Content-Discovery in Wireless Information-Centric Networks

Bachelorarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Arun Sittampalam
2015

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

Abstract

Content-centric networking (CCN) [1] is a new networking approach in which forwarding is based on content names instead of host identifiers. The main goal is to provide a communication infrastructure that is better suited to content distribution and mobility. Through caching and broadcast communication, CCN is more resilient to disruptions and failures. CCN supports infrastructure-based communication, e.g., in wired networks or via wireless access points, but also supports opportunistic wireless communication between devices. Therefore, CCN can work in isolated islands where the network infrastructure, e.g., access points, is not operational. To retrieve content, nodes need to have knowledge of available content names. For example, in a disaster scenario, requesters can not retrieve naming information from central repositories but need to discover content that is reachable within their wireless transmission range.

We base our work on the CCN architecture, which is pull-based, i.e., requesters need to transmit Interests to receive content. Interests are forwarded and matched to content objects based on longest-prefix matching. In this thesis, we design and evaluate different algorithms for opportunistic content discovery based on wireless one-hop broadcast. The main idea behind content discovery is the following: requesters transmit wireless broadcast requests in certain prefixes to retrieve matching content via longest-prefix matching from any node in transmission range. This is in contrast to traditional opportunistic networking approaches, where nodes perform device discovery to find neighbors before they connect to them.

RID (Regular Interest Discovery) and ERD (Enumeration Request Discovery) are based on a previous bachelor thesis [2]. RID discovers the name tree in a depth-first manner by retrieving the first segment of a content object while ERD discovers it in a breadth-first manner (name components on each level of the name tree). In this work, we extended RID and ERD by a resume capability, which is useful in case of mobility where it takes some time until a requester sees all content sources because of intermittent connectivity. Furthermore, we proposed and implemented LFD (Leaves First Discovery), which uses elements of RID and ERD. The main benefit of LFD compared to ERD is the quicker content discovery in structured namespaces, i.e., less browsing is required. Compared to RID, LFD results in less traffic because fewer content messages are transmitted.

All algorithms have been implemented in C and have been integrated into CCNx 0.8.2 [3]. Extensive evaluations have been performed in mobile scenarios with up to 100 nodes by emulations using NS3-DCE on Ubelix, the Linux cluster of the University of Bern [4]. We have evaluated content discovery in diverse namespaces (flat, hierarchical and mixed). Flat namespaces have only one name component and no naming structure. Hierarchical namespaces have names with many name components. Mixed namespaces are a combination of flat and hierar-

chical namespaces including flat names for some content and hierarchical names for others.

The results have shown that LFD performs best in most scenarios. This means it should be preferred if the name structure is not known.

Flat name spaces are ideal for ERD, but the overhead of LFD is negligible. LFD results in nearly the same discovery time and generates only slightly more traffic (one additional message to reach the leaf level).

In hierarchical and mixed namespaces LFD performs significantly better than ERD and RID because less time is required to browse the namespace. Furthermore, we have observed that the data traffic of ERD in hierarchical namespaces can become huge, even more than with RID. LFD, however, results in the smallest traffic overhead.

As future work, LFD may further be improved by implementing an adaptive request strategy on the leaf level (RID or ERD requests) depending on the discovered naming information. If many nodes reply with identical information, RID may be more efficient to detect new information. However, if the content is different, ERD requests may be more efficient because they contain only names and no data segment in the payload resulting in less traffic.

Bibliography

[1] 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*, pages p. 1-12, Rome, Italy, December 2009.

[2] Arian Uruqi: ”Content Discovery and Retrieval Application for Mobile Content-Centric Networks”, Bachelor Thesis, April 2014

[3] CCNx. http://www.ccnx.org/, April 2015.

[4] Ubelix, http://www.ubelix.unibe.ch

Content Discovery in Wireless Information-Centric Networks

Carlos Anastasiades, Arun Sittampalam, Torsten Braun
Institute of Computer Science and Applied Mathematics
University of Bern, Switzerland
Email: {anastasiades, sittampalam, braun}@iam.unibe.ch

***Abstract*—Information-centric networking (ICN) enables communication in isolated islands, where fixed infrastructure is not available, but also supports seamless communication if the infrastructure is up and running again. In disaster scenarios, when a fixed infrastructure is broken, content discovery algorithms are required to learn what content is locally available. For example, if preferred content is not available, users may also be satisfied with second best options. In this paper, we describe a new content discovery algorithm and compare it to existing Depth-first and Breadth-first traversal algorithms. Evaluations in mobile scenarios with up to 100 nodes show that it results in better performance, i.e., faster discovery time and smaller traffic overhead, than existing algorithms.**

***Index Terms*—Information-centric networks, wireless, opportunistic, discovery, CCN**

I. Introduction

Information-centric networking (ICN) is a new communication paradigm to address drawbacks of host-based communication protocols. By addressing and routing information based on names, ICN can identify and aggregate concurrent requests in popular content, and quickly find alternative content sources. In recent years, most ICN works have focused on routing, scalability and security in wired Internet protocols. However, due to its flexibility, ICN is also very appealing for wireless and mobile networks. A node that broadcasts a request can quickly find available content in its immediate neighborhood.

Information-centric networking has been identified as promising approach for opportunistic and delay-tolerant communication [1]. Natural disasters, e.g., floodings, earthquakes or wars, can destroy communication infrastructures and, thus, prevent infrastructure-based communication. While most works have focused on redundancy and infrastructure resilience [2], not much work has been performed to investigate approaches for operation during or after disasters, when fixed infrastructures may be destroyed. Recent work [3] has shown that most communication patterns that take place during disasters are of information-centric nature, e.g., retrieval of disaster information or dissemination of warnings. Information-centric networking can work in isolated islands that are disconnected from central infrastructure to provide any kind of information. For example, first responders could carry storage units in their backpacks with world news [3] or cars could provide multimedia content [4] to remote cities, which lack the required infrastructure. To retrieve content, knowledge of available content names is required. In a disaster scenario, requesters cannot get naming information from central repositories, and even if they could, it may not be meaningful if most content could not be retrieved due to broken links. Thus, if preferred content is not available because of limited reachability, users may also be satisfied with "second-best" alternatives. For example, if a video with the current weather forecast is not available, users may also be satisfied with a textual weather description.

In this work, we investigate algorithms to support opportunistic content discovery based on broadcast requests. We base our work on Content-Centric Networking [5], which is a popular ICN architecture. CCN is based on hierarchical naming, which supports name aggregation and longest-prefix matching. Unlike other approaches, CCN does not require any name resolution procedures. Due to longest-prefix matching, a node can define the scope of discovered content, e.g., requests with the prefix */Publisher* retrieve all content from a publisher while */Publisher/video* only retrieve videos. Based on prior work [6], we have designed and implemented three content discovery algorithms and evaluated them in terms of message overhead and time to discover available content. Since content naming can be performed arbitrarily, we thoroughly evaluate different naming structures for content discovery. *Flat namespaces* have only one name component and no naming structure. *Hierarchical namespaces* have names with many name components. *Mixed namespaces* are a combination of flat and hierarchical names including flat names for some content and hierarchical names for others.

The algorithms have been implemented in CCNx 0.8.2 [7] and we evaluate them in diverse mobile scenarios via emulation using NS3-DCE [8]. Please note that our algorithms are designed to find available content names in the absence of central naming repositories. However, once name collections have been identified, synchronization protocols may be applied [9] to retrieve new content.

The remainder of this paper is organized as follows. A short overview of the CCN architecture and related work is presented in Section II. The discovery algorithms are described in Section III. Evaluation results are shown in Section IV. In Section V, we discuss namespaces for content discovery. Finally, we conclude our work in Section VI.

II. Related Work

A. Content-Centric Networking

CCN communication [5] is based on two messages: *Interests* to request content and *Data* to deliver content. Typically in CCN, files consist of multiple segments that are carried by Data messages. Then, in order to retrieve a file, users need to express Interests in every segment of the file. The CCNx [7] project provides an open source reference implementation of CCN. The core element of the implementation is the CCN daemon (CCND), which performs message processing and forwarding decisions. Links from the CCND to applications or other hosts are called *faces*. A CCND has the following three memory components:

1) The Forwarding Information Base (FIB) contains forwarding entries to direct Interests towards content sources.
2) The Pending Interest Table (PIT) stores unsatisfied forwarded Interests together with the face on which they were received. If Data is received in return, it can be forwarded based on face information in the PIT.
3) The Content Store (CS) is used as cache in a CCN router storing received Data packets temporarily.

Prior to transmission, content is included and scheduled for transmission in a queue. To avoid duplicate content transmissions, an additive random delay AD = [DP, 3DP] is considered during content scheduling (the data pause DP is a configurable parameter). If a node overhears the transmission of the same content from another node, it removes the scheduled content from the content queue to avoid duplicate transmissions, i.e., enable *duplicate suppression*.

Interest messages contain several header fields, which are considered in the forwarding and matching process. The *Interest Lifetime* determines how long an Interest stays in the PIT. The *Scope* (if present) limits forwarding to the local host or neighboring hosts. Furthermore, *Exclude filters* can be used to indicate already known components and, thus, avoid duplicate content retrieval.

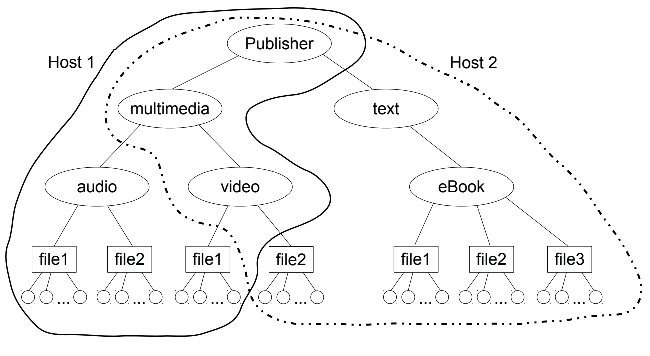


Fig. 1. Hierarchical Name Structure: files may be stored on different hosts.

Content names follow a hierarchical structure composed of one or multiple name components. To ensure globally unique names and support routing, content names may be aggregated by publisher specific prefixes similar to DNS names as illustrated in Figure 1. The ellipses correspond to name components and the rectangles to files. Each file consists of one or several segments denoted by small circles. There are no restrictions on content names and they may be selected arbitrarily. The hierarchical name may not indicate the location of content objects as Figure 1 shows. Content from the same publisher may be downloaded and provided by different mobile hosts. To find content from a specific publisher, an Interest in the general prefix */Publisher* is sufficient due to name aggregation and longest-prefix matching.

B. Discovery in Information-centric Networking

Routing in information-centric networking is equivalent to finding a content source for a given content name. NLSR [10] is a link-state based routing protocol that uses only Interest and Data messages and a synchronization mechanism to exchange collections of prefixes and learn the complete network topology. Other approaches use Bloom Filters to exchange availability information of local caches among neighbors [11], [12], [13] or use exploration phases, where Interests are flooded in the network [14] to find content sources close to requesters. However, routing protocols are used to configure forwarding entries in the FIB in a transparent way, i.e., without users noticing it. In addition, routing protocols may only advertise content prefixes, which is enough for routing, but they do not provide information about content published under these prefixes. In opportunistic scenarios with limited content availability, users may want to know what content can be retrieved.

In this work, we investigate name discovery based on broadcast requests similar to multicast DNS (mDNS) [15] and DNS-based Service Discovery (DNS-SD) [16]. DNS-SD enables requesters to find available services in specific domains or subdomains via service instance enumeration. Since mDNS provides name to IP address mappings, names are in general uniquely assigned to distinct nodes. This means that if the same name already exists, the user needs to select another one. In contrast to DNS-SD and mDNS, CCN content names do not need to be uniquely assigned to specific nodes and it is likely that the same content objects, i.e., copies, may be stored at repositories and caches of multiple nodes.

Earlier works [6], [17], [18] have applied Exclude filters to efficiently retrieve diverse content objects at the same time via broadcast requests. However, all of these works apply Exclude filters in flat namespaces, i.e., all content objects are published under the same prefix. It has been shown [6] that broadcast requests are particularly beneficial if content is stored uniquely in distinct repositories, i.e., no redundant copies, because multiple content objects can be transmitted in response to the same Interest, i.e., *parallel transmissions*.

In this work, we evaluate existing algorithms [6], i.e., Enumeration Request Discovery (ERD) and Regular Interest Discovery (RID), and compare it to a new Leaves First Discovery (LFD) algorithm in mobile environments with up to 100 nodes. We evaluate all algorithms in flat, hierarchical and mixed namespaces.

III. Content Discovery Algorithms

In distributed environments, where connectivity to fixed infrastructures and central repositories may not exist, nodes need to perform content discovery to learn what content names are available. Content discovery algorithms based on Bloom filters [13] or the Sync protocol [9] work only for already known content names. In addition, the Sync protocol triggers the retrieval of missing content automatically independent of whether users want the content or not. In this section, we describe three algorithms for content discovery in naming structures (name trees) such as in Figure 1.

A. Notation and Parameters

We describe Regular Interest Discovery (RID) and Enumeration Request Discovery (ERD), which are based on earlier work [6], as well as Leaves First Discovery (LFD), which is a combination of both. The common parameters for all algorithms are listed in Table I.

TABLE I
Algorithm Parameters.

Parameter	**description**
p	request prefix, initially $p = root$
IL	Interest lifetime: IL_{long}, IL_{short}
EF	Exclude filter in an Interest
to	# timeouts, initially $to = 0$ stop if to equals T
NT	name tree with name components

The request prefix p is initially set to a $root$ prefix, which defines the starting point for discovery, i.e., the root of the name (sub-)tree. Broadcast requests in content prefixes can result in multiple different content replies at the same time. Earlier work [6] has shown that requesters should wait a time interval of $2DP$ after the reception of the first response to an Interest. This enables different responses, which are triggered by the first Interest, to be received at the local cache before the next Interest is transmitted. Thus, this content can then be served from the cache and it does not need to be retransmitted by content sources, reducing the number of duplicate content transmissions drastically. We use two different values for the Interest lifetime IL, i.e., IL_{short} to retrieve content from the local cache and IL_{long} to get content from neighboring nodes. We do not set the Interest scope with IL_{long}, i.e., unlimited scope, but set it to "only local host" with IL_{short}. Interests do not request specific content but contain a general prefix to discover content published under the prefix. To avoid duplicate transmissions and receptions, Interests have an Exclude filter EF. If an Interest times out, the Interest is retransmitted until the number of timeouts to reach a certain limit T. All discovered names are stored in a name tree NT.

B. Regular Interest Discovery (RID)

Algorithm 1 presents Regular Interest Discovery (RID) and we explain it with the help of the name tree in Figure 1. RID is based on recursive expression of Interest messages to browse the name tree via Depth-first traversal. Due to longest prefix matching, a requester can express an Interest in a root prefix $p = /Publisher$ to discover the subtree under that root. After a Data message c has been received (line 10), RID extracts the file name without segment number (line 12), e.g., */Publisher/multimedia/audio/file1*, and stores it in the name tree NT (line 13). Then, the last component, i.e., *file1*, is removed from the request prefix (line 14) and included in the Exclude filter EF (line 15) to request other files such as file2. If only one Interest in the prefix p has been transmitted so far, i.e., *first_request*, and there were no timeouts to (line 16), the algorithm waits for $2DP$ before the next Interest transmission to reduce duplicate transmissions (line 17). After that, the Interest lifetime is set to IL_{short} to retrieve content from the cache. In case of a timeout with IL_{short}, the Interest is retransmitted with IL_{long} (line 21). In case of a timeout with IL_{long}, the timeout counter to is increased before the Interest is retransmitted (line 23). After T timeouts with IL_{long}, RID assumes that all content has been received under that prefix (timeout event) and climbs one level up in the nametree, e.g., to */Publisher/multimedia*, excluding the subtree it came from (line 7), i.e., *audio*. RID always requests the first segment of a content object. Thus, it can quickly go down to a leaf of the next subtree, e.g., */Publisher/multimedia/video/file1*.

A full discovery round is completed when RID has climbed up to the root of the nametree and experienced T timeouts, i.e., $root$ is not a prefix of p_{new} anymore (line 3). In the next discovery round (line 4), which starts after $time$, already known names can be excluded.

Algorithm 1 Regular Interest Discovery (RID)
```
 1: function RID_DISCOVER($p$, $EF$)
 2:     $p_{new}$ = RID_GET ($p$, $EF$, 0)
 3:     if $root \notin p_{new}$ then
 4:         SCHEDULE_NEXT_DISCOVERY ($time$)
 5:     else
 6:         $p_{next}$ = remove_last_comp($p_{new}$)
 7:         last_comp($p_{new}$) $\rightarrow EF$
 8:         RID_DISCOVER($p_{next}$, EF)

 9: function RID_GET($p$, $EF$, $to$)
10:     $c$ = SEND_INTEREST ($p$, $IL_{long}$, $EF$)
11:     while (Data $c$ has been received) do
12:         name = getName($c$)
13:         name $\rightarrow NT$
14:         $p$ = remove_last_comp(name)
15:         last_comp(name) $\rightarrow EF$
16:         if first_request and $to$ == 0 then
17:             wait($2DP$)
18:         $c$ = SEND_INTEREST ($p$, $IL_{short}$, $EF$)
19:     if $to < T$ then
20:         if timeout with $IL_{short}$ then
21:             RID_GET ($p$, $EF$, $to$)
22:         else
23:             RID_GET ($p$, $EF$, $to$ + 1)
24:     else
25:         return $p$
```

C. Enumeration Request Discovery (ERD)

Algorithm 2 shows Enumeration Request Discovery (ERD), which is based on the consecutive expression of enumeration requests and targets content at repositories. An enumeration request for a certain prefix requests all next level components for the prefix at a certain repository. The enumeration response, i.e., list of next level components, is identified by the ID of the corresponding repository, which is based on its public key.

Algorithm 2 Enumeration Request Discovery (ERD)

```
1:  function ERD_DISCOVER($p$, $EF$)
2:      ERD_GET ($p$, $EF$, 0)
3:      $p_{next}$ = next_comp_on_lvl($p$, $NT$)
4:      if $p_{next} \neq \{\}$ then
5:          ERD_DISCOVER ($p_{next}$, $\{\}$)
6:      else
7:          $p_{next}$ = next_lvl_comp($p_{new}$, $NT$)
8:          if $p_{next} \neq \{\}$ then
9:              ERD_DISCOVER ($p_{next}$, $\{\}$)
10:         else
11:             SCHEDULE_NEXT_DISCOVERY ($time$)

12: function ERD_GET($p$, $EF$, $to$)
13:     $L$ = SEND_ENUMERATION ($p$, $IL_{long}$, $EF$)
14:     while (Enumeration $L$ has been received) do
15:         getNames($L$) $\rightarrow NT$
16:         getID($L$) $\rightarrow EF$
17:         if first_request and $to$ == 0 then
18:             wait($2DP$)
19:         $L$ = SEND_ENUMERATION ($p$, $IL_{short}$, $EF$)
20:     if $to < T$ then
21:         if timeout with $IL_{short}$ then
22:             ERD_GET ($p$, $EF$, $to$)
23:         else
24:             ERD_GET ($p$, $EF$, $to$ + 1)
```

ERD starts by expressing an enumeration request in the root prefix, e.g., $p = /Publisher$ (line 13). This may trigger an enumeration response from repository 1 on host 1, i.e., */Publisher/ID1*, containing the next-level component *multimedia*. The received name components, i.e., here only *multimedia*, are then added to the name tree NT (line 15) and the repository ID *ID1* is added to the Exclude filter (line 16). To reduce duplicates, ERD also waits $2DP$ after the *first_request* (line 18). The next Enumeration request uses IL_{short} and excludes *ID1* to retrieve Enumeration responses of other hosts from the cache (line 19). In this example, repository 2 on host 2 has replied with */Publisher/ID2* containing the components *multimedia* and *text* in the payload. ERD continues with the same prefix (but more excluded IDs) until an enumeration request has timed out T times (timeout event). Then, *next_comp_on_lvl* (line 3) checks the name tree for other name components on the same level. If there are other components, discovery continues on the same level (line 5). If there are no more components, *next_lvl_comp* (line 7) continues with the first component on the next level in the name tree, e.g., */Publisher/multimedia* or */Publisher/text*. A discovery run is finished at the leaves of the name tree, i.e., if there are no more name components. Then, the next discovery run (line 11) starts after $time$.

In contrast to RID, ERD payloads contain only content names but no data and, thus, have a smaller size. However, since enumeration responses are identified by repository IDs, redundant information cannot be avoided. For example, if multiple repositories store exactly the same content, their enumeration responses appear to be different (different repository IDs), although they do not provide new information.

D. Leaves First Discovery (LFD)

Leaves First Discovery (LFD), which uses elements from both RID and ERD, is presented in Algorithm 3. Similar to RID, a regular Interest is transmitted first to quickly find content and reach the leaves of the name tree (line 2). After the first Data message has been received, LFD has reached a leaf level and can perform ERD discovery to retrieve enumeration responses from all repositories (line 7). If all enumerations have been received at a level, i.e., T timeouts have been triggered in ERD_GET (line 7), and it is not the root level (line 8), the algorithm climbs one level up in the name tree by removing the last component (line 11), e.g., from */Publisher/multimedia/audio* to */Publisher/multimedia*, excluding the component from the last subtree (line 12) , e.g., */audio*, and performs a RID request to quickly reach the leaves of the next subtree (line 13), e.g., */Publisher/multimedia/video/file*. RID requests are retransmitted up to T times (line 15) before climbing one level up (lines 17-19). Discovery stops with T timeouts at the root of the tree and the next discovery starts after $time$ (line 9 and 18).

Algorithm 3 Leaves First Discovery (LFD)

```
1:  function LFD_DISCOVER($p$, $EF$, $to$)
2:      $c$ = SEND_INTEREST ($p$, $IL_{long}$, $EF$)
3:      if (Data $c$ has been received) then
4:          name = getName($c$)
5:          name $\rightarrow NT$
6:          $p_{new}$ = remove_last_comp(name)
7:          ERD_GET ($p_{new}$, $\{\}$, 0)
8:          if $p_{new} == root$ then
9:              SCHEDULE_NEXT_DISCOVERY ($time$)
10:         else
11:             $p_{next}$ = remove_last_comp($p_{new}$)
12:             last_comp($p_{new}$) $\rightarrow EF$
13:             LFD_DISCOVER ($p_{next}$, $EF$, 0)
14:     else if $to < T$ then
15:         LFD_DISCOVER ($p$, $EF$, $to$ + 1)
16:     else if $p \neq root$ then
17:         $p_{next}$ = remove_last_comp($p$)
18:         last_comp($p$) $\rightarrow EF$
19:         LFD_DISCOVER ($p_{next}$, $EF$, 0)
20:     else
21:         SCHEDULE_NEXT_DISCOVERY ($time$)
```

LFD includes advantages of RID to quickly reach content leaves but does not require as much data overhead as RID because mostly content names are requested and fewer data segments.

E. Unsolicited Content in the Content Store

The discovery algorithms described above transmit Interests (prefixes) via broadcast to all repositories in the vicinity. However, if content has been received at the content store and no Interest can be found in the PIT, e.g., because the Interest has already been satisfied by previously received content, it is considered as *unsolicited content*. To enforce flow balance between Interest and Data messages in the CCNx 0.8.2 implementation, unsolicited content becomes stale immediately, i.e., as if it was expired content. Stale content in the cache is deleted and not returned to Interest messages, even if it would satisfy them. This is a very inefficient strategy because the content has already been received and needs to be retransmitted by content sources. Previous work has shown that it is beneficial to keep unsolicited content in wireless information-centric communication [6], [19], [18] because requesters can address multiple content sources at the same time and benefit from parallel content transmissions by retrieving some content from cache. Therefore, to improve efficiency for wireless communication, the current CCND implementation needs to be slightly modified such that unsolicited content does not become stale immediately.

IV. Evaluation

We have implemented RID, ERD and LFD in the CCNx 0.8.2 framework [7]. The evaluations have been performed by emulations in mobile scenarios with NS3-DCE [8].

A. Evaluation Parameters

The evaluation parameters are listed in Table II. We use 802.11g wireless interfaces and a Log-Distance propagation loss model. The transmission power is set to 16dBm and the energy detection threshold is set to -76dBm, which corresponds to transmission ranges of up to 60m. The playground size is set to 150m $\times$ 150m. Repositories with content are placed in a static grid of 25 or 100 nodes, i.e., in a grid with 25m or 12.5m distance. A mobile requester performes opportunistic (one hop) content discovery, while moving according to the Random Waypoint Model with a pedestrian speed between 1.2 - 1.4m/s. The additive random delay AD for content scheduling uses a data pause DP of 270ms. The Interest lifetimes are set to 875ms (long Interests) and 125ms (short Interests). If a short Interest times out, a long Interest is transmitted. After $T = 1$ timeouts of long Interests, a timeout event is triggered, i.e., the algorithms proceed with the next prefix. If not all content objects could be discovered within one discovery round, discovery is resumed after 2s. Every configuration is evaluated in 100 different runs.

TABLE II
Common simulation parameters.

Parameter	**Value**
interface	1x 802.11g
propagation loss	LogDistance
energy detection threshold	-76dBm
TX power	16dBm
Mobility	random waypoint, speed: 1.2 - 1.4m/s pause time: 1s, skip time: 3600s
playground size	150m x 150m
repository nodes	25 nodes grid with 25m distance 100 nodes grid with 12.5m distance
data pause DP	270ms
Interest lifetime	long: 875ms, short: 125ms
retransmission limit T	1
resume interval	2s

B. Scenarios

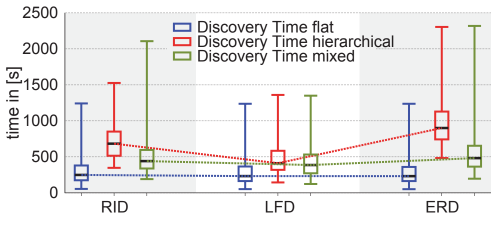
The performance of content discovery depends on the structure of the namespace. Therefore, we perform evaluations in diverse namespaces. In each scenario, we set the number of distinct content objects to 100.

1) *flat namespace:* all 100 content objects are published under the same prefix, i.e., there is no naming structure. Every content has one distinct name component.
2) *hierarchical namespace:* all 100 content objects are published under a hierarchical prefix with 10 name components. We use 4 different hierarchical prefixes with 10 components for the 100 content objects.
3) *mixed namespace:* it is composed of hierarchical and flat names. In total, every prefix has randomly between 1 and 10 components. Every node in the name tree has between 1 and 5 different children. At the leaves, the number of content objects is randomly set between 1 and 20.

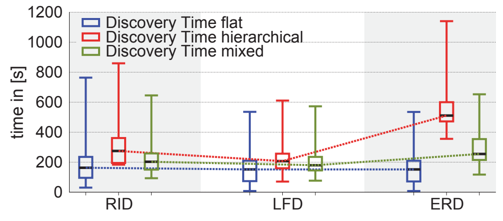
The namespaces are designed such that every name component is 12 bytes long. In every scenario, each node contains 4 different content objects. In the 25 nodes network, every content object is stored uniquely at one node. In the 100 nodes network, content is distributed randomly among all nodes, i.e., there are redundancies. We evaluate RID, ERD and LFD in terms of discovery time, transmitted Interest and received Data messages as well as received duplicate Data messages.

C. Discovery Time

We define the *discovery time* as the time until a requester has discovered all 100 content objects in the network. Figure 2a shows discovery times in the 25 nodes network. The x-axis shows the result for RID, LFD and ERD and the y-axis the discovery time in seconds.



(a) 25 nodes



(b) 100 nodes

Fig. 2. Discovery Time of RID, ERD and LFD in flat, hierarchical and mixed namespaces.

Figure 2a shows that flat namespaces result in shortest discovery times. The difference in discovery time between RID, ERD and LFD is insignificant. LFD requires 0.8% more time than ERD and RID requires 5% more time than LFD. Although RID requests each content object separately, the discovery time is only slightly longer than for LFD and ERD.

This is due to broadcast requests that can trigger multiple distinct content transmissions, i.e., parallel transmissions, from different content sources. If unsolicited content is not discarded, it can be collected quickly by subsequent Interests from the cache [6].

Discovering content in a hierarchical namespace requires more time because algorithms need to climb up and down the name tree, which includes more waiting times due to timeouts. For example, LFD requires 76.7% more time in a hierarchical namespace compared to a flat namespace. Figure 2a shows that LFD handles naming hierarchy better than RID and ERD: it results in 39.8% shorter discovery times than RID and even 54.3% shorter discovery times than ERD.

It may seem surprising that LFD performs even better than RID in the hierarchical scenario, because each node contains 4 content objects with different hierarchical prefixes. This means that every enumeration response with LFD contains only 1 content name. However, LFD can quickly reach the leaf level and then retrieve content names faster than RID because Data messages at the leaf level are smaller, i.e., they contain only names. While RID could benefit largely from parallel transmissions in flat namespaces, fewer parallel transmissions are possible in hierarchical namespaces (only for content with the same prefix).

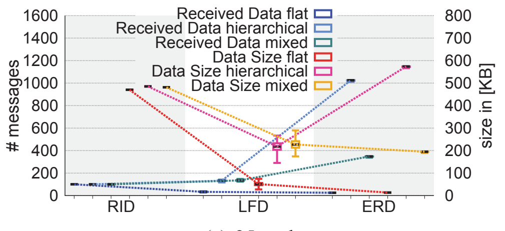
In mixed namespaces, LFD performs on average 17.6% faster than RID and 29.1% faster then ERD. The maximum discovery times of RID and ERD are even 56.1% and 71.8% longer than for LFD since more time is required to browse the name tree, which is disadvantageous in case of mobility.

Figure 2b shows discovery times for RID, LFD and ERD in a network with 100 repository nodes. Due to higher content density, discovery times have significantly decreased compared to the 25 nodes scenario in Figure 2a. In hierarchical namespaces, discovery times have decreased more than in flat namespaces due to fewer climbing operations, i.e., content can be found quicker. RID benefits the most from a higher content density due to more parallel transmissions, while LFD and ERD benefit less, because the number of nodes increases as well, which means that more enumerations have to be requested.

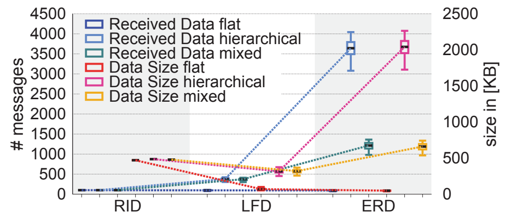
The relative differences between flat and mixed namespaces have decreased in the 100 nodes scenario. For example in the 25 nodes network, RID discovery in mixed namespaces results in 68.5% longer discovery times than in flat namespaces, while in the 100 nodes scenario the difference between mixed and flat namespaces is only 24.0%. If content density is high, discovery in flat namespaces results in more collisions than in mixed namespaces because more content is requested at the same time (broadcast requests).

D. Data Messages

Data messages with RID always contain a payload of 4096 bytes, i.e., first data segment, while Data messages with ERD are smaller because they contain only lists with name components. Figure 3a shows, therefore, not only the number of received Data messages at the requester (left y-axis) but also their size in bytes (right y-axis) for the 25 nodes network.



(a) 25 nodes



(b) 100 nodes

Fig. 3. Transmitted Data of RID, ERD and LFD in flat, hierarchical and mixed namespaces.

In flat namespaces, ERD results in the fewest transmitted Data messages (and fewest transmitted bytes) because only one list needs to be transmitted per node, while RID needs to transmit one Data message per content object. On average, LFD results in 2.8 times more data bytes and RID even in 34.3 times more data bytes than ERD. The data overhead between LFD and ERD is larger than expected, because LFD needs to transmit an Interest to reach the leaf level. Due to multiple nodes in wireless transmission range, an LFD requester may retrieve 7 - 8 segements in response to this Interest (due to parallel transmissions), although only one Data message would be enough to reach the leaf level.

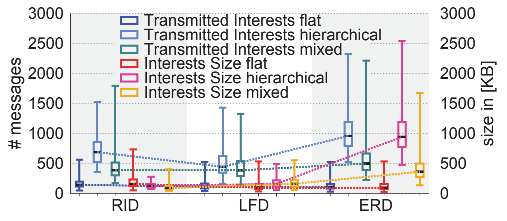
However, ERD does not result in the fewest transmitted bytes in all namespaces. In hierarchical namespaces, ERD results in 2.6 times more transmitted data bytes than LFD and even 18% more bytes than RID. This is because ERD transmits 7.8 times more Data messages than LFD while climbing down the name tree and even 10 times more Data messages than RID. Although individual packets are smaller for ERD than for RID, the sum of transmitted bytes (packet headers and payloads) becomes larger.

In the mixed namespace, LFD transmits on average 52.8% fewer bytes than RID but 14.5% more bytes than ERD. LFD transmits slightly more bytes than ERD because it retrieves a content object for every name subtree to reach the leaf level. However, the situation changes with increased node density. Figure 3b shows the number of Data messages in the 100 nodes network. In the mixed namespace (100 nodes network), ERD results in the most transmitted Data bytes, i.e., 37% more than RID and even 107% more than LFD.

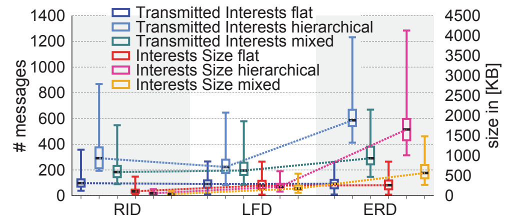
In hierarchical namespaces, LFD results in 35.6% fewer bytes than RID and even in 84.6% fewer bytes than ERD. Only in flat namespaces, ERD performs slightly better than LFD, i.e. 32% fewer bytes, but the difference is negligible (a few KBs) compared to the large overhead in hierarchical and mixed namespaces (several hundreds of KBs), where ERD requires even more Data bytes than RID. In all scenarios, LFD results in significantly fewer Data bytes than RID.

E. Interest Messages

Figure 4a illustrates the transmitted Interests (left y-axis) and their sizes in bytes (right y-axis) in the 25 nodes network. Because Interests only retrieve content, their size may often be neglected. However, Exclude filters can grow with the number of discovered content resulting in large Interest messages.



(a) 25 nodes



(b) 100 nodes

Fig. 4. Transmitted Interests of RID, ERD and LFD in flat, hierarchical and mixed namespaces.

The number of transmitted Interests in the flat namespace is similar, LFD sends 21.4% fewer Interests than RID and 1.5% more Interests than ERD. RID Interests are slightly larger due to larger Exclude filters (more excluded components). Thus, RID sends 36% more bytes in Interests than LFD and ERD.

In the hierarchical namespace, LFD results in 28.2% fewer Interests than RID and even in 43.2% fewer Interests than ERD. However, the size of transmitted Interests with RID is 25.8% smaller than for LFD and even 87.3% smaller than for ERD. This is due to two reasons. First, there are four different hierarchical prefixes, thus, fewer components need to be excluded with RID compared to the flat namespace (namespace partitioning). Second, Exclude filters with LFD and ERD are larger because they include repository IDs, which have a static length of 38 bytes, and not name components, which have a length of 12 bytes in our scenario.

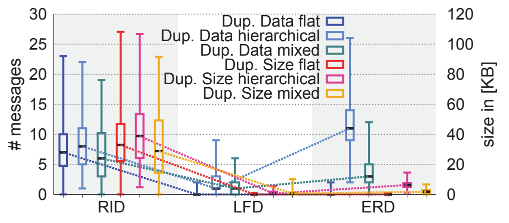
Similar observations can be made in mixed namespaces. However, compared to the hierarchical namespace, LFD and ERD require slightly fewer Interest bytes because more names can be included in the same enumeration responses, i.e., fewer Interests are required.

Figure 4b shows the number of Interest messages in the 100 nodes network. Surprisingly, the number of transmitted Interests can be reduced compared to the 25 nodes scenario. For example in the flat namespace, the number of Interests can be reduced by by 33.9% (RID), 27% (LFD) and 26.6% (ERD) compared to the 25 nodes scenario. Due to higher content density more content can be retrieved via *parallel transmissions*. Consequently, the Interest bytes for RID decrease by 47.6%. For LFD and ERD, however, transmitted bytes increase despite fewer Interests by 143.5% (LFD) and 147.2% (ERD). Due to higher node density, transmitted Interests that are not satisfied from local cache have longer Exclude filters because more repository IDs need to be excluded. Similar observations can be made for the hierarchical and mixed namespace.

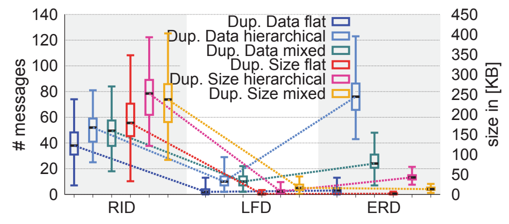
In the worst case (maximum values), the transmitted Interest bytes with ERD can become even larger as transmitted Data messages in bytes (see last subsection).

F. Duplicate Data

Figure 5a shows the received Duplicates (left y-axis) and their sizes (right y-axis) at the requester in the 25 nodes network. It may be unexpected to have duplicates in the 25 nodes network where every content is uniquely stored only at one node. However, a node may request content from a repository and then move out of range for reception. Other nodes may receive the content and keep it in the cache. If the requester comes into the range again, multiple nodes may reply with the same content resulting in duplicates. Although duplicate suppression can reduce the number of duplicates, it cannot completely prevent them [6].



(a) 25 nodes



(b) 100 nodes

Fig. 5. Duplicate Data of RID, ERD and LFD in flat, hierarchical and mixed namespaces.

Figure 5a shows that most duplicates (except for the hierarchical namespace) are received with RID and the fewest duplicates are received with LFD. Since duplicates with ERD and LFD are rather small, RID always results in the most duplicate bytes transmitted. However, compared to the size of transmitted Interest and Data messages, the overhead for duplicate transmissions is insignificant in the 25 nodes scenario.

Figure 5b shows the number of duplicates and their sizes in the 100 nodes network. Due to higher node and content density, the number of duplicates have increased by a factor of 5 or more compared to the 25 nodes network. Figure

5b shows that duplicate bytes are still negligible for LFD and ERD. Although ERD results in 5 times more duplicate bytes than LFD in hierarchical namespaces, which may seem high, it corresponds only to 2.5% of all transmitted Interest bytes with ERD. For RID, however, duplicates become a significant fraction of network traffic as more duplicate bytes are transmitted than Interest bytes.

V. Namespace Design for Disaster Scenarios

Discovery in flat namespaces is considerably faster compared to hierarchical namespaces because no browsing is required. However, if node and content density is high, flat namespaces can result in many data transmissions and collisions. In addition, the size of Interest messages increases since more components need to be excluded. Content providers can, therefore, use hierarchical names (depending on the number of content they provide) to partition the namespace and limit the number of content replies.

If time matters, e.g., in emergency scenarios, flat namespaces or only a few hierarchy levels should be preferred. To facilitate content discovery, authorities may define artificial flat namespaces, e.g., */disaster*, and create *alias mappings* to ”real”, i.e., existing and potentially hierarchical, content. Alias mappings are content objects with artificial names, which include a list of content names in the payload. A requester can then learn ”real” content names by inspecting the payload. This enables authorities to promote already widely distributed content (even from various providers) and mark them as important without re-publishing and, therefore, re-signing the content. Also, multiple alias mappings, e.g., names in different languages, can be defined for the same content. Content retrieval can be performed by the ”real” content name learned from the payload. Thus, requesters can identify identical content in caches despite different alias mappings. Like any other CCN content, alias mappings are signed such that users can lay their trust in alias mappings based on the authority that created it.

VI. Conclusions

If connectivity to a fixed infrastructure is broken, users need to know the names of available content objects before they can retrieve them. In this paper, we have described three discovery algorithms for content discovery, namely RID, ERD and LFD. All algorithms have been evaluated by emulations in mobile scenarios using flat, hierarchical and mixed namespaces.

If the structure of the namespace is unknown, LFD should be preferred because it performs better than ERD and RID in most scenarios. In flat namespaces, which are optimal for ERD, LFD results in only slightly more Data traffic compared to ERD (same discovery time) but in significantly lower Data overhead compared to RID. In hierarchical and mixed namespaces, LFD performs significantly better than ERD because fewer Data and Interest messages are required (lower traffic overhead). Although LFD sends slightly more bytes via Interest messages compared to RID, the overall traffic including Data and duplicate messages is still higher with RID. In addition, LFD results in lower discovery times compared to both ERD and RID.

For higher node densities, LFD and ERD can send fewer Interest messages than in sparse densities but the Interest messages become larger due to longer Exclude filters. In hierarchical and mixed namespaces, ERD performs the worst because it may transmit even more bytes via Interest than Data messages. However, even in high node density networks, LFD still results in a lower traffic overhead and lower discovery times compared to RID.

As future work, an adaptive request strategy for LFD on the leaf level, i.e., send RID or ERD requests depending on whether enumeration responses contain the same information or not, may further improve message efficiency. To support this, it would be beneficial to use hashes from payloads as enumeration identifiers instead of repository IDs.

References

[1] G. Tyson, J. Bigham, and E. Bodanese. Towards an Information-Centric Delay-Tolerant Network. In *2nd IEEE NOMEN*, 2013.

[2] J. Sterbenz, D. Hutchison, E. Cetinkaya, A. Jabbar, J. Rohrer, M. Schöller, and P. Smith. Resilience and survivability in communication networks: Strategies, principles and survey of disciplines. *Elsevier Computer Networks*, 54:1245–1265, 2010.

[3] G. Tyson, E. Bodanese, J. Bigham, and A. Mauthe. Beyond content delivery: Can icns help emergency scenarios? *IEEE Network*, 28(3):44–49, June 2014.

[4] A. Galati, T. Bourchas, S. Siby, S. Frey, M. Olivares, and S. Mangold. Mobile-enabled delay tolerant networking in rural developing regions. In *IEEE Mobile-Enabled Delay Tolerant Networking in Rural Developing Regions*, Silicon Valley, CA, USA, October 2014.

[5] 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*, pages 1–12, Rome, Italy, December 2009.

[6] C. Anastasiades, A. Uruqi, and T. Braun. Content Discovery in Opportunistic Content-Centric Networks. In *5th IEEE WASA-NGI*, pages 1048–1056, Clearwater, FL, USA, October 2012.

[7] CCNx. http://www.ccnx.org/, April 2015.

[8] NS-3: Direct Code Execution. http://www.nsnam.org/overview/projects/direct-code-execution/, April 2015.

[9] CCNx Synchronization Protocol. http://www.ccnx.org/releases/ccnx-0.8.2/doc/technical/SynchronizationProtocol.html, April 2015.

[10] M. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang. NLSR: Named-data Link State Routing Protocol. In *ACM Workshop on Information-Centric Networking (ICN 2013)*, August 2013.

[11] Y. Wang, K. Lee, B. Venkataraman, R. Shamanna, I. Rhee, and S. Yang. Advertising cached contents in the control plane: Necessity and feasibility. In *IEEE Infocom*, 2012.

[12] M. Lee, K. Cho, K. Park, T. Kwon, and Y. Choi. Scan: Scalable content routing for content-aware networking. In *IEEE ICC*, Kyoto, Japan, June 2011.

[13] M. Lee, J. Song, K. Cho, S. Pack, T. Kwon, J. Kangasharju, and Y. Choi. Content discovery for information-centric networking. *Elsevier Computer Networks*, 2014.

[14] R. Chiocchetti, D. Perino, G. Carofiglio, D. Rossi, and G. Rossini. Inform: a dynamic interest forwarding mechanism for information-centric networking. In *3rd ACM sigcomm ICN*, Hong Kong, August 2013.

[15] S. Cheshire and M. Krochmal. RFC 6762: Multicast DNS.

[16] S. Cheshire and M. Krochmal. RFC 6763: DNS-based Service Discovery.

[17] C. Anastasiades, W. El Maudni El Alami, and T. Braun. Agent-based Content Retrieval for Opportunistic Content-Centric Networks. In *12th WWIC*, 2014.

[18] M. Amadeo, C. Campolo, and A. Molinaro. Multi-source data retrieval in IoT via named data networking. In *1st ACM ICN*, 2014.

[19] L. Wang, R. Wakikawa, R. Kuntz, R. Vuyyuru, and L. Zhang. Data naming in vehicle-to-vehicle communications. In *IEEE Infocom Computer communication workshop*, pages 328–333, Orlando, FL, March 2012.