RAPTOR CODING IN MOBILE CONTENT CENTRIC NETWORKS

Bachelorarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Alexander Striffeler
2014

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

Contents

List of Figures

List of Tables

Acknowledgment

Many people have participated directly or indirectly in the development of this bachelor thesis. I am deeply thankful to all of them!

I would like to specially thank my supervisor Carlos Anastasiades for his invaluable advice and support throughout the whole working process. I also wish to thank my co-supervisor Nikolaos Thomos for his help on Raptor coding and network coding issues.

I would like to extend my gratitude to Prof. Dr. Torsten Braun for giving me the opportunity to write this thesis in his research group and for providing an excellent work environment. Last but not least, I wish to thank my girlfriend and my family for their boundless support.

Abstract

In Content Centric Networks (CCN), data transmission is based on content names instead of identifiers as in IP networks. Thus, CCN delivers structures for dynamic mobile networks.

Considering such wireless mobile networks, network coding using Raptor codes is capable of exploiting the broadcast nature of the wireless medium when packets have to be retransmitted due to congestion, packet loss or mobility: Instead of transmitting an exact copy again, another linear combination of the encoded packets is sent which may be useful for decoding for any node within transmission range with high probability.

In this thesis, Raptor coding functionality has been added to CCN using simulations in OMNeT++. The experimental evaluation has shown that Raptor codes lead to a significant performance gain in environments with high requester density. If collisions occur, which is likely to happen in such environments, Raptor coding allows the nodes to transmit another linear combination of the content instead of retransmitting exactly the same packet. This new combination again may satisfy Interest messages sent by other requesting nodes. On the other hand, in environments with low requester density, content transmission using Raptor coding takes more time and nodes send more Interest packets than with regular CCN. As CCN is strictly Interest-based and thus requires every data packet to match an Interest packet previously generated, Raptor coding could not exploit its opportunistic data fetching characteristics.

Chapter 1

Introduction

In contrast to the Internet's early days, todays network usage does not longer deal with end-to-end-connections, but clients are rather interested in retrieving data – no matter at which location or on which node this data is stored. Content centric networks (CCN)[3] facilitate this paradigm change by identifying content by its name and not by its storage location. When focusing on wireless networks, mobile nodes may enter or leave specific radio cells. CCNs support mobility of nodes in a way that each intermediate node caches content. Hence, if a node moves to an adjacent cell, a node higher in the network hierarchy, for instance a router or proxy server, can deliver it instead of requesting the content at the remote content source again.

However, in such mobile wireless scenarios, packet collisions are likely to happen. This might lead requesters to obtain the same packet again, which is a redundant transmission for other nodes who might have overheard the previous packet. Raptor coding [4] fills this gap in the way that each additional packet sent on the wireless channel can be advantageous for any neighbouring node. This can be achieved by Raptor encoding a set of input packets at the content source, which results in a larger set of encoded packets. From an arbitrary set of encoded packets, requesters can then reconstruct the original data packets.

In this thesis, we aim to increase both robustness and throughput of the wireless network by adding Raptor coding to content centric networks.

The aim of this bachelor thesis is to integrate network coding using Raptor codes in content centric networks using the simulation framework OMNeT++ [5] and implied the following tasks:

- Study related work on content centric networks as well as Raptor coding.
- Develop a sample implementation of Raptor encoding and decoding procedures.
- Implement Raptor coding functionality in CCN using the OMNeT++ framework.
- Evaluate the CCN application with Raptor coding using the OMNeT++ framework.

Chapter 2

Related Work

In the following chapter, we briefly summarize related work for the different techniques used in this thesis. Please note that the terms content packet and data packet will be used interchangeably.

2.1 Content Centric Networks

In today's Internet, users are mostly interested in information and not *where* the information is stored. Content centric networks (subsequently also called CCN) deliver an approach to cope with these changed requirements. The most distinct way CCN differs from conventional IP protocol is the way packets are transmitted. This procedure is explained below. Furthermore, by caching content in intermediate nodes, CCN can support mobility in wireless networks.

In CCN, each node operates on three data structures which are of interest for our work [3]

PIT Pending Interest Table: This data structure keeps track of Interests sent upstream in order to forward the content back to the requesting (downstream) nodes.

CS Content Store: Each received data packet that is being received upon an Interest transmission is placed in the content store. This enables each node to directly reply to Interests for this data without forwarding it to other nodes.

FIB Forward Interest Base: This table is used to forward incoming Interests to (potential) content sources. While IP tables only allow one outgoing interface per IP prefix to avoid loops, CCN allows several interfaces. CCN FIBs use longest-prefix matching similar to IP tables.

The sequence of packet transmissions in CCN is illustrated in Figure 2.1: To obtain a file, the requesting node sends out an Interest for the data packet with a particular name. According to the entries in the Forward Interest Base (FIB), the requesting node forwards this Interest on the appropriate face. Each node that receives this Interest then looks up its content store whether the requested content has already been contained in its cache. If so, it returns a data packet which will satisfy the Interest sent out by the requesting node. This Interest would then be consumed at the requester which terminates the current packet transmission. For example, as

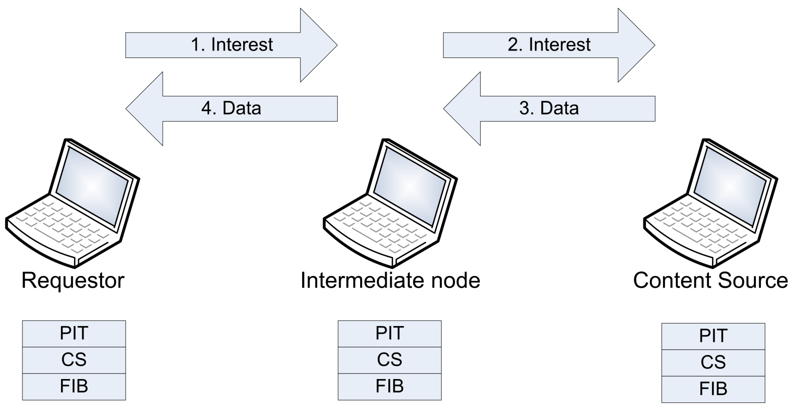


Figure 2.1: Packet transmissions in CCN to obtain a data packet having an intermediate node in addition to content source and requesting node.

seen in Figure 2.1, the second node is an intermediate node and has not stored the requested data. It will then become a requesting node itself, i.e., it will forward the Interest to the neighbouring nodes. Assume the third node (called Content Source in Figure 2.1) that receives the Interest has the data buffered in its content store, then it broadcasts in one hop range the corresponding data packet. The intermediate node now receives this data packet and detects that there is one pending Interest for this incoming packet. Then it buffers the data packet in its own content store for further requests and again performs one-hop broadcast on the face on which the Interest originally arrived.

Since in CCN an Interest with a certain prefix can be used to retrieve data packets, a new concept is needed to identify data. Therefore, the names of CCN data are structured hierarchically: Each name consists of a organizational name, followed by version and segmentation information, for instance

/unibe/lectures/network/slides.ppt/_segment0

We can recognize two things from the example above: first, there can be several versions of a file in parallel and each file is subdivided into several segments. For forwarding data, the nodes perform a *longest prefix match lookup* which means that for instance a data packet named */unibe/lectures/slides.ppt/s0* will satisfy an Interest for the name */unibe/lectures/slides.ppt*.

By just dealing about data content and not about *where* it comes from, there are also security concerns emerging. Rather than trusting in a connection to a server as this is the case in IP, in CCN, all content packets are authenticated with digital signatures from the content provider. If content needs to be private, it is additionally encrypted. Content signing is done on both data and the name. Thus, a receiving node can verify that the data contained in the packet origins from the provider it expects - names can be securely bound to arbitrary names by the publisher.

The fact that CCN can run over IP facilitates the exploitation of its functional advantages.

2.2 Network Coding

In todays networks, a router only forwards packets exactly as it received them. Considering a wireless topology, the broadcast nature of the wireless medium provides the advantage that every connected node can receive the transmitted packets at once. However, when packets have to be retransmitted - which is done by resending an exact copy - there is no knowledge gain in the network. Sending an alternative representation of a packet, however, leads to additional information in the network. Using network coding [6], routers perform operations on all packets that are to be combined – either all packets available in the node or a selected set of input packets. Several input packets are encoded to form a set of encoded packets. Since a wireless network is a broadcast medium, everyone who demands to receive a packet out of the encoded ones will get the same information, i.e. the same set of encoded packets. The task of the receiving nodes now is to decode the packet(s) they need.

Fig. 2.2 shows a simple network topology where all links have unit capacity. There are direct links between s_1 and r_2 and between s_2 and r_1. Sender s_1 wants to send packet X to receiver r_1, while s_2 wants to send packet Y to r_2. However, all packets sent from s_1 to r_1 and s_2, r_2 respectively need to traverse the link in the middle, which then becomes the bottleneck of the whole network. Imagine, as a simple example, a network in which all links have the same capacity with four nodes: sender s_1, sender s_2, receiver r_1 and receiver r_2. Without network coding, the bottleneck needs to be time-shared and thus for sending the two packets, two time-slots are used. However, by using network coding we can combine (i.e. bitwise XOR) both packets and send the encoded packet to both r_1 and r_2 at the same time. Since there are direct connections between s_1 and r_2 and between s_2 and r_1, both receiving nodes can receive the respectively unrequested packet in the same timeslot. Having this information, the receiving nodes can then decode the encoded packet and hence extract the information they need. This is done by again XORing the unrequested content (which is packet X for r_2 and packet Y for r_1) with the encoded information: having two packets x and y which are shown by the blue and the red lines, the purple lines represent the encoded information $x \oplus y$. Requester r_2 can now calculate $x \oplus (x \oplus y) = y$ which equals the packet r_2 requested. By sending the same amount of packets in less transmissions (three transmissions with network coding versus four without), of course, network coding also minimizes the transmission energy used per packet and the contention in the wireless medium.

Another variation of network coding is introduced in [7]. If we look at media streams, packets are often prioritized differently and can be organized in different importance classes. Network coding can conform to these requirements and handle the packets according to their priority by using Prioritized Encoding Transmission [7][8]. Of course, the complexity of computing the coding opportunities increases drastically with such increasing dependencies between packets.

In wireless networks, as a packet traverses multiple hops, its data becomes known to all intermediate nodes which may deliver massive redundancy. While wired networks provide reliable links, wireless networks are affected by unreliable links, limited coverage and furthermore, their characteristics may vary over time. In addition, wired networks come with a known and rather static topology and provide collision detection which is not the case for wireless networks.

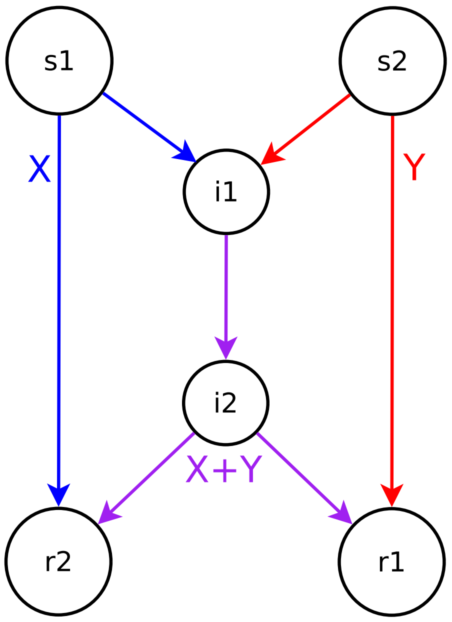


Figure 2.2: Butterfly-topology: A basic example for network coding. Node $s1$ sends packet X to $r1$ and $s2$ wants to send packet Y to $r2$, both over the two relay nodes in the middle. The blue and red lines represent packet A and B, respectively. The purple arrows represent the encoded packet $A \oplus B$.

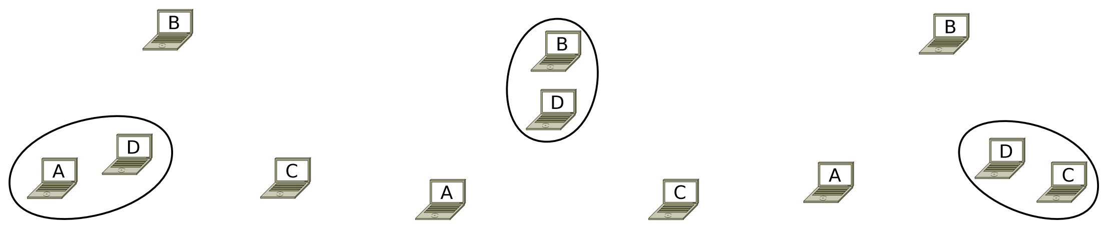
To compensate those characteristics of wireless networks, [1] proposes to exploit the broadcast nature of wireless networks.

Network coding also increases throughput in unreliable networks. Since all encoded packets are linear combinations of the data packets to be transmitted, the sending node does not need to know which packets are still missing at the receiving node - it can just keep sending encoded packets until the receiver is able to decode the information and then send back an acknowledge-packet for the whole file [1].

Focusing on mobility (which is the intention of this thesis), network coding is advantageous as well since the topology may vary quickly and routing updates are costly [1]. However, using network coding, the mobile node can just broadcast random linear combinations of all packets the node in range wants to receive. According to Figure 2.3, as long as any of the encoded packets contains information that is new to any of D's neighbours, the packet is innovative and valuable. Hence, the nodes do not need to track any longer which node has received what packets and can therefore much quicker adapt to changing topologies.

2.3 Raptor Codes

Network coding can improve the throughput of a network connection if all channels are noiseless. However, having congested channels, network coding should be used with an error correction code [9], such as Raptor codes.



(a) Timeslot 0 (b) Timeslot 1 (c) Timeslot 2

Figure 2.3: Node D introduces mobility to the network, following [1, page 6]. Node D moves and is connected to different nodes A, B and C over time.

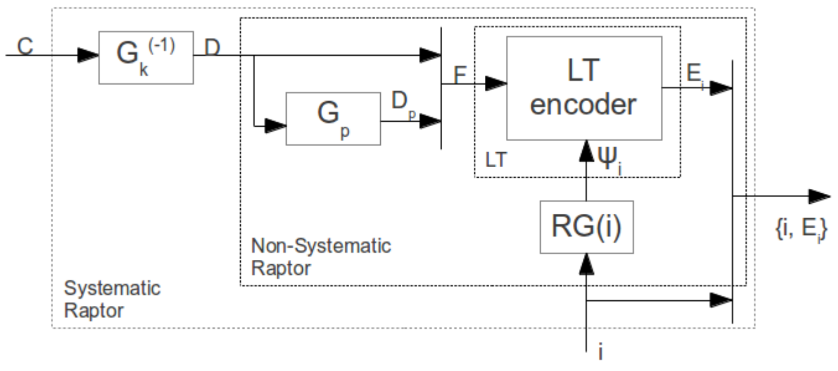


Figure 2.4: Conceptual block diagram of systematic Raptor encoding [2].

For efficient and robust use of network coding, the data needs to be encoded prior to transmission and decoded again when arriving at the destination node. In general, the encoding node therefore generates messages X from the S packets to be encoded $P_i, i \in 1, \ldots, S$ as follows:

$$X_k = \sum_{i=1}^{S} g_k^i \times P_i \tag{2.1}$$

where g_k^i is a coding coefficient chosen from a finite field. As many other codes, Raptor codes [10] are capable of providing the encoded symbols. Raptor codes significantly improve LT codes. LT-codes (Luby Transformation) are a class of Fountain codes which cannot encode with constant computation cost if the number of collected output symbols is close to the number of input symbols [10]. Raptor codes instead allow encoding and decoding with constant cost. As illustrated in Figure 2.4, Raptor codes are a concatenation of two codes: A pre-code which is a fixed rate erasure code or a concatenation of such codes itself and LT-codes.

Fountain codes produce a potentially endless stream of output symbols $z_1, z_2, \ldots$ by encoding an input set of k input symbols $(x_1, \ldots, x_k)$. Every output symbol is generated independently and randomly using the Formula 2.1 on a finite field F_2^k. One of the nice properties of Fountain codes is that as each symbol is generated randomly, a decoding node can receive output symbols from different encoding engines, but from the same set of input symbols and can decode them. To make practical use of Fountain codes, they depend on a fast encoder and decoder. Furthermore, the decoder needs to be able to extract the input symbols from any adequate sized subset of the encoded symbols. Such Fountain codes are called *universal*. LT-codes were the first known class of universal Fountain codes.

Raptor Codes are the first known Fountain codes with both linear encoding and decoding times. For a given amount of input symbols, Raptor codes can produce infinite streams of symbols such that any subset of symbols of length $k(1+\varepsilon)$, where ε is the coding overhead which is close to 5% for Raptor codes for medium to large codeblocks. This overhead is sufficient to recover the input symbols with high probability. The advanced performance of Raptor codes is due to the pre-coding of the input symbols before applying an appropriate LT-code. A LT-code is a Raptor code without any pre-coding at all. LT-codes have optimal space consumption 1 and with an appropriate output distribution $\Omega(x)$, the overhead of an LT-code is $O(log^2(k)/\sqrt{k})$ and its cost is proportional to $log(k)$ [10]. Since LT-codes have no pre-coding, they compensate their lack with a very sophisticated output distribution $\Omega(x)$.

The component codes of Raptor codes introduced above are not systematic which means that the input symbols do not always have to be reproduced by the encoder. However, Raptor codes can be built to be systematic as well (see Fig. 2.4 and [10, page 2563 ff.]).

Chapter 3

Design and Implementation of Raptor Coding in CCN

In this chapter, we will describe our approach to integrate Raptor coding into content centric networks. Section 3.1 introduces the encoding strategy, naming tasks such as content names and segmentation are described in section 3.2 and section 3.3 describes the implementation of Raptor coding in CCN.

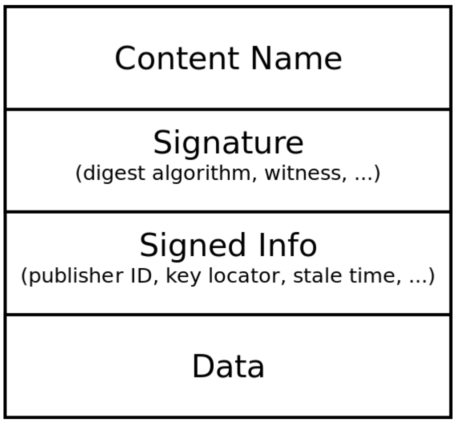
3.1 Raptor Coding in Content Centric Networks

3.1.1 Raptor Coding Strategy

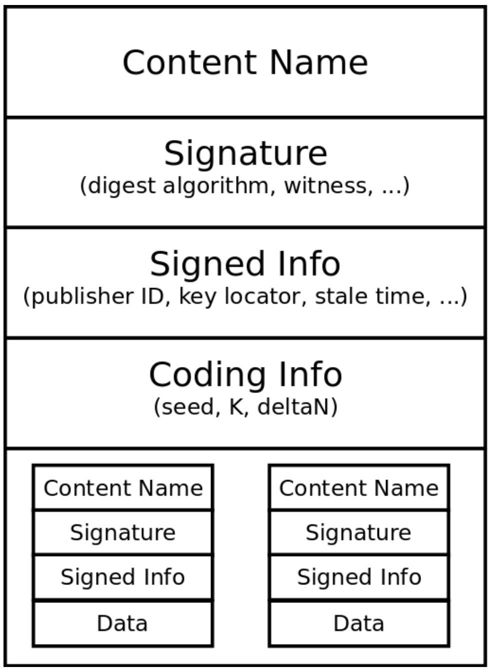
Raptor coding implies that intermediate nodes may not only forward encoded packets, but also encode them. The signature and trust mechanisms embodied in CCN, however, emanate that the packet is signed by the content source. Checking the signature, the requester can avoid receiving vulnerable data from a malicious node. Adapting security mechanisms is out of the scope of this thesis, hence we assume that only content sources encode packets, i.e., no re-encoding is performed at intermediate nodes. Hence, we do not perform network coding but Raptor coding.

There are two strategies to perform Raptor coding in a network: The first approach is to encode different parts of a single file or stream that are requested by the same node. The second approach is to encode packets from different sources that are sent over the same link.

In CCN, it is not possible to encode packets based on their destination because the destination is not included in the data packet and forwarding on specific faces is only based on prior expression of matching Interests. Thus, Raptor coding would require nodes to re-encode content on-the-fly based on forwarding on specific faces. This would increase forwarding delays and may result in different content files, which are forwarded over the same face, being encoded together. However, if content with different names was encoded together, forwarding based on the name, which is a fundamental building block in CCN, would not be feasible anymore. To reduce this conceptual gap between Raptor coding and CCN in our implementation, we only apply Raptor coding to different segments of the same file or stream. This preserves the name prefix and thus does not violate the constraints between name and content.


Content Name
Signature
(digest algorithm, witness, ...)
Signed Info
(publisher ID, key locator, stale time, ...)
Data

(a) Common CCN data packet



(b) Packet with network encoded packets encapsulated in data field

Figure 3.1: Comparison between common CCN data packets and data packets with encapsulated Raptor encoded packets.

3.1.2 Data Structures

There are two possibilities to transmit encoded information. First, the original CCN Data packet is encapsulated and included in the data part of another CCN Data packet as shown in Figure 3.1. The header of the encoded Data packet needs to be extended by additional Raptor coding information, such as seed, number of input symbols K and $deltaN$. These parameters are explained later in more detail in section 3.3.1. Second, instead of encapsulating encoded data packets, only the data part is encoded, i.e., without original header information. This increases the relative amount of transmitted data in the payload of the encoded packet because CCN header information of the original data packet is not included in the payload. However, a receiver could not verify the signature of the original publisher, because it is not available anymore. Therefore, in this work, the encoded packets are encapsulated in another Data packet with an extended Raptor coding header (first approach).

Another issue to consider is the number of packets that are encoded together, i.e. the value of K. Encoding only very few data packets allows a receiver to wait for a shorter time until receiving the necessary amount of packets to decode. However, this way, the overhead due to the coding information included in each encoded packet is rather big in relation to the payload size. On the other hand, with the increasing amount of segments encoded together, the overhead decreases.

For different reasons, we only encode at content sources and decode at requesting nodes: First, since Raptor coding only takes place at the application layer, intermediate nodes would need to retrieve the data as well so that content could be re-encoded at the application layer. We do not consider coding operations at the CCN layer for reasons mentioned in chapter 3.1.3. Second, re-encoded content would also need to be signed by the encoder and trust in the encoders' key is required due to trust constraints entailed in CCN.

As we limit the encoding to content sources, we do not perform network coding but apply Raptor coding in this thesis.

3.1.2.1 Signatures

In CCN, a node can get data not only from its origin but from any node that has cached this data. Every requester can verify the authenticity of the content by verifying the publisher's signature. If we encode these (signed) packets at an intermediate node, the encoded packets are signed by another key than from the original publisher. There are two possibilities handling this:

- **Do not sign the encoded packet at all**. The original data packets are still signed but the encoded packets are not. This does not correspond to the CCN policy since it requires that each and every packet is signed. Furthermore, this leads to the problem that malicious nodes can spread junk packets. Since the packets are not signed, these corrupted packets cannot be recognized and could mess up the decoding if they are considered as legitimate packets.

- **Sign the packet with the encoding node's key.** In this scenario, every time an intermediate node encodes packets, it signs these with its private key. Now, the receiver can immediately see which node has encoded the packets and - if necessary - distinguish volnerable sources from trusted ones. However, this requires trust measurements such as reputation ratings. Additionally, it increases the processing overhead and forwarding delay since packets need to be decoded first before they can be re-encoded.

3.1.3 Packet Processing

There are different possibilities where to process encoded packets. Either we place the whole encoding and decoding logic in the CCN layer itself or delegate the Raptor coding capabilities to the Raptor coding application. Figure 3.2 shows how application layer and CCN layer are connected. In CCN, each node provides an application layer and a CCN layer. The application is attached to the CCN layer but only the lower layer directly connects to the network.

In the first approach, the CCN layer is responsible for the whole processing of encoded packets including the collection of sufficient encoded packets that are linearly independent, decoding the packets if a sufficient large number of packets are received, and encode packets. This approach has the advantage that encoding can be done in the CCN layer directly without having the need to send packets up to the application layer. However, the drawbacks are:

- An application does not know how many (and which) packets have already been received and whether new packets need to be requested.

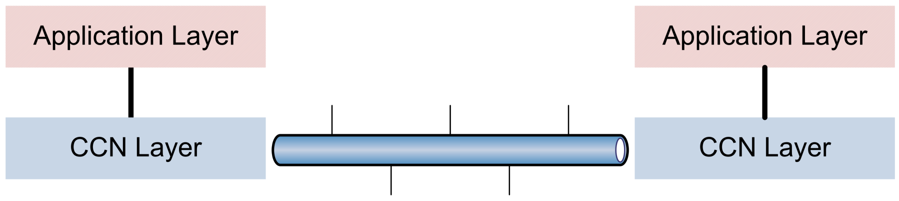


Figure 3.2: Layer architecture in CCN

- Decoded CCN data packets would be included in the cache and, thus, would be deleted after a short time depending on the cache replacement strategies. To persistently store the packets, they need to be stored in a repository application.

In the second approach, Raptor coding is performed at the application layer. Thus, a requester stores all encoded packets temporarily in the content store (CCN layer) and persistently at the CCN application layer. All decoding operations are performed by the Raptor coding application at the application layer. Intermediate nodes, which are no requester of content, only store and forward the packets temporarily at the CCN layer but do not forward and store it at the application layer.

In this approach, the application monitors whether enough packets have already been received for decoding or whether more Interest packets need to be transmitted. However, the main downside is that packets can only be Raptor encoded at the content sources and requesting nodes since the packets are only forwarded and retransmitted at intermediate nodes. The implementation at the application level does not have an impact on regular transmissions without Raptor coding which implies that other applications building on the same CCN layer remain unaffected.

In this work, we decided to include Raptor coding in the application layer.

3.2 Naming

In CCN, every data packet a node wants to receive must be requested by sending an Interest packet. Thus, it is not possible to get packets that were not requested explicitly. However, Raptor coding tries to collect as many data as possible in order to gain further information from additional encoded packets since every encoded packet represents a linear combination of the source data packets. Hence, data retrieval in CCN is quite different from what would be ideal for Raptor coding. To cope with these differences and to exploit the advantages of both worlds, we need to adapt the processing of Interests and data packets in CCN. However, it still needs to be possible to exchange conventional CCN data packets along with encoded content.

As explained in chapter 2.1, Interest prefixes need to match content names to trigger data transmission. This matching is done by performing a longest prefix match of the Interest name on the data name. Whenever a content message is received in reply to an Interest message, it is forwarded and removed from the pending Interest table following the bread crump routing approach.

At the same time, content centric communication should not require messages to be encoded and transmission of regular not-encoded messages should still be possible. Therefore, we tag encoded traffic with an identifier between version and segmentation information. While an example content name of a non-encoded message could be

/unibe/lectures/slides.ppt/_segment

the same name of encoded content now looks like

*/unibe/lectures/slides.ppt/**_rc**/_encodedID*

where the prefix *_rc* stands for Raptor encoded packets. Hence, a requester can request encoded or unencoded messages by including or avoiding the Raptor coding identifier. The suffix `encodedID` specifies the encoded vector. These encoded vectors are being generated at the content source based on the original data packets and are linearly independant. This prefix-based naming only works for packets that belong to the same file.

To decode content and restore the original content, a sufficient number of encoded packets need to be received. To avoid the reception of duplicates, the requester can specify the encodedID it wants to receive in the requested name. Avoiding the encodedID in the content name would lead requesters to add all packets they have already received to exclude filters in the Interest message. Since content may include many encoded packets, the exclude filter and, thus, the Interest message itself may grow significantly in size. Therefore, the requester specifies in the Interest message an encodedID that it has not yet received. By including the encodingID in the Interest prefix, a requester asks for specific, i.e., not yet received packets, from the encoded set, without sending large Interest packets containing long exclude filters. The first `encodedID` equals the seed of the encoding set and it is increased by one for every subsequent encoded packet. We will explain the retrieval of encoded content with an example in Figure 3.3.

The Raptor coding parameters are summarized in Table 3.1. When looking for encoded packets of the file */unibe/lectures/overview.pdf*, the requesting node, i.e., the Raptor coding application running on the node, will generate an Interest packet for content */unibe/lectures/overview.pdf/rc*. If encoded content is already available, the Interest will request the smallest available encodedID. If no encoded but unencoded content is available on a nearby node, the request will trigger the content source to encode the content now. When encoding the content, a content source generates $K + deltaN$ encoded packets from K input packets. The requester will then receive the first encoded packet having the lowest identifier which equals the seed, e.g. */unibe/lectures/overview.pdf/rc/25034*. Knowing the seed, the number of encoded packets K and $deltaN$, the requester knows the ID range of the given set of encoded packets and can request more packets directly with the encodedID, which needs to be in the range of $[seed, \ldots, seed + K + deltaN]$. Every `CCNEncodedPkt` contains an additional header that includes information required for Raptor coding. Please refer to chapter 2.2 for further information concerning the encoding process.

3.3 Implementation in OMNeT++ Simulator

This chapter will give an overview of the changes that were made in order to encode data packets in content centric networks. One important aspect in the whole implementation was to minimize the changes to existing structures to guarantee that Raptor coding would not violate conventional data transmissions in CCN.

K	The number of `CCNDataPkts` combined in this encoding set, i.e. the amount of input packets in the encoding process.
deltaN	The number of encoded packets computed additionally to the number of Input packets. The number of coded packets available for a specific coding set equals $K + deltaN$. This parameter is required to be known by both encoding and decoding nodes.
seed	The seed is a random number computed during the encoding process and identifies the set of encoded packets. In particular, the sets lowest encodedID equals the seed. The range of encodedIDs for a particular set is $[seed, \ldots, seed + K + deltaN]$.
ε	The number of packets a requesting node needs to gather to be able to successfully decode, in addition to the number of K.

Table 3.1: Overview of Raptor coding parameters.

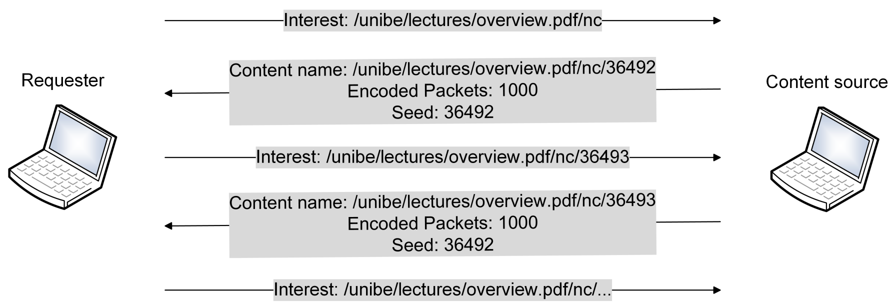

Interest: /unibe/lectures/overview.pdf/nc
Requester
Content name: /unibe/lectures/overview.pdf/nc/36492
Encoded Packets: 1000
Seed: 36492
Content source
Interest: /unibe/lectures/overview.pdf/nc/36493
Content name: /unibe/lectures/overview.pdf/nc/36493
Encoded Packets: 1000
Seed: 36492
Interest: /unibe/lectures/overview.pdf/nc/...

Figure 3.3: Schematic sequence of retrieving encoded packets. Snapshot at the beginning of the transmissions.

3.3.1 CCN Application

The CCN application contains all Raptor coding functions. At a content source, the data packets are encoded upon the reception of the first Interest packet in Raptor encoded content. A requester needs to express Interests in encoded packets and to decode the content if sufficient packets are received. The parameters listed in Table 3.2 need to be known to the application:

encodedPackets	The number of source packets. In the code as well as in subsequent chapters, this value is called K. In our simulation, we always chose K to be equal to the number of segments of the original data.
deltaN	The number of linear combinations that are generated additionally to the number of original data packets K. The number of linear combinations created in total equals $K + deltaN$.
epsilon	The number of additional linear combinations a requester collects until it is able to decode. This value is not estimated in advance but can only be observed by the requester.
rc_prefix	The prefix used to identify encoded content and distinguish it from conventional, non-encoded content without rc_prefix.
provideEncodedContent	This option determines whether received encoded packets are registered by their prefix at the CCND Layer so that Interests from other nodes can be received and forwarded to the CCN Application. In contrast to the CCND content store, encoded packets are stored persistently at the CCN Application layer.

Table 3.2: Parameters of Raptor coding application

The functionality in the Raptor Coding application includes handling of received Interests and encoded Data packets from the lower CCN layer, encoding and decoding data as well as sending Interests in encoded data to the lower layer for transmission.

Encoding and decoding tasks are delegated to the RaptorCoding package and are not performed in the application directly. Processing of received Interest and data packets is visualized in Figure 3.4. If an Interest packet for encoded content reaches the Raptor coding application, it first checks whether the content has already been encoded earlier. If so, the encoded packet is transmitted immediately. Otherwise, the application checks whether the requested content is available in its raw unencoded form, i.e. original data packets. Having the requested content available as original data packets, the application starts the encoding to create a set of $K + deltaN$ encoded packets, which are stored persistently. Responding to the Interest which caused the encoding process, the first packet out of the set of encoded packets is sent to the lower layer. In particular, an Interest packet in the general Raptor coding prefix *.../rc* is answered by the first encoded packet as soon as the encoding process is finished. From then on, requests for the same data will be replied directly using the persistent storage. For the CCN layer to know which data is stored in the application layer's persistent storage, the appropriate name prefix is registered towards the CCN layer whenever data is added to the persistent storage. While the encoded packet's prefix can only be registered after decoding, the prefix for encoded content is registered as soon as the first encoded packets are received. Thus, every requester can respond to Interests from other requesters if the requested encodedID is contained in its persistent storage.

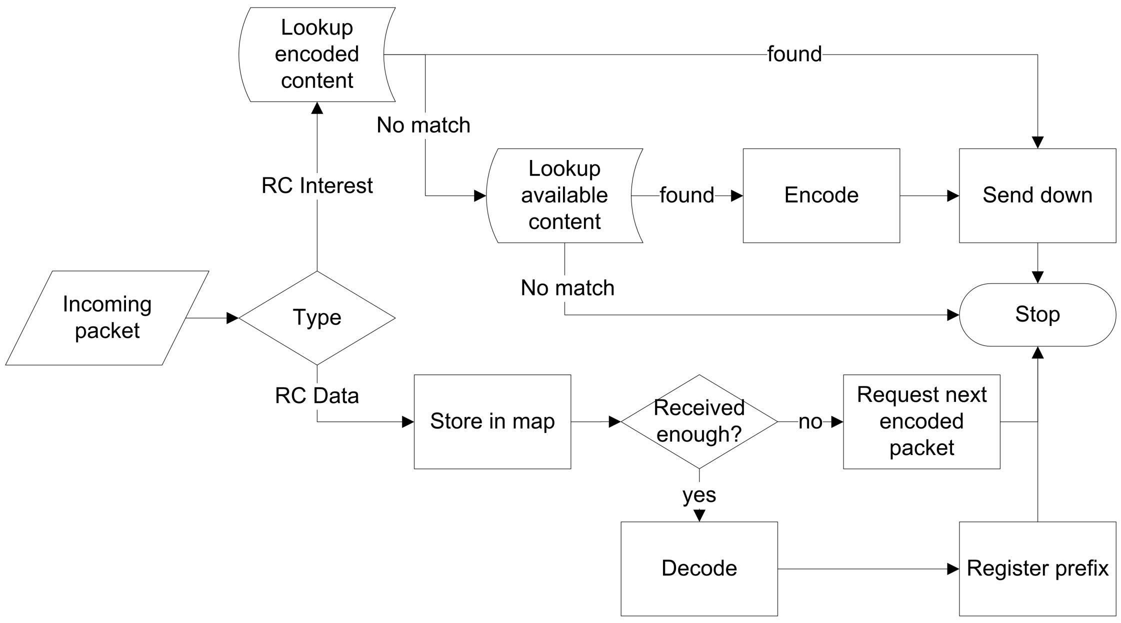


Figure 3.4: Flow chart describing packet handling in the CCN application.

One of the most significant differences between the Raptor coding application and conventional CCN applications concerns the strategy with which the different segments are requested: In conventional CCN applications, all segments are requested in sequential order. In case of a timeout or missing segment, the corresponding segment is re-expressed and the pipeline size is set to 1. However, using Raptor coding, it is not required to obtain all packets in sequential order because they contain linear combinations of multiple segments anyway. Therefore, it is sufficient to obtain an arbitrary set of $K + \varepsilon$ encoded packets to decode and retrieve the original content. If an encoded packet is not retrieved, instead of a retransmission, the next encoded packet is requested. The ability to skip packets depends on the value of $deltaN$, which determines the number of additionally encoded packets from which ε packets need to be retrieved. As some packets can be dependent, this might consume some of the $deltaN$-potential.

Please note that our application never re-expresses an Interest in the same encodedID right away but proceeds to the next encoded packet it has not received yet. The maximum pipeline size is never reduced to one but always kept at the maximum. As the decoding process only requires a set of $K + \varepsilon$ packets, it is indifferent if received encoded packets are in sequential order or not.

Changing to the requesters side, an incoming data packet will be stored in a hash map using its content name, i.e., the content name without Raptor coding prefix and encodedID, as the key. The application then checks whether there are already enough packets available to decode. If

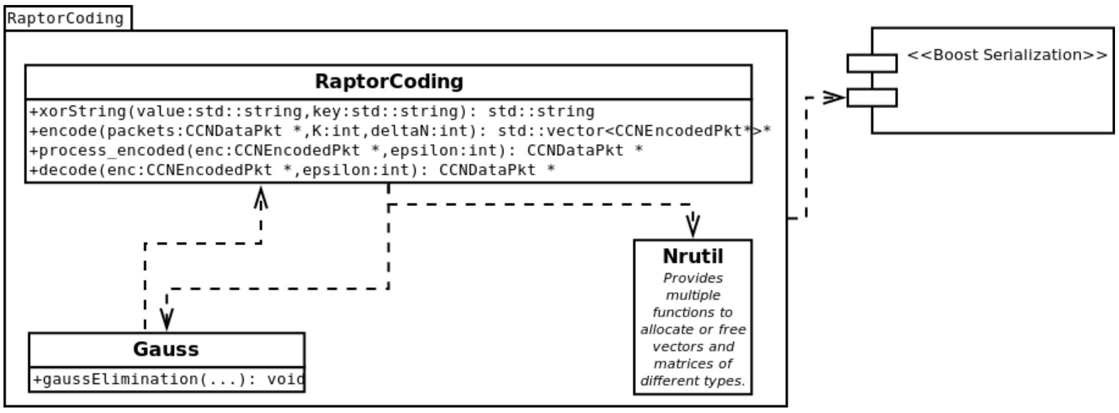

RaptorCoding
RaptorCoding
+xorString(value:std::string,key:std::string): std::string
+encode(packets:CCNDataPkt *,K:int,deltaN:int): std::vector<CCNEncodedPkt*>*
+process_encoded(enc:CCNEncodedPkt *,epsilon:int): CCNDataPkt *
+decode(enc:CCNEncodedPkt *,epsilon:int): CCNDataPkt *
Nrutil
Provides multiple functions to allocate or free vectors and matrices of different types.
Gauss
+gaussElimination(...): void
<<Boost Serialization>>

Figure 3.5: Simplified UML view on functions of Raptor coding package.

yes, the packets are decoded and the resulting original data packets are registered by their prefix to the CCN layer which means that Interests in these packets will be forwarded to the application from now on. Otherwise, if the number of received encoded packets is not yet sufficient for decoding, an Interest packet for the next encoded packet is being expressed.

3.3.2 Raptor Coding Class

This package provides the whole coding functionality required for encoding, requesting and decoding the packets. Figure 3.5 shows a simplified UML diagram of the Raptor coding package hiding attributes and parameters mostly. Please refer to the following subsections for explanations of the methods mentioned. From the application's point of view, the package masks all the complexity of the Raptor coding processes by just providing methods for the encoding of original data packets and for the handling of incoming encoded packets.

3.3.2.1 Process_encoded

Every encoded packet is first passed to this function. Figure 3.6 gives an overview of the processes that follow.

When an encoded data packet is received at the application layer, it is forwarded to the Raptor coding class which then checks whether there were already other packets with the same prefix received earlier. To achieve this, we maintain a hash map which matches a given prefix with a list of already received packets. Hence, each incoming encoded packet is appended to the list. Next, depending on the length of this list and the parameter ε, the class decides whether the node has enough packets to decode or not. In case of enough packets, the decoded packets are forwarded to the Raptor coding application. If decoding is not possible yet, the function returns `NULL` which tells the application to collect further encoded packets.

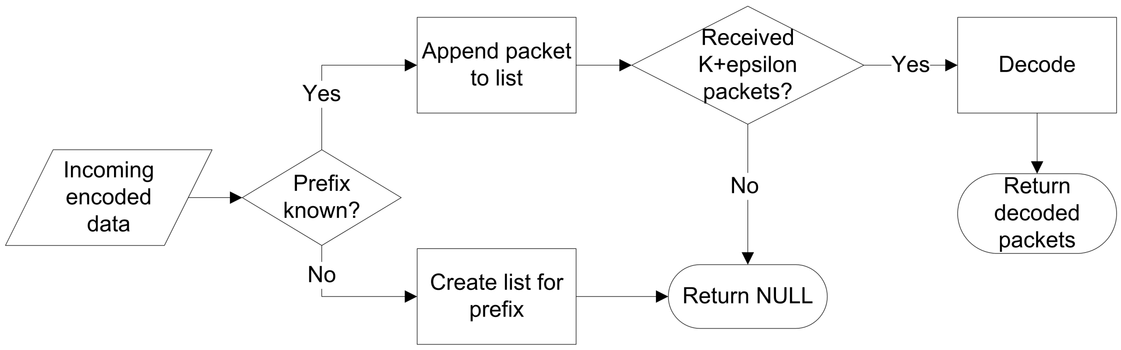


Figure 3.6: Processing of incoming encoded packets

3.3.2.2 Encode

The encoding function takes a set of `CCNDataPkts`, serializes them using the Boost framework [12] and performs the encoding actions introduced in chapter 2.2. Serialization and XOR functions are explained in subsections 3.3.4 and 3.3.5 respectively. Upon calculating the encoded content objects, it creates a set of $N = K + deltaN$ `CCNEncodedPkts`, sets all the required header fields such as content name, seed, encodedID, etc. and then passes these packets to the Raptor coding application.

3.3.2.3 Decode

Decode is the inverse function of encoding: It takes a set of $K + \varepsilon$ `CCNEncodedPkts`, and decodes the encoded packets to retrieve the serialized original packets, de-serializes them and returns the set of K `CCNDataPkts`. After performing matrix operations on the decoded packets, the deserialization process involves solving a system of equations for which it uses Gaussian elimination, mentioned in section 3.3.3.

If there was an error during the decoding process, the function returns `NULL`.

3.3.3 Helper Functions

The Raptor coding application accesses several functions found in other classes in the Raptor coding package such as:

GPPfunctions	This class provides functionalities used during the encoding and decoding processes, mainly used for the calculation of coding parameters. These functions follow the specification of 3GPP [2].
Gauss	The Gauss class is restricted to Gaussian elimination, which, however, is a key component of the decoding process.
nrutil	Nrutil is a helper class used for vector and matrix allocation and also delivers some matrix conversions [11].

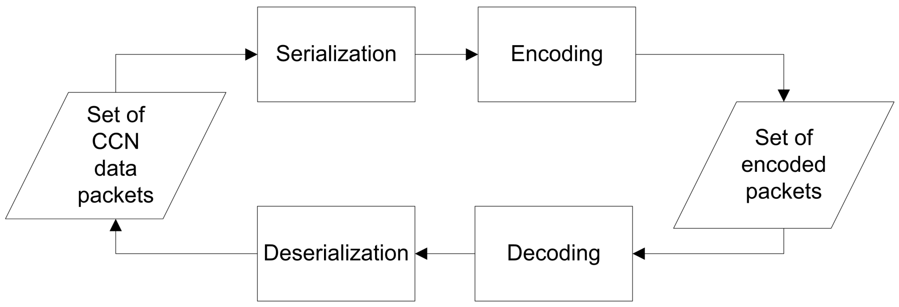


Figure 3.7: Placement of serialization and deserialization in the context of encoding and decoding processes.

3.3.4 Serialization

Performing Raptor coding on CCN data packets requires the input objects to be XOR-ed. However, data packets do not just consist of primitive data types but also include Strings, Signature and SignedInfo, which renders XOR-operators in standard C++ useless. In order to be able to perform the XOR operation using the function `xorString` explained in section 3.3.5, we have chosen to use the Boost libraries[12] to serialize data. Serialization will provide a character sequence containing all packet data. Figure 3.7 shows at which points in encoding and decoding the serialization and deserialization take place.

The source code for each packet type in OMNeT++ is generated automatically from specific declaration files as shown for CCN Encoded Packet and CCN Data Packet in Listing 3.1. The program `opp_msgc` provided by OMNeT++ takes such a `*.msg`-file containing the message declaration and then generates the `*.cc` and `*.h` files which form the C++ class. The automatic generation of source code makes it impossible to implement the serialization functionality in the packet code directly. Hence, we had to use a non-intrusive way to implement serialization. Listing 3.2 shows the serialization of a `CCNDataPkt` which then invokes the code shown in Listing 3.3. The serialized character stream can be further handled using common data types, e.g. `std::string`.

As this packet contains further custom-made objects, i.e., Content, SignedInfo and Signature, each of those has to provide serialization functionality itself.

After the decoding process, the data packets are available in serialized form. To obtain the original `CCNDataPkts`, deserialization has to be performed. For all classes involved in serialization, a corresponding deserialization function has to be implemented.

Listing 3.1: Declaration files for CCN Encoded Packet and CCN Data Packet

```
packet CCNEncodedPkt extends CCNDataPkt{
    int encodedPackets;
    int packetID;
    int seed;
}

packet CCNDataPkt {
    string contentName;
    Signature sig;
    SignedInfo siginfo;
    Content data;
}
```

Listing 3.2: Serialization of CCN Data Packets prior to the encoding process.

```
    for (int i = 0; i < K; i++) {
        inputPackets->at(i) = new std::ostringstream("");

        CCNDataPkt *current = packets[i].dup();

        boost::archive::text_oarchive oa(*inputPackets->at(i));
        oa << *current;

        delete current;
    }
```

Listing 3.3: Boost serialization code for CCN Data Packet

```
/**
 * Serialization using boost. Boost v1.5.0 needs to be installed.
 */
#include <boost/serialization/split_free.hpp>

namespace boost {
namespace serialization {

template<class Archive>
void save(Archive & ar, const CCNDataPkt & p,
        unsigned int version) {

    ar & p.getContentName();
    ar & p.getSig();
    ar & p.getSiginfo();
    ar & p.getData();

    /* Fields from cMessage */
    short int kind = p.getKind();
    ar & kind;

    int64 byteLen = p.getByteLength();
    ar & byteLen;
}

} // namespace serialization
} // namespace boost
BOOST_SERIALIZATION_SPLIT_FREE(CCNDataPkt)
;
```

3.3.5 XOR

This function acts as an internal helper to perform the XOR operation on character sequences and is an important part of the encoding and decoding function. Since serialized data is likely to contain `NULL` characters, it is important not to use strings in C-style which are `NULL`-terminated. If any of the characters in the string matches the string delimiter, all subsequent characters are lost and deserialization becomes infeasible. Hence, if C-strings were used, the string was cut at the first appearance of a `NULL` character, which may be created by the XOR operation on packet data.

Another difficulty is that the XOR function should also perform reliably for inputs of different lengths and empty strings such that the original data can always be retrieved. Therefore, a custom XOR-operation has been implemented that uses std::strings.

Chapter 4

Evaluation

In this chapter, we measure the performance of content centric networking using our Raptor coding approach described in chapter 3 and compare it to the performance of conventional CCN communication. In particular, we examine the following:

- How do the parameters $deltaN$ and ε influence the Raptor coding performance?
- How does Raptor coding performance change when the playground size increases and thus node density decreases?
- What is the impact of different ratios of requesting nodes?
- How does Raptor coding traffic perform compared to regular CCN traffic?

4.1 Simulation Environment

For evaluating the different scenarios named above, we have used OMNeT++ [5] as the simulation framework. The scenarios we have examined all embrace a total of 100 wireless nodes whereas one of them acts as content source (and thus does not request any content). There are scenarios for different numbers of requesting nodes. Each requesting node will provide received encoded packets to other requesters. Each host runs one CCN application, i.e., our Raptor coding application, and uses one 802.11g radio interface. Each scenario was run 100 times. The content to be encoded consists of 1000 segments of 4096 bytes which result in $K = 1000$ input symbols. Table 4.1 gives an overview of the static evaluation parameters. For MAC layer and wireless transmission parameters, we have used standard values. The value of 100s for the parameter *Start Interest Interval* means that all 99 requesting nodes will start requesting content randomly within the first 100 seconds of the simulation. RTS threshold was set to 10'000 bytes which means that RTS/CTS will not be used.

All nodes move according to the Gauss-Markov mobility model [13]. The parameters are listed in Table 4.2

Channel and 802.11 radio parameters	
Carrier frequency	2.4 GHz
Transmitter power	2.0 mW
Signal attenuation threshold	-110 dBm
Path loss coefficient	2
Number of radio channels	1
Thermal noise	-110 dBm
Sensitivity	-85 dBm
SNR Ratio	4 dB

MAC Layer parameters	
WLAN multicast bitrate	2 Mbps
RTS-Threshold	10’000 bytes

CCN App parameters	
Content lifetime	60 s
Segment numbers	1000
Pipelining size	16
Start Interest interval	100 s
Segment size	4096 bytes

Table 4.1: Common parameters for all simulation scenarios.

Mobility Parameters (Gauss-Markov model)	
Alpha	0.9
Speed	10 mps
Angle	0 deg
Variance	40
Margin	30 m

Table 4.2: Parameters for Gauss-Markov mobility model.

Variable parameters	
Playground side length	500, 1000, 5000
deltaN	10, 100, 500, 1000
Ratio of Requesters	10, 50, 99

Table 4.3: Variable parameters for evaluation scenarios.

4.2 Simulation Scenario

Having all the evaluation parameters summarized above, we vary the following three parameters in our evaluations: Playground size, $deltaN$ and requester ratio. Table 4.3 shows the different values we have used in the evaluated scenarios.

The playground size determines the area in which the individual nodes can move. Since transmission power and the number of nodes are constant for all scenarios, the playground size indirectly determines the number of nodes in transmission range, i.e., the node density. The shape of the playground is always square, i.e., the measures are *playground length* x *playground length*. The results for different playground sizes are shown in subsection 4.3.3.

The parameter $deltaN$ describes the number of encoded packets generated in addition to the number of input packets (K). Thus, the total number of encoded packets available equals $N = K + deltaN$. The higher N, the more packets the requester can choose from in order to request the required number of encoded packets. This means that the unavailability of an individual packet affects the requester relatively speaking less since the set of available packets is generally larger and, thus, the requester can just skip to the next packet if the current packet is not available. The results for different values of $deltaN$ are shown in subsection 4.3.2.

Third, we varied the proportion of requesting nodes relative to the number of total nodes to take values of 10%, 50% and 100%. Since the total number of nodes was fixed to 100 and one node always acted as a content source, the scenario of 100% requesting nodes resulted in an absolute number of 99 requesters. The impact of different requester ratios is shown in subsection 4.3.4.

The variable ε describes how many packets the receiving node collects in addition to the amount of input packets before decoding can be performed. This means that any requester will collect $K + \varepsilon$ encoded packets before decoding. As mentioned in chapter 2.2, ε influences the probability for the receiver to being able to successfully decode. The more packets are available for decoding, the lower is the probability of failure. We have evaluated the values $\varepsilon = \{3, 10, 20\}$ and ran the simulations with these parameters 10'000 times without any decoding failures, which shows that the parameters are robust. These results are in accordance with existing studies [4] that confirm that the selected ε results in a very low decoding failure probability.

In all the above scenarios, we focus on one-hop broadcast messages and do not use multi-hop broadcast over several nodes. Content dissemination is achieved by caching and mobility of nodes.

4.3 Simulation Results

This section shows the results of the scenarios described in section 4.2.

4.3.1 Impact of ε

Figure 4.1 shows the time it takes the requesting nodes to receive enough encoded packets for being able to decode on the largest playground measuring 5000m x 5000m. The x-axis indicates the time in seconds while the y-axis denotes the cumulative number of nodes that are able to decode. We can see that the difference between the three scenarios is marginal, i.e., curves are overlapping. This can also be observed on the two smaller playgrounds measuring 500m x 500m and 1000m x 1000m as well as for other values for $deltaN$. The reason for this negligible difference can be found in the small difference of additional packets: Transmitting 1000 packets, the difference from the smallest value for $\varepsilon = 3$ to the biggest value $\varepsilon = 20$ is only 17 packets or 1.7%.

For this reason, we have decided to neglect the effect of the variable ε and to select a value $\varepsilon = 3$ in all remaining evaluations of this chapter. In a real-world implementation, the value for ε could be chosen dynamically: A requesting node keeps on collecting additional encoded packets until the node eventually becomes able to decode.

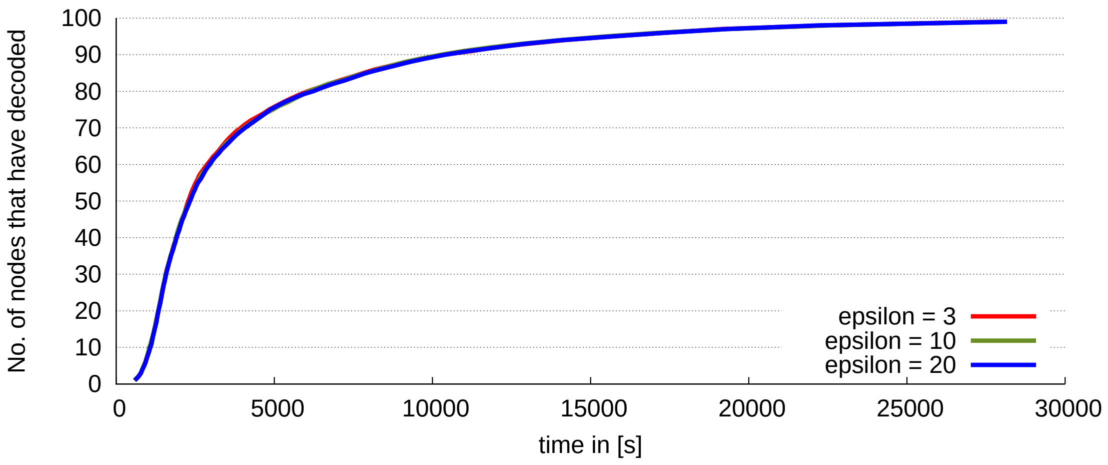


Figure 4.1: Cumulative number of nodes that have decoded for variable ε. Playground: 5000 x 5000, $deltaN = 100$.

4.3.2 Impact of deltaN

In our scenarios, as both the number of nodes and the transmission power are fixed, larger playgrounds always implicate lower node densities and therefore, a requester has fewer neighbor nodes. We can thus introduce a measure for node density with respect to the node's transmission range, which is defined as follows

Playground Side Length	Net Node Density
500	72.38
1000	18.10
5000	0.72

Table 4.4: Net node density for different playgrounds and 100 nodes.

$$Net\ Node\ Density = \frac{No.\ Nodes \times \pi \times (Transmission\ Range)^2}{Playground\ Area} \tag{4.1}$$

This value indicates the average number of nodes in communication range of a specific node. Table 4.4 shows the net node density values calculated using Formula 4.1 for the different playgrounds. We can observe that while the net node density is high on the two smaller playgrounds, the value is smaller than one on the largest playground measuring 5000m x 5000m. This means that in average, nodes will not be able to connect to a neighboring node at all times.

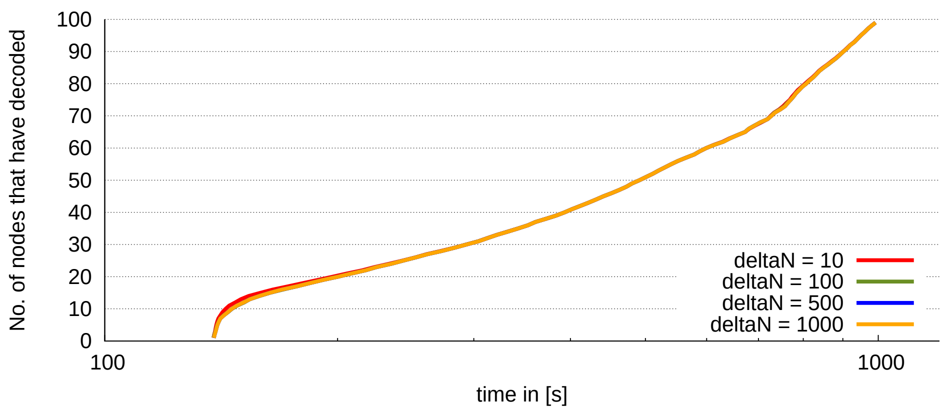


Figure 4.2: Cumulative number of nodes that have decoded for variable deltaN. Playground: 500 x 500, $\varepsilon = 3$

In the first scenario, we evaluated the influence of the parameter $deltaN$. Figure 4.2 shows the cumulative decoding time of 99 requesters in a playground size of 500m x 500m. The x-axis shows the simulation time in seconds while the y-axis shows the number of nodes that have already decoded the encoded packets and thus reconstructed the original data. Figures 4.3 and 4.4 show the same values but for the two larger playgrounds of 1000m and 5000m side length respectively. As we can see on the figures, on the two smaller playgrounds, there is virtually no difference between the different values of $deltaN$. On the largest playground, however, we can observe that high values for $deltaN$ are disadvantageous as the speed of content dissemination is reduced. Having a high value of $deltaN$, the requester can request a broader range of encoded packets to satisfy its demands. If a specific encodedID is not available at a certain time, the node just continues requesting the subsequent encodedID until it reaches the top of the range of available packets, i.e., the packet with encodedID $seed + K + deltaN$. Then, the node starts

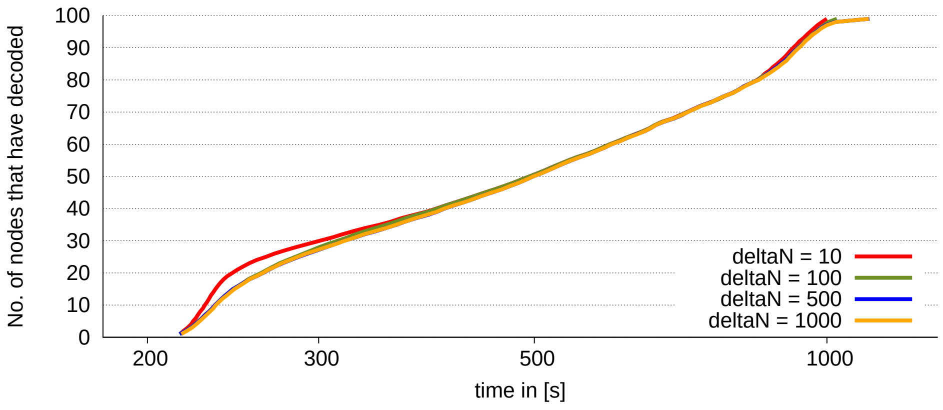


Figure 4.3: Cumulative number of nodes that have decoded for variable deltaN. Playground: 1000 x 1000, $\varepsilon = 3$

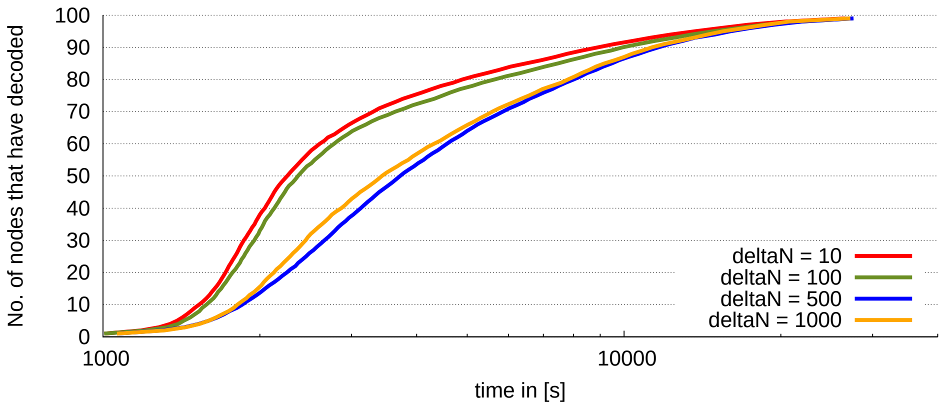


Figure 4.4: Cumulative number of nodes that have decoded for variable deltaN. Playground: 5000 x 5000, $\varepsilon = 3$

again requesting the lowest available encodedID. Hence, for fewer encoded packets available, the probability that a certain node can provide packets which other nodes request, increases. Vice-versa, having high values for $deltaN$, the probability of two requesters asking for the same encodedID decreases and thus the curve gets less steep.

To comply with the CCN request-response scheme, we decided in this thesis that each requester needs to actively request each encoded packet. This means that overheard content only gets known to the Raptor coding application if an Interest in its name has been expressed. To profit from the increased encoded packet diversity due to higher values of $deltaN$, requesters should be able to detect unsolicited encoded packets arriving at the content store and to retrieve them before their content lifetime expires and the packet is removed from cache. The characteristics for different $deltaN$ might of course be different for such strategies.

Another metric is the number of Interest packets a requester has to send prior to receiving the required number of encoded packets. Figure 4.5 shows the value of $deltaN$ on the x-axis and the number of messages on the (logarithmic) y-axis for all playgrounds. We can observe that the number of Interests the requesters require to send do not vary for different values of $deltaN$ on the two smaller playgrounds. On the largest playground measuring 5000m x 5000m, however, the number of Interest messages increases for larger values of $deltaN$. While the number of Interest messages for $deltaN = 100$ is only increased by 5% compared to $deltaN = 10$, the scenarios with higher values of $deltaN = 500$ and $deltaN = 1000$ require much more Interests to be sent: 43.5% more for $deltaN = 500$ and 45.7% more for $deltaN = 1000$, again in comparison to $deltaN = 10$. If a node has few neighbors (0.72 in average on the largest playground) and $deltaN$ is high, as there are more encodedIDs available, the probability that a requester transmits an Interest in the name of a packet its neighbor has cached is low. Therefore, it is less likely that nodes can profit from their neighbor's content store data than with low values for $deltaN$. We can further see in Figure 4.6 that the number of duplicates a requester received while requesting the encoded packets stays about the same for all the four different values of $deltaN$.

Overall, we can observe that smaller values of $deltaN$ are preferable for this requesting strategy. However, the transmission performance heavily depends on the playground size: Due to the lower net node density on large playgrounds, a requester may often request content that is not available within its transmission range.

4.3.3 Impact of Playground Size

In this subsection, we fix $deltaN$ to the value that performed best in the previous evaluations and compare the performance on different playground sizes. Thus, the parameters are set to $deltaN = 10, \varepsilon = 3$.

Figure 4.7 shows the time required for all 99 requesting nodes to receive enough encoded packets for being able to decode. The x-axis shows the time in seconds while the y-axis indicates the cumulative number of nodes that were able to decode. We can observe that the time to receive enough packets to decode increases significantly with increasing playground size. This is due to lower net node density and requesters requiring more time to find a content source. From Table 4.4, we can see that on average, several nodes are within a node's transmission range while on the largest playground it is not guaranteed that a node has a neighbor. Small values implicate

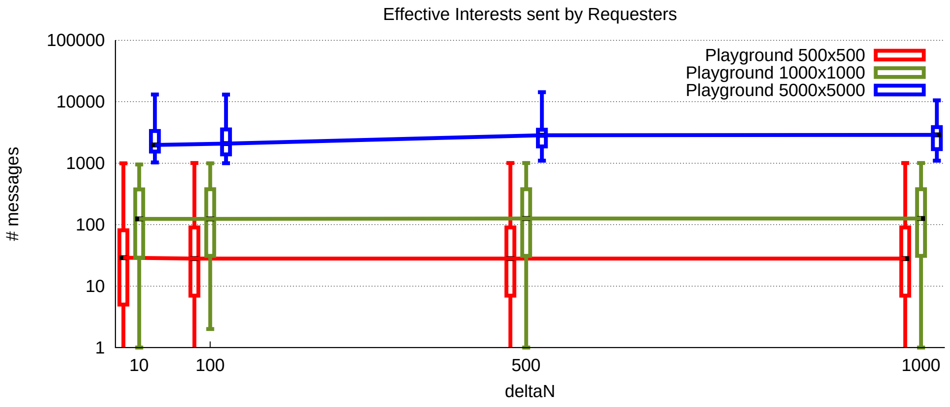


Figure 4.5: Interests sent for variable deltaN on all three playground sizes. $\varepsilon = 3$

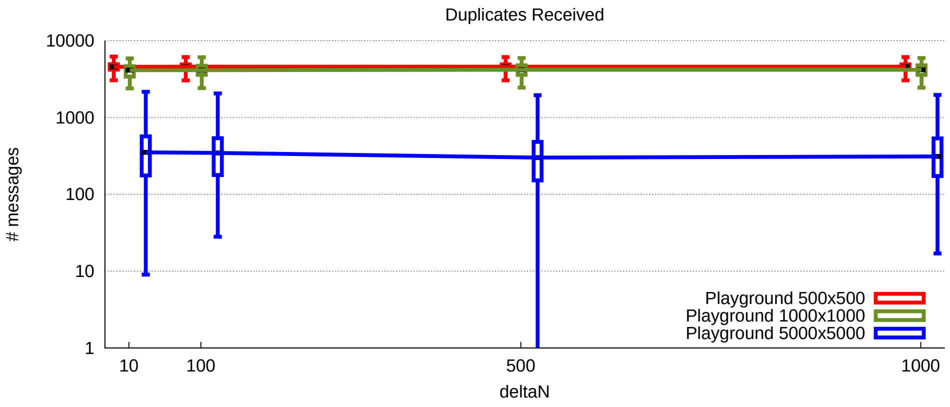


Figure 4.6: Duplicates received for variable deltaN on all three playground sizes. $\varepsilon = 3$

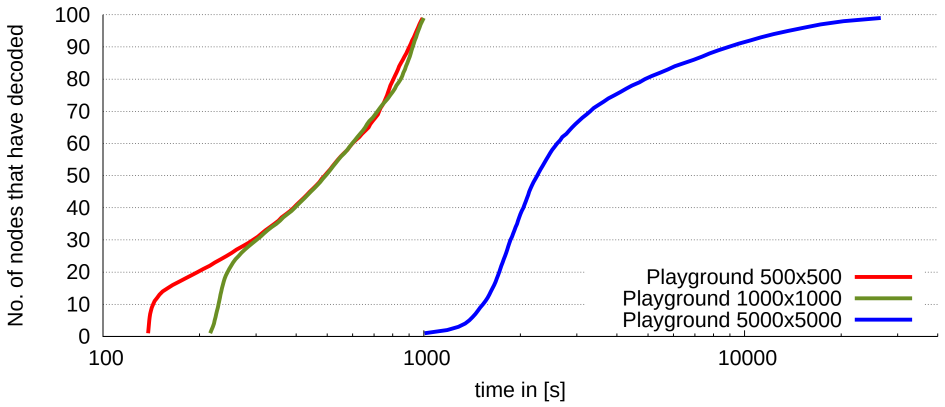


Figure 4.7: Cumulative number of nodes that have decoded for variable playgrounds. $\varepsilon = 3$, $deltaN = 10$.

that nodes might take some time to get within reach of the content source or lose the connection to the content source for some time. Further, as less nodes are within transmission range, the probability that a adjacent node's cache holds the requested packet decreases.

As expected and seen in Figure 4.8, the number of Interests required before nodes can decode rises along with the playground size. In average, the number of Interests sent on the playground 1000m x 1000m compared to the smallest playground increase by 327%. From the medium playground to the largest, there is again an increase in Interest messages of over 1500%. As seen in subsection 4.3.2, the largest playground has by far the lowest net node density. Thus, the probability that a node gets into the transmission range of the content source or another node which can provide the requested content decreases as the playground grows. Since any node can possibly profit from data packets sent by other requesters using one-hop broadcast, requesters require less Interests to send out by themselves in areas with high node densities. Requesters can provide content if they have requested the same content earlier and stored them in their persistent storage or if they have cached a packet at the CCN layer whose content lifetime has not yet expired.

Additionally, having a higher net node density, the probability for a requester to overhear content transmission increases. As an example, if a node sees five other nodes within its transmission range, it will be able to overhear some packets these five nodes request.

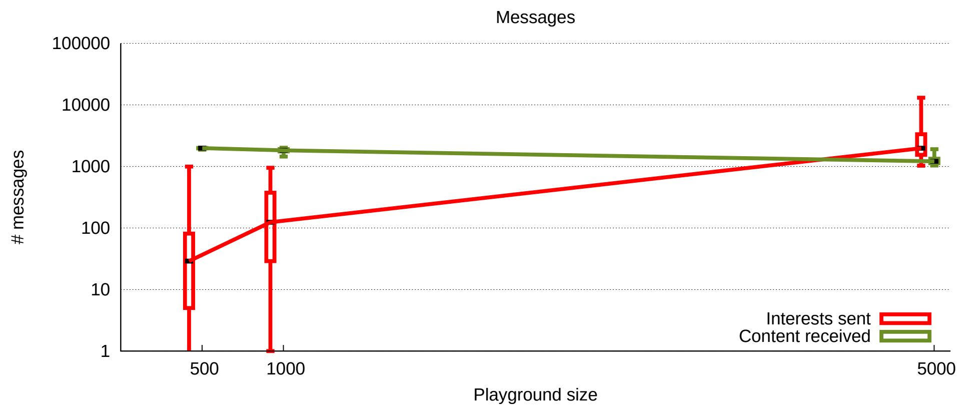


Figure 4.8: Number of Interests sent and content received for different playgrounds with $deltaN = 10, \varepsilon = 3$.

4.3.4 Impact of Requester Ratio

In this subsection, we will compare the performance depending on how many nodes request content, relatively speaking in terms of the total number of nodes. The performance will be measured by evaluating both the time until nodes become able to decode and the number of Interest packets a requester needs to send in order to obtain the required number of encoded packets. The total number of nodes stays fixed to 100 nodes but now, instead of having one content source and 99 requesters, we reduce the number of requesters to 10 and 50 nodes. The coding parameters are set to $deltaN = 10$ and $\varepsilon = 3$.

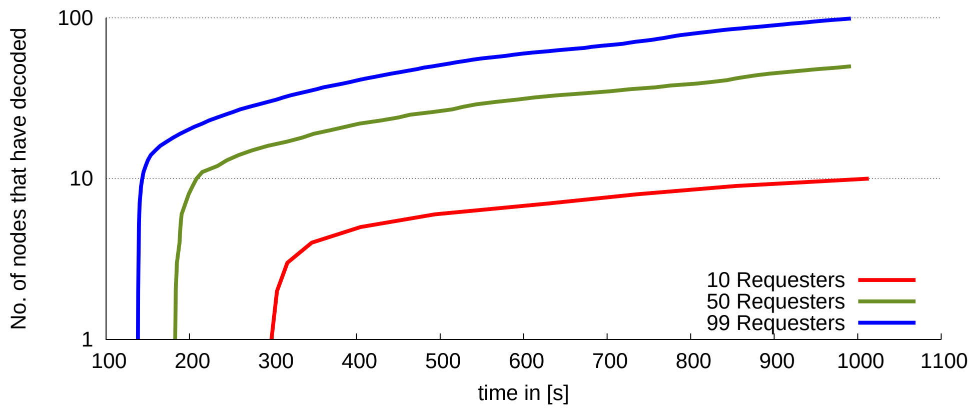


Figure 4.9: Cumulative number of nodes that have decoded for variable number of requesters. Playground: 500 x 500, $deltaN = 10, \varepsilon = 3$.

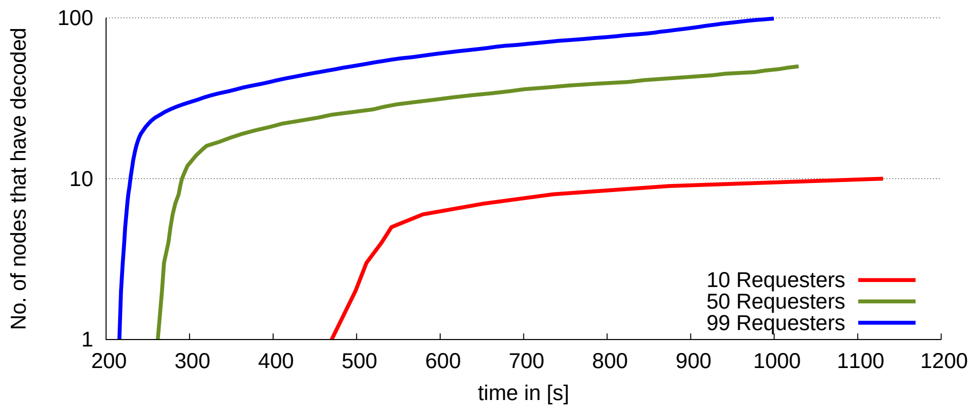


Figure 4.10: Cumulative number of nodes that have decoded for variable number of requesters. Playground: 1000 x 1000, $deltaN = 10, \varepsilon = 3$.

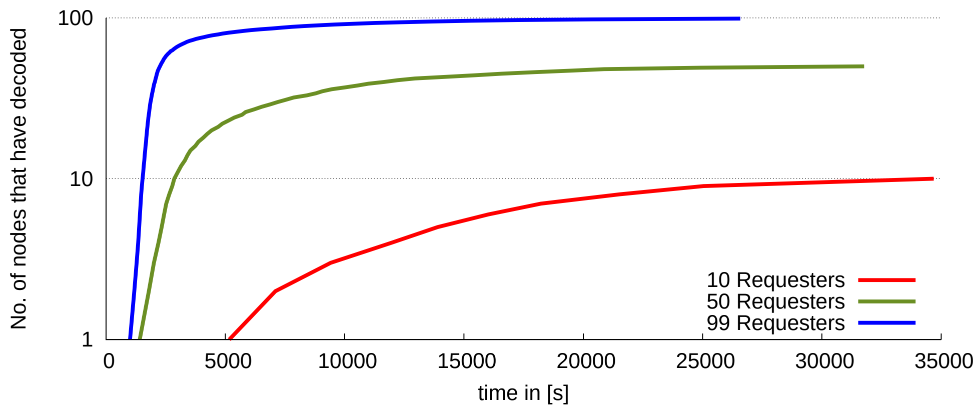


Figure 4.11: Cumulative number of nodes that have decoded for variable number of requesters. Playground: 5000 x 5000, $deltaN = 10, \varepsilon = 3$.

The Figures 4.9, 4.10 and 4.11 show the cumulative number of nodes that retrieved enough encoded packets for being able to decode for the three playground sizes. The x-axis indicates the time in seconds while the y-axis shows the cumulative number of nodes that were able to decode. We can see that on the playground of 500m x 500m, in all three scenarios, it takes about 1000 seconds for all nodes to retrieve the required amount of packets with just a marginal additional delay for the 10 requesters scenario. The larger the playground, the bigger the difference in the times until all nodes have decoded for the different ratios of requesting nodes gets. Again, as the node density is high, the nodes will likely be able to connect to the content source or another requester already providing the requested content. As explained above, requesters store requested encoded packets persistently and provide it to others. The content lifetime only has an influence on cached content at non-requesting nodes but not on persistent storage at requesters. Hence, requesting nodes can provide already received encoded content to other nodes at application level. If less nodes request content, less nodes provide the content they have retrieved, thus, content dissemination slows down which means it takes longer for all nodes to complete. Also, having high requester density, more nodes will profit if a packet is transmitted using one-hop broadcast.

Similar to the net node density measure introduced in subsection 4.3.2, we can define the measure *net requester density*, which only takes requesting nodes in account. The net requester density describes the area covered by all requesting nodes in proportion to the total playground area and can thus be used to measure node connectivity, i.e., the number of nodes in communication range of a requesting nodes in average. It is defined as follows

$$Net\ Requester\ Density = \frac{No.\ Requesting\ Nodes \times \pi \times (Transmission\ Range)^2}{Playground\ Area} \tag{4.2}$$

		Playground size		
		500 x 500	**1000 x 1000**	**5000 x 5000**
No. requesting nodes	**99**	71.66	17.91	0.71
	50	36.19	9.04	0.36
	10	7.24	1.81	0.07

Table 4.5: Net requester density

Table 4.5 shows the values for the different numbers of requesting nodes. Comparing with above Figures 4.9, 4.10 and 4.11, we can again observe that scenarios with high net requester density values perform faster than those with low net requester density.

We can also compare the number of Interest packets the requesters sent out. Figure 4.12 shows the number of Interest packets the application scheduled for different numbers of requesters on the three playground sizes. The x-axis denotes the playground size and the y-axis shows the number of Interest packets. We can see that the scenario with 99 requesters – the one which finished first and which has the highest requester density – also needed fewest Interest packets to be transmitted since nodes can profit from Interests sent by others. As seen in Figures

4.10 and 4.11, the 50 requesters scenario performed faster than the 10 requesters scenario on the two larger playgrounds. The same characteristics can be observed concerning the number of transmitted Interest packets. Having only few requesting nodes, the probability that any adjacent node can provide the requested packet is lower than if each node requests and stores content.

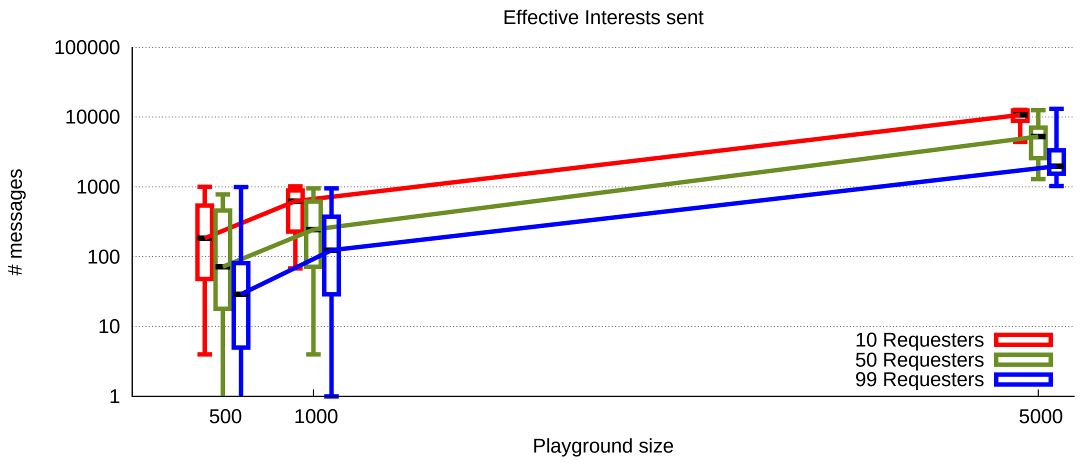


Figure 4.12: Number of Interest packets sent for different playground sizes and different numbers of requesters. $deltaN = 10, \varepsilon = 3$.

4.3.5 Comparison to Regular CCN Traffic

In this subsection, we compare content transmissions via Raptor coding with regular CCN content transmissions without Raptor coding. Figure 4.13 shows the time required for all 99 requesters to receive enough packets to decode. The x-axis indicates the time in seconds while the y-axis shows the cumulative number of nodes that finished content transmission. As we can see, the Raptor coding application outperforms regular CCN by far in terms of transmission time on the two smaller playgrounds. On the largest playground, using Raptor coding, the first few nodes get the complete content earlier than in regular CCN. However, in regular CCN, it takes a shorter time for all nodes to gather the whole content. This can be explained by the different Interest transmission strategies in Raptor CCN and regular CCN.

In regular CCN, data packets are requested sequentially. If a timeout occurs – due to mobility or channel loss for instance – the same packet is retransmitted instantly. If several Interest packets remain unanswered, the requester pauses for four seconds and then again sends an Interest packet. In addition, if a timeout occurs, the pipeline window is set to one which means that a maximum number of one Interest can be pending at a time. For each received segment, the pipeline window will then again be increased by one until it reaches the pipeline size. In the Raptor coding approach, the pipeline window is fixed to 16 which is the maximum size for regular CCN. Hence, a node will always send Interest packets for the next 16 packets it requires. If a segment is not available, it is skipped and the requester proceeds with the next higher segment it has not received yet. This strategy is especially advantageous if node density is high

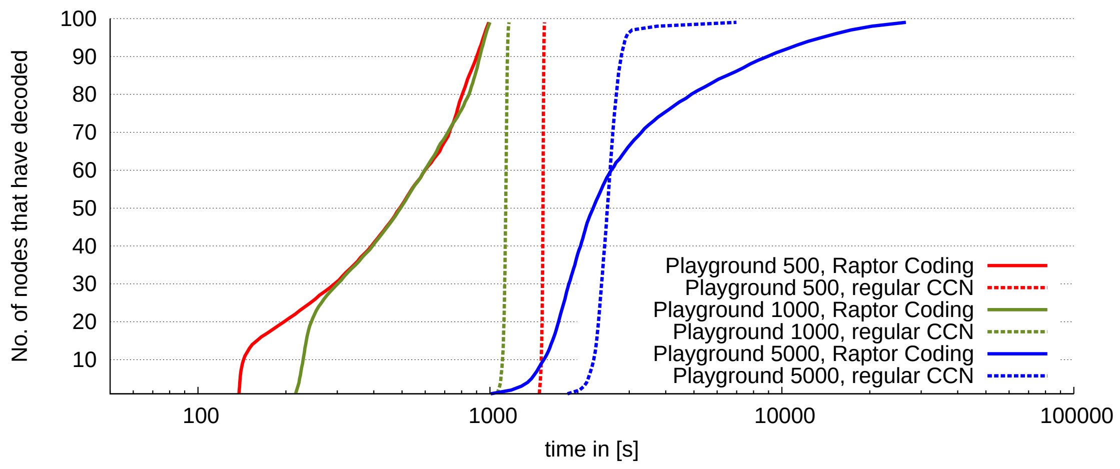


Figure 4.13: Cumulative number of nodes that have decoded for variable playgrounds on logarithmic time scale. $\varepsilon = 3, deltaN = 10$.

since available content can be collected quickly. On the other hand, having low node density, this approach keeps sending 16 Interests at a time and thus generates much Interests in situations where no other node is within transmission range.

Figure 4.14 shows the number of Interest packets the Raptor coding application sends out until the required number of encoded packets is received, denoting the playground size on the x-axis and the number of Interest messages on the y-axis. Please note that the y-axis is scaled logarithmically. We can observe that overall, Raptor coding shows the bigger variance. On the smallest playground, the average number of Interest messages required is 93% lower in the Raptor coding scenario than with regular CCN. Having the playground of 1000m x 1000m, Raptor coding still transmits 80% less Interest packets. However, on the largest playground size of 5000m x 5000m, Raptor coding requires more Interest packets being sent out in order to retrieve all required content. This pattern can also be observed in the scenarios where the ratio of requesting nodes is reduced to 10 and 50 requesting nodes. The effect relates to the reduced connectivity of nodes which is due to the reduced net requester ratio on large playgrounds. If no content source within reach can reply to a requester's Interest packets, it keeps transmitting Interest packets. As the net requester density is low on the largest playground, this is more likely to occur than on the smaller playgrounds. In such scenarios, it would be advantageous to adapt the requesting strategy and reduce the pipeline window if not connected to any node. This would lower the number of Interests being sent to obtain the required number of encoded packets since unnecessary Interests are avoided if the requester is not connected to any node.

Next, we want to compare Raptor coding with regular CCN when not all nodes are requesting content. Figure 4.15 shows the number of Interest packets sent out until the required number of packets has been received for the scenarios with requester ratios of 10 and 50 percent. The percentages are indicated in parantheses in the legend. The x-axis shows the playground size and the y-axis shows the number of transmitted Interest messages. Please note that the y-axis is scaled logarithmically. We can observe that with Raptor coding, fewer requesters lead to an

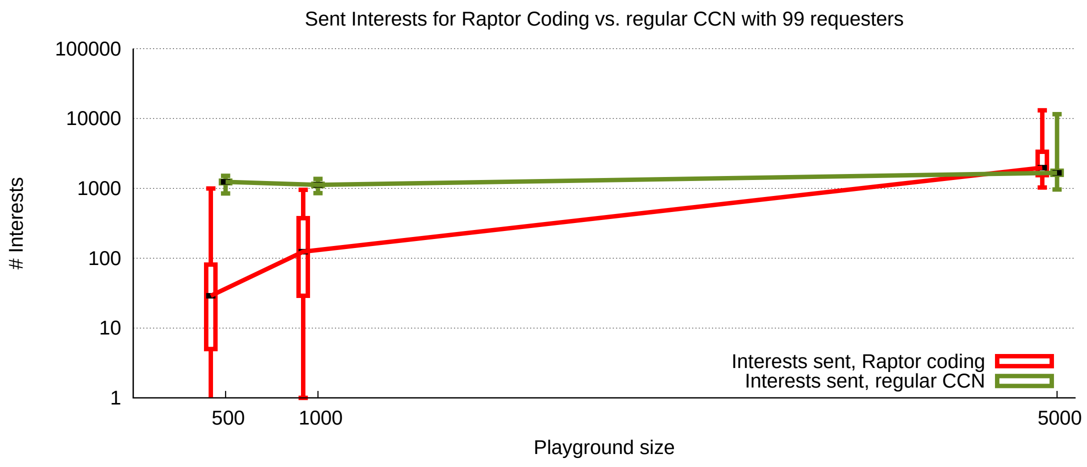


Figure 4.14: Comparison of Interest packets sent using Raptor coding application and regular CCN application on different playgrounds. $\varepsilon = 3$, deltaN=10.

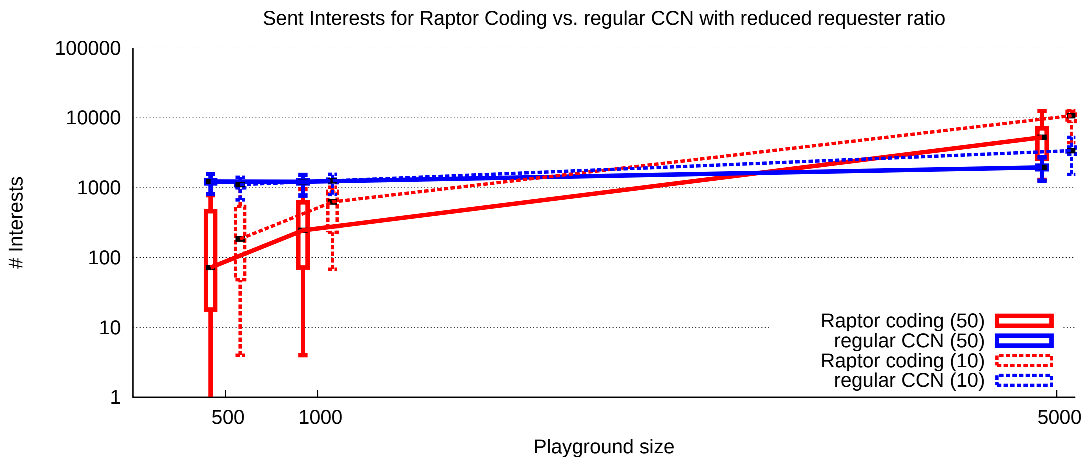


Figure 4.15: Comparison of Interest packets sent having a reduced ratio of requesting nodes. $\varepsilon = 3$, $deltaN = 10$.

increase in Interest packets being transmitted. Regular CCN requires approximately the same number of Interests on the 500m x 500m and 1000m x 1000m playgrounds. However, on the largest playground, the fewer nodes request content, the more Interests need to be transmitted in order to receive the requested. While Raptor coding requires less Interests being sent on the smaller playgrounds with 99 requesters, it transmits 18.75% more Interest messages than regular CCN in average on the largest playground. If the requester ratio decreases, the number of additionally transmitted Interests increases. In the scenario with 50 requesters, Raptor coding sends out 169.8% more Interest messages than regular CCN does. In the 10 requester scenario, Raptor coding even transmits 217.2% more Interest packets than regular CCN.

As requesters also act as content sources for content they have already received, the decreasing number of requesters also decreases the number of potential content sources. This supports the conclusion that Raptor coding is advantageous in scenarios with a high net requester density while regular CCN performs better on areas with a rather sparse net requester density.

		Playground size		
		500 x 500	**1000 x 1000**	**5000 x 5000**
No.	**99**	71.66	17.91	0.71 ▨
requesting	**50**	36.19	9.04	0.36 ▨
nodes	**10**	7.24	1.81 ▨	0.07 ▨

Table 4.6: Net requester density. Regular CCN performs better where cells are shaded.

Based on Raptor Coding and regular CCN performance, the radio parameters and Equation 4.2, we can derive Table 4.6, which links the net requester density to the different simulation scenarios. The cells which are shaded grey identify scenarios in which Raptor coding with the current requesting strategy could not prove to be advantageous over regular CCN. We can see that having a net requester density below 2, regular CCN performs better. Having a higher net requester density, however, Raptor coding can reduce both time and number of Interest messages required to receive data.

Therefore, we can conclude that we either use Raptor coding only in crowded areas with high node density, or we need to increase the net node density by increasing the transmission power of the mobile nodes' antennas (if possible).

4.3.6 Discussion

Having mentioned the three most important questions at the beginning of this chapter, after analyzing the simulation results, we can now answer them:

How do the parameters $deltaN$ and ε influence Raptor coding performance?

The parameter $deltaN$ determines the number of linear combinations that are generated in addition to the number of original data packets. ε on the other hand indicates the number of encoded packets a requester receives until it is able to decode, again in addition to the number of original data packets.

First having a look at the parameter $deltaN$, the simulation showed that smaller values for $deltaN$ are advantageous in terms of time needed until decoding is feasible. This also holds with respect to the number of Interests required in order to get all needed encoded packets. As requesting nodes cannot detect when content is added to their cache, their ability to overhear transmissions is limited.

However, regarding ε, there was hardly any discrepancy between the different values. Therefore, the value for ε should be chosen to comply with a sufficient probability of successful decoding as mentioned in chapter 4.2.

How does Raptor coding performance change when the playground size increases and thus node density drops?

Raptor coding performs much better on smaller playgrounds. When having the same number of nodes, the bigger the playground, the smaller the chance that a node meets another station which can provide the requested content. Having an extended set of packets that may be requested in Raptor coding compared to the input packet set requested in regular CCN, the probability that an adjacent node can provide the requested content even drops lower. Hence, having low net node density, Raptor coding cannot improve transmission performance compared to regular CCN.

Above conclusions are made under the assumption that the number of nodes on the playground stays fixed which implicates that the playground size determines the net node density. We expect that increasing the number of nodes on the largest playground accordingly, i.e., by factor 25, would lead to similar results to the ones observed on the 1000m x 1000m playground.

What is the impact of different ratios of requesting nodes?

On small playgrounds, the different proportions of requesting nodes do not influence the time until all requesters are able to decode in a relevant manner since the net node density remains high in all cases. The larger the playground, however, the bigger the difference: Scenarios with few requesters show a slow data dissemination which results in longer times until all requesters can decode. Likewise, a high number of requesters reduce the average number of Interest messages needed to be transmitted in order to decode the requested content. Both these effects mean that Raptor coding performs better in environments where many nodes request the same content.

How does Raptor coding perform compared to regular CCN traffic?

Focusing on the smaller playgrounds with sizes of 500m x 500m and 1000m x 1000m, Raptor coding delivers a mentionable gain in performance with respect to both the time and Interest messages needed to receive the whole requested content. However, when considering larger environments and / or environments with net requester density below 2, regular CCN takes less time for all nodes to receive the content. Similarly, fewer Interests need to be transmitted on the two smaller playgrounds while the largest playground requires Raptor Coding to transmit more Interest packets than regular CCN.

We have seen that Raptor coding is advantageous in situations where requester density is high. This knowledge could be used to deploy Raptor coding in crowded environments where node density is expected to be high, for instance at airports, shopping malls or sports stadiums. Even time-dependant applications, e.g. at train stations during rush hour, are conceivable.

Chapter 5

Conclusion

5.1 Summary

In this bachelor thesis, we added Raptor coding functionality to content centric networks. In case of collisions or packet loss, Raptor coding allows network nodes to retransmit not exactly the same packet again but another linear combination of the content requested. This may lead to a gain of knowledge for other nodes within the transmission range. Simulating different scenarios using OMNeT++, we measured the time needed to receive all segments of the received content as well as the amount of Interest packets required being transmitted to achieve this goal.

5.2 Conclusions

The evaluation of the different simulations has shown that Raptor coding is advantageous especially concerning the time required until all nodes have received the whole content having environments with high node density and using a relatively small value for $deltaN$. The parameter ε does not influence the transmission characteristics but can be chosen to comply with a sufficient probability of successful decoding or even in a dynamic manner.

However, the full potential of Raptor coding cannot be exploited as the underlying concepts are very different: Raptor coding is based on the ability of network nodes to overhear transmissions on the wireless medium. CCN, on the other hand, requires each and every data packet to match an Interest packet previously generated. Hence, CCN does not support opportunistic overhearing of data packets, which is a key strategy in Raptor coding.

5.3 Future Work

One fundamental element of content centric networks – trust and signature – has not been addressed in this thesis. Thus, when intending to deploy Raptor coding in CCN, a strategy for signing the encoded packets without impairing the current signature mechanisms that ensure trust for data packets needs to be engineered.

Also, we have used an integer value to assign IDs to encoded packets. Together with the seed and the parameter $deltaN$, the encodedID could identify the set of encoded packets a particular packet belongs to. When the number of different encoding sets for one set of input packets (i.e. for one content prefix) increases, the probability of getting overlapping ID ranges or even an overflow of the data type increases. This absolutely has to be avoided since ambiguous content names would corrupt the decoding process beside its undesirability in general. Since there was only one content source producing encoded packets in our simulations, we could neglect this problem. However, having multiple content sources or introducing re-encoding at each node, this issue needs to be addressed. In a real-world implementation, prefix matching also included the signature which is uniquely added to each packet at the content source and which makes the problem less severe. In the simulations, however, we did not consider signatures. For large numbers of input packets, even the encodedIDs data type might be changed to ensure that no overflow occurs.

Running the experiment as OMNeT++ simulation means that processing time is not taken into account. This could, for instance, have the impact that Interests in encoded content might expire while the content source encodes the requested packets due to their expiration time of four seconds. This issue was not relevant for our simulations but should be considered when carrying the scenarios to physical nodes. This solution could for example involve an additional packet which tells the requester that original data packets are currently being encoded. This packet will contain the seed of the encoding by which knowledge the requester can generate Interests in the encoded packets without additional delay.

For playgrounds with low net requester density, the chosen requesting strategy may not be ideal. Thus, the requesting strategy might be chosen according to the net requester density. Also, facilitating the nodes to detect cached content could enhance the ability to overhear content transmissions and thus reduce the number of Interest messages that have to be sent out. A way to solve this could be to introduce Interest messages with local scope, i.e., whose reach is limited to the node the application is connected to.

The thesis has highlighted the huge conceptional gap between strictly Interest-based content reception in CCN and opportunistic data-accumulation in Raptor coding which was further explained in chapter 3. Aiming to make use of Raptor coding or network coding in CCN universally and in an efficient manner, this gap must be conquered.

Bibliography

[1] C. Fragouli, D. Katabi, A. Markopoulou, M. Medard, and H. Rahul, "Wireless network coding: Opportunities x00026; challenges," in *Military Communications Conference, 2007. MILCOM 2007. IEEE*, oct. 2007, pp. 1 –8.

[2] M. Luby, M. Watson, T. Gasiba, T. Stockhammer, and W. Xu, "Raptor codes for reliable download delivery in wireless broadcast systems," in *Consumer Communications and Networking Conference, 2006. CCNC 2006. 3rd IEEE*, vol. 1, jan. 2006, pp. 192 – 197.

[3] 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

[4] A. Shokrollahi and M. Luby, *Raptor Codes*, 2011, vol. 6, no. 3-4. [Online]. Available: http://dx.doi.org/10.1561/0100000060

[5] "OMNeT++ simulation framework," http://www.omnetpp.org, 2014.

[6] P. Chou and Y. Wu, "Network coding for the internet and wireless networks," *Signal Processing Magazine, IEEE*, vol. 24, no. 5, pp. 77 –85, sept. 2007.

[7] A. Albanese, J. Blomer, J. Edmonds, M. Luby, and M. Sudan, "Priority encoding transmission," *Information Theory, IEEE Transactions on*, vol. 42, no. 6, pp. 1737–1744, Nov 1996.

[8] N. Thomos, J. Chakareski, and P. Frossard, "Prioritized distributed video delivery with randomized network coding," *Multimedia, IEEE Transactions on*, vol. 13, no. 4, pp. 776–787, Aug 2011.

[9] N. Thomos and P. Frossard, "Network coding of rateless video in streaming overlays," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 20, no. 12, pp. 1834–1847, Dec 2010.

[10] A. Shokrollahi, "Raptor codes," *Information Theory, IEEE Transactions on*, vol. 52, no. 6, pp. 2551 –2567, june 2006.

[11] “Nrutil, Numerical Recipes Public Domain Software,” http://www.nr.com/pubdom/nrutil.c.txt, 2014.

[12] “Boost C++ serialization library,” http://www.boost.org/libs/serialization, 2014.

[13] T. Camp, J. Boleng, and V. Davies, “A survey of mobility models for ad hoc network research,” *Wireless Communications and Mobile Computing*, vol. 2, no. 5, pp. 483–502, 2002. [Online]. Available: http://dx.doi.org/10.1002/wcm.72