AGENT-BASED DATA RETRIEVAL FOR OPPORTUNISTIC CONTENT-CENTRIC NETWORKS

Masterarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Tobias Schmid
2015

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik

Contents

List of Figures

List of Tables

Acknowledgment

On this page I would like to thank everybody who supported me to write this master thesis.
First I would like to thank my coach Carlos Anastasiades. He supported me trough the whole process of writing this thesis, had an ear for discussing problems with the thesis and gave inputs when I was blocked with my work.
After that I would like to thank Prof. Dr. Torsten Braun who allowed me to write this thesis in his research group. I am also very grateful for the resources that were generously provided by the research group of Prof. Braun.
Last but not least my work colleagues and office mates Jürg Weber and Alexander Striffeler who have been there for solving problems, drinking coffee, killing time and fooling around.

Abstract

Content-Centric Networking (CCN) is a new networking approach where communication is based on names and not host identifiers. Thus, in contrast to host-centric network architectures, content-centric networks have no need to know their neighbors and the whole network topology because data is requested based on a content specific identifier and not a host identifier. For this reason, content-centric networks are suitable for dynamic mobile networks with many moving devices. The CCNx project implements the CCN concepts as open source framework.

In this thesis an agent protocol as well as a requester application in C based on the CCNx framework have been developed. The agent-based approach enables requesters to delegate content retrieval to other nodes, i.e., agents. Agents import retrieved content locally in their content repository. In case of connectivity disruptions, content retrieval can be resumed (from any node) whenever content becomes available again. Requester and agent can negotiate during the delegation phase whether push or pull notifications are used. Once content retrieval has been completed, agents can notify the requester that content is ready for delivery. When a notification has been received, the requester can download the content via unicast communication directly from the agent. Since agents store content in their local repositories, the content density in the environment is increased.

The implemented applications have been evaluated using the network simulator NS3 and the NS3-DCE (Direct Code Execution) module. Evaluation of different scenarios in large playground with up to 100 mobile nodes showed that agent-based content retrieval, compared to broadcast multi-hop content retrieval, is fast and reliable for transferring large files over long distances even under difficult mobility conditions, e.g., sparse node density. The principles of content-centric networks can be exploited to resume interrupted content retrievals without a lot of overhead for finding content sources and building routes. Compared to multi-hop retrieval, agent-based content retrieval does not generate a lot of network load: in each considered scenario the amount of transferred messages was lower than with multi-hop.

Chapter 1

Introduction

When the Internet architecture was developed network devices and computers were not integrated in the environment the same way as today. Mobile computing was just a dream because the devices were too big to carry around. The Internet architecture has been developed to share resources and the communication was limited to wired connections. Since then technology evolved very fast and many constraints have disappeared. Nowadays, it is even possible to use the air to exchange data between mobile devices.

Today's network connections are used mostly to exchange data. It is important to receive a specific requested content but the real location of the content source is irrelevant. With the increasing dissemination of mobile devices such as smart phones or tablets, that fact is reinforced.

For this reason, different information-centric networking approaches have been developed. One prominent approach is Content-Centric Networking (CCN) [1]. The open source framework implementation CCNx [2] is the basis of this thesis.

1.1 Motivation and Task Formulation

1.1.1 Motivation

In opportunistic networks, content retrieval is particularly challenging because contacts between users and their mobile devices are not predictable. If a requester and a content source never meet, they cannot communicate directly. In such a scenario, the requester can delegate the content retrieval to other nodes with different mobility patterns, i.e., agents, to increase the probability of finding the content and increase the content density. To avoid denial-of-service attacks and misuse, the delegation should be performed in a controlled way based on agreements between requester and agents.

A prototype implementation of agent delegation has been developed in [3]. The goal of this master thesis is to develop the protocol further. Agents that retrieved content for others must ensure its authenticity and integrity by providing original headers and signatures. Therefore,

agents need to ask their own repository to retrieve and store the content. Similar to [4], the repository needs to be extended to remember already received segments from incomplete file transfers. Different notification mechanisms (e.g., push and pull notifications) should be implemented and evaluated. Furthermore, flexibility of agent delegation should be improved, so that agent and requester applications can negotiate the notification type and further parameters.
Finally, suitable scenarios for agent-based content retrieval with multiple nodes (up to 100 nodes) and mobility have to be found, implemented and evaluated.

1.1.2 Task Formulation

The goal of this thesis is to develop an application that enables agent-based content retrieval in opportunistic content-centric networks based on the current CCNx framework [2].
The work comprises the following tasks:

- extension of the existing CCNx repository (`ccnr`) to enable opportunistic resume operations.
- implementation of the agent delegation mechanisms in C based on the work in [3] so that requesters can delegate content retrieval to agents.
- implementation and comparison of push and pull based notification mechanisms. Flexible usage by selecting the notification mechanisms during the agent delegation process.
- evaluation of the protocols in a mobile scenario using a network simulator.

1.2 Outline

The remainder of this thesis is organized as follows. Chapter 2 presents the basics of content-centric networking and agent-based content retrieval. Chapter 3 describes the design and implementation of the agent protocol in CCNx. After that in Chapter 4 the developed agent applications are evaluated in different scenarios and the obtained results are discussed. Finally, in Chapter 5 the gained insights are summarized and an outlook for future work in this research area is presented.

Chapter 2

Related Work

2.1 Content-Centric Networks

Content-centric networking (CCN) has been introduced by Van Jacobson [1]. The Palo Alto Research Center (PARC) has implemented a reference implementation of CCN called CCNx[2]. The applications developed in this thesis are based on the source code of CCNx 0.8.2 [5] which was the most recent version at the time of writing this thesis. Further explanations about the CCNx protocol always refers to version 0.8.2.

2.1.1 CCNx 0.8.2 vs. CCNx 1.0

Since June 2015 CCNx version 1.0 has been released. There are significant changes between CCNx 0.8.2 and 1.0 since the source code has been completely rewritten and rearranged in independent libraries [6]. The source code of CCNx 1.0 is no longer publicly available for everyone, only binaries and header files are publicly released. The modifications in CCNx 1.0 [7] have also an impact on agent-based content retrieval because longest-prefix matching and Exclude filters (used in all communication steps, see Chapter 3) are no longer supported. Instead, only exact name matching is performed in CCNx 1.0. However, the new NDNx framework [8], which was initially based on the CCNx source code, continues to support longest-prefix matching [9]. Therefore, we suggest that future versions of agent-based content retrieval are implemented within NDNx.

2.1.2 CCN Packets

In CCN, communication is based on two basic packet types: Interest and Data. Figure 2.1 shows the Interest packet for requesting content and the Data packet to deliver data which is called ContentObject in CCNx.

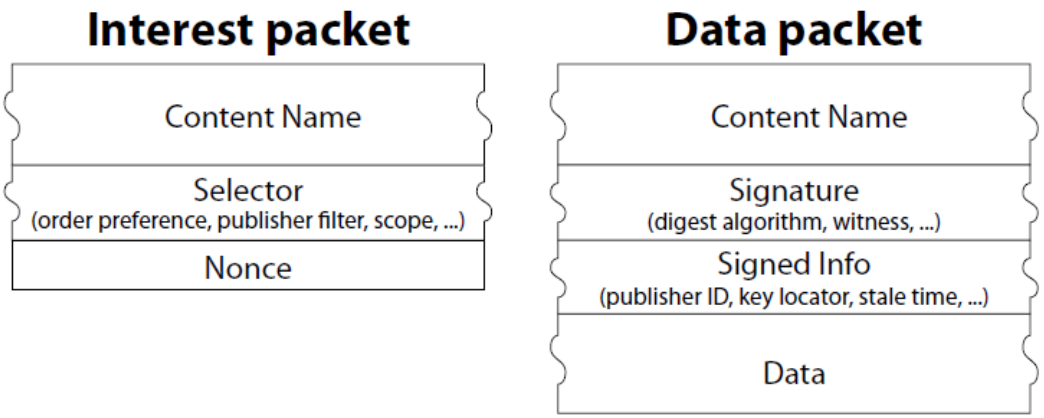


Figure 2.1: CCN Packet Types [1]

An Interest packet contains a prefix or the complete ContentName of the requested ContentObject and some selectors, i.e., `answerOriginKind` and `scope`, which allow to restrict the origin and matching of the Data packet.
A Data packet is defined as follows: It contains the ContentName, the Signature, the SignedInfo and the data itself.
There is an important rule for sending and receiving packets in CCNx; at most one Data packet is sent per Interest packet. If no Data packet arrives in response to an Interest within a given time, the Interest is retransmitted. After several Interest retransmissions, it is assumed that the content is not available and the download is stopped.

2.1.3 Content Naming

Another essential part of content-centric networking is content naming. Because content is requested by name, a clever way of handling content names is necessary. A hierarchical name structure is suitable for this use case.

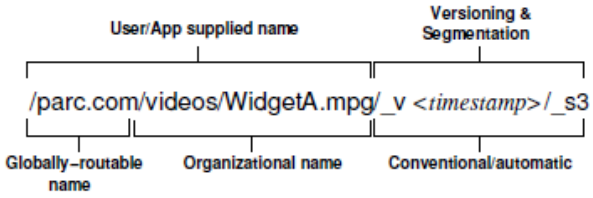


Figure 2.2: CCN Naming

In CCN packets ContentNames are binary encoded, however, ContentNames can be represented by URIs which are easily read and remembered by humans (as shown in Figure 2.2). Every component between two slashes (/) represents a hierarchy level. Furthermore, every ContentObject has a version number and is split into several segments. To indicate version and segment number, markers are used. Versioning is implemented with timestamps. A `_v` marker followed by a timestamp represents the version of the packet. The segment number is given after the `_s` marker. The first segment holds the segment number 0, for each additional segment the segment number is incremented.

Besides segment and version markers the basic name conventions provide extensible command markers (marker C1) [10]. Extensible command marker are used for commands, queries or other special identifiers and have the following appearance: `%C1.<namespace>.<command>`. The `%C1` prefix indicates an extensible marker, followed by a namespace, e.g., the related application (`%C1.R.xyz` for the `ccnr` repository), followed by the command string. Optionally, command arguments can be appended delimited by tilde characters (∼).

2.1.4 Data Storage in CCNx

In CCNx there are different ways of storing data. To store data persistently in CCNx, repositories are used. The ContentStore is used as cache to keep data temporarily.

ContentStore (CS)

The ContentStore is a cache to keep received Data packets temporarily. The received packets are stored in main memory (RAM) which allows a fast access time. If a packet arrives that is already included in the ContentStore it is discarded. The lifetime of Data packets in the ContentStore is limited by the memory size and the *freshnessSeconds* flag which is embedded in the Data packet header and indicates the maximum validity time of the content.

Repository

Repositories enable persistent storage of data on dedicated storage devices.
The repository application written in C (called `ccnr`) has replaced the Java application (called *ccn_repo*) due to its better performance and integrated synchronization facility[11]. To share data over the repository they first have to be inserted. When inserting new content into the repository, it is divided into segments and signed by the publisher's public key. This has to be done only once when including new content, i.e., when synchronizing content among repositories no signatures need to be created, but the existing one is used. All content of a repository is persistently stored in a repository file in the local file system.

2.2 Communication in Opportunistic and Delay-Tolerant Networks

Routing in Delay Tolerant Networks (DTN) [12] has been studied for more than a decade. Since then, many DTN routing protocols [13] have been proposed such as Epidemic Routing [14], where a node copies its messages to all nodes that it encounters, Spray-and-Wait [15], which limits the number of forwarded copies, or prediction-based forwarding based on the history of past encounters [16]. In recent years, increasingly more DTN routing protocols rely on social characteristics [17] to improve message delivery. Based on neighbor discovery [18], nodes can create a contact graph to detect communities as well as extract centrality, similarity and friendship characteristics for more efficient forwarding. Nodes in the same community regularly see each other and are for example reliable message carriers while nodes that have long-lasting and regular contacts may be considered as friends, who may even share common interests [17]. However, mapping contacts to social relations is complex and DTN routing performance heavily depends on how the mapping (contact aggregation) is performed [19].

Several content-centric networking schemes have been proposed for delay-tolerant or opportunistic networks. For example, in the context of a mobile peer-to-peer system [20], where users transmit periodic hello beacons to detect neighbors before connecting to them. Content is then exchanged pairwise with a solicitation protocol. There is no multi-hop communication and routing is replaced by the mobility of the nodes. Slinky [21] is another approach where content is placed in communities (topological structures) ensuring high availability to all nodes. Each node in a community stores only a fraction of all published content and replies to queries for that content. On the downside, if community structures change due to mobility, content needs to be re-distributed in the community, rendering is inefficient for high mobility scenarios.

CEDO [22] is a first approach to extend CCNx with DTN functionality. Interests stay in the Pending Interest Table (PIT) until they are satisfied. Whenever a contact is detected (through periodic hello beacons), a message that summarizes all pending Interests is transmitted. A receiver of such a message sends back all Data messages that it has in the cache. Regular Interest forwarding via FIB is disabled.

Another approach [23] introduces the concept of logical faces for communication in disaster scenarios.It assumes that communities (location where content can be retrieved and from where Interests originated) are static and mobile mules forward messages between communities. Logical faces are used to map communities to physical interfaces. Whenever mules reach the corresponding communities, the logical faces become active and Data can be retrieved based on information in the PIT.

Both approaches [22], [23] keep Interest messages in the PIT until nodes encounter the desired content source or community. However, Content-Centric Networking (CCN) content may contain multiple segments, which need to be requested individually and, typically, the requester does not know the content size until receiving the last segment. Thus, a requester does not know

how many Interests are required to retrieve the content and there is no immediate feedback when content transfer is finished (delay-tolerant retrieval via data mules). If too few Interests are forwarded by a requester, a data mule may not retrieve the complete content and may need to travel several times between communities. Too many Interests, however, result in inefficient memory management in the PIT. Since both approaches [22], [23] modify CCN message processing and message structures significantly, they can not be used in well-connected networks.

In this work, we assume that DTN communication is only used for a certain time or at a certain place, e.g., to cover remote areas, or support communication if central communication infrastructures are temporarily not available. However, eventually people will be using the infrastructure again. Then, they may want to resume incomplete downloads or upload collected information without converting and resigning it. Therefore, we use the concept of agent-based content communication, where content retrieval is delegated to agent nodes. Since delay-tolerant communication is provided as application module without modifications to regular CCN message processing, it can combined with multi-hop communication whenever possible. We base our work on previous work [3] and refine it for more flexible communication in mobile networks (see next subsection for more details).

If all messages are transmitted via broadcast, Interests address multiple content sources at the same time. With pervasive caching, content density increases drastically and even more nodes may be able to respond to Interests. To avoid collisions and enable duplicate suppression, Data messages need to be delayed. Furthermore, unicast data rates are usually significantly higher than broadcast rates because they can be adapted based on the distance (signal-to-noise ratio) between two nodes, whereas broadcast data rates are usually set to the lowest supported rate. Therefore, we use unicast communication as often as possible in this thesis, e.g., content download from agent nodes (see section 3.3.3).

Earlier work [24] showed that Dynamic Unicast, i.e., creating dynamic unicast faces to a content source after initial broadcast requests, result in faster transmission times even for multiple concurrent requesters. Since Dynamic Unicast is independent from the agent protocol, it can be combined with the agent protocol in this thesis.

We compare agent-based content retrieval with multi-hop communication in multiple scenarios with different number of agents, content sources, velocities and file sizes. Because multi-hop communication may experience disruptions, we use an application from previous work ([4], [25]) that can persistently store partial files and resume content transfers in case of disruptions (see section 3.5 for more information).

2.3 Comparison to Previous Work: Agent-based Content Retrieval

In a previous work [3], a prototype implementation of agent-based content retrieval has been developed and evaluated on Android smart phones. This thesis is based conceptually on [3], but differs in design, implementation and evaluation. These differences are discussed in the following subsections.

2.3.1 Design and Implementation

In [3] an agent and a requester application have been implemented using the CCNx Android library as well as the Android SDK. A basic agent handshake has been introduced and evaluated in a static configuration with three mobile phones and a laptop (as shown in Figure 2.3).

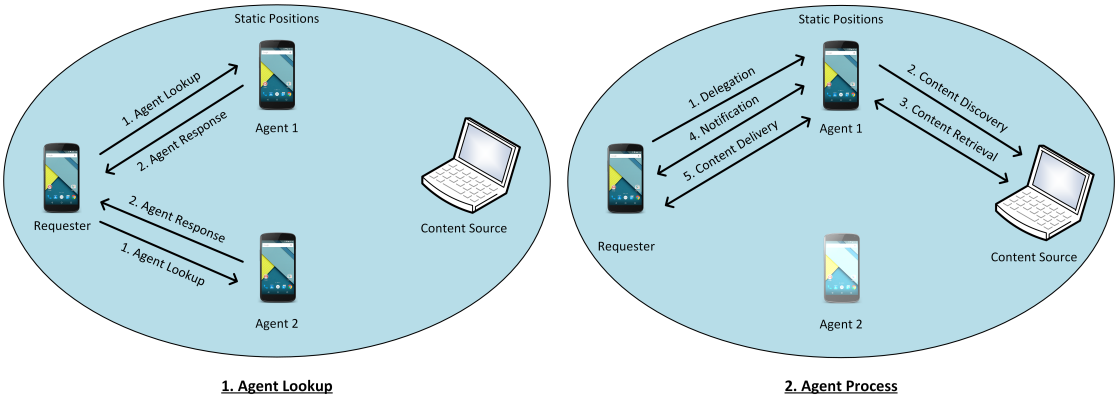


Figure 2.3: Master Thesis Agent-based Content Retrieval: Evaluation Scenario

The main modifications in the implementation are as follows:

- The agent and requester applications have been rewritten in C in order to enable evaluations with NS3-DCE. Please find more information about NS3-DCE in Section 4.2 of Chapter 4.
- The existing agent handshake has been modified to:
  - follow the CCNx basic name conventions with command markers.
  - enable flexibility by defining options such as the notification type, i.e., push or pull, during agent delegation. In previous work only pull notifications were supported.
  - introduce an additional acknowledgment during agent delegation to ensure that agents have received the delegation. Otherwise, the requester may wait for the content although no agent is retrieving the content. See Section 3.5 for more information.

- To support content retrieval in opportunistic networks, the existing content repository has been extended to check whether the entire content is in the repository. This is required because of disruptions in connectivity, which means that only parts of the content could be retrieved by the repository. The modification ensures that if the content is not complete and there is no ongoing content download, the content download is resumed. Based on this extension the agent application is capable of resuming incomplete content imports due to disrupted connectivity.

- Furthermore, at the requester side, the extended content fetch library introduced in [4] has been integrated into the requester application. By using the extended fetch library, content can be retrieved even in case of short contact times due to disrupted connectivity.

2.3.2 Evaluation

In [3], agent-based content retrieval has been evaluated using up to three Android smart phones in a static simulation setup. In this thesis the NS3-DCE simulation framework has been used to simulate wireless communication and mobility in large playgrounds with more than 100 nodes.

Due to Android dependencies and implementation in Java, the application of the former work was not reusable for evaluations with NS3-DCE. Therefore, agent-based content retrieval has been completely rewritten in C. With NS3-DCE it was possible to evaluate agent-based content retrieval and compare it to multi-hop communication in different topology and with different node densities.

Chapter 3

Design and Implementation of Agent-based Data Retrieval in CCNx

3.1 Problem Description

In opportunistic content-centric networks, nodes can exchange information within transmission range via direct ad-hoc communication without being connected to the Internet. However, transmission range and mobility of nodes is limited such that they may never meet a certain content source directly. As shown in Figure 3.1, multi-hop communication may also not be possible if there is no continuous end-to-end path between requester and content source.

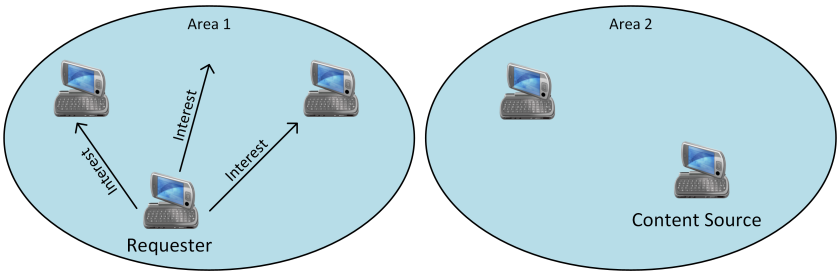


Figure 3.1: Problem Formulation: Disjoint Areas

To address this problem certain jobs, e.g., content retrieval, can be delegated to agent nodes (or agents for short), which move close to content sources and can then execute the jobs, e.g., retrieve the content directly. We design an agent application, which is responsible for finding and fetching content of accepted jobs. When content of a job has been completely retrieved, it is available for delivery to the requester. In the next subsections, agent-based job delegation is described in more detail.

3.2 Agent Process

The whole agent process in content-centric networks can be structured in two roles and four phases. In the remainder of this section these roles and phases will be described in detail.

3.2.1 Roles

The agent process can be structured in two different roles with separate responsibilities.

Requester

The requester is interested in content or information, which it cannot obtain at its location due to limited connectivity to the content source. The requester can then delegate content retrieval to other nodes, which may move to the content source to retrieve the content directly and return it to the requester. The requester follows a predefined handshake routine to delegate content retrieval to other nodes (agents).

Agent

The agent receives job delegations from requesters and decides whether it can accept it or not. If it accepts the job, it will inform the requester. The agent then looks for content of accepted jobs and fetches it if found. After completing the fetch process of an accepted job, the notification mode will be activated to notify the requester that the content has been found and is ready for delivery.

3.2.2 Phases

The whole agent process / protocol can be grouped in four different phases.

Phase I: Job Delegation

Figure 3.2 illustrates the first phase (job delegation).
A requester is looking for suitable agents within the agent probing interval. During this time, the requester sends an init request to its one-hop neighborhood and then probes the cache for incoming responses. After the agent probing interval, the requester may have received (multiple) init responses and can delegate the job to one or multiple agents as shown in Figure 3.3. Agents have to acknowledge received delegations, since agents could have already moved away within the agent probing interval. A job delegation is only valid for a specified time, in the following called expiration time, to ensure agents are not searching for the content for an indefinite amount of time.

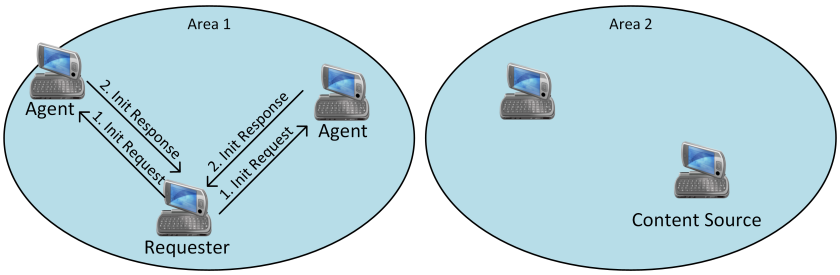


Figure 3.2: Agent Process: Phase Ia - Job Delegation

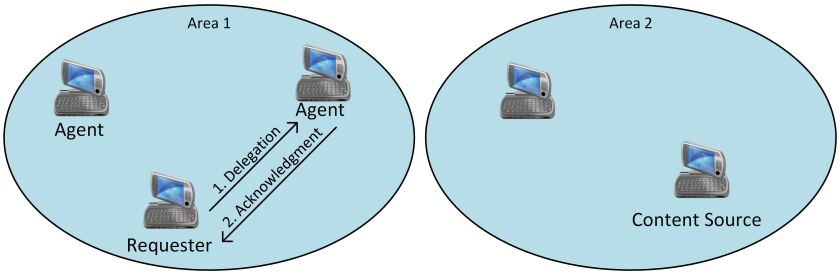


Figure 3.3: Agent Process: Phase Ib - Job Delegation

Phase II: Content Retrieval

When a job is delegated to one or more agents, the agents become active by executing the delegated tasks. They are responsible for fetching and storing the requested content within the defined job expiration time. The agent periodically requests the desired content while moving around. As soon the content is available it will be imported to the local content repository. Partially received content, i.e., in case of disruptions, is stored in the local content repository (persistent storage, keeps original signatures) and the repository import can be resumed later. If the retrieval time exceeds the expiration time, the job will be aborted independent whether the job has been completed or not. Once the content is completely fetched, Phase III (notification) will be activated.

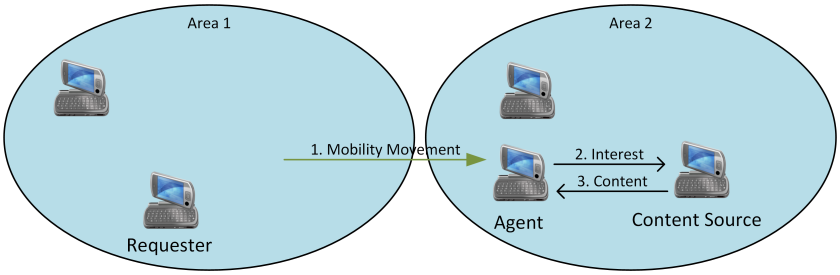


Figure 3.4: Agent Process: Phase II - Content Retrieval

Phase III: Notification

If agents have completed their task, they can notify the requester that the content is available for download as Figure 3.5 shows.

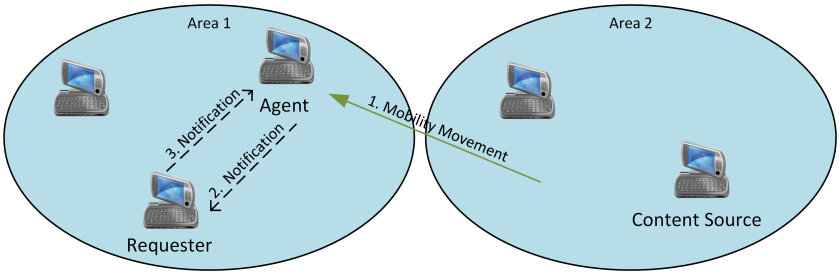


Figure 3.5: Agent Process: Phase III - Notification

We differentiate two different kinds of notifications: pull and push notification.

1. **Push Notification**
   Applying push notification means that the agent has the leading role for notification. After delegating content retrieval to agents, the requesters starts listening for push notifications from agents. As soon as an agent has completely fetched the desired content, it starts to periodically send notification messages to inform the requester.

2. **Pull Notification**
   Pull notification means that after job delegation the requester periodically requests notifications from agents that have completed their task. The agents do not independently send notification messages but only reply to notification from requesters.

Phase IV: Content Delivery

After the notification phase, the requester can fetch the content from the agent as illustrated in Figure 3.6. Since information in the notification messages includes content version and host identifier, e.g., the IP address, the requester creates a direct face to the agent and can directly retrieve the content from the agent's repository. Please note that content retrieval from agents can also be resumed in case of interruptions, i.e., when an agent leaves the transmission range of the requester. In such a case, content delivery can be resumed when the agent comes in transmission range again (if before expiration time has elapsed). Once the content has been completely delivered, the job has successfully finished.

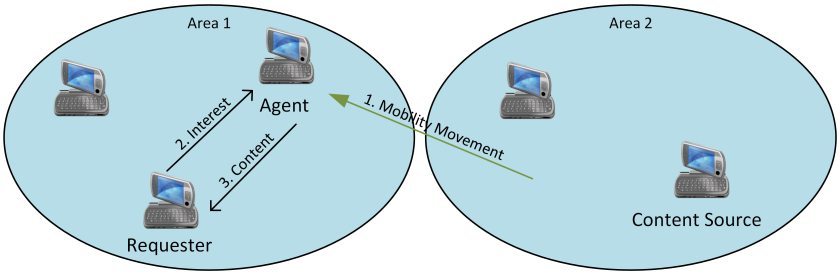


Figure 3.6: Agent Process: Phase IV - Content Delivery

3.3 Agent Handshake - Message Flow

In this section the message flow of the agent handshake is described in more detail. Therefore, the messages between requester and agents are visualized in message flow diagrams.

Table 3.1 lists parameters / variables, which are used in the sequence diagrams as well as in the descriptions and will be explained shortly here.

Term	Description
prefix	Content name of the requested content written as a human readable URI without any marker components.
nodeID	String representation of the public key of a CCN daemon (ccnd), used to distinguish the different running ccnd instances.
host identifier	Identifier, to distinguish network interfaces of different hosts/nodes, e.g., the IP or MAC address.
jobID	Randomly generated string representation, used as a unique identifier for each job.
expireTime	Unix timestamp to indicate the expiration time of the job.

Table 3.1: Term Declaration

According to the CCNx naming conventions [10] additional extensible command marker components appearing as `%C1.<namespace>.<cmd>` have been introduced. In the following table the markers are explained.

Marker	Description
`%C1.A.init`	Request for available agent resources (delegation phase).
`%C1.A.delegate`	Delegate job to selected agent (delegation phase).
`%C1.A.version`	Request version for content delivery (notification phase).
`%C1.A.done`	Announce that content delivery succeeded and job can be removed from job queue (notification phase).

Table 3.2: Additional Command Markers

3.3.1 Delegation

Figure 3.7 shows the message flow for a job delegation. Before a job can be delegated a user has to start the requester application (`ccnar`) with the *prefix*, i.e, content name without content version and segment number, and the desired command line parameters. For example, `ccnar -n pull ccnx:/unibe.ch/cds/presentation.pdf` defines that content with the name `ccnx:/unibe.ch/cds/presentation.pdf` should be retrieved and notification (option: `-n`, see more details in Appendix 6.1) should be performed pull-based. If the user's input is valid, the requester starts searching for potential agent candidates by sending Interest messages with the name `ccnx:/ferrying/<prefix>/%C1.A.init`. The requester may look for agents for a configurable amount of time, i.e., agent request period (in our implementation: 2 seconds). Agent applications are listening for Interests with the *prefix* component `/ferrying/` and respond (if they have available resources) with a Data message. In the ContentName of the Data messages, agents append their *nodeID*.
On the requester side, the appended *nodeID* is stored in a list of potential agents. After the agent request period has elapsed, the requester can select an agent from the agent list (if it is not empty). The requester can then delegate the job to the selected agent using the *nodeID* and additional job parameters such as notification type, *jobID* and *expireTime* to start a job. Since agent delegation is performed within an agent request period, an acknowledgment from the selected agent is required to ensure that it has received the job delegation and not moved away. If no acknowledgment has been received within 1 second, the requester selects another agent from the candidate list or restarts the agent request period if there are no other agents on the agent list. After delegation the requester switches to the notification mode depending on the command line parameter (`-n` option) in the delegation process (see Appendix 6.1 for more information).

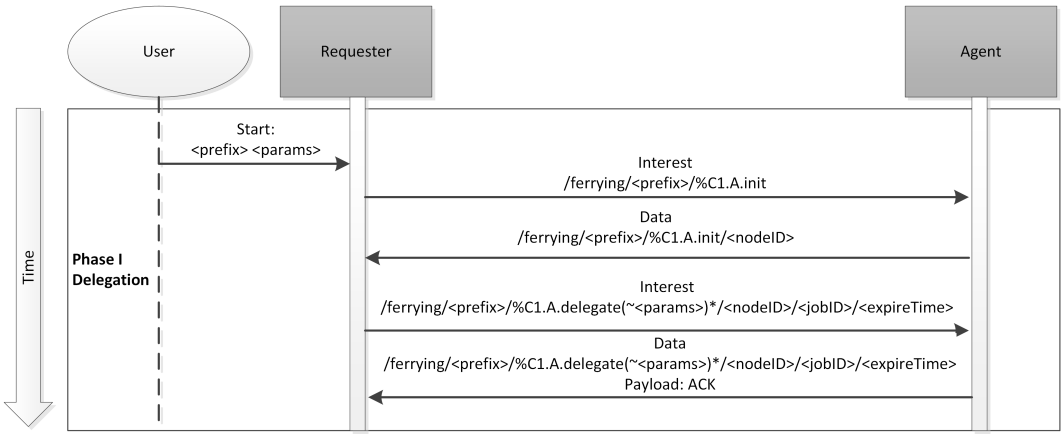


Figure 3.7: Agent Handshake - Delegation

3.3.2 Content Retrieval

Figure 3.8 shows the content retrieval process. The agent has to search and retrieve requested content. To ensure the authenticity and integrity of the retrieved data, original Data headers including signatures have to be provided to the requester. Therefore, the requested Data messages need to be directly imported to the local content repository. To achieve this, the agent first searches for available content by sending version resolving requests (required to find the newest version of a content object). Once a content source responds and the newest version has been identified, the agent starts the repository import with an Interest with name `ccnx:/<prefix>/<version>/%C1.R.sw-c/<parameters>/` (the import process is explained in detail in subsection 3.4.1). The repository application then asynchronously sends Interests and imports incoming Data messages.

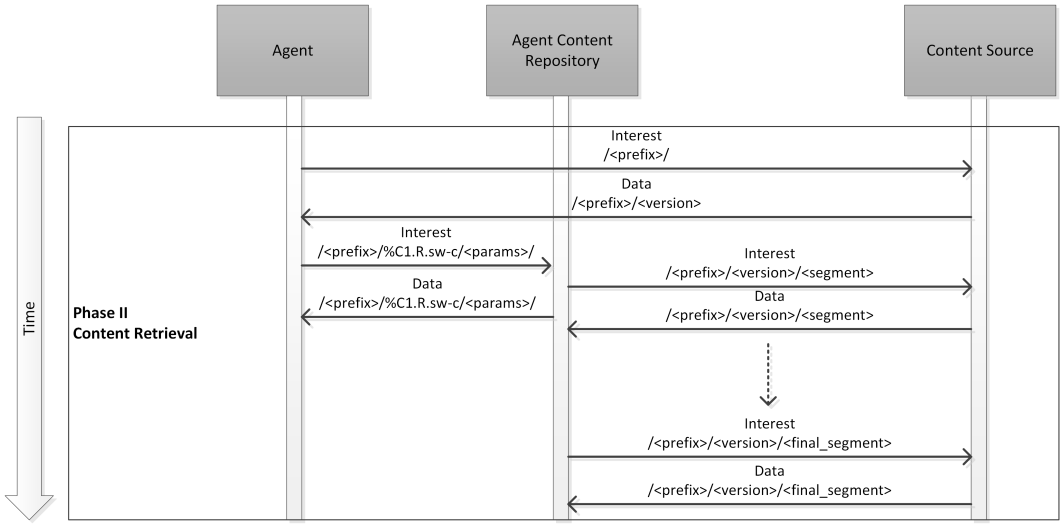

Agent
Agent Content Repository
Content Source
Time
Phase II
Content Retrieval
Interest
/<prefix>/
Data
/<prefix>/<version>
Interest
/<prefix>/%C1.R.sw-c/<params>/
Data
/<prefix>/%C1.R.sw-c/<params>/
Interest
/<prefix>/<version>/
Data
/<prefix>/<version>/
Interest
/<prefix>/<version>/<final_segment>
Data
/<prefix>/<version>/<final_segment>

Figure 3.8: Agent Handshake - Content Retrieval

3.3.3 Content Notification

Push Notification

Figure 3.9 shows the message flow of push-based content notifications. After delegating the job, the requester node sets up an Interest filter listening for Interests with the name `/notify/<jobID>`. Agents transmit such Interests periodically after completing content retrieval. In addition to the *jobID* the agent also appends the version of the retrieved content and some optional parameters. Optional parameters are for example host identifiers, i.e, the IP address, which enables the requester to create a unicast face to directly retrieve the content. This increases download speeds and saves energy because only requesters receive and process the transmitted content. As soon as the requester has completed the content retrieval, it replies an incoming notification messages with a `"DONE"` flag to tell agents that notification beaconing can be stopped and the job can be removed from the job queue.

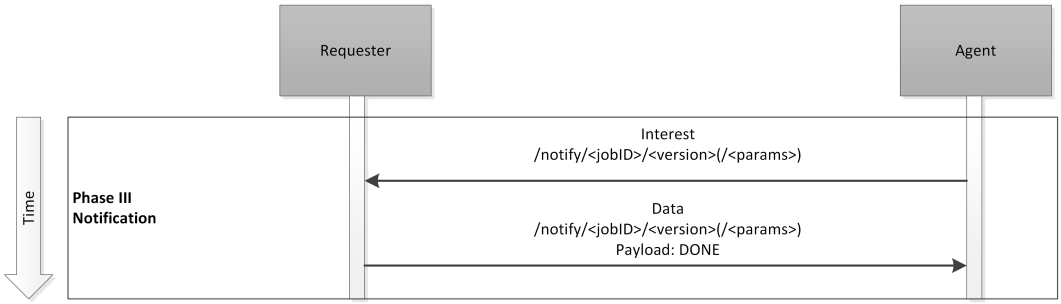


Figure 3.9: Agent Handshake - Push Notification

Pull Notification

The pull notification message sequence is illustrated in Figure 3.10. After delegating the job to an agent, the requester starts looking for agents that have completed their jobs by sending Interests with ContentName `ccnx:/notify/<jobID>/%C1.A.version`. The marker `%C1.A.version` is used to request the ContentVersion of the delegated job, which is required to download the content. Agents install an Interest filter to reply to such Interests as soon as they have completely retrieved the content. In Data replies, agents append the version to the ContentName as well as optional parameters such as, e.g., its own IP address for direct content retrieval via a unicast face.
Once the requester retrieved the ContentVersion, the delivery process is started. After successful download, the requester sends an Interest with the `%C1.A.done` marker to indicate that the job has been completed. The agent acknowledges this message to indicate that notification will be stopped by sending a response message with an `"ACK"` string. Then, the agent can remove the job from the job queue.

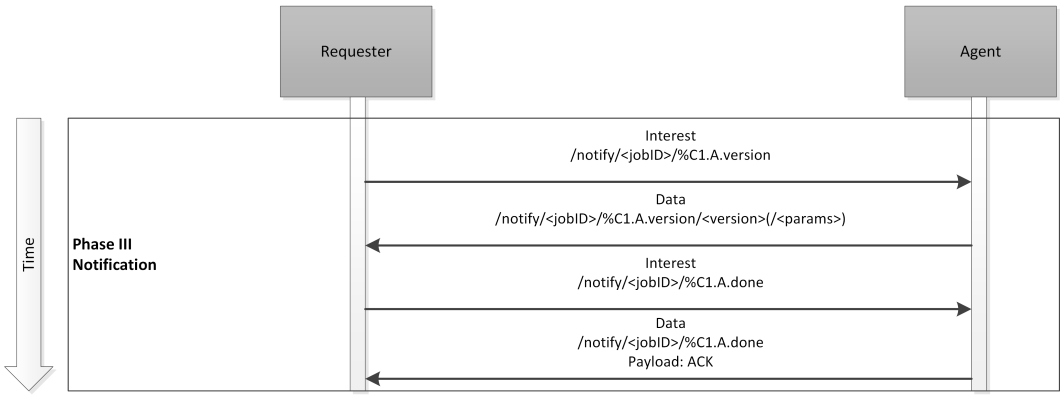


Figure 3.10: Agent Handshake - Pull Notification

3.3.4 Content Delivery

Figure 3.11 illustrates the content delivery process. Because the content version has been announced in the notification phase, the requester can directly request content from the agents' content repository. Data messages starting from segment 0 until the last segment (final bit set) have to be requested with Interest messages. Once the segment with the final flag arrives the content delivery is finished. Note that it is possible to use pipelining to fetch the content via multiple parallel Interests (for more detail see subsection 3.4.3), but the diagram shows only sequential content retrieval for simplicity.

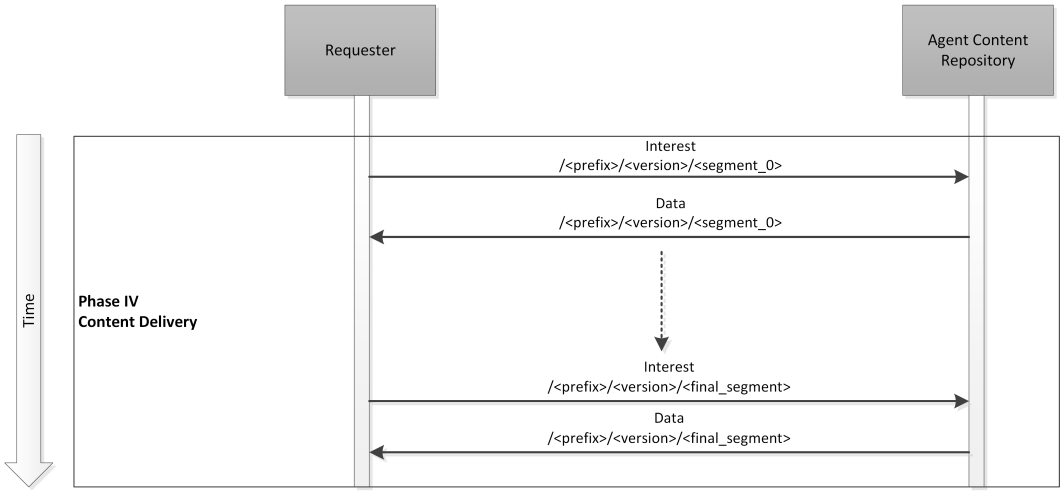


Figure 3.11: Agent Handshake - Content Delivery

3.4 Agent Process Applications

3.4.1 (Extended) Content Repository (ccnr)

To persistently store content and share it among other users, the CCNx framework provides the content repository application `ccnr`. In agent-based content retrieval the agent has to ensure the authenticity and integrity of retrieved data by providing original headers and signatures. Therefore, retrieved ContentObjects have to be completely imported into the agents' content repository. The CCNx repository protocol [26] offers functionalities to import content into an existing content repository.

Existing Repository Implementation

Two command markers under the R (for repository) namespace are available for importing content:

- **Start Write (`%C1.R.sw`)** requests that the repository retrieves and stores content consecutively starting from segment 0 until the final segment.
- **Checked Start Write (`%C1.R.sw-c`)** is similar to start write but the repository first checks whether the content is available in the repository and does not retrieve it again if this is the case. Different from Start Write, a segment number is required as starting point for the content request, thus, the repository import can start from a segment number different than 0.

A repository acknowledges accepted start write import requests with a RepositoryInfo object. The RepositoryInfo object contains a `Type` and an `InfoString` field to return response information. As soon as the repository acknowledges a start write command, it asynchronously imports the content by expressing Interest messages in every segment until the final object is received. This process is completely independent and the repository does not provide any information about the state of the download to the user.
This is problematic in mobile networks with intermittent connectivity, where the connection to a content source may break, i.e., download may be disrupted, before the whole content has been imported into the repository. When the connection is disrupted, the transmitted Interests will time out. The repository reexpresses expired Interests 5 times, before the import process is aborted. Once the process is aborted, it has to be restarted manually.
The current implementation of the repository has no mechanism to detect whether stored content is complete or not. If content is only partially available in the repository only these segments will be shared.

For the agent process it is necessary to store data persistently. Content in the content store (cache) can only be stored for a short time due to its limited size. However, content retrieved by agents may need to be available for multiple hours, days or weeks. Because no mechanism is available to check whether content is stored completely in the repository or not, the default `ccnr` repository is not suitable for the agent application. However, the agent application has to

ensure that content is completely stored before a requester will be notified about its availability. The default `ccnr` repository has been extended by a content validation (for completeness) and automatic resume functionality.

Extension

A new command marker for content validation (`%C1.R.vc`) has been introduced. Figure 3.12 shows the validation process and its output in a flow chart. To start a content validation request an Interest with the name `ccnx:/<prefix>/<version>/%C1.R.vc` has to be sent. The repository first checks if content under the requested ContentName is stored or not. If there is content for that name, the number of segments are counted and it is checked whether there are missing segments. The last considered segment should contain the final flag to indicate that the content is complete. As response a RepositoryInfo object is sent including information about content completeness. If the content is complete, the type is `"DATA"`, otherwise if content is not in the repository or segments are missing the type is `"INFO"`. For `"INFO"` RepositoryInfo objects, an additional InfoString denotes the next segment to request, i.e., segment 0 to start a download for the first time or the last valid segment to resume it.

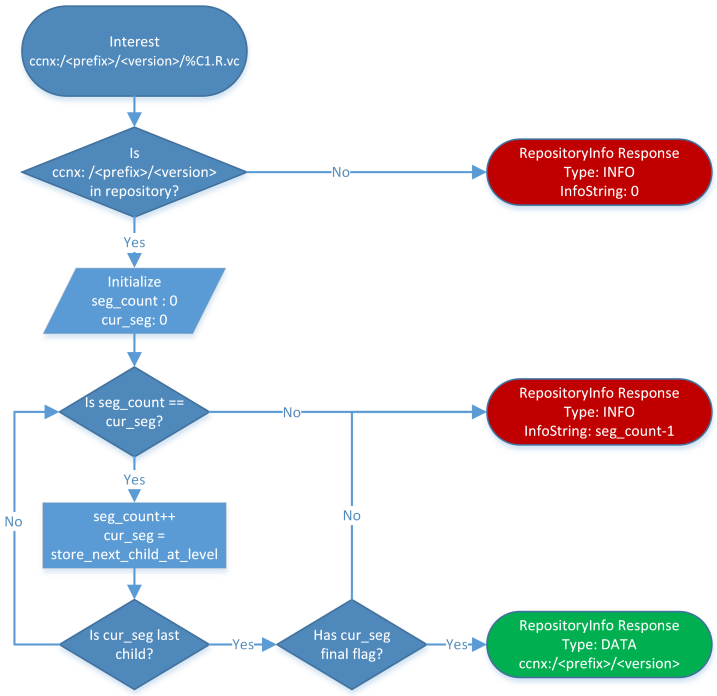


Figure 3.12: CCNR Repository Extension - Validate Content (`%C1.R.vc`)

Combining the content validation (`%C1.R.vc`) with the checked start write (`%C1.R.sw-c`) command enables to resume aborted imports at any time, since content validation provides the resume point for the additional import process.

3.4.2 Agent Application (ccna)

Delegation at Agent Application

The agent application is running on mobile nodes and is responsible for retrieving requested content for delegated jobs. Figure 3.13 gives an overview over the operations of the agent application. In the initialization phase the agent registers an Interest filter with the *prefix* `/ferrying`. Then, the application listens for Interests with `/ferrying` and processes incoming messages. Message processing happens asynchronously. Therefore, multiple requests can be handled at the same time. Depending on the command markers, included in the Interests' name, different actions will be triggered. To limit the processing overhead on the agent, the agent applications maintains a job queue with a limited number of job slots depending on the available resources.

The following command markers will be handled by the agent application:

- **Init Request `(%C1.A.init)`**
  If enough resources are available, the agent responds to the init request with a Data message (init response). The agent declares his identity in the init response by appending the *nodeID* to the content name (see subsection 3.3.1).
- **Delegation Request `(%C1.A.delegate)`**
  An arriving delegation request contains the *nodeID* of the selected agent. Therefore, the agent first checks whether the *nodeID* is equal to his own *nodeID* or not. If *nodeIDs* match, the delegation request will be acknowledged as described in subsection 3.3.1. The mandatory parameters *jobID* and *expireTime* included in the content name will then be stored in the job queue in order to start a new job thread. If the *nodeIDs* do not match, the Interest is ignored.

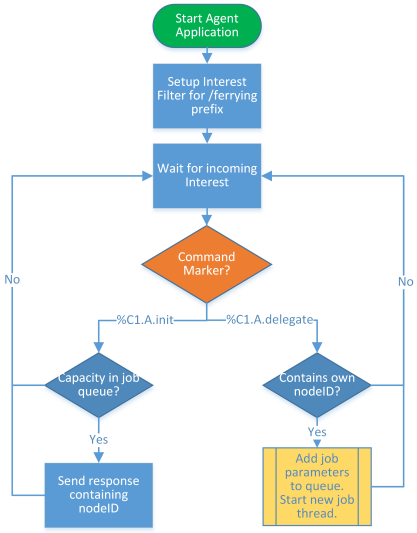


Figure 3.13: CCNA Agent Application - Main Loop

Content Retrieval at Agent Application

Figure 3.14 visualizes the process of a single job thread from the start, i.e., after the job has been delegated, until thread termination (due to expiration or successful completion). A job thread is responsible for content probing, repository import and requester notification.

A job thread is started with a reference to the entry in the job queue. In the initialization phase job parameters, i.e., ContentName, *expireTime*, notification type, etc, will be read from the job queue entry before the content retrieval process starts. Because the content import to the repository is asynchronously performed within the `ccnr` application, the agent application is only responsible for initializing, monitoring and resuming the repository import. Content probing, i.e., checking whether the content download is complete or needs to be resumed is continuously repeated until imported content is valid or job *expireTime* has been exceeded.

The content retrieval process is performed in cycles. In a first step content is searched in the local repository (see subsection 3.4.1). When no content is found locally, the thread periodically transmits content requests to resolve the newest content version. Finding a content version indicates that content is currently available. Therefore, the version number is appended to the ContentName and a repository import is started. While the content repository `ccnr` is importing content (independently), the thread sleeps for a few seconds (in our implementation: 1 second) and then checks for the completeness of the imported content.
The check includes testing for content validity (all segments received) and checking whether the *expireTime* has been exceeded. When the repository import has been finished, the validity check is successful and the content retrieval phase is finished. Otherwise, the content retrieval process continues in another cycle.

If content is found in the repository but not yet all segments have been retrieved, it is checked whether the repository import is still running. To check this, the highest valid segment numbers between two consecutive validation checks (`ccnx:/<prefix>/<version>/%C1.R.vc`) are compared. If the content import is still running, the thread will go to sleep and re-check the situation later. If no repository import is running, repository import is resumed. If content retrieval is successful, i.e., can be completed within the *expireTime*, the job state is updated to `COMPLETE` and the notification phase can start.

Depending on the notification type, an Interest filter for notification messages is registered (pull-based notification) or periodic notification messages are sent (push-based notification). After notification messages have been exchanged, retrieved content will be directly delivered from the content repository (`ccnr`). This content retrieval application is explained in the next subsection. The notification process is repeated until *expireTime* has been exceeded or the requester sends a job done message. Once a job reaches an end state, i.e., expired or done, the job information will be removed from the job queue and the thread will be destroyed.

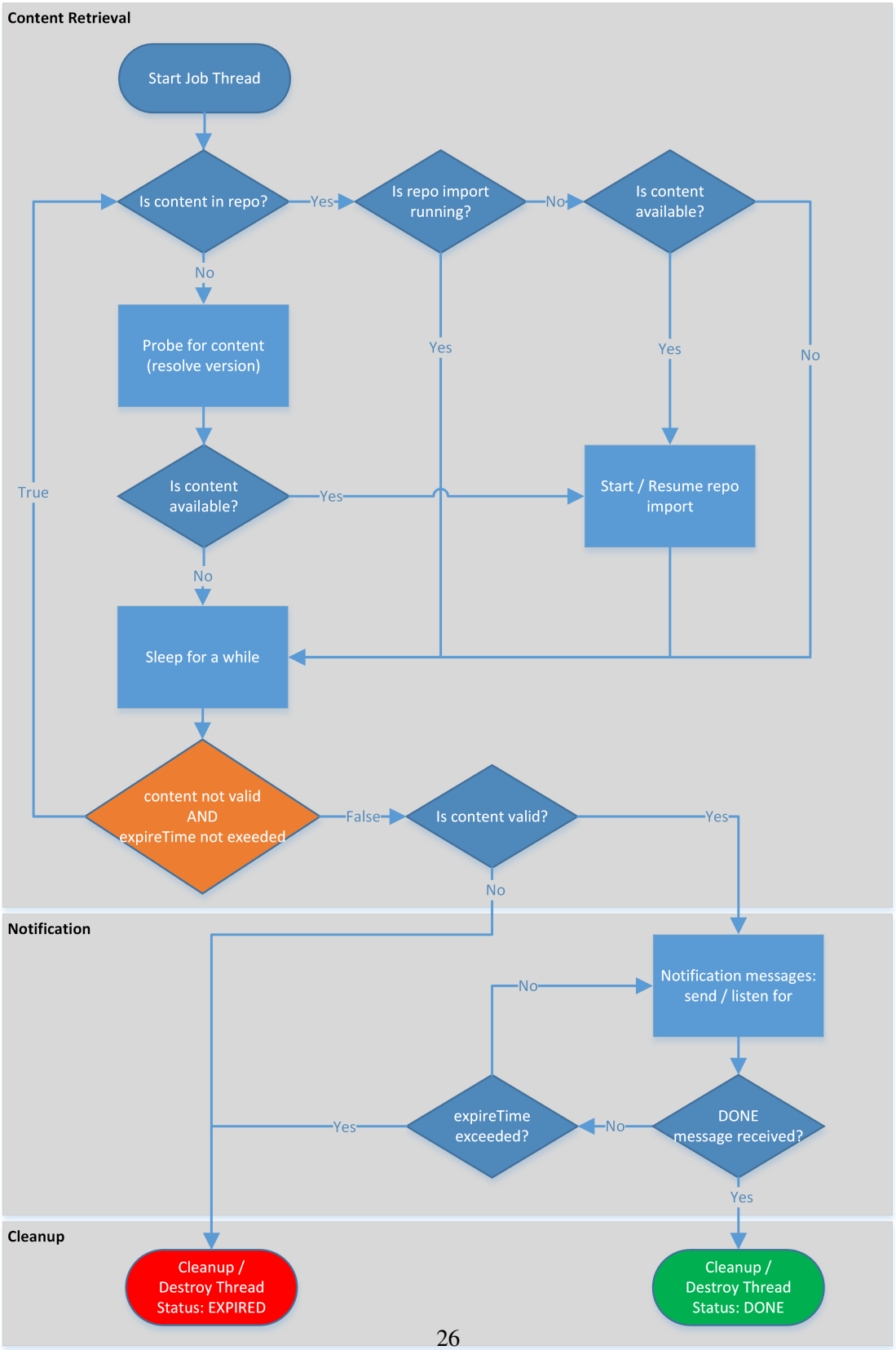


Figure 3.14: CCNA Agent Application - Job Thread

3.4.3 Requester Application (ccnar)

The requester application is responsible for delegating jobs and retrieve content as soon as a content notification has been received from an agent. For each job, a separate requester application (ccnar) instance has to be started, however, multiple jobs can be delegated at the same time by running multiple ccnar instances in parallel. Figure 3.15 shows an overview over the requester process. This process can be split in two subfunctions, namely 1) agent delegation and 2) notification / content retrieval.

Table 3.3 lists processes and parameters that are used in the requester application together with their description.

Term	Description
agent probing	Process of searching for available agent nodes by periodically sending init requests.
probe delay	Time to wait until a new agent probing is started, when no agent is found.
candidate list	List of potential agent candidates found while agent probing.
delegation list	List of agents the job is already delegated to, i.e., to avoid redelegation to the same agent.
delegation interval	Minimum amount of time between to agent delegations, if multiple delegations (for redundancy) are enabled.
maximum number of agents	Maximum number of agents a job is delegated to.

Table 3.3: Requester Application - Term Declaration

First, content retrieval needs to be delegated to agents. As described in subsection 3.3.1, init requests (first init request sent to one-hop-neighborhood, subsequent requests sent to retrieve responses from local cache) are transmitted within an agent request period (in our implementation 2 seconds) to find potential agents in the neighborhood. We call this *agent probing* in the following. Responding agents will be inserted in a *candidate list*. After the agent request period has elapsed, an agent is selected from the candidate list for delegation. In this work, agent selection from the candidate list is performed randomly but future work may investigate more sophisticated selection criterions such as mobility patterns or social parameters (see Section Future Work). Agent probing is performed periodically every delegation interval until the maximum number of delegated agents have been received or the agent delegation has been finished or aborted.

When a delegation is acknowledged, the *nodeID* is stored in a *delegation list* and will be excluded in subsequent delegations of the same job, e.g., if the agent does not return the content in time. Otherwise, if the delegation is not acknowledged, the candidate list will be cleared and after a *probe delay* (10s in our evaluations) the probing will be restarted. The candidate list will be cleared because candidate agents may have already disappeared, i.e.,

within the lifetime of the delegation message. Once the first delegation has been performed the applications activates the notification / retrieval mode. Depending on the notification type, periodic notification requests are transmitted (pull-based notification) or an Interest filter for notification messages is installed (push-based notification). If delegation to multiple agents is enabled, agent probing and delegation is periodically repeated (despite activated notification mode) every agent *delegation interval* until the *maximum number of agents* is reached or the job is finished (done or expired).

In the notification mode the requester application waits for notification messages of the pending jobs. When the agent is in range and transmits a notification message, the delivery process starts.

Content delivery is performed with the extended fetch library, which has been introduced in previous work [4] to support resume operations. Partial content and meta information as well as complete delivered data is persistently stored in a download folder on the hard disk.

If content download from the agent has not been completed, the applications stays in notification mode and waits for further notification messages until the delivery can be resumed or the job time expires. When the content download from the agent is complete, notification will be terminated like as described in subsection 3.3.3 and job data will be removed. After that, the application will be terminated.

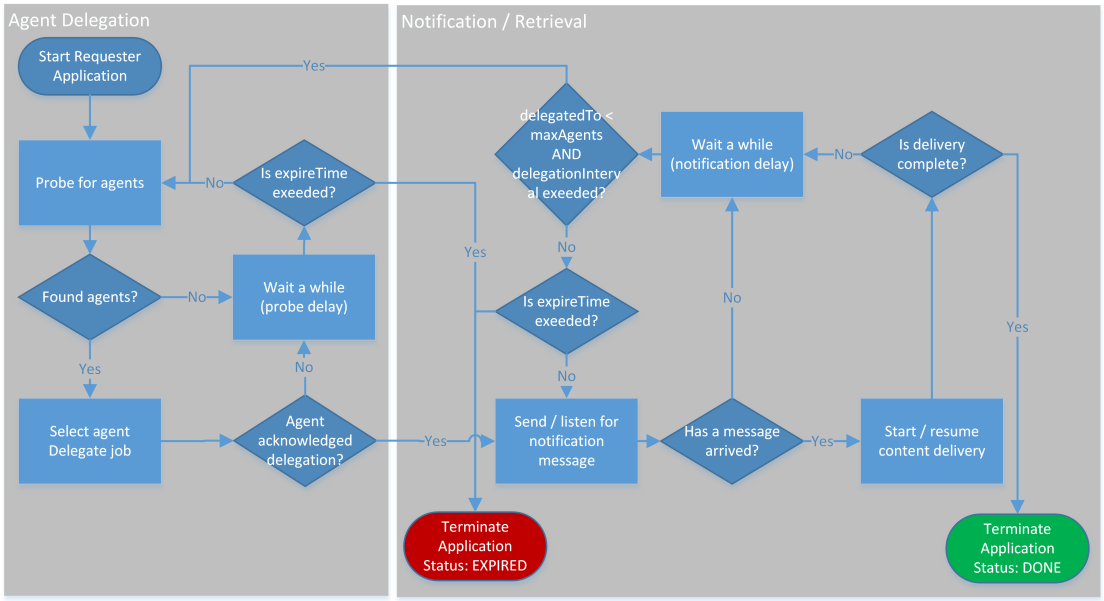


Figure 3.15: CCNAR Requester Application - Process

3.5 Multi-hop Requester Application (ccnacat_resume)

In the evaluation we compare agent-based content retrieval to multi-hop communication. In contrast to agents, requesters do not need to provide received content to others. Therefore, we decided to use a resume application ([4], [25]) at requesters, which stores received partial files and resumes disrupted transfers from where they were stopped. In contrast to the extended repository implementation for agents, retrieved content is not provided to others (private content). On the plus side, requesters can detect disruptions in connectivity quicker because they do not need to periodically check the repository if the file is complete (file transfer is performed only by the resume application). We have extended the resume application with a job expiration time (similar to agent jobs) to define an upper bound for file transmissions.
Without resume application, multi-hop content retrievals may never be completed if the multi-hop path is only available for a short time, i.e., not long enough to complete the entire file download. The resume application extends the default CCNx file transfer application `ccncat`, which has only a simple timeout handling. For example, if the connectivity between two mobile nodes is disrupted, Interests time out. With `ccncat`, Interests are reexpressed three times before the application is terminated and received (incomplete) data is discarded. Another problem occurs if at application start no content source is available, because version resolving needs to be performed before content download can start. In `ccncat`, which assumes continuous connectivity, Interests for version resolving are only retransmitted once and the application terminates as soon as the retransmitted Interests expires. In contrast, the resume application continues version resolving until content is found or the job expiration time has been reached.

Fig 3.16 illustrates the processing steps of the resume application, which has been used at requesters. At application start, the application checks for existing resume information from previous (disrupted) file transfers. If resume information is found, the download is initialized from the segment where it was disrupted the last time (stored in resume information). Otherwise, if there is no resume information available, the download at the beginning with segment 0. The application requests content segments and writes it to a partial file until the file transfer is completed or disrupted (three consecutive timeouts of the same segment denote a disruption). If the content download has been completed, the partial file is renamed and the application is terminated with job state `"DONE"`.
Otherwise, if the content download has been disrupted, i.e., no more content can be currently retrieved, the application checks first, whether the expiration time has already been exceeded. If the job has not been expired yet, resume information, i.e., content name, content version, next expected segment number (see [4] for more information), will be stored locally. After a configurable time interval (default: 10s, but 1s in our evaluations), a resume operation can be performed. This means that the application tries to download the rest of the file starting from the next expected segment number (resume information). If the expiration time has been exceeded and the content download has not been completed, the application terminates with the status `"EXPIRED"`.

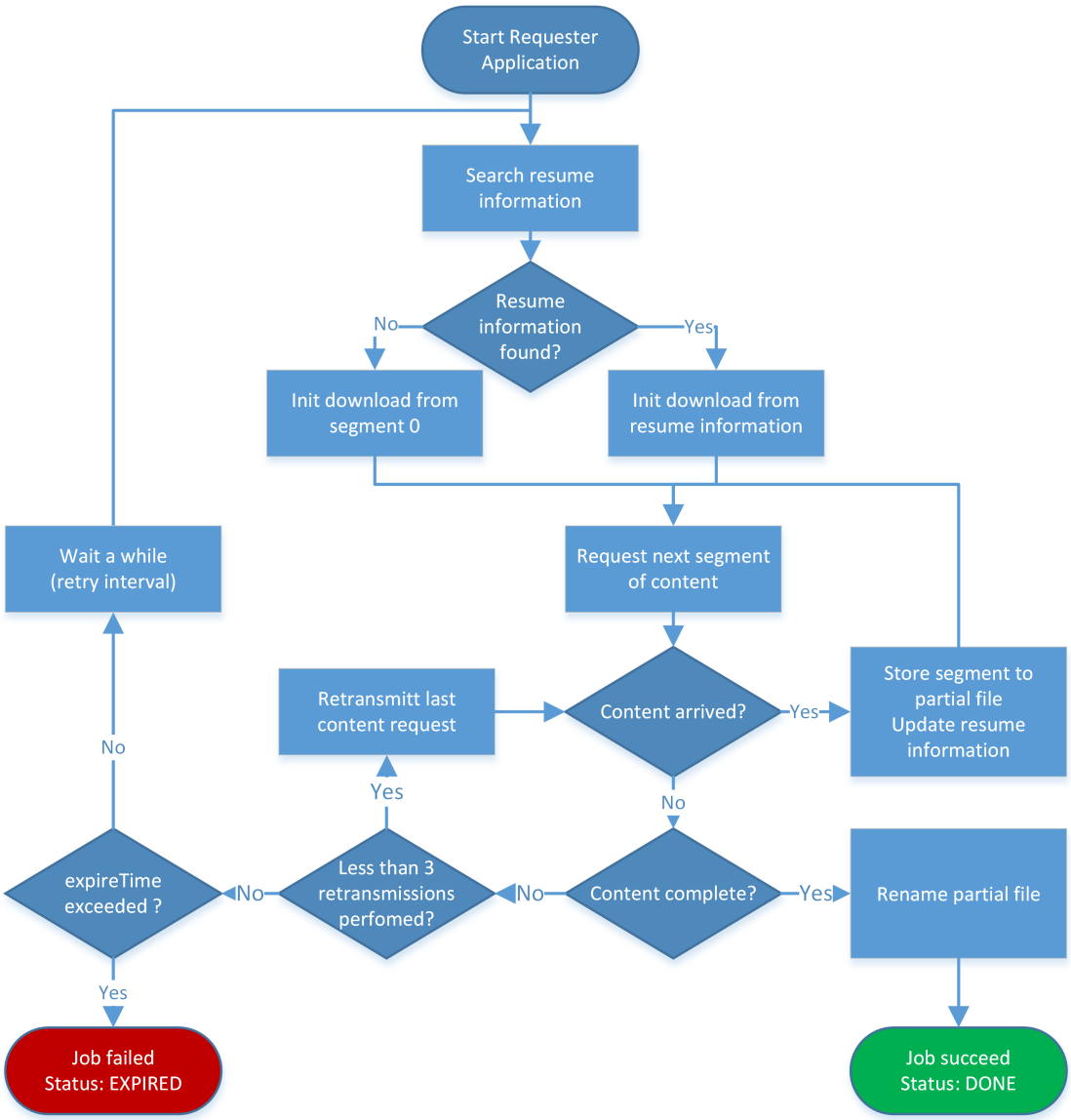


Figure 3.16: Multi-hop Requester Application - Process

Chapter 4

Evaluation

In this chapter the performance of agent-based content retrieval is evaluated and compared to multi-hop communication.
In particular, the following questions should be answered after the evaluation:

- Which notification type is more efficient, pull or push notification?
- When is it suitable to use the agent approach and when to use multi-hop transfer?
- Is it possible to resume interrupted content retrieval from one or more source?
- Can agent-based content retrieval benefit from the routing strategy ”Dynamic Unicast”?

4.1 Evaluation Scenario

The evaluation scenario can be described as an everyday situation and is visualized in Figure 4.1.

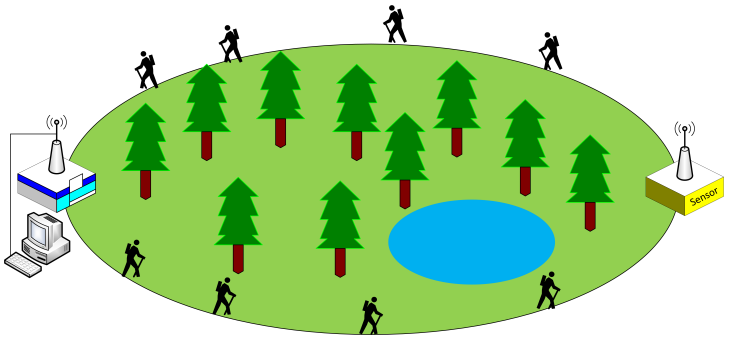


Figure 4.1: Evaluation Scenario

There is a round trip hiking trail to a popular spot on a hill. At this popular spot a solar powered sensor gathering data (e.g., weather data, webcam snapshot, etc...) is installed. At the start of the trail is the tourist office, which is interested in the sensor data. Unfortunately, the sensor is too far away for a direct wireless connection and a wired link is also missing. In order to obtain the data anyway, multi-hop or agent-based content retrieval can be applied. Since a lot of hikers are on this trail their mobile phones can be used to build opportunistic networks.

To support this scenario, a circular mobility model has been implemented into NS3, where hikers (in the remainder of this section called nodes) follow a circular path. The nodes (except requester and content source, which are static) are uniformly random distributed on the whole trail and move with velocities within a specified range. From time to time the nodes take a break to enjoy the beautiful view or eat something. This *pause time* is a configurable parameter. In addition, there may be regular hikers, which move at regular speed, and lazy hikers, which move with slower speed and make longer breaks. In the remainder, we refer to lazy hikers as ”bad nodes”.

4.2 Evaluation Environment

The implementation of the agent process was tested with the Direct Code Execution framework for the ns-3 network simulator. This frameworks has been used to test the native source code of CCNx, i.e, without simplifications, in mobile and wireless networks with many nodes.

> ”Direct Code Execution (DCE) is a framework for ns-3 that provides facilities to execute, within ns-3, existing implementations of userspace and kernelspace network protocols or applications without source code changes.”[27]

In addition to the DCE framework, ns-3 modules have been used to simulate and log the environment, namely node emulation, mobility, wireless connectivity.
The defined scenarios have been evaluated on the grid system of University of Berne (UBELIX).

> ” UBELIX is a Linux High Performance Computing Cluster used for computational extensive tasks. Currently, UBELIX runs about 225 nodes on which up to 2700 jobs can be run simultaneously.” [28]

The UBELIX cluster has been used to run multiple simulation instances of a certain scenario in parallel in order to save time. The UBELIX grid systems allows to submit up to 1200 jobs at once per user. Because the architecture of a grid system differs from a single workstation, the DCE framework has been patched to run on the UBELIX cluster.

To evaluate the huge amount of generated log and result files a comprehensive message analyzer tool suite has been written in Python. Log files are parsed and necessary information is extracted in order to analyze the scenarios.

4.3 Evaluation Setup

In this section the simulation parameters and metrics are explained. In addition, the global parameters, which are used for all evaluations, are defined in this section.

4.3.1 Parameter Description

Table 4.1 contains a list of parameters, which are used in the evaluation scenarios. These parameters are introduced and explained here.

Parameter	**Description**
Topology	
Simulation Mobility	Mobility model of mobile nodes.
Requester Nodes	Number of nodes interested in receiving content.
Source Nodes	Number of nodes offering content.
Mobile Nodes	Number of nodes with agent or forwarder capabilities.
Source Position	Position of the source node in the area.
Requester Position	Position of the requester node in the area.
Node Speed	Speed of the mobile nodes.
Pause Time	Pause time of mobile nodes.
Simulation	
Simulation Length	Maximum number of seconds the simulation runs.
Simulation Skip Time	Number of seconds to run mobility model before events will be started.
Run Number	Number of runs (with different seed) per scenario.
Requester	
File Size	Size of the transmitted data.
Notification (agent only)	Notification type (push / pull).
Expiration Time	Maximum number of minutes to retrieve data.
Max Agents	Maximum number of agent delegations per job.
Notification Interval	Number of seconds between two notification messages.
Delegation Interval	Number of seconds between two agent delegations.
Agent Request Period	Number of seconds until agent is selected from candidate list.

Table 4.1: Parameter Description

4.3.2 Global Parameters

Table 4.2 contains parameters which are the same for all evaluation scenarios.
To simulate common network conditions, the wireless 802.11g standard in the 2.4 GHz channel range has been used. The NS-3 `YansWifi` module (described in [29]) has been used because it includes common radio simulation models (i.e., wireless channel propagation loss and delay). In this simulation a free space area has been simulated by applying the `ns3::ConstantSpeedPropagationDelayModel` (speed of light) and `ns3::LogDistancePropagationLossModel` [30] model. Table 4.2 lists applied parameters for these models. With these parameters, the wireless transmission range is approximately 130m.

Parameter	**Description**
Network	
Wireless Standard	802.11g
Wireless Channel	2.4GHz
Log Distance Exponent	3.0
Reference Loss	40.0 dB
Energy Detection Threshold	-86.0 dBm
CCAMode1 Threshold	-90.0 dBm
Simulation	
Simulation Skip Time	7200s (2h)
Run Number	100
Requester	
Max Agents	1, 5, 10
Notification Interval	1s
Delegation Interval	10s
Agent Request Period	2s

Table 4.2: Global Parameters

4.4 Notification type: Pull vs. Push

In this section the notification types push and pull are compared.

4.4.1 Parameters

In Table 4.3 the evaluation parameters are shown, a description what these parameters mean can be found in Table 4.1. The results discussed in this section consist of all possible parameter permutations which led to 1200 simulation runs per notification type. In our circular evaluation scenario contact time of source and agent nodes is limited due to mobility behavior.

Parameter	**Description**
Topology	
Simulation Mobility	Circle (Radius 250m, Circumference 1570.80m)
Requester Nodes	1
Source Nodes	1
Mobile Nodes	5, 10, 20, 50
Source Position	(100, 400) in 1000 x 1000 m playground
Requester Position	(400, 100) in 1000 x 1000 m playground
Bad Nodes Amount	0%, 50%, 75%
Node Speed (Good Nodes)	Uniform Distributed Random Value: 1 - 1.4m/s
Node Speed (Bad Nodes)	Uniform Distributed Random Value: 0.7 - 1m/s
Pause Time (Good Nodes)	30s
Pause Time (Bad Nodes)	1200s
Simulation	
Simulation Length	21600s (6h)
Requester	
File Size	1MB (name: ccnx:/ccnx.org/4K1024KB, content objects: 251)
Notification (agent only)	push, pull
Expiration Time	180min (3h)

Table 4.3: Push vs. Pull Notification: Parameters

4.4.2 Retrieval Time

In this subsection the retrieval time of agent-based content retrieval is analyzed. The retrieval time is the time elapsed from starting until stopping the requester application due to successful content delivery or job expiry. In agent-based content retrieval, the path length from requester to content source and back has a direct influence on the retrieval time. In the ideal (fastest) case, the agent walks to the source, retrieves the content and returns directly to the requester, i.e., without any waiting times, thus, the time needed to walk from requester to the source and back is given by the movement speed and can not be reduced.

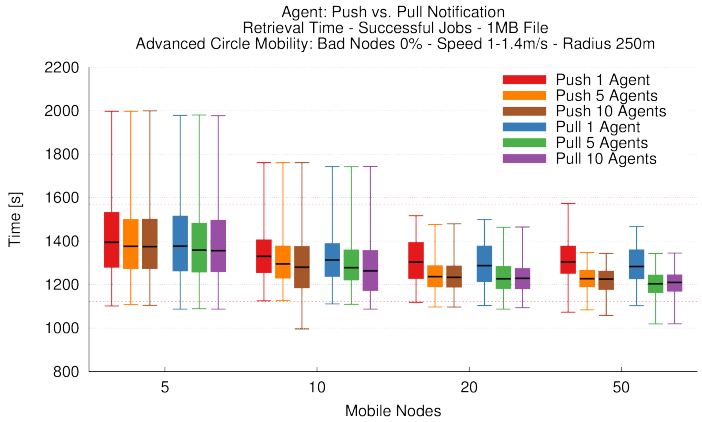


Figure 4.2: Push vs. Pull Notification: Retrieval Time - 0% Bad Nodes

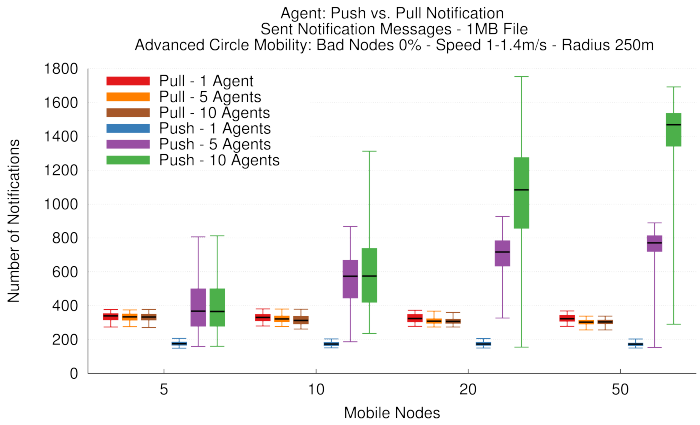
Figure 4.2 shows the retrieval time of pull and push notification without bad nodes. This means that the node speed for each mobile node is in the range 1 to 1.4 m/s and the pause time varies between 0 and 30s. The x-axis shows the number of mobile nodes (potential agents) and the y-axis the retrieval time, i.e., time from agent delegation until receiving the content, in seconds. Per x-axis label six box plots are drawn, the left three boxes show the retrieval time for pull notification with 1, 5 and 10 delegated agents and the right three the same for push notification. Comparing corresponding results for pull and push notification shows that the median retrieval time as well as the inter-quartile range are nearly equal. Thus, the notification type does not have an impact on the retrieval time.

Figure 4.2 reveals two more things:

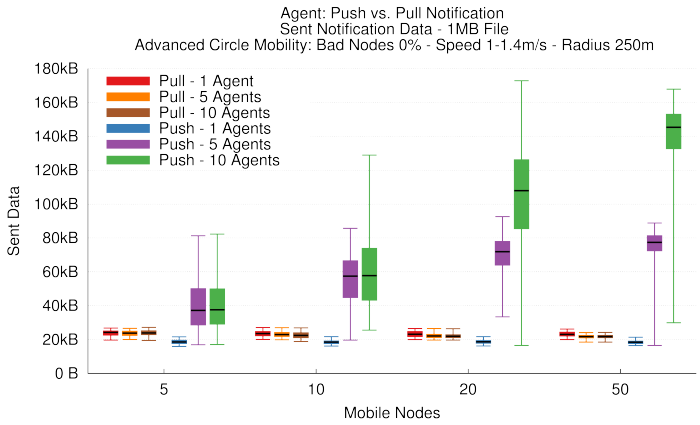
- Delegating jobs to more than one agent, results in shorter retrieval times. Mobile nodes move with varying velocities in a range of 1 to 1.4 m/s and randomly perform occasional breaks between 0 and 30s. Therefore, the chance to select a node which finishes the round trip faster than others increases with increasing number of delegations.
- With an increasing number of mobile nodes on the movement path, the inter-quartile range of the retrieval time decreases. Because more nodes are available on the path, agents can be delegated quicker and, thus, jobs can be finished earlier.

4.4.3 Notification Messages / Data Sent

To avoid medium occupation and save resources for other transmissions as well as energy, transmitted notifications should be reduced to a minimum. Therefore, the number of transmitted notifications as well as the size of transmitted notifications are suitable performance metrics. Because notification messages differ in size for different types (push-based notifications are larger, see Chapter 3 for more details), the number of transmitted messages as well as the size of transmitted messages need to be evaluated.



(a) Sent Messages



(b) Transmitted Data

Figure 4.3: Push vs. Pull Notification: Notifications - 75% Bad Nodes

Figure 4.3(a) shows on the y - axis the number of transmitted notification messages and the x-axis the number of mobile nodes (potential agents) in the network. The number of transmitted pull notification messages is approximately the same independent of the number of delegated agents, while the push notifications increase drastically with the number of delegated agents, i.e., 50 mobile nodes with 5 agents lead to 2.5 times more messages with push-based notifications

compared to pull-based notifications. With 10 delegated agents, push-based notifications result in even 4.5 times more notifications. Furthermore, it is also visible that for push notifications to multiple agents the number of transmitted messages increases with increasing number of mobile nodes (node density). Due to the fact that the node density is higher, agent delegations can be performed faster. For this reason the agents walk close together and start push notifications only with a small time delay.
Thus, push notification generates more messages as soon as two or more agents have been delegated (assuming that they both meet the content source, retrieve the content and start the notification phase). Only if delegation is performed to one agent, push notifications result in fewer notifications. This is due to the fact that push notifications do not start until content retrieval is complete, while we perform pull notifications periodically starting already immediately after job delegations (because we do not make any assumptions when the download will be finished). However, pull based notification could be optimized if certain assumptions would be made, i.e., estimating the agent's return time based on the assumed distance between requester and content source. However, such optimizations have not been considered in this work. Thus, if agent delegation is performed to more than 1 agent, pull notifications should be preferred.

In Figure 4.3(b) we show the size of transmitted notification bytes for both push and pull based notifications.
We would like to emphasize that pull notifications are relatively small (around 70 bytes per message) because they do not contain detailed information, such as content version, parameter payloads including host identifier etc. This information is only included in the response to pull notifications, thus, not periodically transmitted but only returned in response to a pull notification. Push-based notifications, however, contain a parameter payload, e.g., version of retrieved content, host identifier such as IP address to create direct face, etc., and are, therefore, 42% larger, i.e., around 100 bytes. Because notifications are periodically transmitted (beacons), smaller messages are more efficient in terms of transmitted bytes.
Figure 4.3(b) shows that, although we used a very basic strategy for pull-notifications (right after agent delegation), which can be greatly optimized, pull-based notifications result only in 30% more transmitted bytes than push notifications in case of 1 delegated agents. If jobs are delegated to more agents, the overhead for push-notifications increases even more, i.e., for 50 mobile nodes and 10 delegated agents 6.67 times more data is transmitted with push-based notifications compared to pull-based notifications. These results again demonstrate the efficiency of pull notifications, even without any optimizations.

However, homogeneity of nodes mobility behavior like in the previous scenario is not guaranteed, i.e., it may happen that some agents move very slow and need a lot of time to return to the requester. In order to analyze such a case, the scenario has been repeated with 75% bad nodes, i.e., they have a slower speeds between 0.7 and 1 m/s and have longer pause times of 0 up to 1200s.

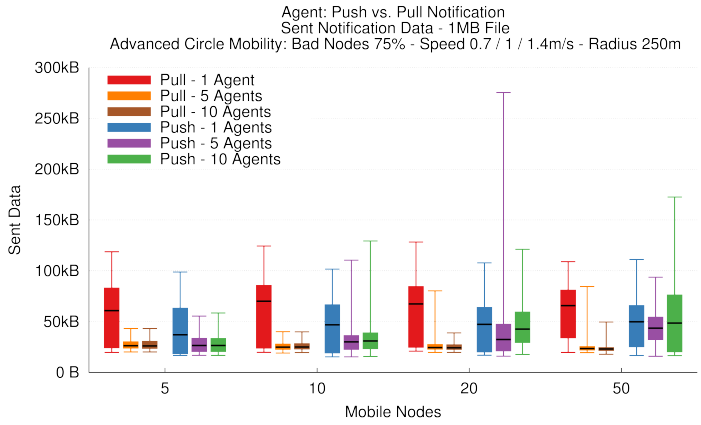


Figure 4.4: Push vs. Pull Notification: Notification Data Sent - 75% Bad Nodes

Figure 4.4 shows the amount of transmitted notification bytes for different node densities. The transmitted notification bytes are affected by bad nodes because bad nodes need more time to deliver the content to the requester, thus, the notification period is longer. Please recall that we use an unoptimized strategy for pull-based notifications, which starts directly after agent delegation. Therefore, in scenarios with only 1 agent, the number of transmitted notification bytes (median, 75-quantile and max values) is significantly worse than in the previous scenario in Figure 4.3(b) without bad nodes. Furthermore, we can observe that pull-based notifications are more affected by bad nodes than push-based notifications. If a bad node is selected as agent, it will require more time to travel the path to the content source and back. Because push-based notifications start only after the content has been retrieved, i.e., at the content source, fewer notifications are sent compared to pull-based notifications, which start immediately after delegation.

4.4.4 Summary

The notification types do not have a significant impact on the retrieval time but the number of transmitted notifications as well as the transmitted bytes. The notification type can be negotiated during agent delegation depending on the network scenario and the number of expected agent delegations. As a rule of thumb, we can say that pull notifications should be used if more than one agent delegation will be performed. The overhead of pull notifications in terms of transmitted bytes for only one agent is relatively small. Therefore, in the remainder of this thesis, we only use pull-based notifications.

4.5 Network Approach: Agent vs. Multi-hop

In this section the requirements and conditions for efficient and successful data transfer using agent-based and multi-hop content retrieval are analyzed.

4.5.1 Fixed Path Length, Variable Density, Fixed File Size

Parameters

In Table 4.4 the applied parameters are shown. An explanation what these parameters mean can be found in Table 4.1. The results discussed in this section consist of all possible parameter permutations which led to a total of 4800 simulation runs.

Parameter	**Description**
Topology	
Simulation Mobility	Circle (Radius 250m, Circumference 1570.80m)
Requester Nodes	1
Source Nodes	1
Mobile Nodes	5, 10, 25, 50
Source Position	(100, 400)
Requester Position	(400, 100)
Bad Nodes Amount	0%, 50%, 75%
Node Speed (Good Nodes)	Uniform Distributed RandomValue: 1 - 1.4m/s
Node Speed (Bad Nodes)	Uniform Distributed RandomValue: 0.7 - 1m/s
Pause Time (Good Nodes)	30s
Pause Time (Bad Nodes)	1200s
Simulation	
Simulation Length	21600s (6h)
Requester	
File Size	1MB (name: ccnx:/ccnx.org/4K1024KB, content objects: 251)
Notification (agent only)	pull
Expiration Time	180 (3h)

Table 4.4: Agent (Pull) vs. Multi-hop: Parameters

Job State

In the first step of the evaluation the job state has been analyzed to find out which approach succeeds under which conditions.

There are three possible job states:

- **Done**: The requested content has been completely retrieved.
- **Expired**: The requested content has not been found within the given expiration time.
- **Interrupted**: Only parts of the content could be retrieved within the given expiration time.

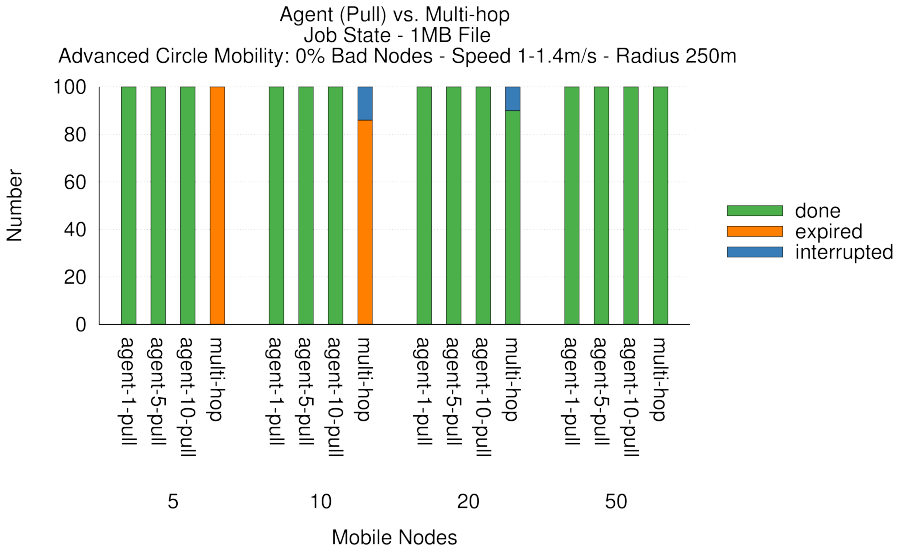


Figure 4.5: Agent (Pull) vs. Multi-hop: Job State - 100 Runs

We compare agent-based content retrieval to multi-hop communication for different node densities. Figure 4.5 shows the job state of agent-based content retrieval compared to multi-hop in a circular topology with a radius of 250m. The x-axis shows the number of nodes in the network and the y-axis the number of runs. In total, we performed 100 evaluation runs for every configuration. If the node density is high, e.g., with 50 nodes corresponding to an average density of 31.42m between nodes, multi-hop communication can always finish. However, with lower densities, multi-hop communication may not always be successful. For example, with 20 nodes corresponding to a density of 78.54m between nodes, only 90 % of the jobs can be finished while 10 % of the jobs are interrupted. Interrupted jobs require more time to finish, i.e., for this 10% interrupted jobs, the requester has retrieved on average only 26% of the file. This illustrates that although node density would suggest that multi-hop communication is possible (i.e., 78.54m between nodes and transmission range of 130m), there may be disruptions since mobile nodes do not move at the same constant speeds, i.e., there is no uniform node distribution. For

10 nodes, the node density corresponds to 157.1m between nodes which is slightly more than a wireless transmission range. As Figure 4.5 shows, even in this case, some packets can still be retrieved with multi-hop in 14% of the cases, however, the nodes retrieve on average only 30% of the file. For an even lower density, i.e., 5 nodes correspond to 314.2m between nodes no multi-hop content transmission can be completed. However, as Figure 4.5 shows, agent-based content retrieval can always finish independent of the node density.

Retrieval Time

Next, we want to evaluate the content retrieval time with multi-hop and compare it to the retrieval time of agent-based content retrieval.

Figure 4.6: Agent (Pull) vs. Multi-hop: Retrieval Time - Successful Jobs - 0% Bad Nodes

Figure 4.6 compares on the y-axis the retrieval time of the agent approach with multi-hop communication for a different number of mobile nodes in the network (x-axis). For every label on the x-axis, four box plots are shown: the left three box plots show agent-based content retrieval for a different number of agent delegations and the box plot on the right shows the multi-hop results. Recall that multi-hop cannot finish content downloads for 5 and 10 nodes, hence, there is no box plot for these densities. Thus, only results for 20 and 50 nodes are available for multi-hop. The shaded area denotes the time that would be required to walk around the circle with a constant velocity between 1 and 1.4m/s.

Figure 4.6 shows that the median retrieval time for all agent parameters is approximately equal and within the shaded area (time to walk around the circle). In case of low node densities, there may be slightly higher maximum values because it may take longer until a node is seen, i.e., until an agent can be delegated. For multi-hop, content can be retrieved for 20 and 50 nodes. In case of 50 nodes, the density is high enough such that a continuous path between requester and source can be guaranteed at all times. In this case, multi-hop communication can result in 388 % faster transmission compared to agent-based content retrieval. However, if node density is slightly lower, e.g., with 20 nodes, the retrieval via multi-hop communication requires in

median 321% more time (and in the worst case even 716% more time). The reason for the higher content retrieval time at lower content densities are disruptions caused by nodes moving at different speeds.

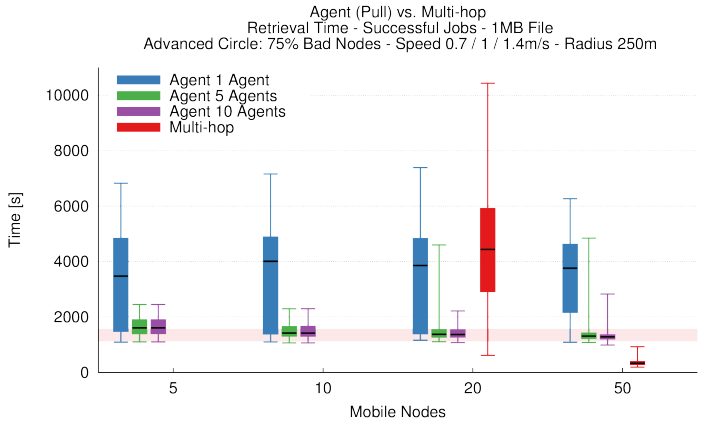


Figure 4.7: Agent (Pull) vs. Multi-hop: Retrieval Time - Successful Jobs - 75% Bad Nodes

Figure 4.7 shows the retrieval times in case of 75% bad nodes. While bad nodes do not have a significant influence on multi-hop retrieval times, i.e., node density is much more important, they may have a large influence on agent-based content retrieval if only one agent has been selected. This behavior is expected because agent-based content retrieval depends on agent mobility, i.e., time until content is retrieved and the agent returns to the requester for content delivery. If there are many bad nodes, e.g., here 3 out of 4 nodes, the probability of selecting a bad node is high. Therefore, in case of expected bad nodes, it may be a better strategy to delegate content retrieval to multiple redundant agents since content retrieval time can then be decreased, i.e., the probability that one of the delegated agents is a good node increases. Furthermore, we would like to note that the probability to select multiple bad nodes increases for an increasing number of mobile nodes, i.e., 75% of them are bad, and the number of agent delegations remains constant in our evaluations (1, 5 or 10). This is visible in Figure 4.7 by the higher maximum retrieval time values for 20 and 50 mobile nodes with 5 and 10 agent delegations.

4.5.2 Variable Path Length, Fixed Density, Fixed File Size

In our next evaluations, we evaluate agent-based content retrieval for different path lengths. To ensure that multi-hop communication is possible, we adapt the number of mobile nodes to ensure a minimum density for multi-hop communication.

Parameters

As Table 4.5 shows, we also evaluate radii of 375m and 500m. The number of mobile nodes has been adapted for these radii, so that the distances between nodes is on average 31.42m and 78.54m. Because of the longer paths with larger radii, the simulation length as well as the expiration time has been extended linearly to the path length.

Radius [m]	**250**	**375**	**500**
Topology			
Mobile Nodes for Density 78.5	20	30	40
Mobile Nodes for Density 31.4	50	75	100
Number of Hops (estimated)	7	10	13
Round-trip distance [m]	1570.80	2356.19	3141.59
Source Position	(100, 400)	(150, 600)	(200, 800)
Requester Position	(400, 100)	(600, 150)	(800, 200)
Simulation			
Simulation Length	21600s (6h)	32400s (9h)	43200s (12h)
Requester			
Expiration Time	180 (3h)	270 (4.5h)	360 (6h)

Table 4.5: Agent (Pull) vs. Multi-hop: Parameters for Radii 250, 375 and 500m

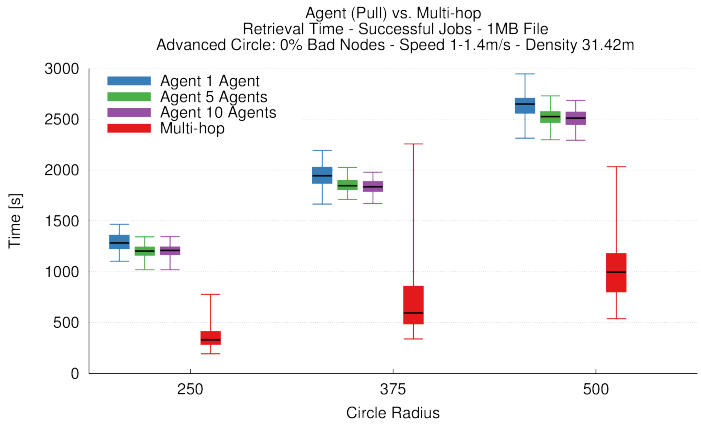
Table 4.6 shows the values of the remaining parameters. Their meaning are described in Table 4.1.

Parameter	**Description**
Topology	
Simulation Mobility	Circle (Radii: see Table 4.5)
Requester Nodes	1
Source Nodes	1
Bad Nodes Amount	0%
Node Speed (Good Nodes)	Uniform Distributed RandomValue: 1 - 1.4m/s
Pause Time (Good Nodes)	30s
Requester	
File Size	1MB (name: ccnx:/ccnx.org/4K1024KB, content objects: 251)
Max Agents	1
Notification (agent only)	pull

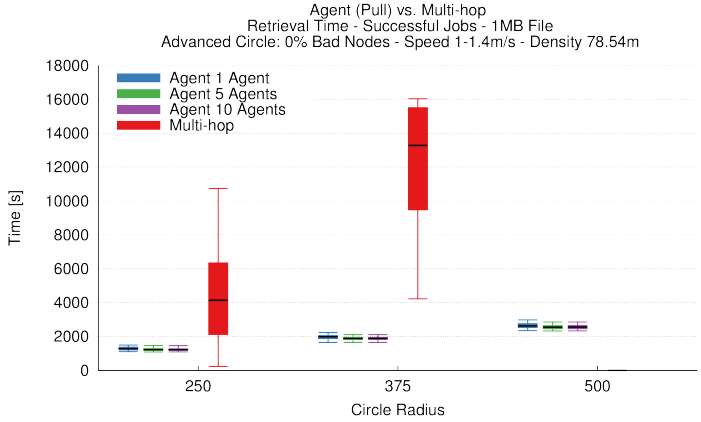
Table 4.6: Agent (Pull) vs. Multi-hop: Parameters

Retrieval Time

Figure 4.8 shows the retrieval times (y-axes) for different radii (x-axis) and different node densities.



(a) Density 31.42 m



(b) Density 78.54 m

Figure 4.8: Agent (Pull) vs. Multi-hop: Retrieval Time - Successful Jobs - 0% Bad Nodes

Figure 4.8(a) shows the content retrieval time for a high node density of 31.42m between nodes. As expected the agent retrieval times increase with increasing radius because the circumference, i.e., path the agent needs to travel, increases. As observed before, retrieval times with 1 agent are slightly (around 5 %) longer than with multiple agents.
Similarly, the content retrieval via multi-hop also increases for increasing radii because more hops are required to reach the content source. Surprisingly, multi-hop communication increases more than linearly with increasing hop distance, i.e., by a factor of 3 from a radius of 250m to a radius of 500m. Figure 4.8(a) shows that if the density is high, multi-hop communication results always in faster content retrieval times compared to agent-based content retrieval.

Figure 4.8(b) shows the same evaluation for a lower node density, i.e., on average 78.54m between nodes. Retrieval times of agent-based content retrieval do not increase much compared to the high density (cf. Fig. 4.8(a)), i.e., the slight increase can be explained by a slightly longer time until agent delegations can be performed. However, multi-hop content retrieval times increase significantly because of more frequent disruptions in case of longer path lengths. For radii of 500m multi-hop retrieval is not even possible anymore, i.e., 77 % of the jobs are interrupted (i.e., they retrieve on average only 11% of the content) and 23% expired (i.e., requester does not retrieve a single content object).

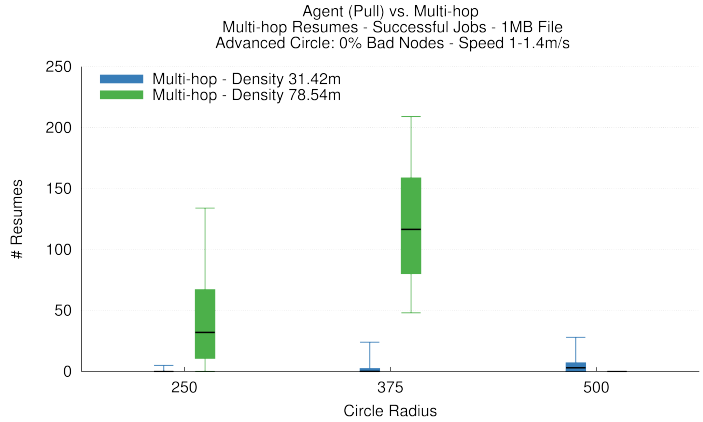


Figure 4.9: Agent (Pull) vs. Multi-hop: Multi-hop Resumes - Successful Jobs - 0% Bad Nodes

Figure 4.9 shows the number of resume operations for successful content retrievals via multi-hop communication in both node densities. The number of resume operations indicates disruptions in path availability. The x-axis shows the different radii and the y-axis the number of resume operations. For every x-label two box plots are shown, on the left side multi-hop transmission with density 31.42m and on the right side multi-hop transmission with density 78.54m. Figure 4.9 shows that the number of resume operations increases with increasing radii. However, with a high density of in average 31.42m between nodes, the median values are always very low, i.e., below 5 for a radius of 500m. But for a low density of 78.54m between nodes, there are already 25 resumes for a 250m radius and more than 100 resume operations for a radius of 375m (recall that the file size is 1MB, i.e., 251 content objects). These values confirm the assumption of frequently disrupted paths for density 78.54m.

Messages - Mobile Nodes

Besides job state and retrieval time, the network load is an important metric to measure transmission efficiency in opportunistic networks. The average number of messages transmitted by all mobile nodes in a simulation run have been evaluated.

The box plots show the statistics over 100 runs. In particular, we focus on transmitted/forwarded Interest messages (Figure 4.10), transmitted Data messages (Figure 4.11) and received duplicate Data messages (Figure 4.12).

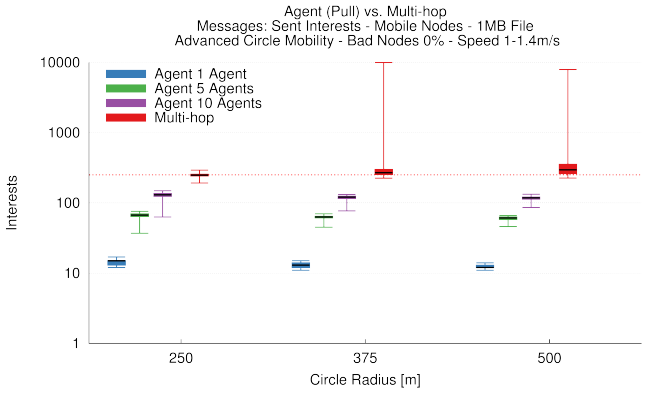


Figure 4.10: Agent (Pull) vs. Multi-hop: Sent Interests - 0% Bad Nodes - Density 31.42m

Figure 4.10 shows the transmitted Interest messages (logarithmic scale on y-axis) by mobile nodes on average for different radii (x-axis). The red dotted line denotes the minimum number of Interests to retrieve the content over one hop.

Agents transmit Interests for content probing, i.e., to find the content source and retrieve content. In our evaluations, we send an Interest for content probing every 1 second. Therefore, a certain number of Interest messages is required for agent-based content retrieval. Although the number of transmitted Interests by agents increases linearly with increasing distance/radius, the number of nodes increases as well (to keep the same density), thus, the average number of Interests for content probing and retrieval stays constant. As expected, the number of transmitted Interests increase with an increasing number of agents.
In the multi-hop case, Interests from a requester are forwarded by mobile nodes to the content source. However, even with 5 or 10 agents, which is a high overhead because 1 agent would be enough, mobile nodes send on average still 81% (5 agents) and 47% (10 agents) fewer Interests than via multi-hop communication. As Figure 4.10 shows, the average number of transmitted Interests by each multi-hop forwarder node is approximately 251 (dotted lines) for 1MB file with 251 segments. For increasing radii with multi-hop communication, the number increases even slightly due to more frequent disruptions and retransmissions.

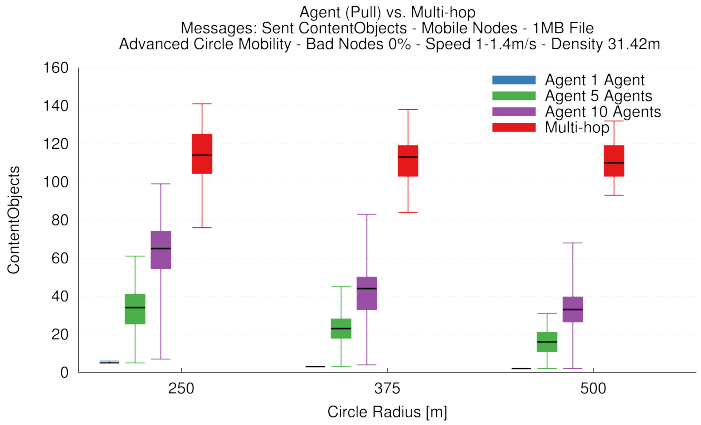


Figure 4.11: Agent (Pull) vs. Multi-hop: Sent Content - 0% Bad Nodes - Density 31.42m

Figure 4.11 shows on the y-axis the Data messages transmitted by mobile nodes on average. If content retrieval has only been delegated to one agent, no broadcast content transmissions will be performed by mobile nodes and the agent only delivers the content, i.e., the 251 Data messages, to the requester via content source. However, due to the number of mobile nodes in the network, the number of content transmissions from the agent via unicast is negligible. For a radius of 250m (50 nodes), a mobile node sends on average 5 content objects, for 375m (75 nodes) it is 3 content objects and for 500m (100 nodes) only 2 content objects.
These are considerably fewer content objects than for multi-hop communication where on average 112 Data messages, i.e., 45% of all Data messages, are transmitted.
For more agent delegations, i.e., to 5 or 10 agents, the number of transmitted content objects increases because agents include content to their repository and may, therefore, provide it to others. This illustrates the importance of agent delegation and in particular, the number of agents that need to be delegated. In this scenario with high node density and no bad nodes, agents are delegated in a short time frame and, thus, perform content retrieval near the content source at approximately the same time. Since content requests are transmitted via broadcast, agents may retrieve content from the content source as well as other agents. Furthermore, mobile nodes (non-agents) may also overhear content transmissions, keep it in the cache to provide it to others. However, even with 10 agent delegations, the number of content transmissions is still approximately 50% lower compared to multi-hop (250m) and even 75% lower for 5 agent delegations.

We can observe that the number of transmitted content objects decreases with agent-based content retrieval for an increasing distance because the number of delegated agents (active nodes) stay the same, i.e., 1, 5 or 10, and the number of mobile nodes (passive nodes) increase to keep the same node density. However, the number of content objects transmitted via multi-hop broadcast does not decrease for an increasing node density and stays approximately the same. This illustrates that broadcast content transmission is not efficient for long path lengths and high node densities.

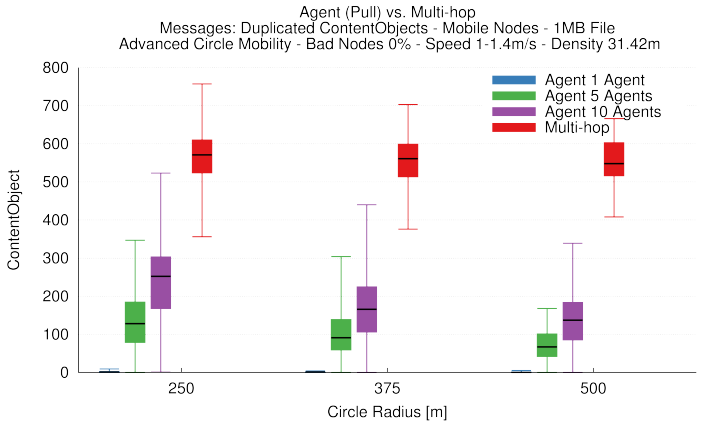


Figure 4.12: Agent (Pull) vs. Multi-hop: Duplicated Content - 0% Bad Nodes - Density 31.42m

Figure 4.12 shows the average number of duplicated ContentObjects (y-axis) per mobile node for a node density of 31.42m between nodes. It shows that for such a high node density, the number of received duplicate content objects is huge, i.e., more than twice the amount of needed content objects. Because broadcast messages are forwarded to all nodes in the vicinity, nodes in range of a content source receive the ContentObjects and keep it in the content store (cache). As soon as these ContentObjects are requested again (e.g., retransmissions or Interests forwarded via alternative path), all nodes will answer and therefore produce a lot of duplicates.

During agent-based content retrieval, duplicate content transmissions are also possible in case of multiple agents. Nodes can overhear content transmissions and respond to other agents requesting the content.

4.5.3 Variable Path Length, Fixed Density, Variable File Size

Finally, agent-based content retrieval is compared to multi-hop content retrieval with variable file sizes. Content transfers with agents depend on the contact time between agent and content source. In this evaluations, we have selected file sizes, which can be downloaded during this contact time. In order to have a constant path for multi-hop transmission, a node density of 31.42m between nodes has been used. Table 4.7 shows the applied scenario parameters for different radii in order to reach equal node densities.

Parameters

Radius [m]	**250**	**375**	**500**
Topology			
Mobile Nodes for Density 31.4	50	75	100
Number of Hops (estimated)	7	10	13
Round-trip distance [m]	1570.80	2356.19	3141.59
Source Position	(100, 400)	(150, 600)	(200, 800)
Requester Position	(400, 100)	(600, 150)	(800, 200)
Simulation			
Simulation Length	21600s (6h)	32400s (9h)	43200s (12h)
Requester			
Expiration Time	180 (3h)	270 (4.5h)	360 (6h)

Table 4.7: Agent (Pull) vs. Multi-hop: Parameters for Radii 250, 375 and 500m

Table 4.8 shows applied scenario parameters for the topology as well as for the requester. The parameters are defined in Table 4.1.

Parameter	**Description**
Topology	
Simulation Mobility	Circle (Radii: see Table 4.7)
Requester Nodes	1
Source Nodes	1
Bad Nodes Amount	0%
Node Speed (Good Nodes)	Uniform Distributed RandomValue: 1 - 1.4m/s
Pause Time (Good Nodes)	30s
Requester	
File Size	1MB (name: ccnx:/ccnx.org/4K1024KB content objects: 251)
	3MB (name: ccnx:/ccnx.org/4K3072KB content objects: 751)
	5MB (name: ccnx:/ccnx.org/4K5120KB content objects: 1251)
Max Agents	1
Notification (agent only)	pull

Table 4.8: Agent (Pull) vs. Multi-hop: Parameters

Retrieval Time

Figure 4.13 compares the retrieval time in seconds (y-axis) to retrieve files of 1, 3 and 5MB with multi-hop and agent-based content retrieval for different circle radii (x-axis). We can observe that retrieval times of agent-based content retrieval, i.e., median as well as inter-quartile ranges, stay approximately constant for different file sizes and same radii as long as they can be finished during the contact time of agent and content source.

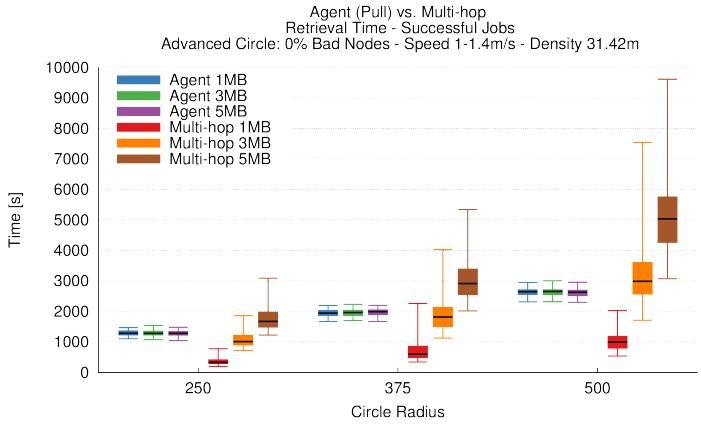


Figure 4.13: Agent (Pull) vs. Multi-hop: Retrieval Time - Successful Jobs - 0% Bad Nodes - Density 31.42m

The retrieval time increases linearly with the radius because agent-based content retrieval is highly depending on agent mobility, i.e., the path length from source to requester and back. However, content retrieval times over multiple hops increase for increasing file sizes even for the same radius. Therefore, we can observe that agent-based content retrieval can be even more efficient than multi-hop communication in high density networks for larger file sizes. For example, for a radius of 250m, agent-based content retrieval may be more efficient than multi-hop communication for content transmissions of 5MB. The larger the distance, the more efficient becomes agent-based content retrieval. For example for a radius of 500m, agent-based content retrieval is already more efficient for file transmissions of 3MB. Because agent-based content retrieval exploits agents' mobility and retrieves content directly from the content source, it is independent of node density and disruptions on the path.

4.5.4 Summary

Agent-based content retrieval works reliable even under conditions where multi-hop communication fails (sparse mobile node density) or works only with bad performance (a lot of connection disruptions). In any case applying agent-based content retrieval results in less transmitted messages compared to multi-hop content retrieval, since content retrieval happens only over one hop by a few selected nodes, whereas with multi-hop communication messages are flooded to all available network nodes. Evaluations have shown that agent-based content retrieval is fast and reliable when transferring large files over long distances. However, agent-based content retrieval is highly depending on the mobility of agent nodes. The minimal time to get desired content, i.e., called retrieval time, is limited by the distance from source and requester.

4.6 Interrupted Content Retrieval: Multiple Sources

Agent-based content retrieval depends on the contact time between agent and content source. If the content transfer can not be completed at once, it needs to be resumed at a later time. In previous experiments the file size was chosen, so that content retrieval could be completed at once in the available contact time to one content source. In these evaluations, we want to evaluate the performance of resume operations of the agent application in mobile environments, i.e., content retrieval from multiple content sources. Therefore, we place multiple content sources in the topology and select file sizes that cannot be downloaded during the contact time to one content source. Furthermore, we also evaluate a higher node mobility, which results in shorter contact times between agents and content sources.

4.6.1 Parameters

Table 4.9 lists the parameters of this scenario. Three identical content sources (with the same repository) are placed on the circle. Like in previous experiments, agent nodes are mobile and move based on our circular mobility model, while requester and content sources are static. Requester and the three content sources are placed in equidistance on the circle, i.e., the distance is a quarter of the circumference (see coordinates of requester and sources in Table 4.9). We evaluate two different mobile speeds and adapt the retrieved file sizes accordingly. For pedestrian mobility of 1.0-1.4m/s, we set the file size to 20MB and for vehicular mobility, we set the file size to 2MB. Please note that file sizes with faster speeds are smaller because of shorter contact times to the content sources. In total 400 runs have been performed.

Parameter	**Description**
Topology	
Simulation Mobility	Circle (Radius 500m, Circumference 3141.59m)
Requester Nodes	1
Source Nodes	3
Mobile Nodes	40, 100
Source Position	(200, 200), (200, 800), (800, 800)
Requester Position	(800, 200)
Bad Nodes Amount	0%
Node Speed (Good Nodes)	Uniform Distributed RandomValue: 1.0 - 1.4m/s & 10 - 14 m/s
Pause Time (Good Nodes)	30s
Simulation	
Simulation Length	43200s (12h)
Requester	
File Size	20MB (name: ccnx:/ccnx.org/4K20480KB, content objects: 5001) 2MB (name: ccnx:/ccnx.org/4K2048KB, content objects: 501)
Notification (agent only)	pull
Max Agents	1
Expiration Time	360 (6h)

Table 4.9: Interrupted Content Retrieval: Parameters

4.6.2 Message Overhead

In order to analyze the resume capability of the agent application, the message overhead has been evaluated and compared to a static scenario where the content can be downloaded from the same content source (no disruptions).

Source Nodes - Sent ContentObjects

Table 4.10 shows the relative overhead of transmitted Data messages by content sources compared to a single static content source (average over 100 runs).

File Size	**2 MB**	**20 MB**
Speed	10 - 14 m/s	1 - 1.4 m/s
Contact Time	Short	Long
# content objects	501	5001
40 Mobile Nodes		
Average of retrieved content objects	518.45	5188.1
Std Dev	4.09	15.07
Overhead compared to static content source	**3.48%**	**3.74%**
Std Dev (Relative)	**0.82%**	**0.30%**
100 Mobile Nodes		
Average of retrieved content objects	516.72	5164.55
Std Dev	3.51	12.74
Overhead compared to static content source	**3.14%**	**3.27%**
Std Dev (Relative)	**0.68%**	**0.25%**

Table 4.10: Interrupted Content Retrieval: Sent ContentObjects 3 Sources - Average 100 Runs

Due to node mobility, a requester may request segments from a content source while still in range but then be too far away for the reception, i.e., missed content transmission resulting in overhead because more messages are transmitted by content sources. For higher node speeds (10-14m/s), the overhead is slightly smaller than for slower speeds (1-1.4m/s), however, the difference is negligible. We can see that the overhead for resuming file transfers from different content sources is for both speeds very low, i.e., below 4%. Since no connection setup needs to be performed, agents can exploit the complete contact time for content retrieval. Furthermore, we can see that the overhead is slightly smaller for 100 mobile nodes compared to 40 nodes. This is because for a higher node density, the probability is higher that other mobile nodes may have overheard missed content transmissions and provide it to the agent before reaching the next content source. In this case, the next content source does not need to provide the same content objects again.

Agent Nodes - Sent Interests

Table 4.11 shows the relative overhead of total transmitted Interests by agent nodes.

File Size	**2 MB**	**20 MB**
Speed	10 - 14 m/s	1 - 1.4 m/s
Contact Time	Short	Long
# content objects	501	5001
40 Mobile Nodes		
Average of transmitted Interests	633.5	6826.40
Std Dev	18.98	94.96
Overhead compared to static content source	**26.45%**	**36.50%**
Std Dev (Relative)	**3.79%**	**1.90%**
100 Mobile Nodes		
Average of transmitted Interests	629.47	6786.19
Std Dev	21.17	61.25
Overhead compared to static content source	**25.64%**	**35.70%**
Std Dev (Relative)	**3.36%**	**1.22%**

Table 4.11: Interrupted Content Retrieval: Sent Interests Agent Node - Average 100 Runs

The Interest overhead is significantly higher than the content overhead, since more Interests are transmitted for content probing, i.e., looking for a content source. In our implementation, agents probe periodically for content every second. For higher node speeds, the overhead is lower, because periods without connectivity to content sources are shorter (the distance can be traveled faster requiring fewer probing Interests).
Nevertheless this overhead does not drastically affect network load since Interest messages are relatively small (around 75 bytes, compared to Data messages with around 4500 bytes). Furthermore, Interest forwarding is limited to only one hop.

4.7 Content Retrieval with Dynamic Unicast

Agent-based content retrieval uses one-hop broadcast communication to find and retrieve content. Unfortunately, the throughput of broadcast communication is very small compared to unicast communication. Therefore, the content retrieval with unicast is faster than with broadcast communication. Thus, more content can be transferred during a given time, i.e., in case of short contact times due to mobility. In another ongoing master thesis [31] a dynamic routing mechanism has been introduced. This routing mechanism is able to create unicast routes based on overheard broadcast content. In this section this routing mechanism, called Dynamic Unicast, has been applied to the already known scenario. Due to its implementation in the CCNx daemon, it is not required to change the agent application. The agent application transmits Interests via broadcast first and the routing mechanism creates a unicast face to the content source as soon as content has been received via broadcast. Subsequent Interests will be sent via the unicast face until content is completely transferred or Interests time out. When Interests transmitted via unicast face time out, the face is deleted and further Interests will be transmitted again via broadcast.

Table 4.12 shows the parameters applied in this evaluation. Dynamic Unicast (with SFF forwarding strategy) has been tested in the circle scenario with 3 content sources. The mobile nodes move with vehicular mobility (speed: 10-14m/s).

4.7.1 Parameters

Parameter	**Description**
Topology	
Simulation Mobility	Circle (Radius 500m, Circumference 3141.59m)
Requester Nodes	1
Source Nodes	3
Mobile Nodes	100
Source Position	(200, 200), (200, 800), (800, 800)
Requester Position	(800, 200)
Bad Nodes Amount	0%
Node Speed (Good Nodes)	Uniform Distributed RandomValue: 10 - 14 m/s
Pause Time (Good Nodes)	30s
Simulation	
Simulation Length	43200s (12h)
Requester	
File Size	10MB (name: ccnx:/ccnx.org/4K10240KB, content objects: 2501) 20MB (name: ccnx:/ccnx.org/4K20480KB, content objects: 5001)
Notification (agent only)	pull
Max Agents	1
Expiration Time	360 (6h)

Table 4.12: Dynamic Unicast: Parameters

4.7.2 Message Overhead

Source Nodes - Sent ContentObjects

Table 4.11 shows the relative overhead of transmitted Data messages by content sources compared to a single static content source (average over 100 runs).

File Size	**10 MB**	**20 MB**
# ContentObjects	2501	5001
100 Mobile Nodes (absolute values)		
Unicast Average	2496.03	5005.02
Broadcast Average	6.58	7.21
Unicast + Broadcast Average	2502.61	5012.23
Unicast + Broadcast Overhead	**0.06%**	**0.22%**
Unicast Std Dev	0.82	55.8
Broadcast Std Dev	1.16	1.65
Unicast + Broadcast Std Dev	1.13	56.07
Unicast + Broadcast Std Dev	**0.05%**	**1.12%**
100 Mobile Nodes (relative values)		
Unicast Average	99.74%	99.86%
Broadcast Average	0.26%	0.14%
Unicast + Broadcast Average	100%	100%

Table 4.13: Dynamic Unicast: Sent ContentObjects - 3 Source Nodes - Average 100 Runs

Dynamic Unicast results in an overhead of less than 0.3% Data messages. The overhead for the 20MB file is slightly higher because with Dynamic Unicast the 10MB file can be mostly retrieved from one source, whereas content retrieval of the 20MB file has to be resumed from a second source. As Table 4.13 shows Data messages are mainly transmitted via unicast links. Receiving only a few Data messages via broadcast is enough to build a unicast face directly to the content source.

Table 4.14 summarizes relative overhead values of broadcast (as shown in Table 4.10) and Dynamic Unicast evaluations (as shown in Table 4.13).

Type	**File**	**Speed**	**Overhead**
Broadcast	2MB	10-14m/s	3.48%
Dynamic Unicast	20MB	10-14m/s	0.22%
Difference - Equal Speed		10-14m/s	**3.26%**
Broadcast	20MB	1-1.4m/s	3.74%
Dynamic Unicast	20MB	10-14m/s	0.22%
Difference - Equal File	20MB		**3.52%**

Table 4.14: Dynamic Unicast: Content Overhead Sources - Broadcast vs. Dynamic Unicast

Comparing the overhead of transmitted Data messages of Dynamic Unicast with one-hop broadcast shows that the overhead is around 3.25% smaller for Dynamic Unicast. Please note that for file sizes of 20MB with one-hop broadcast the content source contact time is longer since the node speed is smaller by a factor of 10, while for the 2MB file the contact time is equal to this scenario (same speed). Content retrieval via unicast links is more reliable and efficient than via one-hop broadcast, because retransmission in case of collisions can performed on the MAC-Layer. Furthermore, applying Dynamic Unicast helps reducing network load since messages are not flooded to all nodes in the network but only forwarded to the required nodes.

Agent Nodes - Sent Interests

Table 4.15 shows the relative overhead of transmitted Interests by agents compared to a single static requester (average over 100 runs).

File Size	**10 MB**	**20 MB**
# ContentObjects	2501	5001
100 Mobile Nodes (absolute values)		
Unicast Average	2504.13	5014.85
Broadcast Average	414.66	809.63
Unicast + Broadcast Average	2918.79	5824.48
Unicast + Broadcast Overhead	**16.70%**	**16.47%**
Unicast Std Dev	3.81	56.22
Broadcast Std Dev	18.9	53.88
Unicast + Broadcast Std Dev	20.09	86.69
Unicast + Broadcast Std Dev	**0.80%**	**1.73%**
100 Mobile Nodes (relative values)		
Unicast Average	85.79%	86.10%
Broadcast Average	14.21%	13.90%
Unicast + Broadcast Average	100%	100%

Table 4.15: Dynamic Unicast: Sent Interests - Mobile Nodes - Average 100 Runs

Like already seen in Table 4.11 of the previous section, the Interest overhead is significantly higher than the content overhead, since more Interests are transmitted to find content (content probing). Table 4.16 summarizes the overhead of transmitted Interests for broadcast (as shown in Table 4.11) and Dynamic Unicast (as shown in Table 4.15) with the same node speed. The overhead of transmitted Interests with Dynamic Unicast is about 9% smaller than with broadcast.

Type	**File**	**Speed**	**Overhead**
Broadcast	2MB	10-14m/s	25.64%
Dynamic Unicast	20MB	10-14m/s	16.47%
Difference - Equal Speed		10-14m/s	**9.17%**

Table 4.16: Dynamic Unicast: Interest Overhead Agents - Broadcast vs. Dynamic Unicast

With Dynamic Unicast the Interest overhead compared to one-hop broadcast is reduced because of two main reasons. First fewer Interest retransmissions need to be performed with Dynamic Unicast, because collision handling is performed on the MAC-Layer. Second because larger files (more Data messages) can be transmitted with Dynamic Unicast the relative overhead for content probing is smaller.

File Size	**10 MB**	**20 MB**
# ContentObjects	2501	5001
100 Mobile Nodes (unanswered Interests - absolute values)		
Unicast Average	8.26	19.71
Broadcast Average	297.98	683.91
Unicast + Broadcast Average	306.24	703.62
100 Mobile Nodes (unanswered Interests - relative values)		
Unicast Average	0.33%	0.39%
Broadcast Average	71.86%	84.47%
Unicast + Broadcast Average	10.49%	12.08%

Table 4.17: Dynamic Unicast: Unanswered Interests - Mobile Nodes - Average 100 Runs

Additionally, Table 4.17 shows the difference between transmitted Interest and received Data messages. Comparing the values shows that less than 1% unicast Interests do not receive a Data message in response. This demonstrates the fact that unicast routes are removed relatively fast if content is no longer available, i.e., the content source is out of connectivity range and the Interest lifetime is exceeded. In this case, Interests, i.e, to resume content repository import as well as to periodically probe for content, are transmitted via broadcast. The amount of unanswered broadcast Interests is significantly higher (around 70 - 85%) due to its utilization for content probing.

4.8 Discussion

After analyzing the results of the different scenarios we are able to answer the questions formulated at the beginning of this chapter.

Which notification type is more efficient, pull or push notification?
The choice of the notification type depends on the network scenario and the number of expected agents. However, pull notifications should be used if more than one agent delegation will be performed since the amount of transmitted bytes is smaller than with push notifications. However, applying pull notifications for jobs with only one agent delegation results in a small overhead of transmitted bytes.

When is it suitable to use the agent approach and when to use multi-hop transfer?
Agent-based content retrieval is suitable for transferring large files fast and reliable over long distances even under difficult mobility conditions, e.g., sparse node density. In any case applying agent-based content retrieval results in less transmitted messages compared to multi-hop content retrieval because messages are only transmitted by one node and not forwarded by multiple nodes. Due to high dependency on the mobility of the agent nodes, the minimal time to get desired content, i.e., called retrieval time, is limited by the distance from source and requester.

Is it possible to resume interrupted content retrieval from one or more source?
By using the developed repository extension, the completeness of content can be validated and the last valid segment can be identified. Repository import can then be resumed using the identified segment and results only in a small overhead of transmitted Data messages by content sources.

Can agent-based content retrieval benefit from the routing strategy "Dynamic Unicast"?
Agent-based content retrieval can benefit from the routing strategy "Dynamic Unicast" in several ways. Retrieving content via unicast increases the download throughput, i.e., more data can be transferred during the limited contact time. Furthermore, network load is reduced since unicast connectivity prevents network flooding by transmitting messages only to required nodes.

Chapter 5

Conclusions

5.1 Summary and Conclusion

In this master thesis we developed an agent and requester application with resume capability in order to evaluate agent-based content retrieval in environments with node mobility. The performance and behavior of the implemented agent protocol has been measured and compared to multi-hop content retrieval. During the evaluation important parameters to improve performance have been searched and found.

Evaluations have shown that pull notifications are more efficient than push notifications in most cases, i.e., for delegations to more than one agent. Further, we found out that agent-based content retrieval enables reliable and fast transfer of large files over long distances even in difficult mobility conditions. Compared to multi-hop retrieval, agent-based content retrieval does not generate a lot of network load: in each considered scenario the amount of transferred messages was lower than with multi-hop.

The principles of content centric networks can be exploited to resume interrupted content retrievals without a lot of overhead for finding content sources and building routes.

5.2 Future Work

The developed agent protocol as well as the implemented agent applications with resume capabilities demonstrate the potential of the agent-based content retrieval approach. Nevertheless, there are open questions and potential optimizations to address.

In the implemented protocol agent selection is not considered (random selection of agents). Agent selection is a very wide an complex topic which contains material for another thesis: to apply agent selection algorithms additional information about the environment and its members are needed. Existing forwarder selection mechanisms in DTN networks are based on community graphs or neighborhood maps that are created by collecting hello beacons from neighboring nodes. However, in content-centric networking, we would like to avoid such hello beacons because the hosts where content is stored is not important. Therefore, new agent selection criteria and algorithms need to be developed, e.g., to create a content map based on overheard transmissions.

In principle, the CCN approach provides more flexibility because the requesters can define agent selection criteria in the init request and only agents that fulfill these criteria may answer. This means that agent selection criteria can be different for every application. For example, if content from a certain geographic region needs to be requested, the init requests can include those coordinates such that only agents that travel towards those coordinates may answer.

Another approach would be to combine multi-hop data retrieval with agent-based retrieval into a hybrid solution. In a first step content is searched via multi-hop. If for a given time no content could be found, data retrieval can be delegated to agents. Furthermore, it may be possible to enable agents to delegate jobs further to other agents or enable agents to perform multi-hop content retrieval over a limited number of hops.

Chapter 6

Appendix

6.1 CCNA Requester Application

```
./ccnar [-m] [-e expire time] [-n (push|pull)] [-a max agents] [-i
   delegation interval] ccnx:/a/b
 Start agent request for given ccn URI
 -h Produces this message
 -m Automatically registers a multicast face for ccnx:/ferrying and
    ccnx:/notify prefixes
 -e Set how long (in minutes) the agent request is valid.
        Default 120min.
        Expire time > 0.
 -n Notification type (push or pull).
    Default push.
 -j Pull notification interval (in seconds).
    Default 10s.
 -a Set max. number of agents per job.
         Default: 5.
 -i Set delegation interval of a job (in seconds).
         Default: 30s.
 -p Set number of pipelines for retrieval stream.
         Default: 4.
```

Listing 6.1: Command Line Reference - ccnar

6.2 CCNA Agent Application

```
./ccna - CCNx agent daemon
Processes delegated content retrieval request in dynamic environments
 -h shows this usage text
 -m Automatically registers a multicast face for ccnx:/ferrying and
    ccnx:/notify prefixes
 -i Push notification interval (in seconds).
        Default: 10s, Max: 99s.
```

Listing 6.2: Command Line Reference - ccna

Bibliography

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, ser. CoNEXT '09. New York, NY, USA: ACM, 2009, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/1658939.1658941

[2] "Project CCNx," https://ccnx.org, 2015. [Online]. Available: https://ccnx.org

[3] W. El Maudni El Alami, "An agent-based content-centric networking application for data retrieval," Master's thesis, University of Bern, September 2013.

[4] T. Schmid, "Data exchange in intemittently connected content-centric networks," 2013.

[5] "CCNx version 0.8.2 on github," 2015. [Online]. Available: https://github.com/ProjectCCNx/ccnx/releases/tag/ccnx-0.8.2

[6] "CCNx 1.0 evolution from experiments," 2015. [Online]. Available: http://www.ccnx.org/pubs/hhg/1.7%20CCNx%201.0%20Evolution%20From%20Experiments.pdf

[7] "CCN progress report," October 2014. [Online]. Available: http://www.ccnx.org/pubs/CCNProgressReport20141021.pdf

[8] "NDNx library and ndnd forwarding daemon on github," 2015. [Online]. Available: https://github.com/named-data/ndnx

[9] "NDN packet format specification - interest packet," 2015. [Online]. Available: http://named-data.net/doc/ndn-tlv/interest.html

[10] "CCNx basic name conventions," April 2015. [Online]. Available: https://www.ccnx.org/releases/latest/doc/technical/NameConventions.html

[11] "CCN_REPO(1) manual page," October 2015. [Online]. Available: http://www.ccnx.org/releases/latest/doc/manpages/ccn_repo.1.html

[12] S. Jain, K. Fall, and R. Patra, "Routing in a Delay Tolerant Network," in *ACM SIGCOMM*, Portland, OR, USA, September 2004, pp. 145–158.

[13] M. Liu, Y. Yang, and Z. Qin, "A Survey of Routing Protocols and Simulations in Delay-Tolerant Networks," *Wireless Algorithms, Systems, and Applications*, vol. 6843, pp. 243–253, 2011.

[14] A. Vahdat and D. Becker, "Epidemic routing for partially connected ad hoc networks," Duke University, Tech. Rep. CS-200006, 2000.

[15] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks," in *ACM SIGCOMM workshop on Delay Tolerant Networks (WDTN)*, Philadelphia, PA, USA, August 2005, pp. 252–259.

[16] A. Lindgren, A. Doria, and O. Schelen, "Probabilistic Routing in Intermittently Connected Networks," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 7, no. 3, pp. 19–20, 2003.

[17] Y. Zhu, B. Xu, X. Shi, and Y. Wang, "A Survey of Social-Based Routing in Delay-Tolerant Networks: Positive and Negative Social Effects," *IEEE Communication Surveys & Tutorials*, vol. 15, no. 1, pp. 387–401, 2013.

[18] N. Orlinski and N. Filer, "Neighbor discovery in opportunistic networks," *Ad Hoc Networks*, vol. 25, pp. 383–392, 2015.

[19] T. Hossmann, T. Spyropoulos, and F. Legendre, "Know Thy Neighbor: Towards Optimal Mapping of Contacts to Social Graphs for DTN Routing," in *IEEE INFOCOM*, San Diego, CA, USA, March 2010, pp. 866–874.

[20] O. R. Helgason, E. A. Yavuz, S. T. Kouyoumdjieva, L. Pajevic, and G. Karlsson, "A mobile peer-to-peer system for opportunistic content-centric networking," in *Proceedings of the second ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds (MobiHeld)*, New Delhi, India, August 2010, pp. 21–26.

[21] V. Kawadia, N. Riga, J. Opper, and D. Sampath, "Slinky: An adaptive Protocol for Content Access in Disruption-Tolerant Ad Hoc Networks," in *In Proceedings of MobiHoc Workshop on Tactical Mobile ad Hoc Networking*, 2011.

[22] R. N. dos Santos, B. Ertl, C. Barakat, T. Spyropoulos, and T. Turletti, "CEDO: Content-Centric Dissemination Algorithm for Delay-Tolerant Networks," in *Proceedings of the 16th ACM international conference on Modeling, analysis & simulation of wireless and mobile systems*, Barcelona, Spain, November 2013, pp. 377–386.

[23] E. Monticelli, B. M. Schubert, M. Arumaithurai, X. Fu, and K. K. Ramakrishnan, "An information centric approach for communications in disaster situations," in *20th International workshop on Local & Metropolitan Area Networks (LANMAN)*, Reno, NV, May 2014, pp. 1–6.

[24] C. Anastasiades and T. Braun, "Dynamic Transmission Modes to Support Opportunistic Information-Centric Networks," in *International Conference on Networked Systems (NetSys)*, Cottbus, Germany, March 2015.

[25] C. Anastasiades, T. Schmid, J. Weber, and T. Braun, "Opportunistic Content-Centric Data Transmission During Short Network Contacts," in *IEEE Wireless Communications and Networking Conference (WCNC)*, Istanbul, Turkey, April 2014, pp. 2516–2521.

[26] "CCNx repository protocols," April 2015. [Online]. Available: https://www.ccnx.org/releases/latest/doc/technical/RepoProtocol.html

[27] "Direct code execution," https://www.nsnam.org/overview/projects/direct-code-execution/. [Online]. Available: https://www.nsnam.org/overview/projects/direct-code-execution/

[28] "UBELIX - university of bern linux cluster," http://www.id.unibe.ch/content/services/ubelix/overview/index_ger.html. [Online]. Available: http://www.id.unibe.ch/content/services/ubelix/overview/index_ger.html

[29] M. Lacage and T. R. Henderson, "Yet another network simulator," in *Proceeding from the 2006 Workshop on Ns-2: The IP Network Simulator*, ser. WNS2 '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1190455.1190467

[30] "Log-distance path loss model," https://en.wikipedia.org/wiki/Log-distance_path_loss_model, 2015. [Online]. Available: https://en.wikipedia.org/wiki/Log-distance_path_loss_model

[31] J. Weber, "Dynamic adaptation of transmission modes for opportunistic content-centric networks," Master's thesis, University of Bern, 2015.