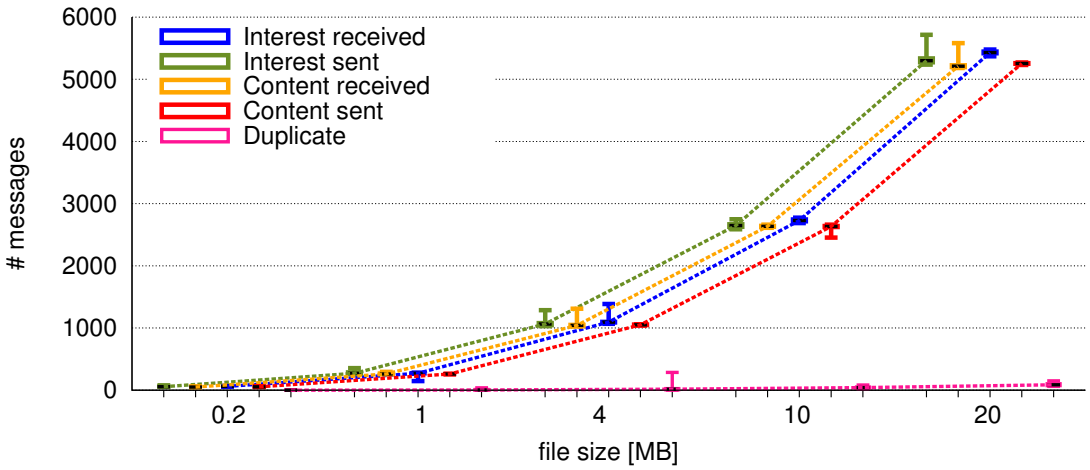


Figure 5.9: Message exchanged at the agent node tested using different file sizes.

The number of received Interests at the agent is slightly higher. The agent transmits an Interest for every content segment in the content retrieval phase (step 2 in Figure 5.3 c)) and receives an Interest from the requester for every content segment after the notification (step 4 in Figure 5.3 c)). Additionally, the agent receives at least two Interests during the Agent delegation phase (step 1 in Figure 5.3 c)), i.e. the *initialInterest* and *delegationInterest*. Depending on the transmission time of the content, the agent can even receive more Interests, namely, one Interest after every agent interval, i.e., in this evaluation set to 2min, with which the requester asks for a new agent, and one Interest every probe interval, i.e., in this evaluation set to 5s, with which the requester asks for a notification from existing agents (step 3 in Figure 5.3 c)).

Therefore, the number of received Interests increases for increasing file sizes because of longer transmission times. As shown in Table 5.5, the number of received Interests is higher at the agent by around 2.4-7% for all transferred file sizes, expect the file size of 1MB. Since the communication between the agent and content source is performed via multicast, it is more susceptible to collisions than with unicast where retransmission is performed automatically by the MAC layer. Therefore, the agent may send more than one Interest for each segment and then the number of transmitted Interests may be larger than the number of received Interests. Thus,

the difference between received and transmitted Interests may be less than three (initialInterest, delegationInterest, notification request). In this evaluation, the agent interval was set to 2min. If file transmission takes longer than two minutes, the agent will receive an additional initialInterest but will not answer it because it already replied to the agent delegation.

The numbers of received and transmitted content objects are approximately the same. There are two more content object transmissions than receptions. The agent will receive and send a Data message for every received segment. Every selected agent transmits an additional Data message during the agent retrieval (step 1) and one Data message as notification reply after the file is completely retrieved (step 3).

file size	Interests sent	Interests received	ContentObjects received without duplicate	ContentObjects sent without duplicate
200KB	57	61	51	53
1MB	271	272	257	259
4MB	1063	1095	1025	1027
10MB	2648	2729	2561	2563
20MB	5299	5429	5121	5123

Table 5.5: Medium values of exchanged messages in scenario 1.

5.4.1.4 Influence of Probe Interval

In this section, we evaluate the influence of the probe interval on the transfer time and on the number of the notification requests sent by the requester in scenario 1 shown in Figure 5.2 a). Figure 5.10 shows the relation between the probe interval and the transfer time. It illustrates also the steps performed during the agent-based content retrieval process in this scenario. The requester starts by transmitting Initial Interest, i.e. Int1. At this point the transfer time is started. After receiving a reply from agent 1, i.e. Data1, it transmits Delegation Interest to it. When the agent receives this Interest, it can start requesting the content and therefore, the content retrieval time on the content source starts. After the requester has delegated the content, it probes every probe interval the environment to see whether one of the delegated agents has received the corresponding content and can reply with a notification. After an agent interval of not receiving a notification, the requester can delegate the content retrieval to a new agent. This procedure is repeated until a notification is received or the request lifetime is over.
The transfer time can be expressed by the formula:

$$t_t = t_{rc} + t_{dc} + t_{ra} + t_{dr} \tag{5.1}$$

where
t_t : total transfer time
t_{rc}: effective transfer time between agent and content source
t_{dc}: disconnection time between agent and content source
t_{ra}: time until requester retrieves the complete file from agent.
t_{dr}: time until requester detects that agent has received the complete content.

Thus, transfer time depends on 1) time until the agent sees the content source and 2) time until the requester gets the notification from the agent. Depending on the mobility of the nodes, the probe interval may result in a quicker detection of the agent (t_{dr}). With increasing probe interval, also the detection granularity increases and, therefore, possibly also t_{dr}. As the probe interval increases so that t_{dr} does. However, with decreasing probe interval, the number of transmitted notification requests increases. Every notification request will only trigger a response if an agent has retrieved the file completely.

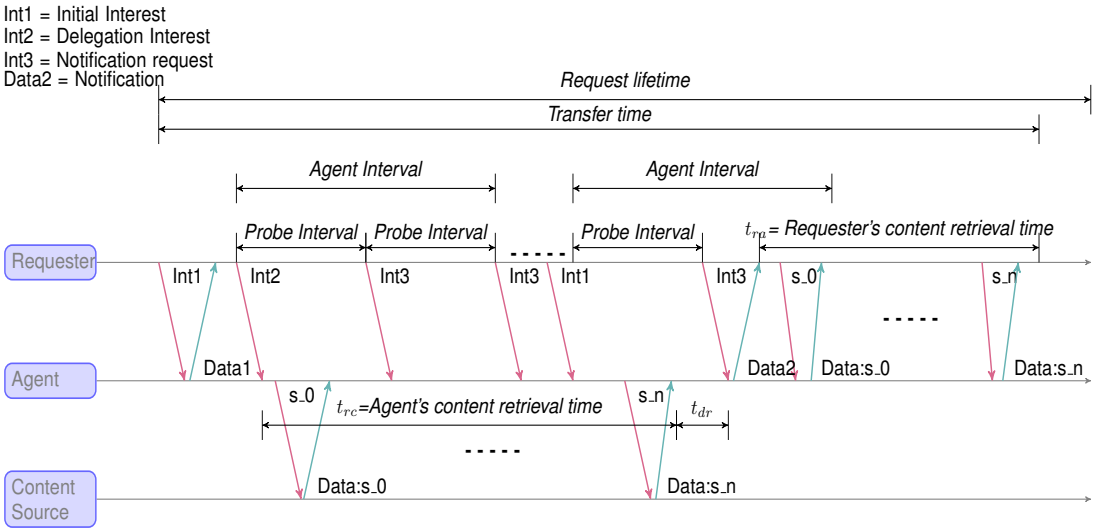


Figure 5.10: Relation between Probe Interval, Agent Interval and transfer time.

We tested the transfer time and the number of transmitted notification Interests using probe intervals of 1s, 5s, 30s, 60s and 120s and file sizes of 1MB, 4MB and 10MB. Figure 5.11 shows the number of transmitted notification requests and Figure 5.12 shows the transfer time until the requester receives a notification and retrieves the complete content. The x-axis shows the selected file sizes. Table 5.6 and 5.7 show the median values of the number of transmitted notification requests and the transfer times respectively.

	Probe interval				
File size	1s	5s	30s	60s	120s
1MB	5	6	1	2	1
4MB	23	20	4	3	2
10MB	64	41	10	6	3

Table 5.6: Medium values of the number of notification requests

As Figure 5.11 shows, the shorter the probe interval, the more notification requests need to be transmitted. For a file of size of 10MB, 56% more notification requests are transmitted when using a probe interval of 1s compared to a probe interval of 5s. And for a file size of 4MB still 15% more notification requests are transmitted. As the figure shows, a probe interval of 30s

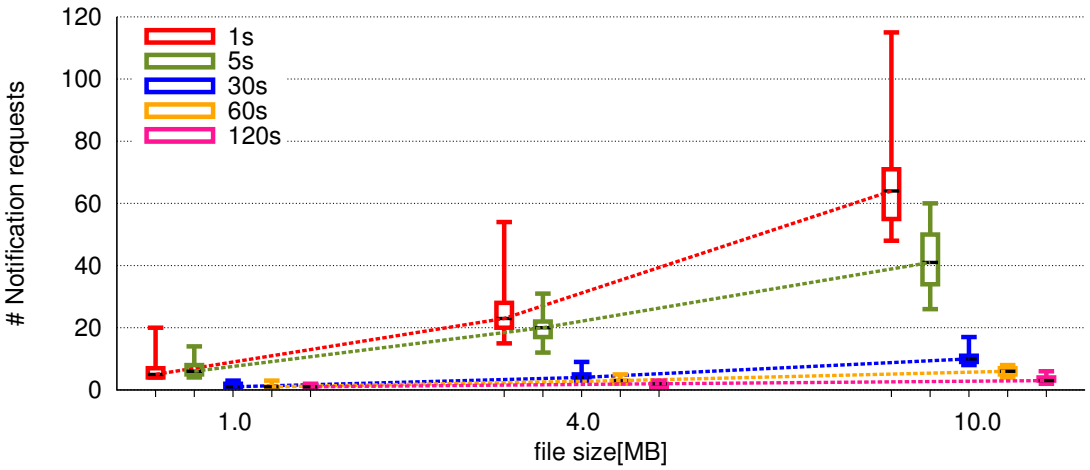


Figure 5.11: Number of notification requests using different probe intervals with different file sizes.

can result in much fewer transmitted notification requests than with 1s. For a file size of 10MB, there are 84.4% fewer notification requests. The number of notification requests when using a probe interval of 1s is around 5-6 times higher than with probe interval of 30s.
Of course, the probe interval also influences the transfer time until the requester can retrieve the complete file as figure 5.12 shows. Large probe intervals may result in large transfer times because the time until the requester detects the availability of an agent that holds the desired content may be larger, i.e. t_{dr} in Figure 5.10 may be larger. As figure 5.12 shows, a probe interval of 60s results in 84% longer transfer time than 1s for 1MB and 32% longer times for 4MB. For file transmissions of 10MB or larger, the median transfer time is approximately the same for all probe intervals because the overall transfer time is much larger compared than the differences in probe intervals. A good probe interval value seems to be 30s, because compared to a probe interval of 1s, the notification requests decrease by 80-84% for file sizes between 1MB and 10MB. At the same time, the transfer time only increases by 8% for a file size of 4MB and only 5% for a file size of 10MB.

	Probe interval				
File size	1s	5s	30s	60s	120s
1MB	45	59	57	83	143
4MB	179	187	194	237	299
10MB	435	430	458	466	480

Table 5.7: Medium values of transfer time in seconds.

5.4.2 Scenario 2: Synchronization between Mobile and Home Repository

In scenario 2 as configured in Figure 5.2 b). The topology comprises two nodes, which communicate wirelessly via a unicast face. The agent is running on an Android smartphone and

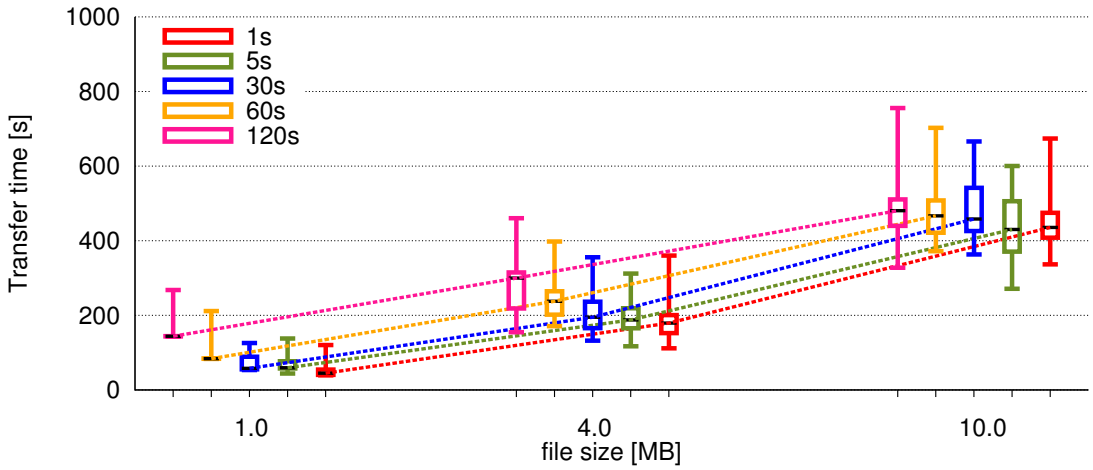


Figure 5.12: Transfer time using different probe intervals with different file sizes.

the home repository on the laptop. The throughput during the synchronization between agent and agent proxy, i.e. home repository, is measured. The intent of this test is to evaluate the synchronization between the agent proxy and the agent node when the CCNsynchronization application is used. We tested the throughput for file sizes of 1MB, 4MB, 10MB, and 20MB. For each file size, 50 test runs were performed.

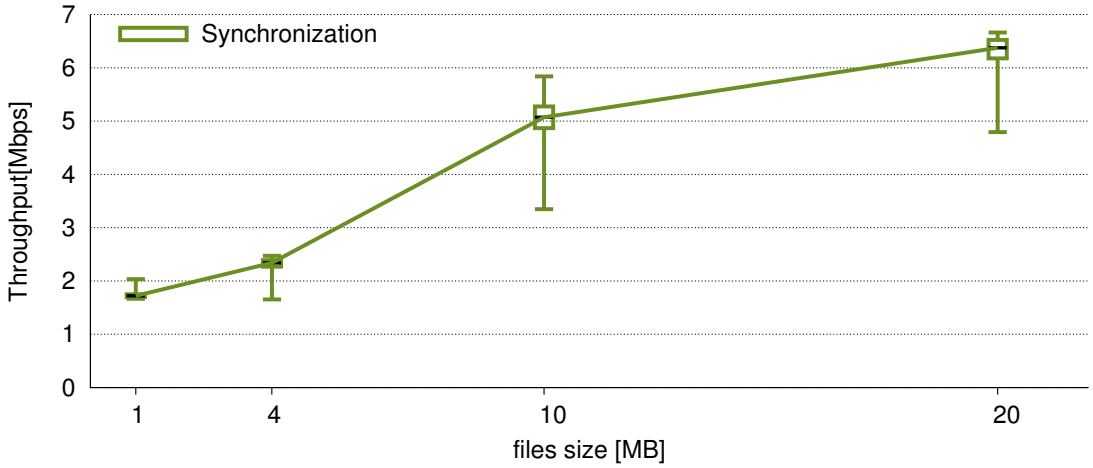


Figure 5.13: Throughput of the synchronization mechanism between the home and mobile repositories with different file sizes.

Figure 5.13 shows the throughput in Mbps during the synchronization of a file of varying size. Synchronization is performed in unicast between the agent and the home repository since the address of the home repository is always fixed. As explained in section 4.5, the agent, i.e. its

mobile repository, would need to contact the home repository prior to synchronization to inform it about its current IP address. This step is omitted here and we assume that the home repository already knows and has configured a face to the agent.

The results are shown in Figure 5.13. As we can see, the throughput from the implemented synchronization mechanism is around of 5 Mbps during the transfer of 10MB and 6 Mbps during 20MB file transfers. The throughput when synchronizing collections increases for larger file sizes because Root Advise Interests are transmitted depending on the number of unsynchronized content objects and not dependent on the content size. Therefore, in our scenario of synchronizing one file, no more Root Advise Interests need to be transmitted but only more Interests to retrieve the content.

5.4.3 Scenario 3: Influence of Agent Interval

In scenario 3 as configured in Figure 5.2 c), we investigate the influence of the agent interval. This interval is used by the requester to decide when to contact the next agent if no notification is received. We evaluate this in two cases. In Subsection 5.4.3.1, the first agent would never meet the content source emulated by an interrupt. Then, it cannot retrieve the content at all. In Subsection 5.4.3.2 both agents can retrieve the content.

5.4.3.1 Two Agents - Interruption at the first Agent

Figure 5.14 shows the message exchanged during scenario 3. The communication is performed in four steps:

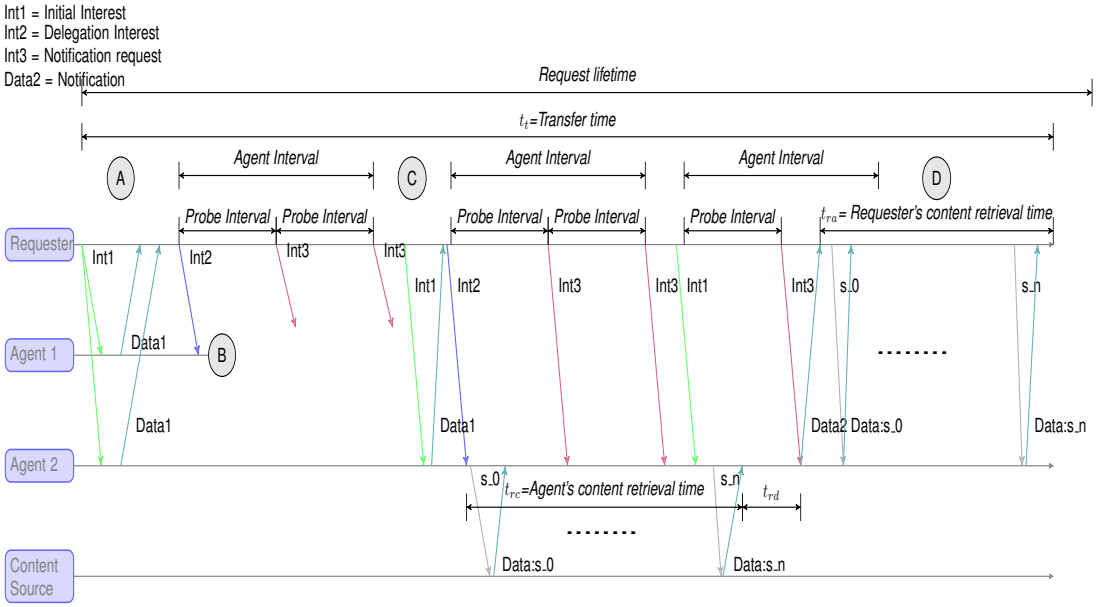


Figure 5.14: Message exchanged during scenario 3: interruption at the first agent.

1. In the initial setup there is one requester, two agents and a content source.
2. The requester transmits multicast request, i.e. Initial Interest (Int1), looking for the available agents. Then, both agents, i.e. agent 1 and agent 2, answer to the multicast request. The requester selects the agent 1 and delegates the content retrieval to it. This is done in step A in Figure 5.14.
3. Agent 1 receives the delegation Interest (Int2) but is not in range of a corresponding content source and can therefore not retrieve the desired content. This is done in step B of Figure 5.14.
4. Since the requester cannot retrieve the notification from an existing agent, i.e., agent 1, it will delegate the content retrieval to a second agent after the agent interval has passed. This is performed in step C of Figure 5.14.
5. Agent 2 will ask and retrieve the content. The requester will receive the notification after some probe intervals. After that, it can retrieve the content from the agent, which is performed in step D of Figure 5.14.

File size	4MB
Number of agents	2
Reading Interval	1s
Writing Interval	30s

Table 5.8: Evaluation settings of scenario 3

To evaluate this mechanism, agent 1 is slightly modified so that no content retrieval is performed, which is equivalent for the agent to being out of range from the content source.

The evaluation was performed for agent intervals of 1 minute, 5 minutes and 10 minutes and probe intervals of 5s, 30s and 60s. The values of the remaining parameters are shown in Table 5.8. The evaluation results are shown in the Figure 5.15 and 5.16. The x-axis shows the used agent intervals.

Figure 5.15 shows the number of transmitted notifications requests and figure 5.16 the transfer time to retrieve the content from the agent. Table 5.9 and 5.10 show the corresponding median values.

	Probe interval		
Agent interval	5s	30s	60s
1min	26	8	5
5min	53	14	9
10min	81	22	15

Table 5.9: Median values of the number of notification requests

The agent interval determines when the requester delegates the retrieval to a new agent. If the agent interval is shorter than the time a node requires to retrieve the content, too many

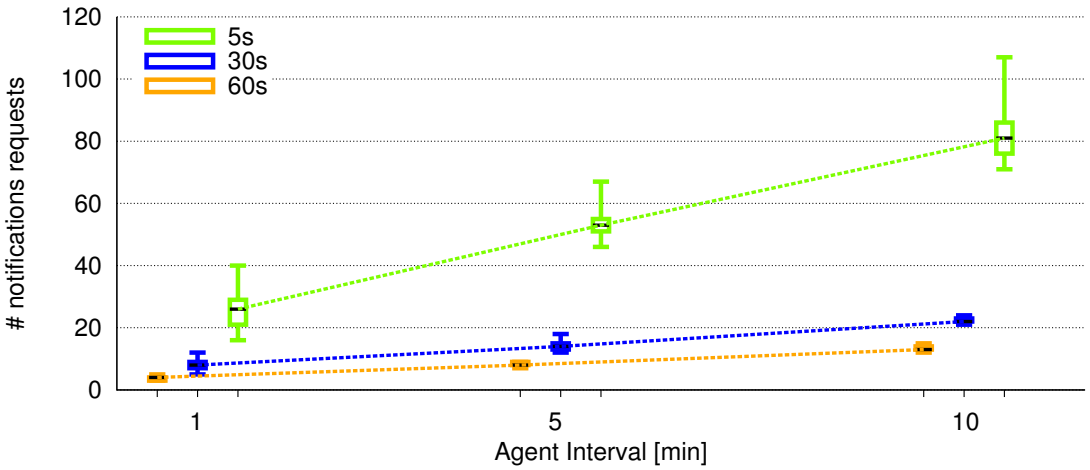


Figure 5.15: Number of notifications requests using different probe intervals with different agent intervals.

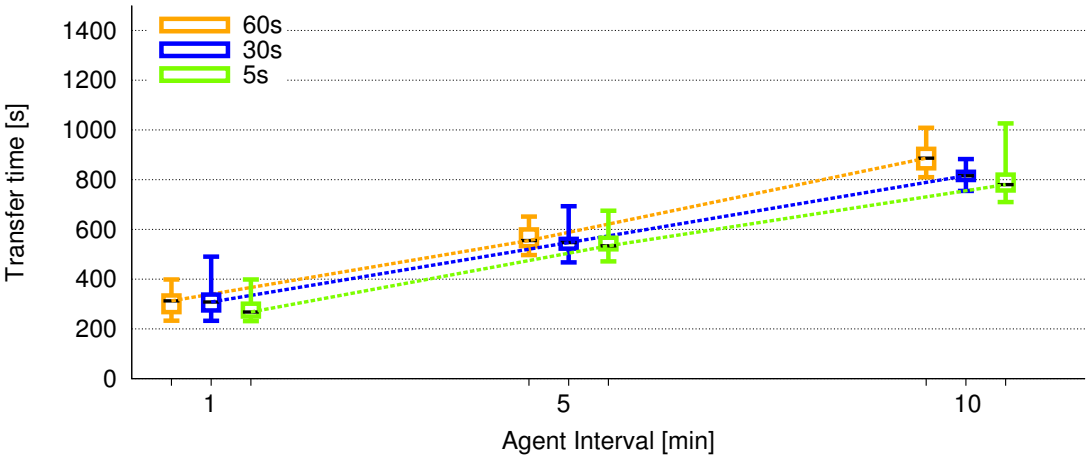


Figure 5.16: Transfer time using different probe intervals with different agent intervals.

	Probe interval		
Agent interval	5s	30s	60s
1min	268	307	312
5min	533	546	555
10min	780	815	886

Table 5.10: Median values of transfer time in seconds.

unnecessary agent delegations may be performed because also the existing agents may have retrieved the content successfully. However, if agent 1 cannot retrieve the content as in this

scenario, a shorter agent interval obviously results in a faster transfer time because agent 2 is selected quicker. In this scenario, the time required to retrieve the content by the requester can be defined by the following formula:

$$t_t = t_{ai} + t_{rc} + t_{dc} + t_{dr} + t_{ra} \tag{5.2}$$

where
t_t: total transfer time
t_{ai}: agent interval
t_{rc}: effective transfer time between agent and content source.
t_{dc}: time until agent meets the content source, in this example equals 0.
t_{dr}: time until requester detects that agent has received the complete content.
t_{ra}: time until requester retrieves the complete file from agent.

Figure 5.15 shows the number of transmitted notification for different probe and agent intervals. Obviously, the number of notification requests increases with increasing agent interval because more time is required until the content is delegated to a new agent. In the meantime, more probing requests are transmitted. According to Table 5.9, with an agent interval of 1min, the number of transmitted request messages decreases by 70% when increasing the probe interval from 5s to 30s. For an agent interval of 10min it decreases by even 73%.

In this scenario, transfer time depends on 1) time until the agent sees the content source, 2) time until the requester gets the notification from the agent and 3) agent interval (t_{ai}). As the probe interval increases so that t_{dr} does, which results in increasing of transfer time. Therefore, as Figure 5.16, the transfer time increases for an agent interval of 1min by 14.5% from a probe interval of 5s to 30s. For an agent interval of 5min, it increases by 2.4% and for 10min, it increases by 4.5%.

According to the Formula 5.2, as the agent interval (t_{ai}) increases so that transfer time (t_t) does . Therefore, as shown in Figure 5.16, for a probe interval of 30s, the transfer time increases with increasing agent interval by 78% from 1min to 5min and by 165% from 1min to 10min. Because agent 1 is not retrieving the content, a short agent interval is favorable in this evaluation test.

5.4.3.2 Two Agents - Parallel Content Retrievals

In the previous subsection, we found that a short agent interval is favorable if agent 1 is not working and in this subsection, we evaluate the same scenario but this time agent 1 can retrieve the file. Figure 5.17 shows exchanged messages and the relations between the different time intervals. As shown the Figure, in step A the requester starts by transmitting multicast requests to the available agents. In step B, the requester delegates the content retrieval to agent 1. At that time, agent1 starts retrieving the content from the content source. After an agent interval, the requester can delegate the retrieval to another agent. It waits until the next probe interval is due and then performs a new agent delegation immediately after the probe interval (if no notification is received). The requester delegates the content retrieval to agent 2 (step C) before agent 1

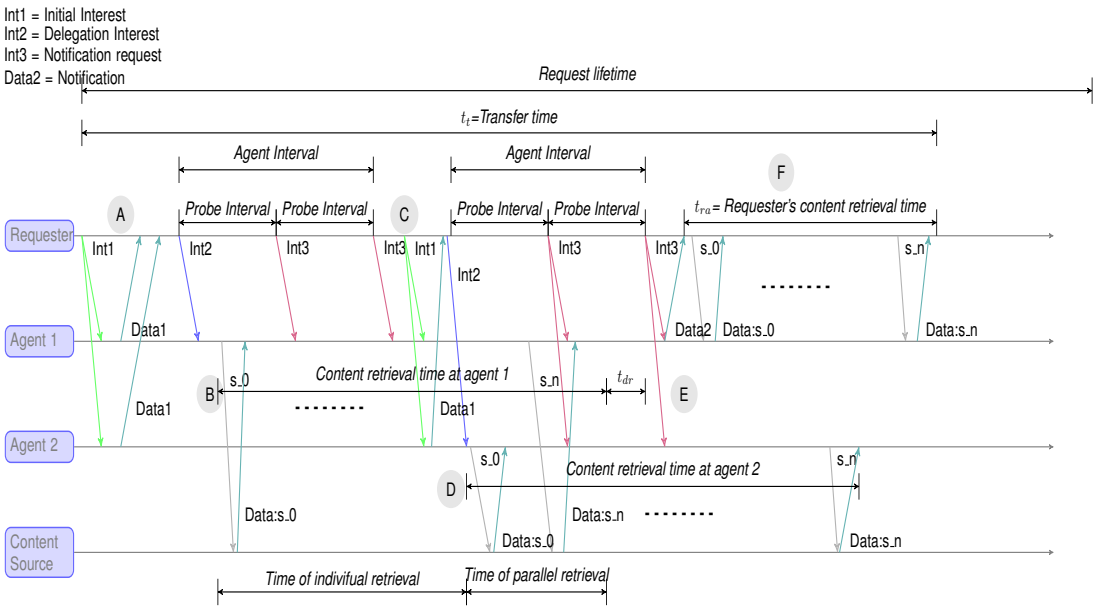


Figure 5.17: Message exchanged during scenario 3: parallel content retrievals, probe interval is smaller than agent interval.

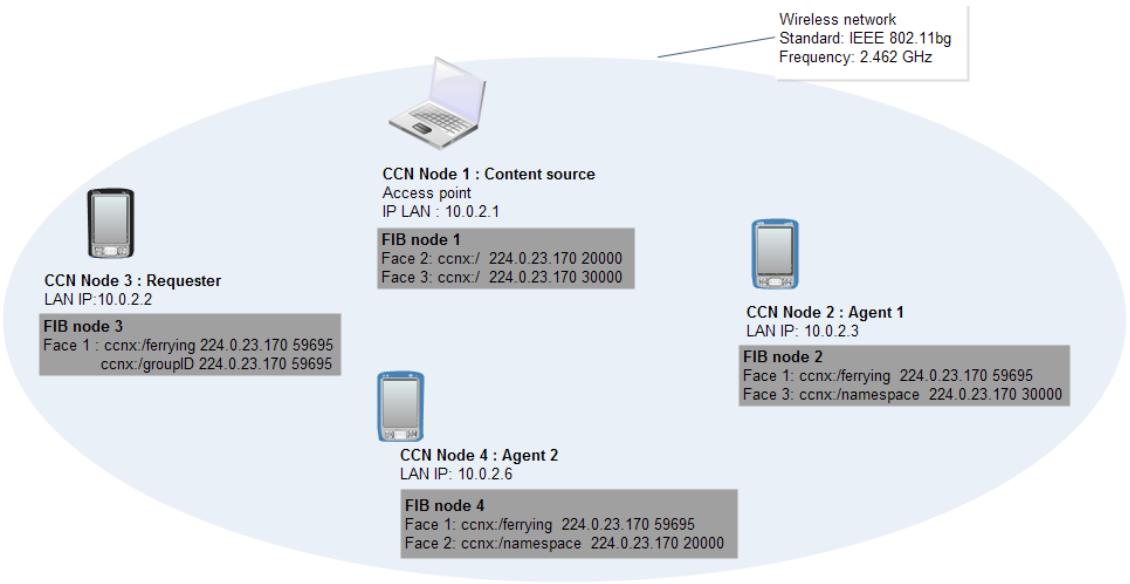


Figure 5.18: FIB configuration of scenario 3.

has received all segments. In step D, agent 2 starts also to retrieve the content from the content source. At this point, the content source transfers the content to the both agents. Therefore, resources on the medium are wasted. In step E, the agent will be finished to retrieve the content and then a notification can be sent to the requester which starts to retrieve the content from the

agent 1 (step F).

To evaluate parallel content retrieval from the same content source, in our setup we need to guarantee that the second delegated agent does not receive the content already requested from the first agent. For this reason, we configure two different multicast faces in each agent as shown in Figure 5.18. The first one, with the port number of 59695, is for listening to the requests by the requesters. The second one is for forwarding Interests to the content source and is different for both agents so that they cannot receive traffic from each other. For example, in Figure 5.18 agent 2, which listens to port 20000, can not hear the content requested from agent 1, which transmits on port 30000. The content source needs to create two multicast faces, i.e. face 2 and face 3 in Figure 5.18, with the corresponding multicast ports used by the two agents. This FIB configuration was only required for this tests so that both agents forward Interests to the content source. In a real scenario, both agents may use the same multicast port 59695.

The evaluation was performed for agent intervals of 0.5 minute, 1 minute, 2 minutes and 5 minutes, probe intervals of 5s, 30s, 60s and 120s and file size of 4MB. The values of the remaining parameters are the same as in Table 5.8. The transfer time is shown in Figure 5.19. The x-axis shows the selected agents intervals.

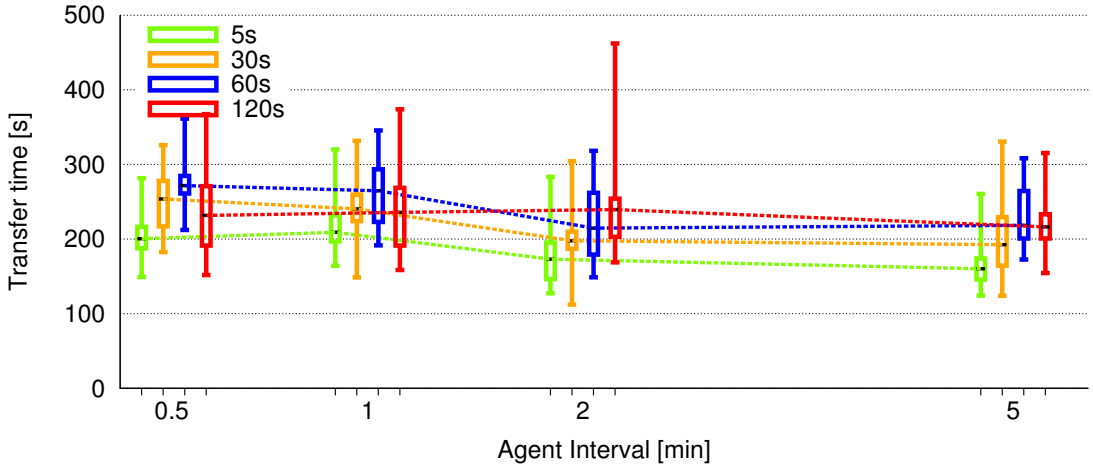


Figure 5.19: Transfer time using different probe intervals with different agent intervals.

	Probe interval			
Agent Interval	5s	30s	60s	120s
0.5min	200	253	271	231
1min	209	240	264	235
2min	172	197	214	239
5min	160	192	218	216

Table 5.11: Medium values of transfer time in seconds.

The medium values of the transfer times are also listed in Table 5.11. As we can see, for

a probe interval of 30s, the transfer time of an agent interval of 0.5 min compared to 1min is longer by 4.9% and decreases even by 24.1% for an agent interval of 5min. This is because the agent interval of 0.5 min or 1min is shorter than the time a node requires to retrieve the content. Then, a new delegation is performed to a new agent, which will trigger the content retrieval from the same content source reducing the throughput for an individual agent. Therefore, the second retrieval slows down the first retrieval by the first agent because the content source needs to reply to the both agents in parallel. As shown in Figure 5.17, the time required to agent 1 to retrieve the content from the content source can be defined by:

$$t_t = t_i + t_p \tag{5.3}$$

where
t_i: time the first agent can retrieve the content alone
t_p: time of parallel retrieval, where both agents retrieve the content in parallel and need to share the bandwith.
The shorter the agent interval, the longer is the time of parallel content retrieval (t_p), which increases the total transfer time because the bandwidth is shared between both agents for a longer time.

For agent intervals larger than 2 min, the time transfer becomes approximately constant because in most cases 2 min is enough for the agent to request the content. Therefore, in this case, the notification is replied before the content retrieval is delegated to agent 2 and t_p equals zero.

The value of the agent interval after which the transfer time is approximately constant depends on the file size to be transmitted, i.e. the transfer time. Therefore, it may be favorable for efficient agent-based file transfers to know the file sizes in advance.

When using a probe interval of 120s, the transfer time does not increase significantly. Since before every new agent delegation, also a notification request needs to be transmitted. After probe interval of 120s, the agent has most certainly already retrieved the complete 4MB file and then it will transmit a notification. This is because in the implementation of the mechanism, there is only one thread that checks the probe and the agent intervals. When the probe interval is over, a notification request is sent and then the same thread checks if the agent interval has passed to send a new delegation. Therefore, if the agent Interval is over, it is only detected after a probe interval. Thus, the agent interval depends to the probe interval. Alternatively, when using two threads, agent and probe interval could be checked independently of each other. However, this may result in the transmission of more notification requests because before every agent delegation a separate notification request needs to be transmitted to check whether an agent has already retrieved the content or agent delegation is necessary. Figure 5.20 illustrates the agent retrieval mechanism where the agent interval is shorter than the probe interval. Before performing a new agent delegation, the requester always checks if there is an existing agent that has already received the desired content by transmitting a notification request. Therefore, the next notification request is always transmitted prior to a new delegation. If the probe interval is 2 min but the agent interval is 0.5min, the requester does not send a new delegation but only after that it sends a notification request to agent 1. Then only after 2 min a new delegation can be performed. However, after the probe interval of 120s, agent 1 has most certainly already

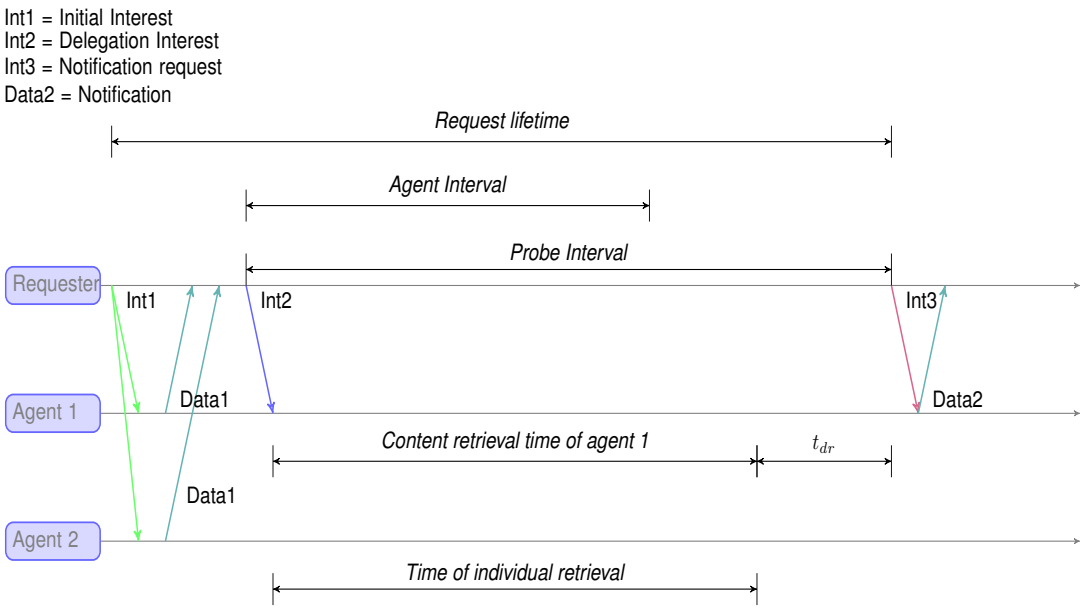


Figure 5.20: Messages exchanged during scenario 3: probe interval is higher than the agent interval

received the file of size 4MB. Therefore, the time of parallel retrieval (t_p) equals zero. The value of the probe interval when the transfer time is approximately constant depends on the file size to be transmitted. This probe interval value should not be larger than the agent's content retrieval time from the content source, so that the t_{dr} in Figure 5.20 will not be large. A large t_{dr} would increase the total transfer time.

5.5 Summary

We can summarize the results of the evaluation in the following points:

* For large files, the throughput of the complete agent-based content retrieval via unicast communication is faster by a factor of 5 compared to multicast communication.

* Comparing the agent-based content retrieval with the standard ccngetfile content retrieval application over two hops, i.e. unicast on the first hop and multicast on the second hop, showed a throughput gain of 20% compared to ccngetfile downloads because the transmissions over both hops are performed subsequently and not in parallel. Therefore, the maximum transfer capacity of each link can be used.

* The unicast throughput over two hops of the standard ccngetfile content retrieval compared to the agent-based content retrieval is higher by a factor of 3. This is because during ccngetfile transmissions the content is forwarded to the requester only from the cache. However, in the agent-based retrieval, the content needs to be additionally stored in the agent's mobile repository and retrieved later when the requester asks for it. However,

the repository can result in higher throughput in dense environments with many content transmissions and many cache replacements. Instead of forwarding requests from other requesters to the content source, it can be answered by the repository of an agent.

* A good probe interval value is 30s since the number of notification requests decreases by around 80% compared to a probe interval of 1s and transfer times when using the same probe intervals only increases by 5%.

* The agent interval has an influence on the transfer time: if the first delegated agent may not reach the content source, then a shorter agent interval may result in a faster transfer time. Otherwise, the agent interval should be large enough so that the first agent has enough time to retrieve the content. If the agent interval would be too short, multiple agents would retrieve the content at the same time decreasing the individual throughput on a shared medium.

Chapter 6

Conclusion

6.1 Conclusions

When the requester and a content source never meet, they cannot communicate directly and the requester will never receive the content needed. A possible solution is to delegate the content retrieval to mobile nodes, i.e. agents, that may move closer to a content source and get the requested content. In this Master thesis, we implemented the mechanism of agent-based content retrieval for CCN. With this implementation, mobile devices may receive requested content from agents even if they cannot reach the content source directly. This approach is also useful in case of unpopular content that needs to be found in a confined geographical area because agents replicate the content source for a specific file and therefore increase content density.

During the evaluation, we measured parameters at the requester in order to minimize the content retrieval time as well as the number of transmitted messages. The influence of the probe interval, the time to ask for a notification from an agent, and the agent interval, which determines when the requester delegates the retrieval to a new agent, have been examined. A probe interval of 30s is better than a smaller probe interval because it decreases the number of notification requests by around 80% compared to a probe interval of 1s and increases the transfer time by 5% only. An appropriate agent interval depends on the file size and the intercontact time between an agent and the content source.

The agent delegation results in a handshake mechanism at the beginning and a later notification to check whether the content is available at an agent. The additional overhead is therefore at least three Interests (initial interest, delegation interest and notification requests) and two content objects (delegation data and notification). For large files, the throughput of agent-based content retrieval is higher by around 20% than the regular content retrieval over two hops (first hop in unicast communication and multicast communication in the second hop). Thus, the agent delegation is better than forwarding Interests over two hops for large files, because content retrieval is performed subsequently and not in parallel. Therefore, the number of transmitted messages can be set individually. A slow link, e.g., during multicast communication on the second hop does not slow down unicast communication on the first hop. Also, there are fewer unicast retransmissions because the transmitted unicast Interests have timed out and need to be reexpressed. For small files the overhead of agent delegation is higher and forwarding in the regular retrieval

is more efficient. However, in case of intermittent and opportunistic connectivity, file transfers with ccngetfile are not feasible because neighbor nodes are not known and, therefore, it is not clear where to forward Interests. Additionally, long forwarding paths decrease the throughput and are more susceptible to node mobility because the content needs to travel on the reverse path. The agent-based approach exploits the mobility of each agent for content-retrieval.
The forwarding Interests using unicast communication in both hops achieves a three times higher throughput than the agent-based retrieval. This is because in the regular retrieval, the forwarding of the content is performed only from the cache and then the transfer is faster. In agent-based content retrieval, the agent's repository additionally stores every segment requiring more time. Then, later when the requester asks for the content, the agent needs to retrieves the content from the repository and then it can be transmitted to the requester. However, secondary storage is also advantageous in networks with many concurrent file transmissions. Because cache replacements may be performed quite quickly, secondary storage may prevent Interests to travel all the way back to the original content source.

6.2 Future Work

6.2.1 Reliability

To avoid denial of service attacks, the agent delegation between requester and agent needs to be restricted. Since an agent delegation is inexpensive compared to content retrieval, the relationship between requester and agents should be trusted and only a limited number of agents should be allowed.

6.2.2 Interval Values

The values for the agent interval and probe interval depend on the file size and the mobility of the nodes, i.e., the intercontact time between agent, requester and content source. Evaluations with mobile nodes are required to investigate appropriate values for them. The knowledge or estimation of file size may help to set appropriate probe and agent intervals.

In this work, the notification is requested by the requester. Investigations on appropriate content notification may enable agents to quickly inform the original requester that the content has been found and no further delegation is needed. However, this may result in content notifications even if no requester is in the vicinity.

6.2.3 Synchronization

As explained in section 4.5 , the agent may change its IP address if attached to a new network. Then, it is required that the mobile agent sends its new IP address to the home repository prior to the synchronization. Additionally, an application is required in the home repository to receive the new agent's IP address and create a unicast FIB entry.
Furthermore, the actual synchronization mechanism between the mobile and the home repository enables only the synchronization of collections under a predefined prefix. However, the agent needs to synchronize all the data that it needs to retrieve based on the delegated content prefix.

This requires exchanging information about the collection prefix between the mobile and home repository prior to synchronization.

6.2.4 Resume Operations from the Repository

The repository does currently not have a memory and does not remember already received content segments. Therefore, if a content transfer is aborted and requested at a later time, all segments are requested again from the neighbors if they are not in the cache anymore although stored persistently in the repository. The storage of meta information to perform a resume operation needs to be implemented. This would avoid the loading of segments from the repository prior to a disrupted file transfer as implemented within this thesis.

6.2.5 Android Sleeping Screen

The Android devices do not receive any Data packets transmitted via multicast if the screen is in sleep mode. Only unicast communication is received with sleep screen. Therefore, it is required that the screen is alive if applications depend on multicast communication.

Chapter 7

Appendix

7.1 Ad-hoc Networking Support for Android

7.1.1 Using the iwconfig program

iwconfig is used to display and change the parameters of the network interface which are specific to the wireless operation (e.g. interface name, frequency, SSID) .
The Linux wireless tools are not installed on Android devices by default . Therefore, it is required to compile wireless tool using NDK. Afterwards, iwconfig program can be added to android devices. The following instructions are used in order to create an ad hoc network in Google Nexus 4 device:

```
ifconfig wlan0 down
iwconfig wlan0 mode ad-hoc
iwconfig wlan0 channel 11
iwconfig wlan0 essid <network name>
iwconfig wlan0 up
ifconfig wlan0 <Ip address>
```

After that, the created network will be visible to all non-android devices and it is possible to connect to it. The same instructions are applied on second Google Nexus 4 device with the same network name and channel but with different IP address.

7.1.2 Configure wpa_supplicant manually

wpa_supplicant is a WPA Supplicant for Linux and it is designed to be a ”deamon” program that runs in the background and acts as the backend component controlling the wireless connection [31].
wpa_cli tool is a command line client to control wpa_supplicant. It allows to manage events from wpa_supplicant.
This method uses wpa_cli tool to force the driver to go to ad-hoc mode. The following instructions are used in Google Nexus 4 device in order to join existing ad hoc network:

```
add_network ---> it will return a number ,which represents network id .
```

```
set_network <network id> mode 1
set_network <network id> ssid "<network name>"
set_network <network id> bssid <bssid>
set_network <network id> frequency <frequency>
set_network <network id> key_mgmt NONE
enable_network <network id>
ap_scan 2
select_network <network id>
```

To obtain the network information, such as bssid, frequency and network name of desired ad hoc network, *scan_results* command can be used .

7.1.3 Edit wpa_supplicant.conf

wpa_supplicant.conf is the configuration file for wpa_supplicant that lists all accepted networks and security policies, including pre-shared keys.
This method forces wpa_supplicant to accept an ad-hoc network by editing directly the configuration file and adding the ad-hoc network information. The following information should be added to wpa_supplicant.conf :

```
network={
ssid="<network name>"
scan_ssid=1
key_mgmt=NONE
mode=1
priority=1
}
```

Afterwards, the modified wpa_supplicant.conf should be pushed on /data/misc/wifi, Finally, it is required to chown ownership back to system.wifi by running the following command:

```
chown system.wifi /data/misc/wifi/wpa\_supplicant.conf
```

7.1.4 Edit Android Open Source

This method modifies the android source code by making the following change in order to enable ad-hoc wifi connection:

In WifiSettings.java file the code that does not allow to display IBSS type of stations needs to be commented as follows:

```
// Ignore hidden and ad-hoc networks.
```

```
if (result.SSID == null || result.SSID.length() == 0) { // ||
//result.capabilities.contains("[IBSS]")) {
continue;
}
```

Location of that file is packages/apps/Settings/src/com/android/settings/wifi

In wpa_supplicant_8, the wpa_supplicant/events.c file should be edited as below in order to allow IBSS connection :

```
static struct wpa_ssid * wpa_scan_res_match(struct wpa_supplicant *wpa_s,
int i, struct wpa_scan_res *bss,
struct wpa_ssid *group) {

. . .

if (bss->caps & IEEE80211_CAP_IBSS) {
wpa_dbg(wpa_s, MSG_DEBUG, " skip - IBSS (adhoc) "
"network");
//continue; //Allow IBSS connection.
}

. . .

}
```

7.1.5 Superuser Access

The process of rooting varies widely by device: to root Google Nexus 4 devices [25] , su binary should be copied to a location in the current process's PATH (e.g. /system/xbin/su) and granting it executable permissions. To limit the access to su binary file, SuperSU application is used. SuperSU is an application whose purpose is to protect access to su binary file. When an application tries to access the root device functionality, SuperSU triggers and asks the user if it is desired to or not grant root priviledges to the application.

7.2 Code Source

The implemetation of this thesis is based on CCNx Java Library.

7.2.1 Source Code of the CCNRequester application

7.2.1.1 Newly Created Classes

/src/ccnx/android/apps/requester/CCNRequester.java

/src/ccnx/android/apps/requester/CCNxRequesterMain.java

/src/ccnx/android/apps/requester/CCNHandleResponse.java

7.2.1.2 Edited Classes from CCNx Java Library

/src/ccnx/android/apps/requester/SimpleFaceControl.java
It was not possible to pass specific prefix and specific port as parameters to the available method that creates multicast or the unicast face. The available method in this class creates a face with prefix "ccnx:/". In our implementation, it was required to register specific prefix on the multicast face. Therefore, openMulicastInterface and connectTcp methods have been modified in this class in order to satisfy our needs.

7.2.2 Source Code of the CCNAgent application

7.2.2.1 Newly Created Classes

/src/ccnx/android/apps/agent/CCNAgent.java

/src/ccnx/android/apps/agent/HelperThread.java

/src/ccnx/android/apps/agent/CCNSynchronization.java

/src/ccnx/android/apps/agent/HanlderRootAdvice.java

7.2.2.2 Edited Classes from CCNx Java Library

/src/ccnx/android/apps/agent/io/CCNAbstractInputStream.java
As explained in section 4.3.2, the reading process must be performed from the repository and should not be from the neighbors. Our implementation uses the reading mechanism provided by the CCNx Java Library. However, the available mechanism may send Interests to neighbors. Therefore, it was required to set the scope of every Interest to value 1 within this class.

/src/ccnx/android/apps/agent/SimpleFaceControl.java

The modification performed in SimpleFaceControl.java class in requester package has been done also in this class. In addition, to unregister a given prefix from the multicast face, there was no method in this class that performs the unregistering of a given prefix. The available method can only delete the face. So, it was required to add the method that unregisters a prefix from the multicast face without deleting the face.

Bibliography

[1] Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H., & Braynard, R. L. (2009, December). Networking named content. In Proceedings of the 5th international conference on Emerging networking experiments and technologies (pp. 1-12). ACM.

[2] "Project CCNx,"`https://ccnx.org`, 2013.

[3] `http://www.ccnx.org/releases/latest/doc/manpages/ccngetfile.1.html`

[4] `http://www.ccnx.org/releases/latest/doc/technical/InterestMessage.html`

[5] `http://www.ccnx.org/releases/latest/doc/technical/RepoProtocol.html`

[6] `http://www.ccnx.org/releases/latest/doc/technical/SynchronizationProtocol.html`

[7] Anastasiades, C., Uruqi, A. & Braun, T. (2012). Content discovery in opportunistic content-centric networks. LCN Workshops (p./pp. 1044-1052), : IEEE. ISBN: 978-1-4673-2130-3

[8] Schmid, T. (2013, March). Data Exchange In Intermittently Connected Content-Centric Networks. Bachelor-Thesis, University of Bern.

[9] Weber, J. (2013, March). Automatic Detection Of Forwarding Opportunities In Intermittently Connected Content-Centric Networks. Bachelor-Thesis, University of Bern.

[10] Hermans, F., Ngai, E., & Gunningberg, P. (2012, June). Global source mobility in the content-centric networking architecture. In Proceedings of the 1st ACM workshop on Emerging Name-Oriented Mobile Networking Design-Architecture, Algorithms, and Applications (pp. 13-18). ACM.

[11] Han, D., Lee, M., Cho, K. , Kwon, Ted "T." & Choi, Y. (2012, August). PMC: Publisher Mobility Support for Mobile Broadcasting in Content Centric Networks. In AsiaFI 2012 summer school, Kyoto, Japan.

[12] Ravindran, R., Lo, S., Zhang, X., & Wang, G. (2012, June). Supporting seamless mobility in named data networking. In Communications (ICC), 2012 IEEE International Conference on (pp. 5854-5869). IEEE.

[13] Hermans, F., Ngai, E., & Gunningberg, P. (2011, June). Mobile sources in an information-centric network with hierarchical names: An indirection approach. In Proc. of the 7th Swedish National Computer Networking Workshop.

[14] Lee, J., Kim, D., Jang, M. W., & Lee, B. J. (2011, January). Proxy-based mobility management scheme in mobile content centric networking (CCN) environments. In Consumer Electronics (ICCE), 2011 IEEE International Conference on (pp. 595-596). IEEE.

[15] Lee, J., & Kim, D. (2011). Proxy-assisted content sharing using content centric networking (CCN) for resource-limited mobile consumer devices. Consumer Electronics, IEEE Transactions on, 57(2), 477-483.

[16] Kim, D. H., Kim, J. H., Kim, Y. S., Yoon, H. S., & Yeom, I. (2012, August). Mobility support in content centric networks. In Proceedings of the second edition of the ICN workshop on Information-centric networking (pp. 13-18). ACM.

[17] Lee, J., Cho, S., & Kim, D. (2012). Device mobility management in content-centric networking. Communications Magazine, IEEE, 50(12), 28-34.

[18] Oh, S. Y., Lau, D., & Gerla, M. (2010, October). Content centric networking in tactical and emergency manets. In Wireless Days (WD), 2010 IFIP (pp. 1-5). IEEE.

[19] Meisel, M., Pappas, V., & Zhang, L. (2010, September). Listen first, broadcast later: Topology-agnostic forwarding under high dynamics. In Annual Conference of International Technology Alliance in Network and Information Science (p. 8).

[20] Varvello, M., Rimac, I., Lee, U., Greenwald, L., & Hilt, V. (2011, January). On the design of content-centric MANETs. In Wireless On-Demand Network Systems and Services (WONS), 2011 Eighth International Conference on (pp. 1-8). IEEE.

[21] Wang, L., Waltari, O., & Kangasharju, J. (2013). MobiCCN: Mobility Support with Greedy Routing in Content-Centric Networks. In: IEEE Global Communications Conference 2013: NGN - Next Generation Network

[22] Su, J., Scott, J., Hui, P., Crowcroft, J., De Lara, E., Diot, C., ... & Upton, E. (2007). Haggle: Seamless networking for mobile applications. In UbiComp 2007: Ubiquitous Computing (pp. 391-408). Springer Berlin Heidelberg.

[23] Jacobson, V., Braynard, R. L., Diebert, T., Mahadevan, P., Mosko, M., Briggs, N. H., ... & Thornton, J. D. (2012). Custodian-based information sharing. Communications Magazine, IEEE, 50(7), 38-43.

[24] Batista, B., & Mendes, P. (2013, April). ICON-An Information Centric Architecture for Opportunistic Networks. The 2nd IEEE International Workshop on Emerging Design Choices in Name-Oriented Networking (NOMEN 2013)/Infocom 2013.

[25] http://nexus4root.com/nexus-4-root/how-to-root-nexus-4-windowsmac-osxl

[26] https://groups.google.com/forum/#!topic/spandev/eVUTW4gK6sY

[27] http://forum.cyanogenmod.com/topic/72659-nexus-4-is-able-to-see-but-cannot-c

[28] http://forum.xda-developers.com/showthread.php?t=2114572

[29] http://android.stackexchange.com/questions/47877/
ad-hoc-network-in-android

[30] http://forums.androidcentral.com/google-nexus-4/
288653-adhoc-network-problem.html

[31] http://linux.die.net/man/8/wpa_supplicant