Content Discovery and Retrieval Application for Mobile Content-Centric Networks

Bachelorarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Arian Uruqi
2014

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

Contents

List of Figures

Acknowledgement

I would like to express my gratitude to Prof. Dr. Torsten Braun for giving me the opportunity to write a thesis in the research group Communication and Distributed Systems, which he heads.

Furthermore, I would like to thank Carlos Anastasiades for coaching me in his field of research, Content-Centric Networking. He was able to motivate me and share the joy of working in a very promising and prospering networking paradigm field, by having lively exchange with me and letting me participate in a submission included in this report.

Last but not least, I am very thankful for the support and encouragement my family and friends have provided, especially during periods of stress.

Abstract

Content-centric communication may yield benefits in mobile and dynamic ad hoc networks, because communication is more resilient to individual node mobility: content can be discovered (and retrieved) without having to locate possible content providers first. This thesis' goal is to explore these benefits in single-hop communication.

For this purpose, an application has been implemented using the Java API of the open source CCN implementation CCNx. It supports two content discovery algorithms:

First, enumeration requests retrieve a list of content names at a repository that matches the requesting prefix and are formed using special Interest prefixes. By subsequently excluding the ID from the responding repository supplied with the response, the request is re-expressed until no new lists can be found and it times out. The retrieved lists are then accumulated to represent the available content.

Second, regular Interest messages can be used for content discovery as well, by excluding retrieved content names in subsequent Interest expressions. With this method, content is not only discovered in repositories but also nearby caches.

Evaluations have been performed using VirtualMesh, a hybrid network emulation framework. A dedicated server simulates ad-hoc networking as well as location and distance of nodes. Message processing is performed on Xen-virtualized guest domains. Communication with other nodes is routed through a virtual network interface on the virtual machines.

To automate application testing and finally collection of evaluation data, a framework has been established automating the steps of initialization and program execution. For a particular node configuration, it enables automatic evaluation of CCNx applications by executing and collecting information from CCNx applications running on the hosts.

Chapter 1

Paper: Content Discovery in Opportunistic Content-Centric Networks

Content Discovery in Opportunistic Content-Centric Networks

Carlos Anastasiades, Arian Uruqi, Torsten Braun
Institute of Computer Science and Applied Mathematics
University of Bern, Switzerland
{lastname}@iam.unibe.ch

***Abstract*— Host-based mobile ad hoc communication requires the transmission of periodic hello beacons to identify neighbors. Drawing conclusions from received beacons, e.g., containing information about existence and neighbor nodes, to available or demanded content is not possible and the gathered information may be outdated quickly due to dynamic environment changes. Therefore, content-centric networking results in more flexible communication without the need of neighbor information. Instead, information about available content is required. In this paper, we will investigate two different content discovery strategies and discuss their efficiency for mobile communication. The algorithms have been implemented in the CCNx framework and evaluated in VirtualMesh, a hybrid emulation tool for wireless mobile ad hoc networks.**

***Index Terms*—discovery, content-centric, opportunistic, ad-hoc networks**

I. Introduction

Content-centric networks are a new networking paradigm for the future Internet. Routing is performed based on content identifiers instead of IP addresses. The approach addresses scalability, security and efficiency concerns of the current host-based Internet architecture. Many different architectures have been proposed, [1], [2], [3], [4], which mainly differ in the way content is named. Hosts need to express interests in, or subscribe to, names to get the corresponding objects published by a content source. In this paper, we focus on the Content-Centric Networking approach proposed in [4], which is hereafter referred as CCN. Although current research mainly targets wired networks, CCN has already been identified [5] as promising approach for mobile and dynamic networks since communication is more resilient to individual node mobility. Instead of trying to reach a specific host, the user tries to get a specific piece of content that can be provided from any other node that holds the content. Most current research works target caching, security and forwarding strategies in CCN. In this work, we investigate a more fundamental requirement, namely, the discovery of available content in a distributed wireless broadcast environment. This is required in scenarios without centralized directories where content objects are generated dynamically and names cannot be predicted deterministically. It enables users to learn available content objects and services without demanding the entire content. Instead of connecting individually to each host, the requester can express an Interest and receive an answer from any reachable content source given that the content is available.

The remainder of this paper is organized as follows: In section II we shortly review the basic functionality of CCN. Related work to CCN is reviewed in section III. Section IV describes content discovery algorithms. Evaluation results are presented and discussed in section V. Finally, in section VI, we conclude our work and give prospects to future work.

II. Content-Centric Networking

In this section, we will briefly describe the main concept of CCN. Readers may refer to [4] for more information.

A. CCN Messages

CCN communication is based on two basic messages: *Interest* and *Data*. Content is organized in segments similar to chunks in BitTorrent [6]. File transfer is pull-based, and thus, users have to express Interests in every segment to obtain the entire content. The CCNx project [7] provides an open source reference implementation of CCN.

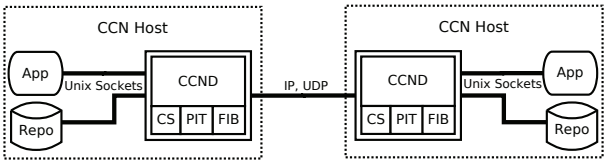

Fig. 1: Forwarding Architecture

The core element of the implementation is the CCN daemon (CCND), which performs the message processing and forwarding decisions. The connections from a CCND to other entities or local applications are called *faces*. In case of applications, the face corresponds to local Unix sockets. In case of mobile hosts, it corresponds to UDP or TCP sockets over IP. In this work, we use a multicast face using UDP and a multicast address to avoid individual IP addressing of mobile nodes.

Figure 1 shows the processing and forwarding architecture of CCNx. If an application on a node requests content, it sends an Interest message via a local face to the CCND on the local host. The CCND processes the message based on its information in the content store (CS), the pending Interest table (PIT) and the forwarding Interest base (FIB). Upon the reception of the Interest, the CCND will first check if the content is already in the CS, i.e. serving as a cache, and returns it if available and not expired. If the content is not available, it

will check the PIT whether the same request has already been expressed. The lifetime of an Interest will determine how long it stays in the PIT. If it is already in the PIT, the Interest can be discarded, because the corresponding data message is already pending. If there is no entry in the PIT, the host considers the FIB to check whether the host knows where to get the content from. If there is an entry, the Interest is inserted into the PIT and forwarded to a remote CCND, e.g., via the wireless channel using UDP as transport protocol. At the remote CCND, the procedure of checking the CS, PIT and FIB will be repeated. Content is persistently stored and shared with others in content repositories. CCN hosts that run a content repository can register the available prefix to their local CCND resulting in an additional FIB entry. Every Interest will result in at most one Data message and retransmission of the same Interest will result in the same Data message. To receive new content, it is required to either adapt the Interest prefix or add already received objects in the exclude field of the Interest.

B. Content Names

In CCN, content names follow a hierarchical structure as illustrated in Figure 2. The ellipses correspond to name components and the rectangles to data files. Each data file consists of one or several segments (not depicted in the figure). There are no restrictions on content names and they can be selected arbitrarily. The hierarchical name structure may not indicate the location of content objects as Figure 2 shows. Content objects may be stored on one, multiple or all hosts. In contrast to flat name spaces, it is not required to agree on common keywords. These keywords should be diverse enough to describe all possible objects but not too diverse to avoid confusions with similar keywords. The hierarchical structure supports the discovery with general prefixes such as IDs from specific publishers. A user may look for specific content names relative to the publisher's name space, e.g., '/publisherA/video/' or '/publisherA/audio/'. Content consumers may learn the naming schemes of their favorite publishers such as BBC, iTunes or netflix. This also enables the integration of social structures to identify reliable content publishers.

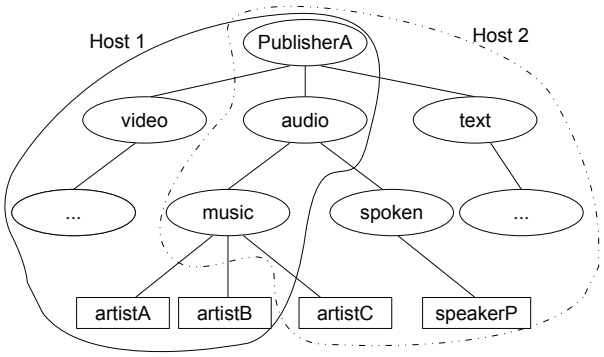

Fig. 2: Hierarchical Name Structure: files may be stored on different hosts.

III. Related Work

Previous work in [8] investigated the applicability of existing MANET routing protocols for mobile CCN based on analytical models. Routing in CCN is equivalent to finding a content source for a given name. The authors conclude that structured solutions such as geographic hash tables should only be used in networks without host churn whereas unstructured flooding is beneficial in small networks with high host churn. In CCN, Interests are routed towards content based on content-specific routing table (FIB) entries whereas content objects travel the same path back from where the Interest arrived. In a broadcast domain, this would result in unbounded Interest forwarding until the entire network is covered. CCN does not consider any multihop suppression mechanisms to avoid unnecessary transmissions or collisions.

In [9], CCN is applied to artificial battlefield scenarios featuring group mobility and hierarchical network topology. Content publishers distribute meta data of generated content to their neighbors and domain gateways. This distributed information is then used by requesting nodes to forward Interests to content publishers. Additionally, a content pushing approach is proposed to distribute information from a command center via a backbone network to specific locations using geographical routing. In a simple testbed, the benefits of CCN over existing routing mechanisms such as the Optimized Link State Routing protocol (OLSR) are shown, but CCN relies on the hierarchical structure and meta data distribution for forwarding.

Resilience to individual node mobility independent of the network topology can only be achieved by broadcast communication, since no individual nodes need to be configured and any node may answer requests. Although introducing flexibility, unbounded broadcast transmission may quickly result in broadcast storms [10]. In [5] and [11], the authors introduce the Listen First, Broadcast Later (LFBL) algorithm, which limits forwarding of Interest messages at every node based on its relative distance to the content source. Additional header fields in the messages indicate the hop distance from the previous forwarder to the destination. These fields are modified at every hop and messages are only forwarded by nodes closer to the destination than the previous forwarder. Although the approach targets the suppression of unnecessary messages, it may not reach that goal reliably. Not protected by the author's signature, the distance fields may yield imprecise information due to node mobility, particularly if messages are transmitted from caches or in case of multicast communication.

In this work, we want to investigate content discovery mechanisms in distributed environments that are independent of potential subsequent file downloads. Based on the discovery information, users may decide which content files to download. We limit the communication to single-hop connections similar to communiation in delay-tolerant opportunistic networks [12]. Therefore, we rely on the suppression mechanism in CCN, which cancels a scheduled transmission if received from the same face. Routing is replaced by the mobility of nodes, caching and reexpression of Interests. In contrast

to peer-to-peer based communication, e.g., in delay-tolerant networks such as Haggle [13] or PodNet [14], where all hosts periodically transmit hello beacons to keep track of neighboring hosts, no beaconing is required with CCN. A host expresses an Interest in a content file and receives data if it is available. Maintaining the neighbor list drains the energy of mobile devices, because beacons are transmitted periodically and independently of any data communication. In these systems, hosts subsequently connect to neighboring nodes to ask for available content. In case of dense urban environments and mobility, many subsequent connections may be required to find the desired content. In CCN, the requester may broadcast the Interest and any host that receives it and holds the corresponding content may respond. This may result in a faster discovery time, which is a crucial criterion if contact times are short.

Once the available content collections are known, a rich set of approaches exist in literature to discover availability of objects from these collections. Existing works in mobile ad hoc networks (MANETs) or delay-tolerant networks (DTNs) apply Bloom Filters [14] or attenuated Bloom Filters [15] to increase the efficiency of content or service discovery. If the synchronization of collections is targeted, the CCNx repository synchronization mechanism may be applied. It is based on a set reconciliation algorithm [16] that calculates the hashes of collections in a structured way. Such mechanisms may therefore optimize our discovery mechanisms, if the preferred collections are known, but can not replace them.

IV. Content Discovery Mechanisms

Content discovery mechanisms are required in distributed environements where content objects are generated dynamically and names cannot be predicted deterministically. In the absence of centralized directories and periodic beaconing, users need to learn available content names before selectively requesting specific objects. This information is required to avoid the download of all available content objects resulting in congestion on the wireless medium.

Therefore, we describe two discovery approaches based on *name enumeration requests* and *regular Interests* hereafter.

As motivated in section I, the algorithms target single-hop communications. We rely on the suppression of data transmissions in CCN, which cancels the transmission of scheduled content if received on the same face: for example, if another node has already responded to an Interest with the same content object. We assume that every node runs a repository with persistent storage extending its local temporary cache and that these repositories are not synchronized among each other. Although answers from secondary storage are slower than from primary storage, additional repositories may help if cached copies are not available anymore or too far away.

The discovery mechanisms described in this section are based on the same idea: the discovering node expresses an Interest with a general prefix and waits for responses. Based on the response, subsequent Interests may be expressed. If no answer is received within a timeout period, the content is assumed to be unavailable. Since Interests are broadcasted, nodes cannot rely on MAC layer acknowledgements from the destination as for unicast transmissions and the sender cannot detect any collisions. Therefore, reliability functionality to identify collisions and to differentiate them from unavailable content needs to be performed by the requester. We achieve this by a retransmission counter: if no answer to a discovery request is received within the specified time, the requester reexpresses the request until a configurable limit of transmission attempts has been reached. When reaching the limit without receiving an answer, the content is assumed to be unavailable. In order to reduce the collision probability in the first place, the content sources answer a discovery request after an additional random answering delay. To adapt to dynamic changing environments and discover newly available content, the algorithms may be repeated periodically or based on external events such as overheard traffic or on-demand. In contrast to periodic beaconing the users may discover new available content instead of new peers.

A. Enumeration Request Discovery

The Enumeration Request Discovery (ERD) requires the expression of name enumeration requests which are addressed only to local and remote repositories. A name enumeration request for a certain prefix requests the enumeration of first-level names under the prefix that are locally available at the repository. The requests are based on regular Interests but include command markers to indicate the enumeration. Figure 3 shows a sample message exchange for the naming tree in Figure 2. For simplicity, we do not show the command markers that are included in the requesting prefix and the returned enumeration name. The initial enumeration request for the prefix */publisherA/* triggers an answer including the next level components on both hosts, i.e., video, audio or text. The discovering node will process the first message received from host 1 and then reexpress the same Interest excluding the repositoryID from host 1, which is included in the received enumeration name and based on the repository's public key, in the third step. An answer to this request will be served either by the local CCND that cached the previous answer from host 2 or by host 2 itself if the cached entry expired. We always ask for the latest list version of the repository and let cached name enumeration requests timeout quickly.

The discovery procedure is described by algorithm 1. To discover the entire available name space, the algorithm starts from the top of the name tree with the shortest possible prefix and sequentially moves down to the leaves by increasing the prefix with the discovered name components after every timeout. At every iteration and level, the requesting user receives a list of available name components from the repository. We assume that the mobile repositories are not synchronized among each other and the content collections are not known. Therefore, the requesting user has to address each repository separately excluding the IDs from previous repositories to avoid inconsistencies. If the requester does not receive an

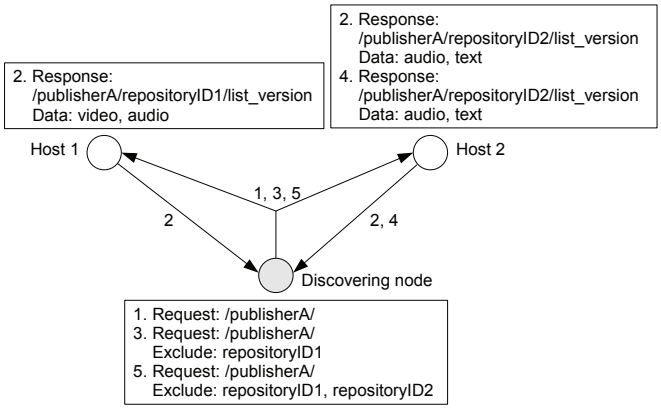

Fig. 3: Enumeration Request Discovery Sequence

Algorithm 1 Enumeration Request Discovery

```
1: p: requested starting prefix
2: L[r]: prefix list of level r, initially r = 0
3: l_i: component list of ith request
4: id_i: id of ith repository
5: function ENUMERATION(p, r)
6:     e: exclude = {}
7:     {l_i, id_i} = SEND_ENUMERATION (p, e)
8:     if timeout then
9:         return
10:    while no timeout do
11:        for all components c_j in l_i do
12:            q: prefix
13:            p + c_j → q
14:            if q ∉ L[r] then
15:                q → L[r]
16:        id_i → e
17:        i → i + 1
18:        {l_i, id_i} = SEND_ENUMERATION (p, e)
19:    for all q in L[r] do
20:        ENUMERATION(q, r + 1)
21:    return

22: function SEND_ENUMERATION(p, e)
23:    broadcast enumeration Interest message with prefix p
24:    and exclude list e containing all received ids
```

answer within a timeout period, it is assumed that no more content is available on any reachable host on that level. The algorithm procedes with the next component until receiving a timeout for all leaves of the tree.

B. Regular Interest Discovery

The Regular Interest Discovery (RID) is based on the recursive expression of regular Interest messages. The user requests an Interest and receives the first data segment in the response. Although this leads to overhead because only the content name and no data is required, it is still more efficient than retrieving all data segments in complete file downloads.

A sample sequence for the name tree of Figure 2 is shown in Figure 4. The Interest expression in the prefix */publisherA/* will reach both hosts and trigger them to answer with the first segment of a matching content object. After the reception of the first segment of /publisherA/audio/music/artistA, the discovering node requests a new name component excluding the received artistA. Since host 2 has already sent this message in step 2, this request will not be forwarded to the wireless medium but answered from the local CCND's cache. Content segments must not expire as quickly as ERD content lists because they do not correspond to temporary repository listings but to existing content objects. Interests may therefore be satisfied from cache. We only discover the human readable part of the name, i.e., neglecting versions because these may be found by a version discovery once the name is known. The procedure is described by algorithm 2.

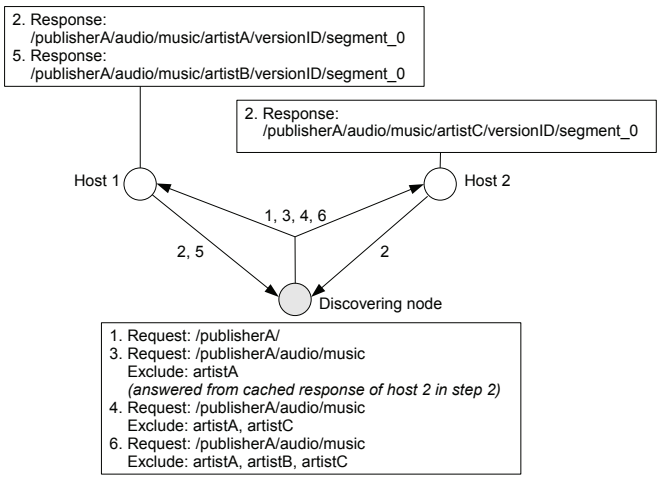

Fig. 4: Regular Interest Discovery Sequence

Algorithm 2 Regular Interest Discovery

```
1: p: request prefix
2: L: name list, initially L = {}
3: c: received content name with c[i], i = 1, ..., n components
4: s: prefix size

5: function DISCOVERY(p)
6:     e: exclude = {}
7:     s = size(p)
8:     c = SEND_INTEREST (p, s, e)
9:     if no timeout then
10:        RECURSIVE(c, s)
11:    return

12: function RECURSIVE(c, s)
13:    e: exclude = {}
14:    do {
15:    if size(c) > s+1 then
16:        RECURSIVE(c, s + 1)
17:    else
18:        c → L
19:        if size(c) == s then
20:            return
21:    c[s+1] → e
22:    c = SEND_INTEREST (c, s, e)
23:    }
24:    while(no timeout)
25:
26:    return

27: function SEND_INTEREST(b, s, e)
28:    broadcast Interest with first s components of
29:    name b and exclude list e
```

The algorithm starts by expressing a general prefix in a name space in the discovery function. After the reception of the first data segment, the mechanism knows the complete name of a content file at the leaf of the tree. By excluding the last components of the received objects, the algorithm searches only for new names. Because every transmitted packet contains only one content name, other nodes that overhear the traffic may cancel the transmission of redundant content objects. In case of a timeout, i.e., when the limit of the transmission counter has been reached, it is assumed that no additional content is available and the algorithm can move up one level by shortening the prefix by one name component and excluding this component in the Interest. The algorithm stops after a timeout at the initial discovery prefix, e.g., '/publisherA/' when all available next level components are excluded. Compared to the Enumeration Request Discovery, RID quickly finds available content objects at the leaves but requires the expression of new discovery requests for every component while ERD starts from the root and continously discovers multiple name components until receiving a content object.

V. Evaluation

A. Evaluation Tools

We implemented the Enumeration Request Discovery (ERD) and the Regular Interest Discovery (RID) algorithms described in section IV and integrated it with the CCNx source code v0.4.2. The implementations are evaluated by emulations with VirtualMesh [17]. VirtualMesh is a hybrid emulation tool that combines the real network stack and the CCNx implementation running on virtualized hosts with simulations of the wireless communication. The wireless communication is simulated by the OMNeT++ [18] network simulator using the INET framework with the default 802.11b MAC layer implementation. All CCNx messages are broadcasted using the default parameters and a static contention window of $32 \times 20\mu s = 640\mu s$. We do not consider any additional bit error models but only transmission errors due to collisions.

B. Emulation Scenarios

The algorithms are evaluated in a static setting of 5 nodes. Due to single-hop single-radio communication, we assume that all nodes can directly communicate with each other. One node, the *discovering node*, performs the discovery operation and the other nodes are hosts running repositories containing different content files. We differentiate between two basic content distributions in our evaluations:

1) *Common case*: all repositories store exactly the same content objects and
2) *Distinct case*: every content object is uniquely stored at only one of the repositories.

All content objects are named under the same hierarchy level by '/prefix/<content #>'. The discovery algorithms are implemented as applications forwarding Interests via the local face to the CCND. If the content is in the CCND's cache, it will be returned immediately without forwarding the Interest to the wireless medium, otherwise it is forwarded to other nodes and temporarily included in the PIT, as explained in subsection II-A. All received content information remains valid in the cache for the entire duration of the discovery. Before every discovery evaluation is started, all CCND caches are cleared. Both discovery algorithms will express Interests in the general prefix '/prefix/' to discover the available content objects at all repositories. Based on the reception of a discovery response, the mechanism will express the next Interest excluding already received information as explained in section IV. The discovery responses differ for ERD and RID: in case of ERD, it is the ERD content list containing the available content names at the corresponding repository. In case of RID, it is the first segment of a content object. Therefore, ERD requires the expression of one Interest per repository node to receive all content lists while RID requires the expression of one Interest per available content object. We use the default segment size of 4096 Bytes. The Interest lifetime is set to 0.5 seconds and we perform a retransmission of the same Interest after a retransmission delay of 0.6 seconds if no response has been received. The retransmission delay is slightly larger than the Interest lifetime to ensure that the existing PIT entry has expired and the retransmitted Interest can be forwarded by the local CCND. If not stated otherwise, we use a retransmission limit of two retransmissions before a timeout is assumed. Since discovery mechanisms should not overload the medium with traffic, we evaluate different delay parameters influencing the number of retransmissions and the discovery time in subsection V-C. The discovery time is defined as time until the discovering node has discovered all content names and detected a timeout. Based on these findings, we evaluate both discovery mechanisms for different numbers of content objects and distributions in subsection V-D.

C. Discovery Delays

Broadcast requests may trigger potentially many responders. Therefore, we will evaluate different delay parameters and their impact on retransmissions and duplicate content transmissions in this subsection. The evaluations apply to both ERD and RID but due to space limitations we only show the results for RID. The answering delay (AD) defines the interval [AD, 3AD] within which each host randomly selects a time to answer a request. Once scheduled, the content object stays in the senders' send queue until the answering delay is due; then it is forwarded to lower layers for transmission. A long AD may increase the individual discovery time but enables other hosts to detect concurrent responses.

Figure 5 shows the performance of RID discovery if the network comprises 40 different content objects, which are all stored on all hosts, i.e. the common case. The x-axis denotes the different answering delays in milliseconds. The figure shows the transmitted requests and received messages at the discovering node as well as the time to discover all content objects, i.e. the discovery time. As expected, the number of received content duplicates is higher with short answering delays and decreases significantly with higher values. At an AD of 10ms, the number of received duplicates is even higher

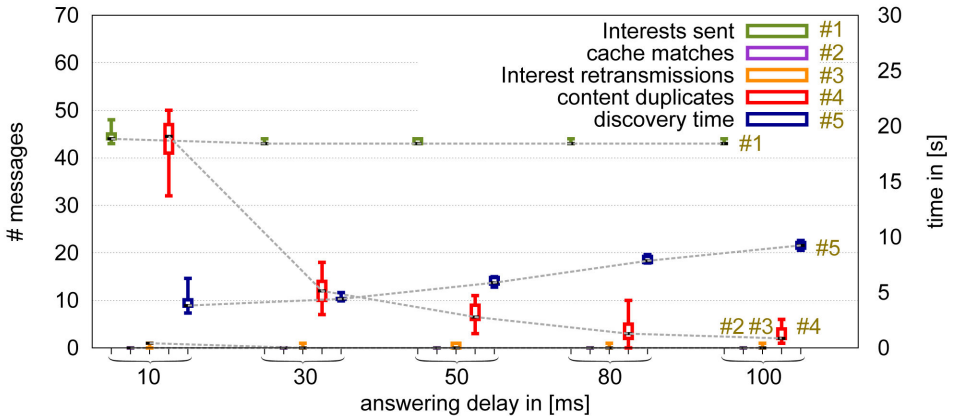

Fig. 5: RID discovery of 40 content objects in the common case

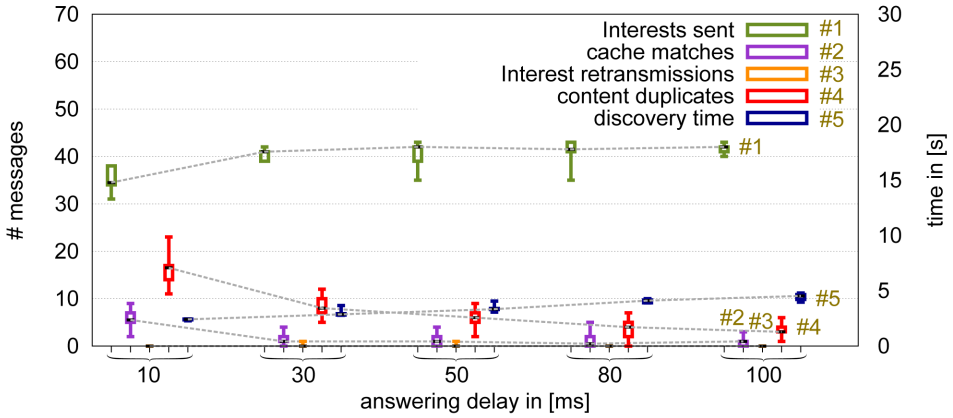

Fig. 6: RID discovery of 40 content objects in the distinct case

than the number of transmitted Interests. Because of the small AD value, the hosts schedule their content transmission almost at the same time not leaving enough time to detect and suppress concurrent transmissions. As soon as the messages are forwarded from the send queue to the lower layers, no cancelation is possible anymore. The number of required Interest retransmissions is suprisingly low: at an AD of 10ms, every Interest is retransmitted at most once. For the discovery of 40 content objects, such retransmissions occured at most five times when using an AD of 10ms and at most once when using an AD of 30ms or higher. Since all content objects are stored on all hosts, every Interest will trigger the same answer from all hosts. Given that the discovering node's cache is empty when starting the discovery operation, no Interest requests can be matched from the local CCND's cache. Therefore, to discover 40 objects, the discovering node transmits at least 43 Interests: 40 Interests to discover the objects and 3 additional Interests to detect a timeout using the retransmission limit of 2.

Figure 6 shows the results for the discovering node when using RID discovery of 40 content objects stored uniquely at different nodes, i.e. distinct case. Since every content object is only stored at one node, every request with the general */prefix/* will trigger different responses from the repositories. Since the transmitted content is not the same, the hosts do not cancel their scheduled content transmission in case of overheard transmissions because they cannot uniquely relate their content transmission to the same Interest. Therefore, the discovering node's CCND may receive multiple content objects per Interest but only one content object per Interest is forwarded to the discovering application. Subsequent Interests may then be matched from CCND's cache and may not be transmitted over the wireless medium anymore but the percentage of cache matches is quite low as Figure 6 shows. The discovery time for RID is approximately halved compared to the common case since different hosts may reply to the same Interest with different content objects resulting in a faster discovery.

Surprisingly, although all content objects are uniquely stored at only one host, content duplicates occur for all AD values in Figure 6. The reason for that is the fact that subsequent Interests are expressed immediately after the reception of a content object resulting in duplicates in case of unsynchronized repositories. We illustrate the problem with the help of Figure 7 where two hosts store different content objects. An Interest in the general prefix '/publisherA/' triggers different responses from both hosts. While host 1 answers

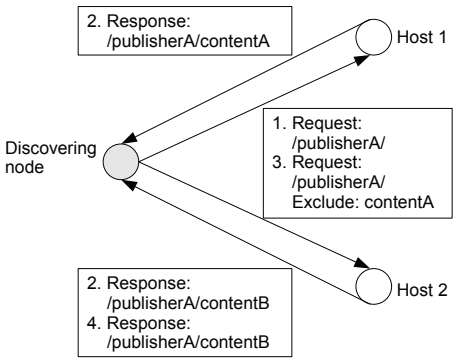

Fig. 7: Content duplicates due to unsynchronized repositories

with 'contentA', host 2 may schedule the transmission of 'contentB'. If the discovery node would express a subsequent Interest immediately after receiving 'contentA' from host 1 but before receiving 'contentB' from host2, it would address host 2 twice. If host 2 has already scheduled the content, i.e. removed from it's send queue and forwarded to lower layers, it cannot remember the previous transmission and sends a duplicate content. Therefore, we apply an additional Interest transmission delay (TD) at the discovering node. Whenever a discovery response is received, the discovering node delays the transmission of the subsequent Interest by TD. We set $TD = 2 \times AD$, i.e. the maximum time difference between two content transmissions. This enables the reception of answers from other nodes before the expression of the next subsequent Interest. If different content objects are received, the Interest may be satisfied from the local CCND's cache. Otherwise, it is forwarded to the wireless medium.

Figure 8 shows the differences in transmitted Interests and local cache matches when applying TD=2AD. It can be seen that if TD is applied, three times more Interest requests can be satisfied from the cache and, therefore, fewer Interests have to be transmitted over the wireless medium. In Figure 9 the differences regarding received content duplicates and discovery time are shown. For TD=2AD we can avoid the reception of any duplicates relieving the wireless medium from unnecessary transmissions. The discovery time increases only moderately for small AD values.

In the following evaluations, we set AD=50ms and TD=2AD. This avoids all duplicates in the distinct case and results in a low number of content duplicates in the common case. It is not possible to avoid duplicates in the common case completely because two senders may always select the

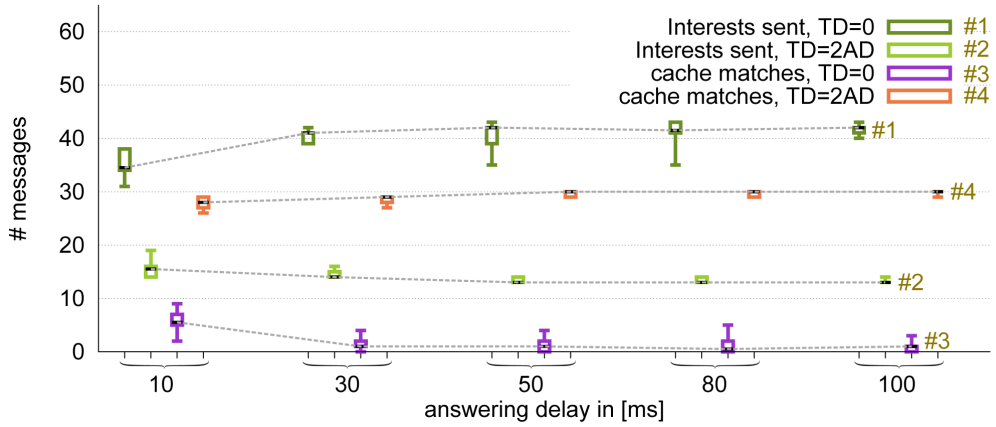

Fig. 8: RID discovery with TD=2AD vs. TD=0 in the distinct case

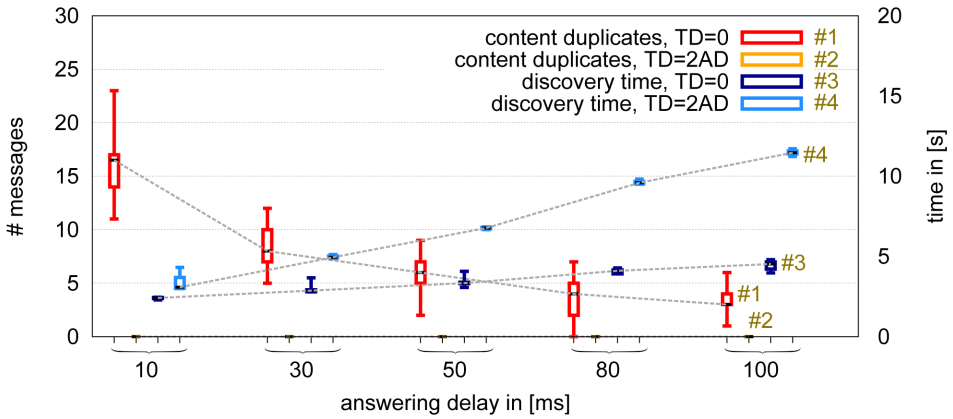

Fig. 9: RID discovery with TD=2AD vs. TD=0 in the distinct case

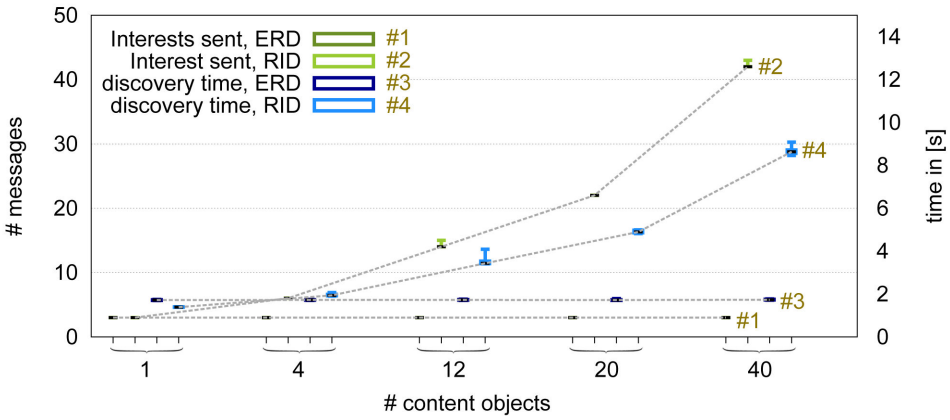

Fig. 10: Comparison between ERD vs. RID in the common case.

same answering time with a certain probability depending on the AD length. We observed in our evaluations that AD values above 50ms do not reduce the number of received duplicates significantly but result in a much larger discovery time. Since in all our evaluations, retransmissions occured very infrequently and at most once per Interest, we set the retransmission counter limit to 1. Although higher network congestion levels may require higher retransmission limits to discover content, it may result in even higher congestion aggravating the traffic situation. In highly congested networks, we can therefore consider the corresponding content objects as (temporarily) unavailable.

D. Enumeration Request vs. Regular Interest Discovery

In this subsection, we compare Enumeration Request Discovery (ERD) and Regular Interest Discovery (RID) with respect to time and number of transmitted and received messages. We use an answering delay of AD=50ms and set TD=2AD. The retransmission counter limit is set to 1 retransmission resulting in 2 unresponded Interest transmissions before a timeout is detected.

The efficiency of ERD and RID is evaluated in the same 5-nodes scenario as described in subsection V-B. We consider different numbers of content objects and content distributions, i.e. common and distinct, in the network. All hosts either comprise 1, 4, 12, 20, or 40 content objects. Figure 10 shows the difference in number of transmitted Interests and discovery time if all hosts have the same content. The x-axis denotes different numbers of content objects. If only one content object is available, RID is more efficient, since it cannot learn anything new after the first request and the discovery stops quickly. On the contrary, ERD requests the ERD content list from all hosts. Only after checking all names on all lists, the discovering node can be certain to have received everything. This requires multiple discovery requests. However, due to the general ERD request prefixes, subsequent requests for content lists may be answered from the cache. The number of ERD requests does not depend on the number of content objects on hosts but the number of hosts in the vicinity. Therefore, the number of ERD requests and the discovery time is constant in our setting for all content configurations. On the contrary, the number of transmitted Interest messages and discovery responses increase significantly for RID with increasing number of content objects. If all hosts store the same content objects, the requester has to express an Interest for every single content object. Since RID requests ask for the first data segment, the transmission of the corresponding responses requires considerably more time.

The figure showing the results in the distinct case is omitted due to space limitations. If only one content object is stored in the distinct case at one node, only this node will respond to requests. Therefore, ERD performs similar to RID: only one request needs to be transmitted. However, RID performance degrades with increasing number of discovered content objects due to the increased number of discovery messages similar to the common case. As observed in subsection V-C, the same Interest request may trigger different answers from different hosts resulting in approximately 70% fewer transmitted Interests compared to the common case. This results in a slightly reduced discovery time because many packets may be satisfied from cache and only every fourth request needs to be transmitted.

The Enumeration Request Discovery (ERD) shows good performance in our evaluations, because it is independent of the number of content objects. However, the approach depends on the number of nodes in the network that store the requested content. Therefore, the approach may be inefficient in mobile scenarios with many nodes. Compared to RID, the ERD content lists of all repositories need to be processed and accumulated to know which content names are available. Therefore, if all hosts store the same content objects, ERD requires all nodes to request and process all content lists without learning something new. RID is more efficient to detect small differences in collections, because it can ask specifically for new content. Redundant information can already be excluded in the header to avoid duplicates. RID is also faster in finding a content object in a highly structured name space with many name components. In our evaluations, we considered a flat

name space where ERD can perfom well. In a more structured name space ERD would require subsequent traversing through all name components until reaching the content objects. Therefore, the combination of both approaches may be promising. An initial RID request may quickly find the full name of a content object. By expressing an ERD request with the prefix of the received content object, a list containing other objects may be received from one repository. Instead of reexpressing other ERD requests, the requester may express RID requests excluding the learned information from the received ERD content list.

Discovery information from RID requests may stay for a longer time in the cache compared to ERD content lists since they correspond to existing content objects but not to temporary repository listings. Therefore, with RID discovery multiple nodes may collaborate and benefit from each others' discovery operations.

Both discovery mechanisms are not optimized yet and modifications are required to increase their efficiency. For example, ERD content lists may be identified by the hash of their content names instead of the repository identifiers to avoid multiple transmissions with the same information. On the other hand, Interests may be extended by a discovery flag, avoiding the transmission of the entire data object.

VI. Conclusions and Future Work

Discovery of available names is very important in mobile CCN to learn what content is available. Users require this information to retrieve content in subsequent transmissions. We described two methods for content discovery: ERD is based on name enumeration requests and RID on regular Interests. The discovery algorithms target the wireless broadcast environment. Since wireless broadcast communication is unreliable and no MAC layer acknowledgments are available, discovery mechanisms have to account for occasional loss and collisions. Therefore, we included a retransmission counter that initiates a retransmission if no information is received within a timeout period. Evaluations showed that a retransmission limit of only one retransmission is enough to detect timeouts in our scenario. This can be interpreted as additional information being temporarily unavailable. In case of very congested networks, more collisions may occur resulting in higher counter limits but retransmitting discovery requests may even aggravate the situation. Avoiding collisions and received duplicates is another important factor for discovery. Evaluations showed that delaying the transmission of content objects helps reducing collisions and duplicate transmissions but this is not enough. In case of unsynchronized repositories, delaying the subsequent expression of discovery Interests may reduce the number of transmitted Interests and avoid received duplicates. ERD shows constant discovery performance in static settings independent of the number of contents in the network while RID decreases with the number of content objects. On the downside, ERD transmits a request to every node in the network and will therefore perform worse in mobile dynamic networks with many nodes or multiple nodes that have only a few common content objects. Since ERD responses carry the discovery information in the data, it is not possible to detect and suppress duplicate transmissions. RID holds the information in the name and hosts are therefore able to suppress duplicate transmissions. RID can efficiently and quickly find only a few content names in a dynamic network. As part of our future work, we target to extend our algorithms to multi-hop communication by developping adequate suppression mechanisms based on overheard traffic.

Acknowledgments

The work presented in this paper was partially supported by the Swiss State Secretariat for Education and Research under grant number C10.0139.

References

[1] M. Caesar, T. Condie, J. Kannan, and K. Lakshminarayanan, "ROFL: Routing on Flat Labels," in *ACM SIGCOMM*, Pisa, Italy, September 2006, pp. 363–374.

[2] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A Data-Oriented (and Beyond) Network Architecture," in *ACM SIGCOMM*, Kyoto, Japan, August 2007, pp. 181–192.

[3] M. Srel, T. Rinta-aho, and S. Tarkoma, "RTFM: Publish/Subscribe Internetworking Architecture," in *ICT-MobileSummit*, Stockholm, Sweden, June 2008, pp. 1–8.

[4] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Network Named Content," in *5th ACM CoNEXT*, Rome, Italy, December 2009, pp. 1–12.

[5] M. Meisel, V. Pappas, and L. Zhang, "Ad hoc networking via named data," in *5th ACM MobiArch*, Chicago, USA, September 2010, pp. 3–8.

[6] B. Cohen, "Incentives Build Robustness in BitTorrent," in *1st P2PEcon*, Berkely, USA, June 2003, pp. 1–5.

[7] CCNx Project. [Online]. Available: http://www.ccnx.org

[8] M. Varvello, I. Rimac, U. Lee, L. Greenwald, and V. Hilt, "On the Design of Content-Centric MANETs," in *8th WONS*, Bardonecchia, Italy, January 2011, pp. 1–8.

[9] S. Y. Oh, D. Lau, and M. Gerla, "Content Centric Networking in Tactical and Emergency MANETs," in *IFIP Wireless Days*, Venice, Italy, October 2010, pp. 1–5.

[10] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu, "The broadcast storm problem in a mobile ad hoc network," in *5th ACM/IEEE MobiCom*, Seattle, USA, August 1999, pp. 151–162.

[11] M. Meisel, V. Pappas, and L. Zhang, "Listen First, Broadcast Later: Topology-Agnostic Forwarding under High Dynamics," in *ACITA*, London, UK, September 2010, pp. 1–8.

[12] G. Karlsson, V. Lenders, and M. May, "Delay-tolerant Broadcasting," *IEEE Transactions on Broadcasting*, vol. 53, no. 1, pp. 369 – 381, March 2007.

[13] J. Su, J. Scott, P. Hui, J. Crowcroft, E. D. Lara, C. Diot, A. Goel, M. H. Lim, and E. Upton, "Haggle: seamless networking for mobile applications," in *9th UbiComp*, Innsbruck, Austria, September 2007, pp. 391–408.

[14] V. Lenders, G. Karlsson, and M. May, "Wireless Ad Hoc Podcasting," in *4th IEEE SECON*, San Diego, USA, June 2007, pp. 273–283.

[15] F. Liu and G. Heijenk, "Context Discovery using Attenuated Bloom Filters in Ad-Hoc Networks," in *4th WWIC*, Bern, Switzerland, May 2006, pp. 13–25.

[16] D. Eppstein, M. T. Goodrich, F. Uyeda, and G. Varghese, "What's the difference? Efficient Set Reconciliation without Prior Context," in *ACM SIGCOMM*, Toronto, Canada, August 2011, pp. 218–229.

[17] T. Staub, R. Gantenbein, and T. Braun, "VirtualMesh: an emulation framework for wireless mesh and ad hoc networks in OMNeT++," *SIMULATION*, vol. 87, no. 1-2, pp. 66 – 81, January 2011.

[18] A. Varga, "The OMNeT++ Discrete Event Simulation," in *ESM*, Prague, Czech Republic, June 2001.

Chapter 2

Automatisation Framework to Support Evaluations

Various parameters may have an influence on content discovery mechanisms. To facilitate evaluations, an automatisation framework based on Linux Shell and Perl scripts has been developed. Subsection 2.2 provides a walkthrough of this framework.

The framework essentially ensures initialization, configuration, and reset of emulation scenarios and network nodes. Automatisation facilitates evaluation and enables reproducible results. Arguments are lists containing different values used for evaluation.

2.1 VirtualMesh Platform

The emulation tool VirtualMesh[1] consists of two parts: emulated nodes, which run real communication software, i.e., CCNx[2], and a model, which simulates the wireless communication between nodes.

For this purpose, two dedicated servers are put up in operation (as shown in Figure 2.1): 'Real' nodes are virtualized by the Xen Hypervisor on *vm-host02*, whereas *vm-model02* hosts the simulation model to which the virtual machines register.

Both servers are configured in the same way: they feature two network interfaces, *eth0* and *eth1*. The latter is used for connections to the Internet for remote access. The first interface, eth0, is used to interconnect the machines directly. On vm-host02, this interface also functions as bridge for the hosted virtualized instances. All simulation traffic, i.e., the encapsulated simulated traffic, is exchanged over this bridged link between the virtualized instances and simulation server vm-model02 in order to decrease simulation latency to a minimum.

The Linux distribution used on the servers as well as on the virtualized node instances is *Debian Squeeze* (version 6.0.5). It is set up without a graphical user interface for performance reasons.

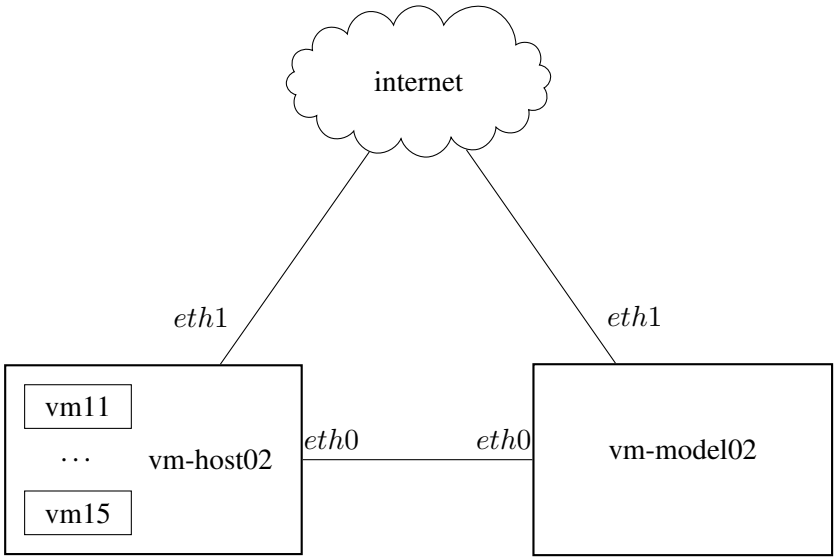

Figure 2.1: Emulation Environment: Overview

2.2 Stages of the Automatisation Framework

As illustrated in Figure 2.2, the framework is divided into following stages: preparation of the simulation model and discovery node, discovery execution followed by evaluation data gathering, reconfiguration of repository nodes and discovery settings, and finally teardown of model and nodes. These are described in the following subsubsections to provide an overview of the involved processes.

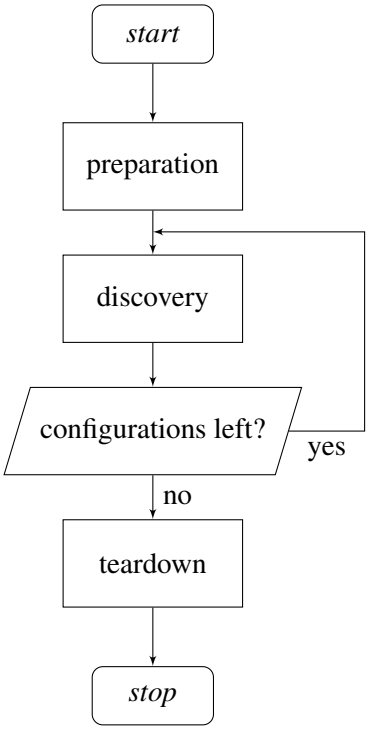

Figure 2.2: Flow chart of the evaluation framework: Overview

2.2.1 Preparation: Initialize Simulation Model and Discovery Node

Figure 2.3 presents all preparation steps that are used. First, all listed, running VMs on the virtualization server vm-host02 are shut down. The current VirtualMesh emulation instance - if existing - is stopped on the corresponding server vm-model02.

Since the network topology is static throughout all evaluations, the model is initialized with the *Star-formation-50m* as illustrated in Figure 2.4: Four outer nodes (designated repositories) are placed evenly 50m apart and around a node in the center (designated discovery node). All nodes are in transmission range of each other.

To further improve the stability of the model, the corresponding process on host *vm-model02* is assigned a higher priority.

Next, the discovery node instance is set up, which involves the following stages: First, the virtual machine is instantiated from a shared Xen virtual machine image which differs only in

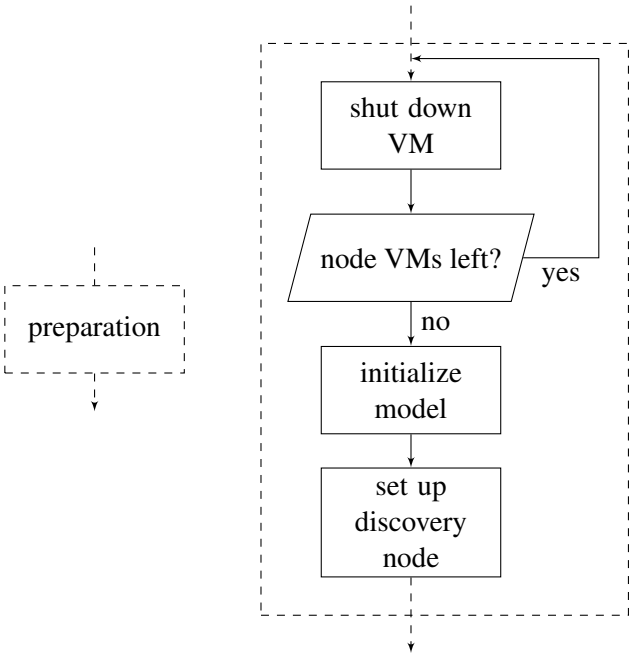

Figure 2.3: Flow chart of the evaluation framework: Preparation

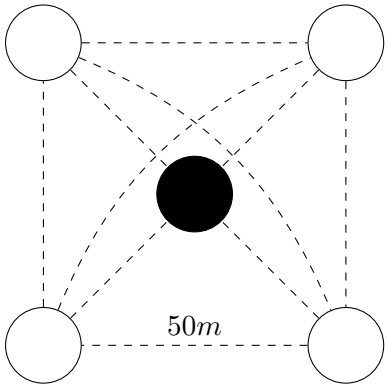

Figure 2.4: Network topology Star-formation-50m

the host name, e.g., *vm11*, and the IP address, e.g., *10.0.0.11*.

Since the single Ethernet interface on the virtual machine is already used for remote access and connection to the model's host itself, a virtual wireless device dedicated to simulated traffic needs to be created. Messages sent to this interface are encapsulated and transferred to the simulation model using the Ethernet interface. The encapsulated traffic is transmitted and exchanged via the simulated network. A receiving node can then obtain the original data on its virtual interface after decapsulating it. To differentiate between physical and virtual networks, the virtual interface uses another private IP address than the physical interface.

Communication is based on multicast only, using the IANA registered multicast IP address *224.0.23.170*, which is routed in the simulated network only, i.e., via the virtual interface.

The network set-up is completed by registering the emulated node with the simulation model. This enables simulated communication between emulated nodes via their representations in the simulation model, using the virtual wireless interface.

As a last step of the preparation phase, available patches are applied and the newest revision of binaries such as Java executables are deployed where applicable.

2.2.2 Configuration: Set Up Repository Nodes and Adjust Discovery Settings

Evaluations are started with different configurable parameters in subsequent phases as illustrated in Figure 2.5.

Repository Count

The first parameter that is considered is the number of repositories in the evaluation. If multiple evaluations are scheduled with a varying number of repository nodes, the repository counts need to be organized in ascending order. This way, existing repository nodes can be reused and supplemented with new nodes.

For example, let *"01;04"* be the repository counts to evaluate: In the first round, one repository is set up and evaluated. The next configuration uses four repositories, i.e., the already existing one from before and three new ones.

Repository nodes are set up similarly to the discovery node, i.e., the single node available in every configuration designated to execute the content discovery. The set-up is described in detail in subsection 2.2.1. Connection of repository nodes to the simulation model is tested by pinging the IP address of the discovery node bound to the model, i.e., the IP address assigned to the virtual interface.

Content Distribution Type

The second parameter is the content distribution type. We define two types: common and distinct.

In the common case, all repositories, i.e., the repository application on the corresponding nodes, offer synchronized content. This means that all repositories contain exactly the same content objects. Repositories use an image called the *repo file*, which persists all repository related informations such as repository ID and offered content objects, i.e., binary data including

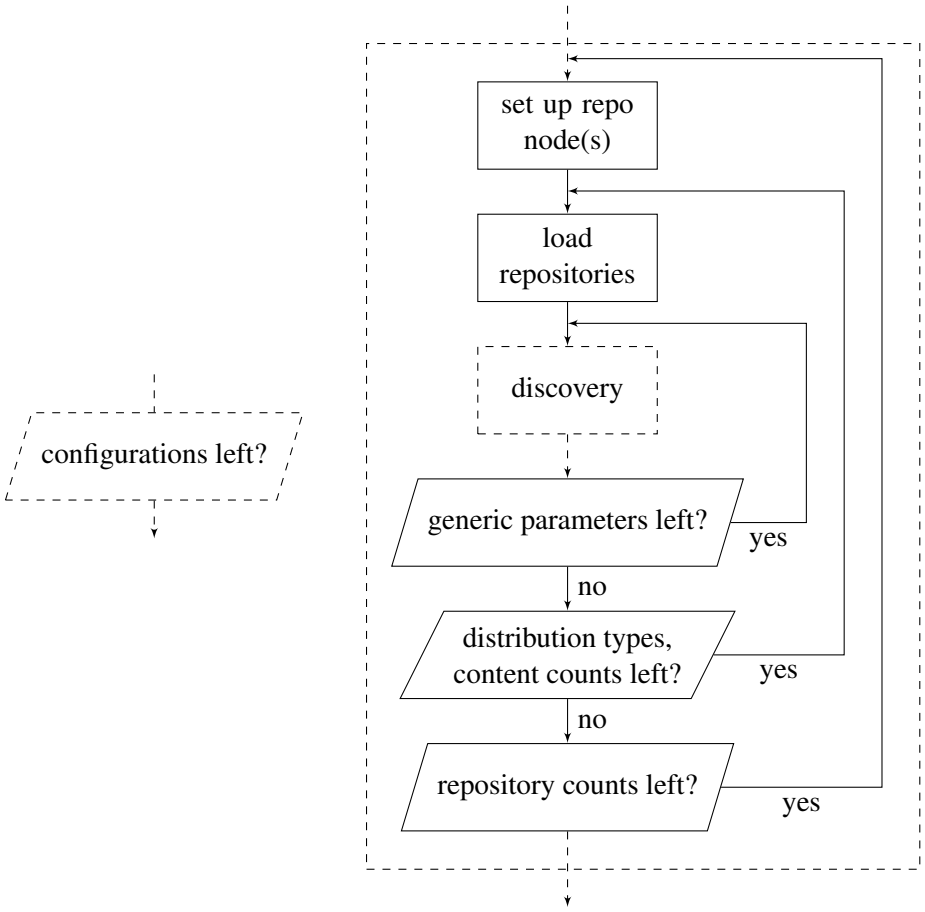

Figure 2.5: Flow chart of the evaluation framework: Configuration

CCNx meta data like content names, signatures and so on. By starting all repositories with the same, distributed repo file, content is ensured to be entirely common.

In the distinct case however, the content objects are unique throughout all possible repositories, thus, every content is stored uniquely at only one repository. The content objects, numbered in the form *c1*, *c2*, etc. and backed by generated binary data, are distributed subsequently to different repositories.

Content Count

The third parameter is the content count, which defines how many files are included in the repositories. File sizes are uniformly distributed but do not have a significant impact on discovery because only the content name and possibly the first data segment are required.

Content objects are either made available on all repository nodes simultaneously, i.e., common case, or distributed evenly among them, i.e., distinct case. The content count should be a multiple of the repository nodes count in the *distinct* case so that the same number of content objects can be stored in each repository.

Only when both properties (distribution type and content count) are defined, the repository can be initialized and loaded.

Other Parameters

The remaining parameters are independent of the repository configuration and only affect the discovery and the ccnd configuration.

Forwarding of content on multicast faces can be delayed by a random time interval to reduce duplicated transmissions. The content-send delay, i.e., *answering delay AD* defines the range of these time intervals, i.e., the maximum delay and is a ccnd configuration parameter.

Parameters concerning the discovery are described in subsection 2.2.3: Interest lifetime, retransmission threshold, retransmission delay, transmission delay and discovery algorithm.

2.2.3 Discovery: Reset Nodes, Execute Discovery and Gather Evaluation Data

As shown in the overview in Fig. 2.5, discovery follows the repository setup.

Each evaluation run consists of the preparation for and the execution of the discovery application itself. Data is gathered after all configured runs have been finished. This process is visualized in Fig. 2.6.

Reset CCNx

To ensure that each evaluation run is independent from previous runs, the cache needs to be cleared after each run. This is performed by stopping and restarting the ccnd on all nodes.

Several configurations are provided to the nodes: the list of repository nodes is used by the discovery node in its application's context to control the repositories' data gathering. The parameter *answer delay AD* is exported as CCNx's environment variable *CCND_DATA_PAUSE_MICROSEC* that is read by the ccnd on start-up.

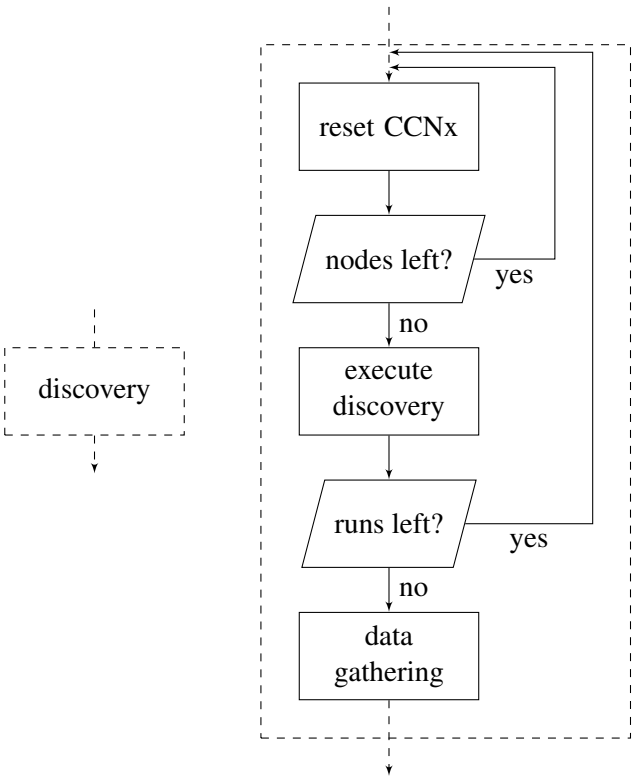

Figure 2.6: Flow chart of the evaluation framework: Discovery

The location of the ccnd log file and the verbosity level are also specified with exported CCNx environment variables *CCND_LOG* and *CCND_DEBUG* respectively. As of now, however, they're not configurable through parameters but rather need to be (statically) specified in a special file on the host running the framework.

If the verbosity level is set to 30, it means to "include Interest messages (2) and details (16), content messages (4) and matching details (8)". Inherently, all basic messages are included with any non-zero level. If discovery times are evaluated, logging of debug information should be avoided by setting the debug level to zero.

After the debugging level is set, the faces need to be configured with the ccnd configuration utility *ccndc*. Discovery is performed via multicast and, therefore, a multicast face needs to be configured. It uses the IP address 224.0.23.170 and port number 59695.

Execute Discovery

A number of introduced configuration parameters affect the discovery algorithm and are thus provided as application parameters. The *Interest lifetime (IL)* defines, how long an expressed Interest is valid and kept pending. After it has timed out, the application can retransmit the Interest message up to a configured number of times as defined by the *retransmission threshold*. The *retransmission delay (RD)* defines the delay between subsequent Interest expressions. All Interest expressions can be additionally delayed through the *transmission delay (TD)* parameter. Content is discovered using the configured *discovery algorithm*.

Furthermore, the *content count* is supplied as a parameter so that the application knows if it has discovered all content names. This information is only used for sanity check since an application assumes that it has discovered all content objects after the number of retransmissions of the same Interest, due to timeouts, reaches the threshold.

Immediately before and after the discovery method call, the application requests all nodes (including its host) to store ccnd log and status informations. The log is reset before and preserved after the discovery, thus only the relevant logging during the method call is preserved. Snapshots for the status of all faces before and after the discovery are preserved as well. Numbers relevant to the method call are obtained as differences between these records. Furthermore, additional discovery run informations such as the discovery time, total number of discovered content names, number of retransmissions etc. are logged at the discovery node.

Data Gathering

As illustrated in Figure 2.6, when all evaluation runs have finished, ccnd log and status files, as well as the discovery run information are collected on all nodes. Data is collected per configuration run and then aggregated.

The raw data is stored on vm-host02 in a specific folder hierarchy where names correspond to used evaluation parameters. Figure 2.7 shows a sample directory hierarchy.

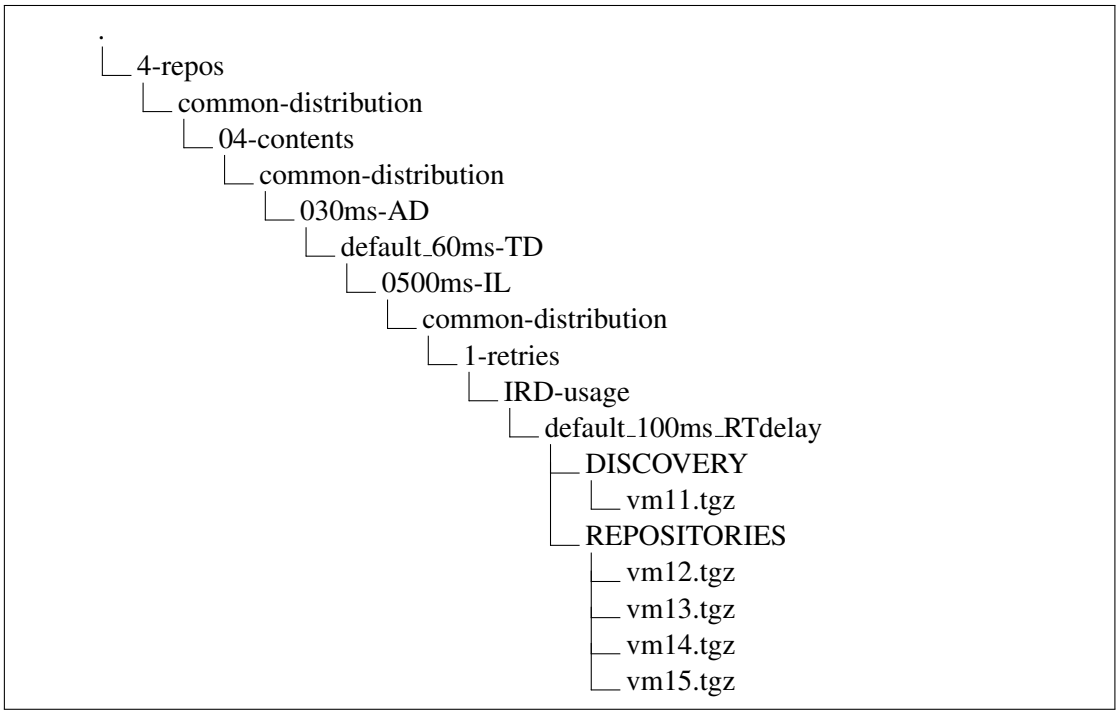

Figure 2.7: Example directory tree of filed evaluation data

Chapter 3

Evaluation Challenges

As examinations of the involved mechanisms have shown, both the emulation tool VirtualMesh and the open-source networking implementation CCNx need to be modified and extended in order to render the intended evaluations possible.

3.1 VirtualMesh's Support for Multicast

One of the first hurdle to overcome was unexpected and concerning the emulation framework's abilities to simulate multicast networking. Ethernet frames transmitted via the multicast face were never transmitted and consequently never received by other nodes.

Graphically debugging the simulation framework OMNeT++[3], which VirtualMesh is based on, revealed problems regarding multicast communication in the Inet framework that provides the link layer to OMNeT++.

Nodes in the VirtualMesh simulation do not join the multicast group and are thus unable to receive the multicast Ethernet frames sent to the multicast address, e.g., the IANA registered multicast address for CCNx.

To enable multicast communication, the class *Ieee80211Mac* in the inet framework representing the data link layer needed to be modified. Transmission has been rendered possible by sending all multicast packets to the broadcast address. Listing 3.1 shows the adaptation in code released on July 23, 2010. The called method *isMulticast()* has a bug in this code release, which prevents it to properly detect multicast MAC addresses. The patch is documented and included on the attributive digital medium.

```
--- a/linklayer/ieee80211/mac/Ieee80211Mac.cc    2010-07-22 20:10:17.000000000 +0200
+++ b/linklayer/ieee80211/mac/Ieee80211Mac.cc    2012-03-29 22:40:04.000000000 +0200
@@ -253,6 +253,11 @@

     ASSERT(!frame->getReceiverAddress().isUnspecified());

+    if(frame->getReceiverAddress().isMulticast()) {
+        frame->getReceiverAddress().setBroadcast();
+        //frame->setReceiverAddress("ff:ff:ff:ff:ff:ff");
+    }
+
     // fill in missing fields (receiver address, seq number), and insert into the queue
     frame->setTransmitterAddress(address);
     frame->setSequenceNumber(sequenceNumber);
```

Listing 3.1: Ieee80211Mac.cc patch

An alternative approach would require a modification at the receiver to detect multicast packets. This hasn't been followed due to IGMPv2 group membership messages being exchanged as a result, which in mobile CCNx communication are of little use.

3.2 Multi-Hop Communication with CCNx

The focus of the content-centric networking paradigm as elaborated by Van Jacobson et al.[4] lies on wired networks with multiple interfaces and unicast faces. Whereas the implemented suppression mechanisms are applicable to single-hop networks, such as in access point based wireless mesh networks, they may not be applicable in multi-hop ad-hoc networks.

Interest messages are forwarded based on the Forwarding Information Base *FIB*, which can be configured using the daemon configuration tool *ccndc*. Content messages that are received in response to Interest messages travel back to the requester on the face the Interest was received. This information is stored in the pending Interest table *PIT*. If a forwarded Interest doesn't get answered within its lifetime, it is removed from the PIT.

Interest messages cannot be forwarded on the face they were received from. By adding a second multicast face, which differs in port numbers only, Interest messages can propagate on alternating faces, thus enabling multi-hop communication.

We will explain this with an example in Figure 3.1. Transmitted Interest messages are denoted by solid arrows, content messages by dashed ones. Message flow in the upper half is forwarded on face 1 and in the lower half on face 2. In this scenario, node a receives an Interest message on face 1. If it is unable to answer the request, it can forward the Interest message on face 2. Node b, which receives the message on face 2, can forward it on face 1 and so on. On the way back, responding content would arrive on face 1 at node b and leave it on face 2, corresponding to the faces where the Interests were received and forwarded.

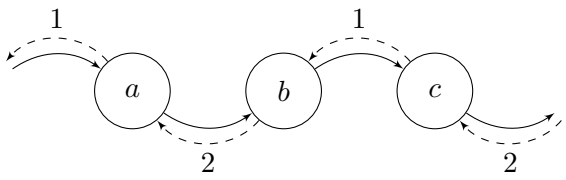

Figure 3.1: Multi-hop communication using two multicast faces

However, multi-hop communication via two configured multicast faces is tricky in case of mobility. Consider the changed message flow as illustrated in Figure 3.2. Node a has an entry in the PIT for the defined IL as in the previous example. While waiting for matching content, node b moves out of range and node c comes into range of node a. Node c receives matching content on face 2 and forwards it on face 1 according to its PIT entry. The requested content hence arrives on face 1 at node a instead of face 2 and can therefore not be forwarded to face 1 because it is already received from face 1.

Therefore, the content would be cached at node a but not further forwarded. A requester would need to re-express the Interest in the content and send it to node a to retrieve the cached

content from a.

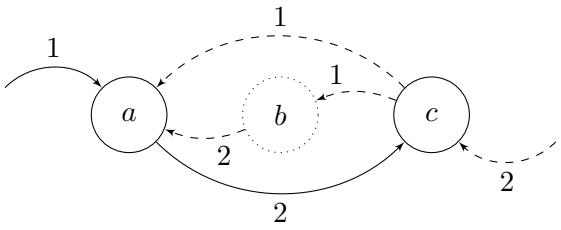

Figure 3.2: Multi-hop communication using two multicast faces: deadlock

Multi-hop ad-hoc communication would require adaptation of existing suppression mechanisms, which are beyond the scope of this thesis. Therefore, we focus only on single-hop communication.

3.3 Adapting the Interest Lifetime

The Interest lifetime *IL* is a property of the Interest message and can therefore be adapted in every Interest message.

An unanswered Interest message is removed from the PIT after the IL time has passed. Received Interests are not forwarded in case of existing PIT entries to avoid duplicate Interest transmissions. Thus, the IL has a direct effect on when Interests can be reexpressed.

Especially when discovering content in one-hop distance, the default IL of four seconds is too long: if content is available, most Interests are answered within 10ms on average. Modifying the IL to perform faster reexpressions can considerably reduce discovery times.

With CCNx release 0.8.1 as of October 9th, 2013 and any previous releases, the Java API does not support the encoding of the IL bytes set, leaving it at the default value as logged by the ccnd. This is due to the Interest class instance method *encode(XMLEncoder encoder)*, which is called on serialization of the Interest message and encodes all Interest properties but the IL. Customizing the IL through the API has thus no effect on the actual Interest lifetime. By extending the Interest encoding, the customized IL is considered and read by the ccnd. Listing 3.2 shows the patch based on CCNx release 0.4.2.

```
--- a/ccnx-0.4.2/javasrc/src/org/ccnx/ccn/protocol/Interest.java        2011-12-07 21:03:35.000000000 +0100
+++ b/ccnx-0.4.2/javasrc/src/org/ccnx/ccn/protocol/Interest.java        2012-05-04 05:46:21.046021391 +0200
@@ -552,6 +552,9 @@
                if (null != nonce())
                        encoder.writeElement(CCNProtocolDTags.Nonce, nonce());

+               if (null != interestLifetime())
+                       encoder.writeElement(CCNProtocolDTags.InterestLifetime, interestLifetime());
+
                encoder.writeEndElement();
        }
```

Listing 3.2: Interest.java patch

Chapter 4

Summary and Future Work

Two discovery algorithms have been developed and evaluated.

The Enumeration Request Discovery *ERD* uses Interests with a specialized prefix, appending a command marker to the prefix. These Interest messages are answered with lists of next name components for a given prefix. The repository ID supplied in the response's content name identifies lists from different repositories and can be used to exclude the retrieved list in future requests.

ERD is advantageous in rather static environments, with highly distinct content distribution among nodes. In such scenarios, the content awareness increases rapidly with subsequent requests. Since retrieved lists of matching name components are supplied as data, already retrieved enumerations can be excluded only using the sender repository's ID and list version. However, it is not possible to know prior to receiving the list, whether it contains new information.

The Regular Interest Discovery *RID* uses Interests with non-specialized prefixes, which retrieve the first data segment of a matching content object. RID proves to be useful in potentially dynamic environments, where content is 'popular' among repositories, i.e., available on numerous nodes. These popular content objects are discovered only once and can then be excluded effectively in any future requests, independent of the number of copies in other repositories. While discovery only operates with content names, regular Interests retrieve the first data segment as well, which can be considered as overhead because it is not required to discover content names.

In future work it may be beneficial to include the list's hash in the response's content name, so that responses to ERD requests from different repositories with the same content can be omitted. Otherwise, two synchronized repositories could respond with the same list of content names but it would be treated as different content because of a different repository ID. In this case, when using a hash to identify content lists, a repository may not respond to a request with exactly the same list. Such enumerations with meaningful content names could be retrieved by appending a different command marker to the prefix.

The option to skip the inclusion of data in responses (possibly using additional flags in the Interest message) may decrease data overhead and network load of discovery when using RID, but it requires the modification of Interest messages.

Combining both discovery approaches may be beneficial. For example, a regular Interest for prefix *ccnx://ccnx.org/data/music/* may retrieve a content object named *[. . .]/Michael Jack-*

son/Greatest Hits/Thriller. Assuming that the found track is included in a music album, an ERD request for the prefix *ccnx://ccnx.org/data/music/Michael Jackson/Greatest Hits/* could retrieve a list of available file names on the answering repository, i.e., available tracks for this album. Other missing tracks could be discovered from other repositories via RID requests excluding all discovered tracks from the ERD request.

Bibliography

[1] T. Staub, R. Gantenbein, and T. Braun, "VirtualMesh: An Emulation Framework for Wireless Mesh and Ad-Hoc Networks in OMNeT++," *SIMULATION*, vol. 87, pp. 66–81, 2011.

[2] Project CCNx. [Online]. Available: http://www.ccnx.org

[3] OMNeT++. [Online]. Available: http://www.omnetpp.org

[4] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," pp. 1–12, 2009. [Online]. Available: http://doi.acm.org/10.1145/1658939.1658941