UCLA
UCLA Electronic Theses and Dissertations

Title
Reliable Content Distribution using ICN

Permalink
https://escholarship.org/uc/item/0327t9kx

Author
Angius, Fabio

Publication Date
2014

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

University of California
Los Angeles

Reliable Content Distribution using ICN

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Fabio Angius

2014

© Copyright by
Fabio Angius
2014

Abstract of the Dissertation

Reliable Content Distribution using ICN

by

Fabio Angius

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2014

Professor Mario Gerla, Chair

Information Centric Networking [ADI12] has an intrinsic distributed nature, which eases decentralizing formerly centralized protocols. Nevertheless, implementing fully distributed protocols on a large scale remains challenging due to the concurrent nature of network elements.

Cache inconsistency becomes a problem whenever different content, or multiple versions of the same content, can answer the same query, in the first part of this book we focus on the detrimental impact that this has on **Privacy** and **Communication Many-To-Many**.

With regard to privacy we show that hierarchical caching introduces a considerable latency in updating Access Control Lists. We study the problem analytically and via simulation and propose two possible solutions: one for the generic problem of privacy in ICN and one for the specific case of **Social Networking**.

In the latter case of many-to-many communication, we prove that the current state of the art limits the throughput and frequently suffers data loss.

The second and last part of the book switches topic to the impact of ICN on **wireless ad-hoc networks**. On this matter, our work consists of an original algorithm for message dissemination; and implementing seamless path redundancy and multi-source download.

The dissertation of Fabio Angius is approved.

Ali G. Sayed

Majid Sarrafzadeh

Lixia Zhang

Mario Gerla, Committee Chair

University of California, Los Angeles

2014

To everyone who has been with me during my Ph.d Program

Table of Contents

List of Figures

List of Tables

Vita

2003-2010	Bachelor and Master Degree at University of Studies in Turin
2008-2014	Graduate Student Researher at NRL UCLA
Winter 2011	Teaching Assistant for Peer to Peer Networks (CS114), Computer Science Department, UCLA.
Winter 2012	Teaching Assistant for Modeling Uncertainty in Information Systems (CS112), Computer Science Department, UCLA.
Spring 2013	Teaching Assistant for Formal Languages and Automata (CS181), Computer Science Department, UCLA.
Spring 2014	Teaching Assistant for Formal Languages and Automata (CS181), Computer Science Department, UCLA.

CHAPTER 1

Introduction

Due to its always increasing demand of performance the online industry relies everyday more on Content Delivery Networks (CDN) that guarantee both high-throughput and low latency by deploying cache memory directly in the local area networks (LAN) of the Internet Service Providers (ISP).

Since CDNs are closed proprietary pay-per-use services, the new challenge for network engineers became to reproduce the same functionality openly on a global scale, at the Network layer. Information Centric Networking (ICN) [ADI12] is the result of these efforts. This new network design exploits dropping memory cost trends and extends routers with extra caching memory that allows them to cache content for future requests. The final goal is to limit the use of long network links favoring, instead, *opportunistic data retrieval*, i.e. fetching data from their nearest copy.

A key aspect of Information Centric Network architectures, and of CCN in particular, is to connect users to data instead of setting up connections between machines. Namely, ICN enables location/identity separation: a file has a unique name that is independent from the address of the machine where it is stored. The goal is to obtain *what* a user wants without worrying about *where* he is getting it from. The user only specifies an interest for a name, and lets the network find matching copy of the content.

This radical change re-opens many issues that were considered solved and that have now to be carefully reconsidered. A representative example is *routing*: due to the cumbersome address space of content names and the multi-path nature of ICN routing protocols must be re-designed in order to limit the network overhead while maintaining routing tables consistent.

Many other problems arise due to the distributed nature of ICN where data can be replicated massively with no central coordination and no global versioning. Problems arise especially when multiple versions of the same content are concurrently stored in the network, since every router has only local knowledge about the date it is impossible to determine if it has cached the most recent version of a content. Note that, a content request can always be forced to skip every cache and be sent directly to the original server, however using this functionality at any moment would make network caching purposeless and therefore it should not be used if not in particular cases.

Throughout the book we will show how due to this limitation, at the current state, ICN cannot implement efficiently and reliably **privacy** as well as **many-to-many communication**.

We studied both problems thoroughly using a bottom-up approach that starts from a theoretical model and moves up to the architectural details, in both cases we were able to formalize and implement a feasible and efficient solution to the problem.

Ultimately, we diverge from the initial topic discussing ICN for mobile ad-hoc wireless networks (MANET). ICN reshapes the way of intending Manets due to the fact that it deprecats end-to-end communication in favor of data dissemination, this solves many of the fundamental problems that limited the use of ad-hoc networks in the past.

1.1 Privacy

The challenge of implementing a privacy protocol in ICN is due to the fact that at a given time there might multiple copies of the same content encrypted using different credentials. On this topic, our contribution is three-fold: Firstly, we define a simple but complete performance framework for comparing current and future solutions. Secondly, we conjecture and prove the existence of *unsafe replicas*, namely cached content that remains available to users whose access has been revoked. Thirdly, we propose a performant protocol that solves the problem of unsafe replicas without tampering with the caching functionality of ICN.

1.2 Social Networking

We propose WARP, an architecture based upon Information-Centric Networking (ICN) designs, which expands the scope of the ICN architecture beyond media distribution, to provide data control in social networks. The benefit of our solution lies in the lightweight nature of the protocol and in its layered design. With WARP, data distribution and access policies are enforced on the user side. Data can still be replicated in an ICN fashion but we introduce control channels, named *thread updates*, which ensures that the access to the data is always updated to the latest control policy. WARP decentralizes the social network but still offers APIs so that social network providers can build products and business models on top of WARP. Social applications run directly on the user's device and store their data on the user's *butler* that takes care of encryption and distribution. Moreover, users can still rely on third parties to have high-availability without renouncing their privacy.

1.3 Many-To-Many Communication

This scenario is troublesome whenever different users produce content under the same name. We show evidence that with the current state of the art, content updates can be lost when generic shared names are used, due to the fact that interest packets - i.e. data requests - can be generated at a fixed bounded rate. As a solution, we propose an algorithm, inspired by the frequency hopping algorithm used in wireless systems, named Prefix Hopping (PH) to implement high rate many-to-many channels. The results show how adopting PH is possible to arbitrarily reduce data losses at the cost of a negligible overhead. We have implemented PH and show that it results in a dramatic performance improvement.

1.4 ICN and MANET

Manets have been studied for decades and along with them the possibility of disseminating data opportunistically, to this day, however, there is still not a real implementation that allows developing wireless peer-to-peer applications.

The emerging ICN architectures solve this problem by offering a new network stack that revolves entirely around data and opportunistic data retrieval. In this work we propose our ICN solution for wireless networks. Our contribution consists of: (1) adopting the BlooGo algorithm for message dissemination, which reduces sensibly the number of transmissions needed to deliver a message, (2) seamless path redundancy, which allows to recover a broken path without need of coordination between the peers, and (3) efficient multi-path download, without breaking the fundamental design of ICN, which removes the need of endpoints, we let the client have a control on which path to use for the download.

CHAPTER 2

Information Centric Networking

2.0.1 Background

Information Centric Networking[ADI12] describes network architectures that use data retrieval primitives, i.e. *put/get*, in place of the primitives for machine-to-machine message delivery, i.e. *send/listen*. The final goal is to decouple applications from topology and fetch data from anywhere in the network, including transparent caches. Traffic in the Internet follows a power law distribution where a significant fraction of the traffic comes from a rather restricted number of items. For this reason, as the cost of storage decreases much faster than the cost of bandwidth, inserting storage within the network as proposed by ICN protocols appears to be a valid alternative to the current architecture.

ICNs usually rely on two types of packets, data and requests. Data packets simply contain the content object. The content is uniquely named, signed and, since there is no access control on the network caches, encrypted by their producer for security reasons. Requests, intuitively, are the messages used to fetch data. How to route content requests is a critical issue for ICN protocols. Some architectures, such as [KCC07, DKO13], use name resolvers to locate content. This solution, however, is less responsive to changes and is exposed to attacks, such as DoS. Alternatively, architectures such as [JST09a, SSH07] use routing tables or hybrid schemes. For name resolution, the reader is reminded of [AK13, JBD12].

By design, ICN architectures secure the data instead of the communication channel. However, since the producer has no control over the replicas of its data, their application to social networking is very limited. A trivial example is the following: let us imagine that a content, say A, is initially shared by Alice with all her friends. After a few of her friends

have downloaded A, A is replicated on several distributors. If now Alice wants to withdraw the permission of reading A from Bob, she must first ensure that all the copies of A stored in caches are voided and replaced with the new version of A. This is required to avoid that Bob fetches the first version of A that he is able to decrypt. The task is made difficult for Alice since the data could have been replicated several times on caches that are unknown to her.

For CCN and NDN, the communication pattern is based on two types of packets *data* and *request*, otherwise known as *interest*.

When a host sends a request to the network this travels according to routing tables, also known as *Forwarding Information Base* (FIB), until it reaches a router that has a copy of the content which is then forwarded back to the requester. Ideally content should always be delivered by the closest cache in order to limit the use of resources in the peripheral areas of the network. However different policies that can be adopted.

2.0.2 Named Data Network

The fundamental design directives of NDN can be summarized as follows: **Content Naming -** Content is named instead than machines in order to decouple application logic from network topology. **In-Network Caching -** Network components, such as routers, may be able to cache traversing traffic in order to serve future requests locally rather than remotely. **Thin waist -** Differently than CDNs, ICN is meant to become a public infrastructure. Consequently, as explained in the original paper [JST09a], its design must maintain the same thin waist of TCP-IP building one layer on top of the others. The network must adapt to traffic demand and failures remaining agnostic of what applications are requesting and serving the data. Indeed this is a very limiting restriction in comparison with CDNs that are application specific and can be heavily optimized.

2.0.3 NDN Primer and Distributed protocols in NDN

NDN names data hierarchically so that each packet embeds the name of the content it belongs to. In addition, as a convention, each name should include (1) a **Globally Routable Name** - used to locate the content; (2) an **Organizational Name** - that identifies the content within its site; (3) a **Version** number; and (4) a **Segment** number. Applications fetch data by requesting its name with messages named *Interest*. As the majority of the ICN architectures, packets are also cached on routers for the purpose of serving future requests. This approach has three aims: First, relieving the infrastructure of unnecessary traffic. Second, overcoming the issues of TCP when connectivity is scattering, e.g mobile wireless network. Third, facilitating the development of content-centric services, such as Youtube, by decoupling the application logic from the details of the network topology.

Interest are looked at intermediate router to check if data matching the interest is available in its *Content Store (CS)*. If not, the router looks at its *Pending Interest Table (PIT)* to see if a similar interest is already in flight and the router is already expecting a response. If multiple interests are sent for the same data, then only one is forwarded to retrieve the data. The others are only used to return the data to the consumers, if they are different of each other. If there is no match in the PIT either, then the interest is forwarded towards the source of the named data.

ICN comprises all the architectures where data can be addressed without regard of its location in the network. The final goal is, first, to decouple the application logic from the details of the network topology and, second, to reduce the load on the network by caching content on the routers. The general idea consists in the clients fanning out content requests to the network without need of specifying a location. The requests are then forwarded by the routers towards the content replicas, unless the router itself has a cached version of the content in which case the request is served locally. By convention, content always travel backwards on the trace left by the content request.

2.0.3.1 Named Data Networking

Named Data Networking has been proposed by Van Jacobson et al as in [JST09b, JSB09]. The peculiarity of the protocol is the data naming. Each packet is identified by a unique hierarchical name that, by convention, specifies its producer; what content it belongs to; its version; and its sequence number. Packets are requested one by one using special messages named *Interest*, this method is also known as *flow balance* rule [JST09b]. NDN has been designed as an alternative solution for the world wide Internet and it does not quite take into consideration the dynamics of an ad-hoc network, there is although a preliminary work from Meisel et al [MPZ10] that to the best of our knowledge it has never been implemented.

2.0.3.2 Haggle Mobility Framework

Haggle targets heterogeneous mobile wireless networks, in particular Personal Area Networks (PAN). Differently than other protocols, Haggle allows to request content from its metadata instead of its name. For example, it is possible to request any picture with a certain tag, or any mp3 of a specific rock band. Differently than NDN there is no flow balance between requests and data packets, once the content has been requested a TCP connection is established between the parties for the data exchange.

2.0.3.3 Dona and Mobility First

Both the projects aim at becoming the future Internet architecture leveraging on names resolution to keep track of all the replicas in the network. At the current state of the art, while the architecture takes into consideration mobility and long range wireless communication, ad-hoc networks are not in the picture for both the projects.

CHAPTER 3

Privacy in ICN

This chapter tackles specifically the problem of privacy for ICN, which is inherently critical given that private information can be replicated several times without the control of its owner.

There is obviously a wide agreement on encrypting data before moving them on the public infrastructure but, unfortunately, still no formalization of a privacy protocol. The general idea consists in encrypting the content using a secret key (sk) and a symmetric encryption algorithm, for example AES, so that it can be safely stored on public caches. Therefore when a user wants to access the original data he has to, firstly, request sk from whom generated the content; secondly, download the encrypted content; and, lastly, use sk to decrypt the cyphertext. The key distribution uses an asymmetric encryption protocol of convenience so that privacy is guaranteed.

Limited prior work on ICN privacy exists and is mainly focused on exchanging encryption keys with little regard to system engineering and communication performance.

As a first contribution of this paper, in Section 3.4 we introduce a generic model for privacy protocols in ICN that is designed to justify our architecture choices as well as guide future work on this area. A second contribution is the identification of the existence of **unsafe replicas**, namely, copies of a content that are still present in the caches even though their access control list (ACL) has changed. We show that expecting the content to be eventually removed from the memory of routers is not an acceptable solution since unsafe replicas of popular content can be kept in the network for an unlimited amount of time even without any malicious intervention. To the best of our knowledge this is the first paper to address the unsafe replicas issue, which is a critical flaw that must be tackled in this early

stage of the design.

Ultimately, we use the lessons learnt from our analysis and experiments to formalize a performant privacy protocol that solves the problem of unsafe replicas. The proposed solution offers the following advantages: (1) fine control over access to content (2) timed access policy enforcement in the entire network (3) minimal impact on ICN's pervasive cache performance and (4) no extra communications between the routers.

The remaining of the chapter is organized as follows. In the next section we introduce related work and further extend our motivation. In Section 3.2 we discuss ICN design directives and review existing encryption schemes. Section 3.4 formalizes the properties of privacy protocols. The Section 3.5 presents the simulated behaviour of real network topologies under different settings. The Section 3.7 discusses the results obtained so far and our proposed solution. The paper is concluded with an outlook to future work.

3.1 Related work and Motivation

For what regards security and access control, big content providers rarely ever share willingly the details of their installations. Facebook is an exception giving in [BKL10, HBR13] the details of their photo storage while in [GAL07, HHH12, AGH12] the authors have reverse engineered YouTube and Netflix.

Web-scale services rely on CDNs for scalability and performance. Their architecture layout typically comprises three parties, as in Fig 3.1a: source of content, distribution servers, and domain name systems (DNS). Intuitively, whoever is producing the original content, stays at the top and feeds content to the distributors. DNS has the fundamental role of mapping content requests to machine addresses. Replicas are chosen in function of network location, system load, availability and other heuristics [DMP02].

On the provider's side, privacy is enforced by authenticating the user over a TCP connection secured using SSL or an equivalent standard. By authenticating the user, the server always knows what content the user can access. Bulky content is hosted on the CDN-side,

where this information is not always available, so the CDN servers cannot discriminate what content a user is allowed to see.

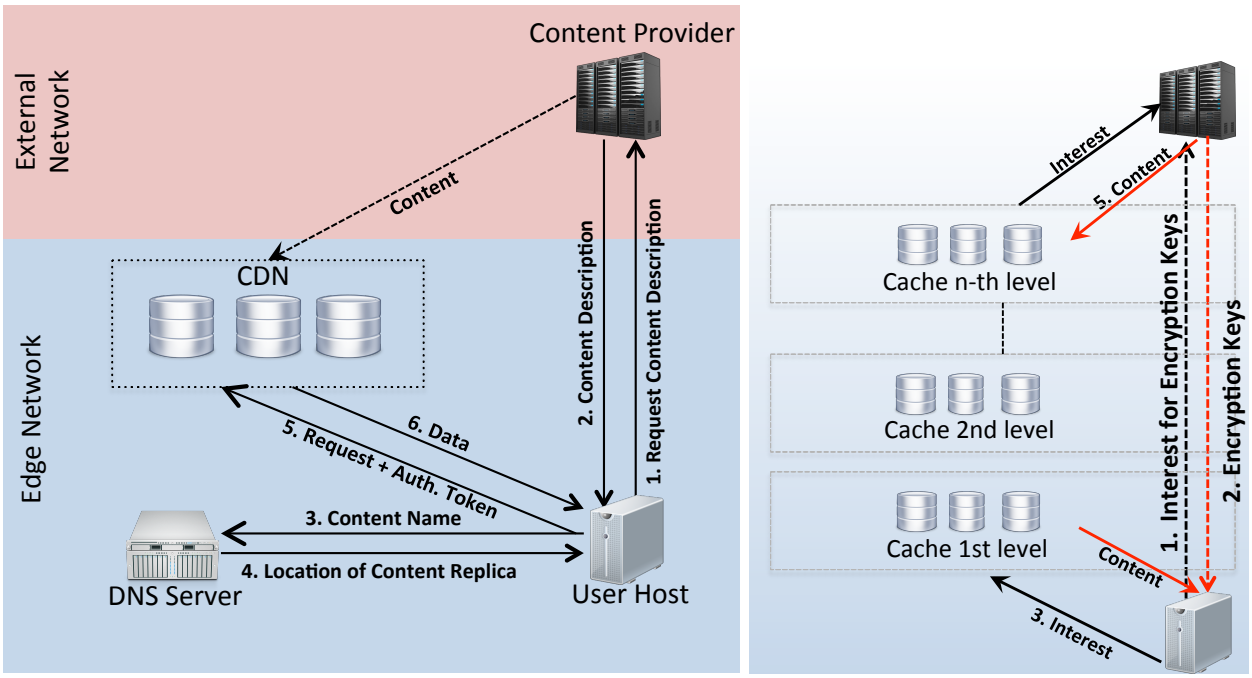
Access control is implemented using an *authentication token* (AT) consisting of a random generated number typically 128 to 256 bits long that the client attaches to its request as a *certificate* of being authorized to see the content. Note that the AT is distributed directly by content provider, who relies on the authenticated connection to know what content the user should be able to fetch.

From a security standpoint, access control is strictly enforced since by-passing the provider trying to guess the AT of a content is physically impossible in reasonable time. Moreover, seldom the name of the content is humanly readable. Instead a flat identifier is used, again 128 to 256 bit long, so to prevent names from: (1) being enumerated (2) leak information about the content, e.g. "Me and Susan at Hawaii."

The same cannot be said of access revocation that, on these systems, is very weak: knowing the content name and its AT is enough to download the content. Anyone can verify this by recording an HTTP request, while logged in to the service, and then replaying from an anonymous connection; the content will be downloaded regularly without any other form of authentication. For instance, anyone that knows the (long and randomized) name of a Facebook image can access it, independently of friendship status with the owner of the image.

Substantially, this glitch is justified by the practical assumption that the software, - e.g the website or the mobile application - is tamperproof for the average end-user, who does not record HTTP traffic, while it is clearly exposed to whoever else has a minimal knowledge of network communications. For the sake of completion, we are confident that ATs are periodically changed in order to avoid exposing the content for too long. However, if that is the case, we did not experience it in our experiments.

In conclusion CDNs, while functional, are not immune to privacy leaks. The questions that remains to be answered is whether ICN can offer the same quality of privacy without sacrificing performance. Throughout the paper we will show that, from the privacy point



(a) Content Delivery Network

(b) ICN

Figure 3.1: Content Delivery Architectures

view, ICN can achieve equivalent performance to CDNs without suffering the problem of unsafe replicas or replayable requests.

3.2 Background

3.2.1 Information Centric Networking

There are many ICN architectures but for security and privacy, they mostly revolve around the same directives. Therefore throughout the whole paper we will refer, unless otherwise stated, to Named Data Network (NDN) [JST09a] as our ICN architecture. However, the security and privacy properties of other ICN architectures could be analyzed in similar manner.

3.2.2 Privacy Frameworks

Throughout the paper we will use a lower case c for referring to content, $\tilde{c}$ to refer to its name, $U_t(c)$ for the set of users u that should have access to it at time t, otherwise referred as the *audience* of c.

We will use standard security notation, $A \rightarrow B : \{m\}_k$ stands for A delivers to B m encrypted with k. Additionally we will write $A^* \rightarrow B : \{m\}_k$ whenever the message can be delivered by A or any cache that has a copy of the message.

In its simplest form privacy is enforced using a Public Key Infrastructure (PKI) [SHF99] or its equivalent decentralized version [RAD03]. Over the years, *broadcast encryption* schemes have been proposed as well to optimize communication.

The main difference between PKI and broadcast encryption lies in the efficiency of communication.

PKI uses RSA [ARS83] or ECC [KMV00] for asymmetric encryption. Both schemes are designed to deliver a confidential message *one-to-one*. Namely, every user u_i has two keys: a public one $K_{pub}(u_i)$ and private one $K_{prv}(u_i)$. They are such that a cyphertext encrypted with $K_{prv}(u_i)$ can only be decrypted using $K_{pub}(u_i)$ and vice-versa.

Translating this into ICN terms, given a content c, $\forall u_i \in U_t(c)$ the communication protocol is:

$$u_i \rightarrow u_P : \{K_{pub}(u_i), \tilde{c}\}$$

$$u_i \rightarrow u_P : \{\tilde{c}\}$$

$$u_P \rightarrow u_i : \{sk\}_{K_{pub}(u_i)}$$

$$u_P^* \rightarrow u_i : \{c\}_{sk}$$

Evidently the requester has to send two interests for every content she wants to download. One interest will fetch a copy of the content encrypted with a secret key while the other will retrieve the secret key encrypted with the user's own public key.

Broadcast encryption schemes achieve better communication performance by encrypting *one-to-many*. Namely, given a message encrypted with the public key, there are many private

keys that can decrypt the cyphertext.

For the purpose of content distribution, this has the advantage that the symmetric key can travel together with the encrypted content, hence the communication protocol can be written as:

$$u_i \rightarrow u_P : \{\tilde{c}\}$$

$$u_P^* \rightarrow u_i : \{\{c\}_{sk}, \{sk\}_{K_{pub}}\}$$

The drawback of using broadcast encryption is key-revocation. When access rules change and a key is revoked, all the others might have to be redistributed as well. The implications of this are later discussed in this paper.

Attribute Based Encryption (ABE) [GPS06], is one type of broadcast encryption that has often been associated with content distribution networks. [YWR10] offers a method for fine control over the keys that can decrypt a cyphertext. Every private key is defined in terms of *attributes*: an attribute is a variable that can be given a boolean or an integer value. Reciprocally content is encrypted using a *policy* consisting of any logical expression over some attributes. For example:

$$(\text{friends} \wedge \text{colleague}) \vee \text{family}$$

can be decrypted by any private key containing the attribute "family" alone or any key that has both "friends" and "colleague". Integer based attributes can be compared to scalar values using equalities or inequalities.

ABE's weakness is that cyphertexts grow linearly with the complexity of the encrypting policy[1]. This issue limits the chances of using integer values, since the encryption algorithm expands every comparison as $a < val$, into up to $\lceil log_2(val) \rceil$ different boolean expressions. The reader is reminded to the original paper for the details [GPS06].

[1]For completeness, new ABE schemes that reduce the length of the cyphertext while incrementing exponentially the length of the private key have been proposed. However due to lack of investigation, they have not been included in this work

A different approach to broadcast encryption is represented by Proxy-ReEncryption schemes (PRE) that use third entities, named proxies, to "transform" the cyphertext encrypted with the public key of a user, say u_A, to a cyphertext encrypted with the public key of another user, say u_B. The communication protocol considering a proxy Pr can be written as follows:

$$u_B \rightarrow u_A : \{\tilde{c}\}$$

$$u_A^* \rightarrow u_i : \{\{c\}_{sk}, \{sk\}_{K_{pub}(u_A)}\}$$

$$u_B \rightarrow Pr : \{sk\}_{K_{pub}(u_A)}$$

$$Pr \rightarrow u_i : \{sk\}_{K_{pub}(u_i)}$$

The benefit of using proxies lies on the fact that they can handle key-revocation in a very efficient way. Specifically they are not able to transform cyphertext for users that have been revoked of their keys by the content producer. Some relevant works that applies PRE schemes to ICN is [WU] [JMB11].

It should be clear by now that the performance of a privacy protocol strictly depends on how fast encryption keys are encrypted and decrypted. For a matter of efficiency and because of the very nature of ICN, it is reasonable to expect secret keys to be cached once they have been decrypted. See Table 3.1 for a comparison of different cryptographic methods.

Throughout the rest of the paper we will consider the clients to have two LRU caches: a content cache, where they store the data objects, and a key cache, where they keep cryptographic material.

A client will then fetch a content as follows: if the content is already stored unencrypted in its local cache the request is served locally, otherwise the content is fetched from the network.

Upon receiving the content the client verifies if its encryption key is cached. If not then:

- **for PKI** it will fetch a new decryption key;
- **for Broadcast Encryption** it decrypts the key again.

Both the content and its encryption key are then moved to the top of the reciprocal caches.

	RSA	ECC	CP-ABE
	[Sym, Cry]	[Sym, Cry]	[GPS06]
Encryption	$0.16ms$	$5.65ms$	$\sim (0.1n)s$
Decryption	$6.08ms$	$3.98ms$	$\sim (0.001n)s$
Cyphertext length	$32B$	$\sim 32B$	$128 + (a + 168)n$
PRE			
	Easier	Chow et al.	
	[JMB11]	[CWY10, WU]	
Encryption	$\sim (0.03n)s$	$\sim 0.1s$	
Decryption	$\sim (0.01n)s$	$\sim 0.1s$	
Cyphertext length	$128 + (a + 212)n$	$3\|\mathbb{G}\| + \|\mathbb{Z}_q^*\|$	
Key-Transformation	N/A	$> 1s$	

Table 3.1: Quantitative properties of Cryptographic Frameworks - ABE-based: the parameter a is the bit length of the attribute names, n is the number of attributes used in the encryption policy. Chow et al. [CWY10] show the cyphertext length is in function of the setup parameters

3.3 Attacks Models

In this paper we want to offer a fair comparison between the state of the art of content distribution, typically based on CDNs, with the solutions available for ICN. For this reason we will not take into consideration security threats that rely on coalitions of users or coalition of users and infrastructure providers, moreover there is very little that can be done in these cases since the problem of malicious users sharing their encryption keys in mathematically unsolvable as in -cite-. A feasible countermeasure, that we will not discuss in this paper, are instead *traitor tracing* algorithms that allow to identify malicious peers so that they can face the consequences of the damage or loss they caused.

A condition that we want to hold for every ICN solution taken into into consideration is that privacy-aware content is never at any time copied unencrypted on public machines. ICN typically solves this problem by design, however problems might arise when third parties, e.g. encryption proxies, participate to the encryption/decryption. More interestingly, we want the content cached in the network to be unaccessible to users that are no longer part of its audience, note that while we can give an asymptotic estimate of the cost of meeting

this condition in the general case we will often relax this assumption limiting the maximum amount of time that an unsafe replica - a replica of the content encrypted with an old key - will be available in the system.

3.4 Analysis

In this section propose a simple, yet insightful, performance model to help interpret the results of our simulations and our following design choices.

3.4.1 Communication Efficiency

We previously mentioned that ABE-based encryption generates cyphertexts of length proportional to the complexity of the encrypting policy. Some PRE schemes and all the PKI algorithms do much better by keeping the cyphertext within a constant length. In general, however, having a secret key for every piece of data poses the problem of communication efficiency without regard to what type of asymmetric encryption is being used.

On the end user side, one would want to use its resources, i.e. bandwidth, to retrieve actual information rather than encryption keys. Hence we define the communication efficiency as:

$$\eta_C^e = \frac{D(\{c\}_s)}{D(\{c\}_s) + D(\{s\}^e)}$$

where $D(\{c\}_s)$ is the length of the content and $D(\{s\}^e)$ is the length of the secret key encrypted using the cryptographic protocol e. In order to keep $\eta^e \sim 1$, one can either use a cryptographic algorithm that keeps the encrypted secret key small in size, or it can re-use s to encrypt larger chunks of data.

One practical example is the following: a Tweet is roughly 256bytes. Using PKI to secure its content would still mean wasting 12% of the resources for delivering the decryption keys. On the other extreme, if we use the same secret key to encrypt several thousand Tweets it becomes almost irrelevant what type of asymmetric encryption is being used.

In the same way, on the server side, keeping the cyphertext of secret keys small is pur-

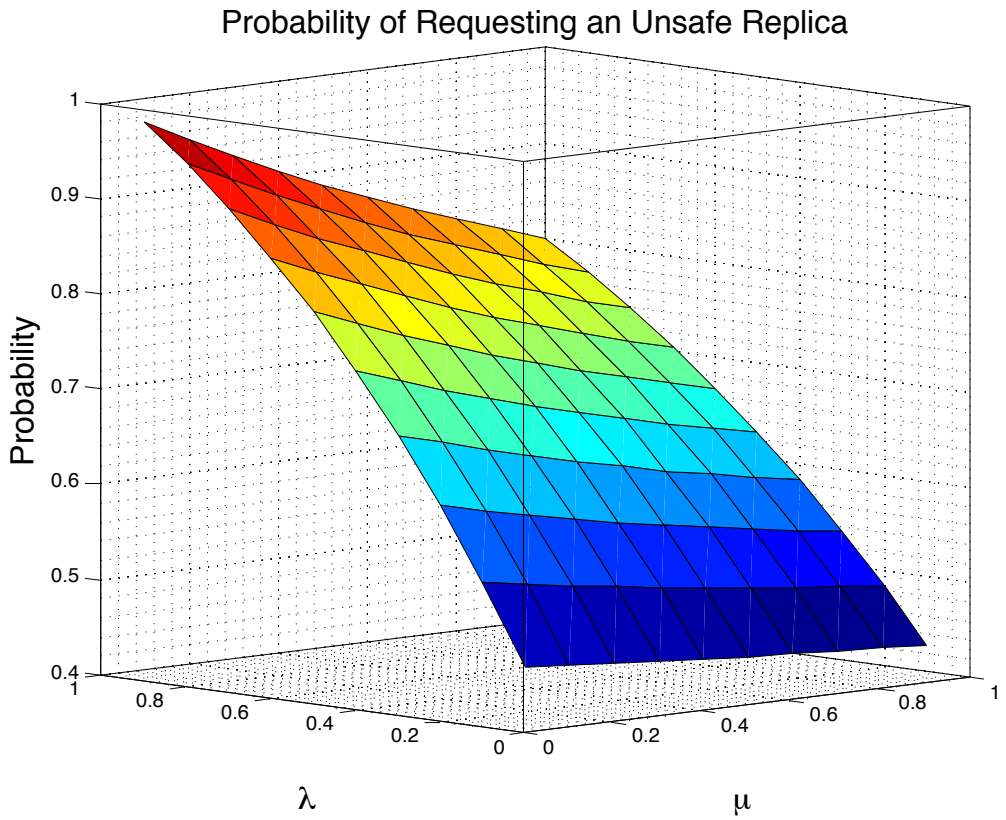


Figure 3.2: Likelihood of unsafe replicas

poseless if then the server has to serve a key to each single user, which can be several millions. Instead the server could encrypt the key once, say with ABE, and then let caching do its job. Hence we define the communication efficiency for the server as:

$$\eta_S^e(t) = \frac{D(\{c\}_s)}{D(\{c\}_s) + \alpha U_t(c) D(\{s\}^e)}$$

. where α is the expected ratio of the audience that has to be served a copy of the secret key.

As a direct consequence of the previous statements, the network topology and its ability of caching must be taken into consideration when designing a privacy protocol for ICN.

3.4.2 Delay and Throughput

A content becomes available to the client once it has received the cyphertext of the encryption key; decrypted it; and ultimately used the retrieved key to decrypt the content.

Putting everything into a sum:

$$T_A^e = rtt_k + T_{enc}(\{s\}_e) + T_{dec}(\{s\}_e)$$

where T_{enc}^e and T_{dec}^e are respectively the encryption and the decryption time of s; and rtt_k is the round trip time for fetching the encryption key. By construction we can conclude that the goodput of the system is then bounded by $1/T_A^e$ for content of unitary length.

For the PKI, its round-trip-time seems to be the discriminating factor since encryption and decryption times are roughly of the same order of magnitude as the latency of a LAN.

ABE has an encryption time that is sensibly higher, albeit the secret key is encrypted only once by the server hence the average latency is supposedly characterized by the decryption time which can be both fast or slow depending on the encryption policy.

PRE performs worse than any other framework since not only the decryption is very slow, the proxy must also transform the cyphertext for the client, which is a slow process and sums up to the extra latency of the network.

Under the same assumption of the previous sections we define define the good-put efficiency as:

$$\eta_T^e = \frac{\frac{D_c + rtt_P}{\mu}}{T_A}$$

where $\frac{D_c}{\mu} + rtt_P$ is the download time for a content c of length D_c given that link has capacity μ and round-trip-time rtt_P. η_T^e implies that if the time needed to obtain the secret key is large then the content must be large enough to make it worth the wait.

3.4.3 Unsafe Replicas

In the introductory part of the paper, we described unsafe replicas as old versions of a content that are still accessible to some clients that have been their access to it revoked. In this section we give a more formal definition of the problem before analyzing it numerically. With the help of Fig. 3.3, let us consider the following events:

t_1**:** u_i has the encryption key s to decrypt $\{c\}_s$

t_2**:** u_i's access to c has been revoked, $\{c\}_{s'}$ is created

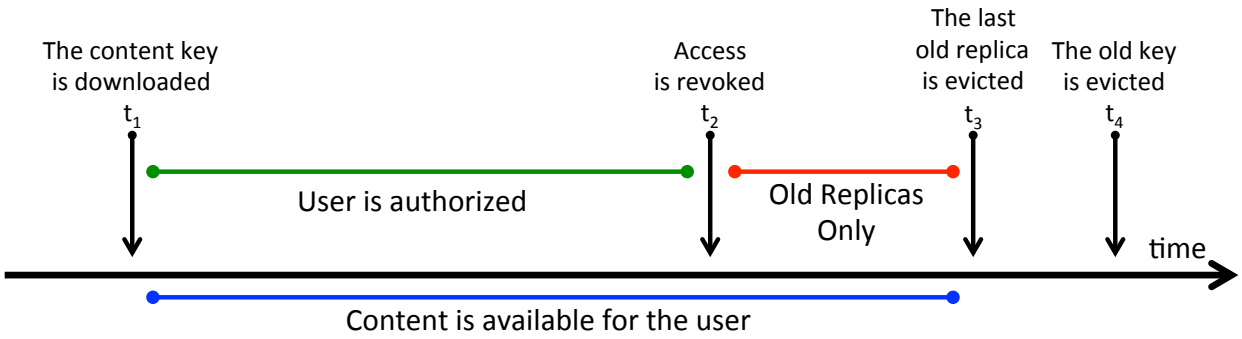


Figure 3.3: Timesequence of unsafe replicas

t_3**:** u_i evicts s from its key cache

t_4**:** There are no remaining copies of $\{c\}_s$, only of $\{c\}_{s'}$ if any.

The quantity that we are looking for is the probability that a request for the content ends up content in the interval $[t_2, t_3]$. The hardness of the task stems from the fact that t_1, t_2, t_3 are all aleatory variables.

Luckily enough, Che et al. gave in [CWT01] an elegant and close approximation for the sojourn time of a of an element in an LRU cache. Namely, they proved that the length of the interval $[t_1, t_3]$ converges asymptotically to a constant, otherwise known as the *characteristic time* of the element. The characteristic time is, in other words, the maximum time lapse between two consecutive cache hit for the same element.

Characteristic times are not known a priori, although considering N cacheable elements, and a cumulative distribution F_i for the request frequency of the element $i \in [1, .., N]$, the following equality holds:

$$\sum_{i=0, i \neq j}^{N} F_i(t < \tau_i) = L_k$$

where $\tau_i \, for \, i \in [1 \dots N]$ is the characteristic time of i and L_k is the cache dimension. A solution for this equality can be easily found using the fixed point method.

For the purpose of our analysis we only need to consider that, given a cache dimension, content accessed more frequently will have the highest characteristic time and that, for a given request distribution, the characteristic time of the elements will grow if the cache size

grows.

Going back to our initial statement problem, knowing that we can express t_3 as a constant, we can determine the probability of a request arriving within the interval $[t_2, t_3]$.

Given a pdf $f_{rev}(t)$ for the revocation frequency of a content, and given a cdf $F_{req}(n, t)$ for the number of requests arriving in a time interval of length t; and under the assumption that the two distributions are independent, we can write:

$$\pi = \int_0^{t_3} f_{rev|t_1}(t) F_{req}(0, t)\, dt + (1 - F_{rev}(t_3))$$

for the probability nπ of not having any access in the critical interval, marked in red in Fig. 3.3.

Assuming that requests are exponentially distributed with frequency λ_{req} then the probability of not having requests in the given interval is Poisson distributed with intensity $\lambda_{req}(t_3 - t_2)$, further assuming t_2 to be exponentially distributed with parameter μ the formula can be rewritten as:

$$\int_0^{t_3} \mu e^{-\mu t} e^{-\lambda(t_3 - t)}\, dt + e^{-\mu t_3}$$

$$\frac{\mu e^{-t_3\lambda + \lambda t - \mu t}}{\lambda - \mu}\Big|_0^{t_3} + e^{-\mu t_3}$$

In Fig. 3.2, we plot the function of for μ and λ_{req} normalized over the length of t_3.

The chart confirms what logic suggests: the more frequently the content is requested the higher is the chance of one request ending up in between a revocation and the eviction of the stale key. At the same time, if credentials are unlikely to be revoked in between the interval $[t_1, t_3]$ the probability drops.

In conclusion we should also consider that content that is accessed more frequently is more likely to be found on the network caches, since it has the longest characteristic time. Hence even if credentials are revoked, the chances of having unsafe replicas in a real network can be very high. We show this experimentally in the next sections.

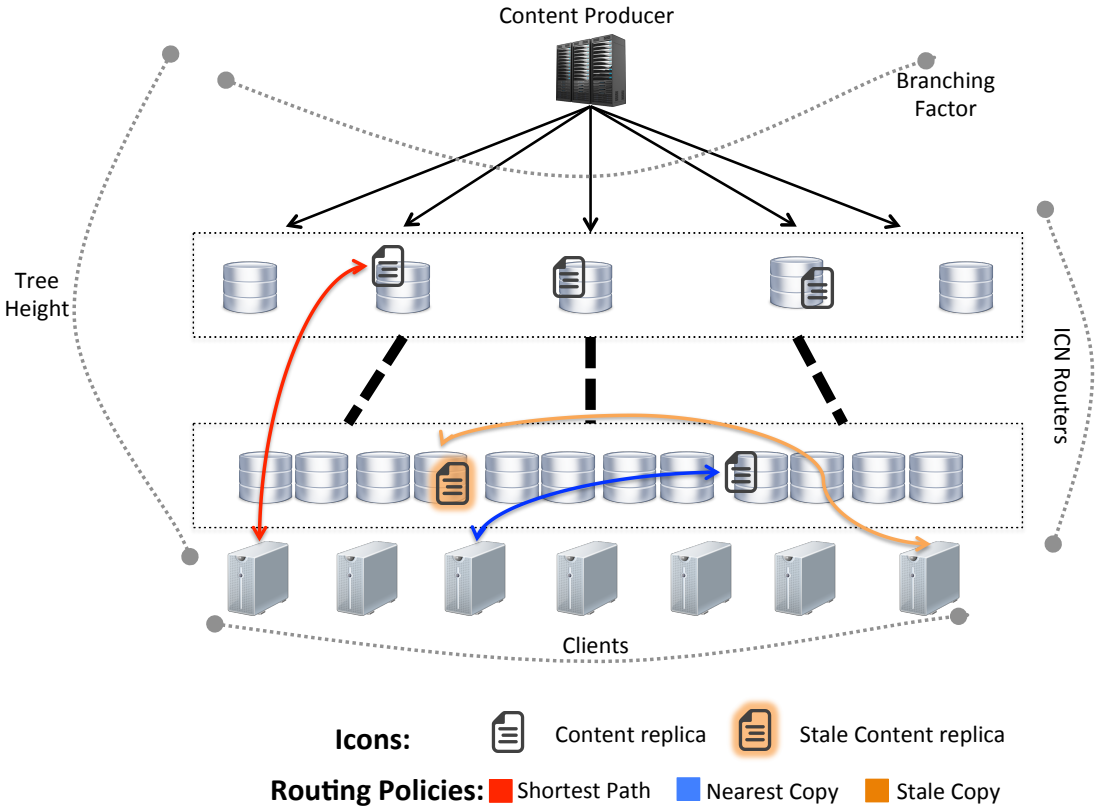


Figure 3.4: Simulated Topology

3.5 Simulation

We developed a simulator in order to study the evolution of a real network. As in [FLT13], we considered a tree-shaped topology, Fig. 3.4, with branching factor $b \in \{2, 3, 4\}$ and height $h \in \{5, 3, 7\}$. Content producers are placed at the root of the tree while clients stay on the leaf nodes.

Every simulated router is equipped with an LRU cache of dimension L_c. Clients have two caches, one for content and one for the decryption keys, which are not cached by the routers. The dimension of the key cache, L_k, is expressed as a multiple of L_c. Content requests are generated at rate λ_{Req}, while content popularity is Zipf distributed with parameter $\alpha \in \{0.6, 0.99, 1.5\}$. The set of all the documents has cardinality $N_d = 10,000$.

We modeled routing accordingly to previous work on ICN and implemented Shortest-Path (SP) and Nearest Copy (NC) which respectively route towards the root of the tree

and, intuitively, towards the closest replica from the requester (if there is one), Fig. 3.4.

We add one extra routing protocol that is meaningful for our analysis that is Stale Content (SC), namely if the leaf node has a stale key its request is routed toward a stale replica of the content, if there is any. Else the request is SP-routed and the new decryption key is fetched from the server. This last protocol has practical application since the original NDN paper [JST09a] proposes to embed the version identifier of a content within its name. If that is the case, a client could, maliciously or not, demand the only version of the content that it is able to decrypt.

A new version of randomly chosen content is created with a frequency λ_{new}, the content to be renewed is chosen based on its popularity. Once the a new version is created, the first request that reaches the server will cause the network to contain two, or more, concurrent versions of the same content. The simulator keeps track of which versions are stored in the network and which versions the clients are able to decrypt so that we can study the phenomenon of unsafe replicas.

Additionally, routers can be configured with a Fixed Caching Time (FCT) that defines the maximum continuous amount of time that a content can sojourn in its cache. Once the time expires, content is evicted from the cache without regard to its position in the LRU cache.

3.6 Results

3.6.1 System Load and Response Time

Firstly, we analyzed how frequently a client has to request the decryption key given different cache sizes, Fig. 3.5a shows the probability of this happening for the most popular document as a function of the Zipf exponent. Fig. 3.5b shows the same for the fifth most popular content in order to show the increment for more unpopular content.

Experimentally $\alpha = 0.9$ is the most realistic value for web requests [FLT13]. In Fig. 3.5c and 3.5d given a key cache with capacity equal to the 5% of the total content, we show the

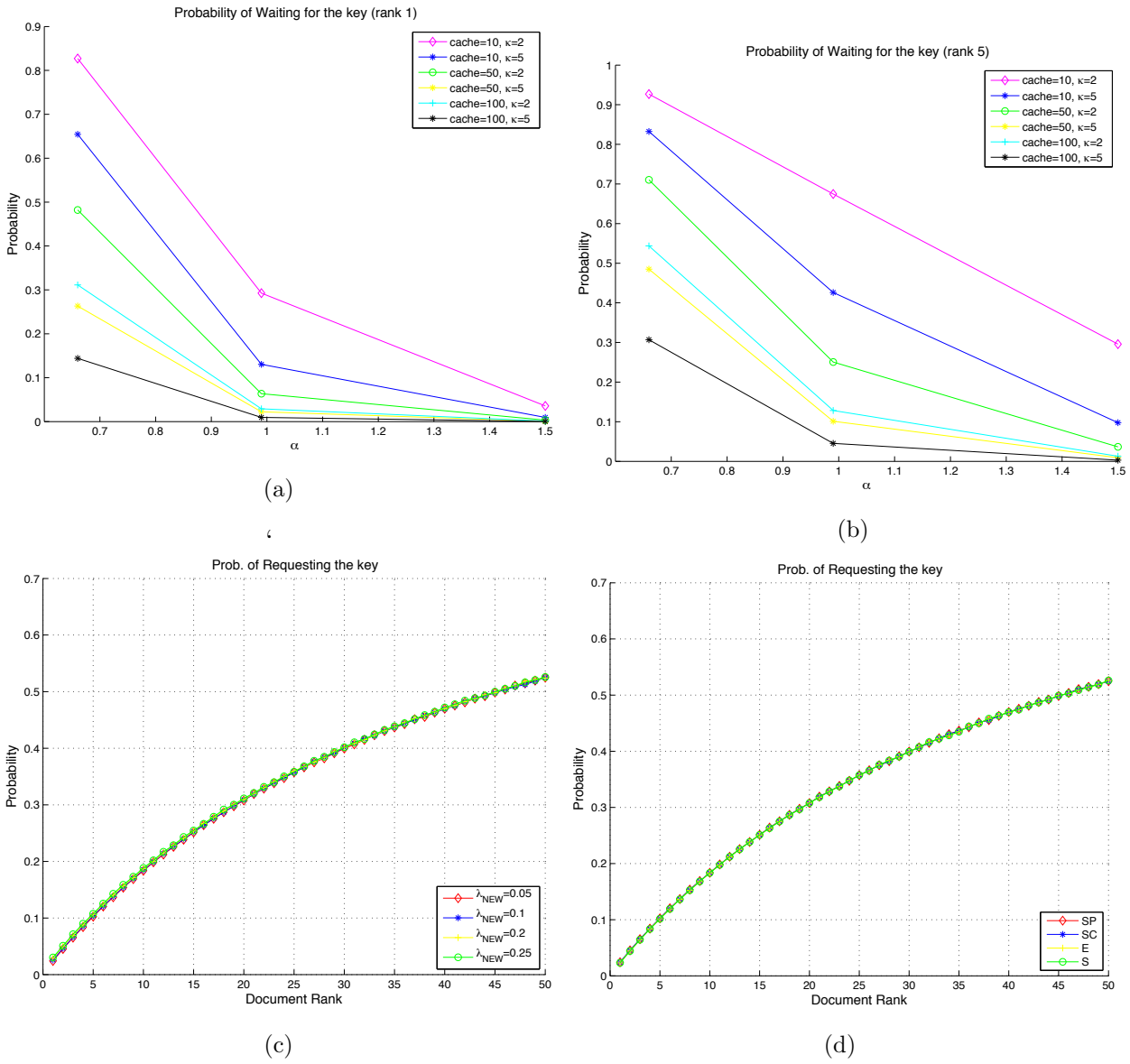
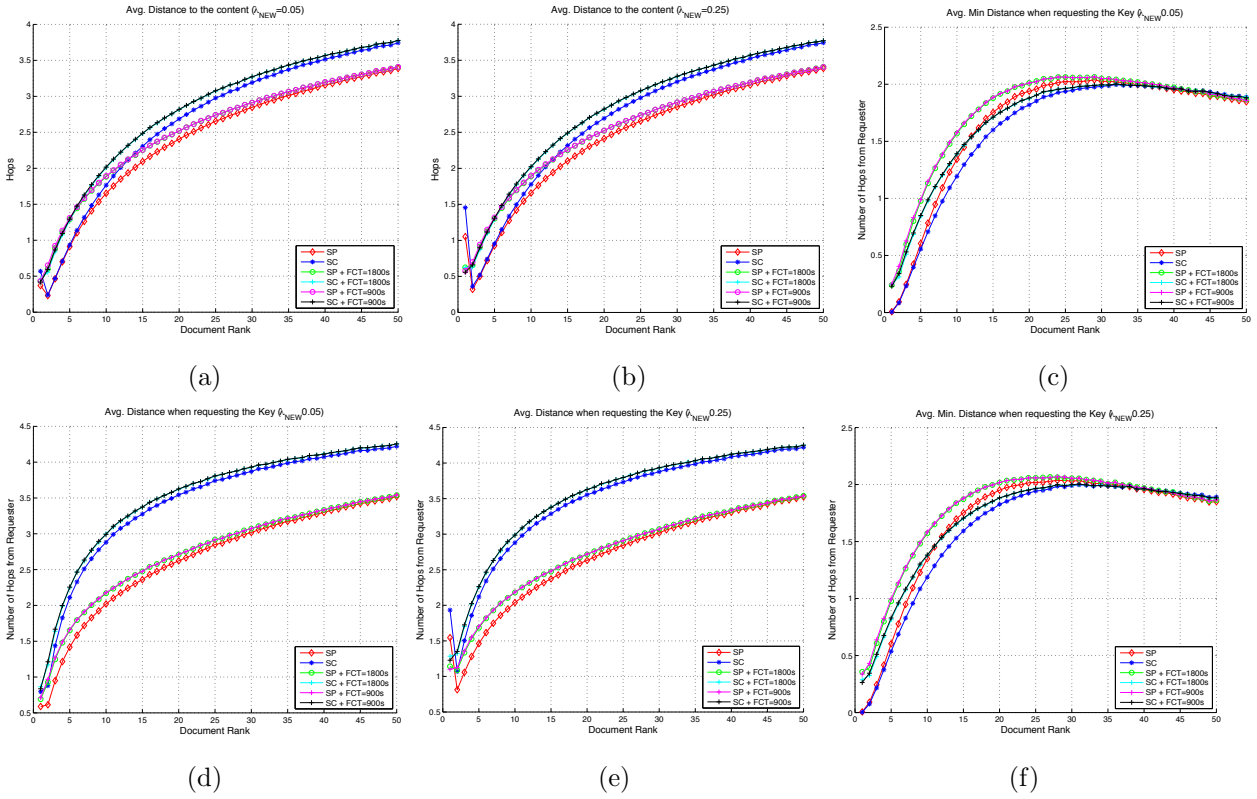


Figure 3.5: Probability of cache miss for the encryption key

probability of key faults given the other parameters of our simulator.

More interestingly, in Fig. 3.6a and 3.6b we show where content is served from for different settings. SP routing offers in general the least delay, the distance between the requester and the content grows as FCT decreases–content is evicted sooner from the routers. SC, follows the same trend but following longer routes. This should not surprise since the requests are always routed towards the stale copies as long as there is any. However the gap is minimal for popular content.



(a) (b) (c)

(d) (e) (f)

Figure 3.6: Distance of the content from the client

Another observation is that the gap between low and high rate of updates is significative only for popular content–the one updated more often and most likely cached in the network.

We said before that the whole purpose of ICN is to favor opportunistic forwarding but in the previous section we also explained how access to the content is bounded by the time needed to fetch the decryption key. In Fig. 3.6c, 3.6d, 3.6e, 3.6f we show this gap by plotting the average and the average minimum distance of the closest replicas when the client does not have the decryption key in its cache.

The chart must be read *cum grano salis*. Specifically one must understand that key-faults are unlikely to happen for the most popular contents. Nevertheless, from Fig. 3.5d it is clear that starting from the 10th most popular content, the chances of having to refetch the decryption key go beyond 33%. This result shows how basic asymmetric encryption is unsuitable for content that is massively replicated, or that is generally expected to be opportunistically retrieved.

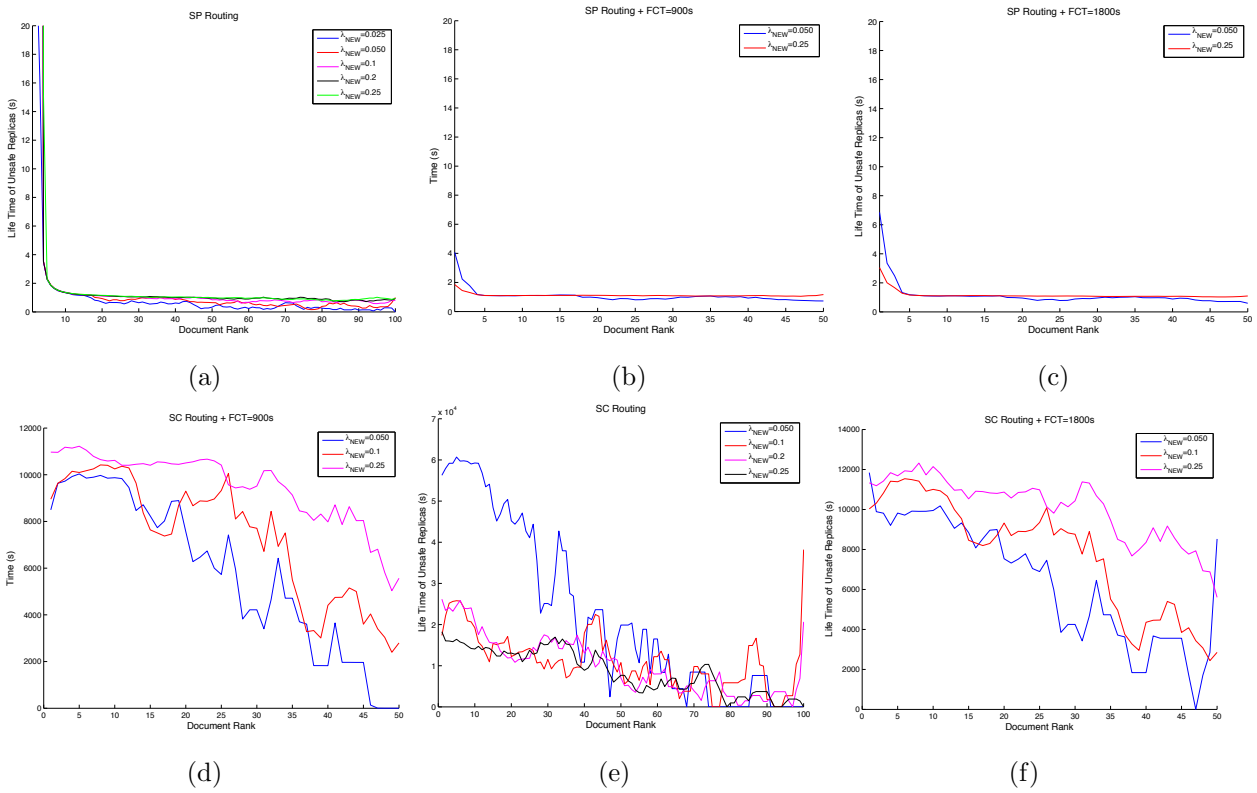


Figure 3.7: Sojourn time of unsafe replicas using SP and SC

3.6.2 Stale content persistency

We experimentally verified our conjectures on stale replicas. This section shows that they are not rare events, even in the simple setting of a small network. In the charts above (Fig 3.7) where requests are generated at a very low rate, $\lambda_{Req} = 10\frac{request}{second}$, but still the sojourn time of stale replicas is almost as long as the simulated experiment.

The charts in Fig. 3.7 show how long the unsafe replicas stay in the system as a function of their popularity. Fig. 3.7a (respectively 3.7b,3.7c) shows the results of routing using SP with no FCT (respectively FCT = 1800s /30 mins, FCT=900s/15 mins). Fig. 3.7d,3.7e,3.7f follow the same pattern but using SC as routing protocol.

At first glance the reader must notice that, on the one hand, we reproduced unsafe replicas using SP as a protocol and we solved the issue by setting FCT on the routers. On the other hand, perhaps surprisingly the problem still remains for SC routing.

The results can easily be explained as follows: using SC, for as long as there is one unsafe replica in the network this can be fetched by any of the clients. Every time the stale copy is fetched from across the network one or more copies are stored on different routers which will start their own timer thus prolonging the sojourn of the stale copy in the network.

This game can be replayed infinitely many times, potentially keeping the replica there forever. Note that this can happen also using NC and SP as routing protocol but the chances of witnessing it in a small scale simulation are unlikely since both the protocol allow only a minimum number of hops between client and content (6 hops in this case).

3.7 GFTC and Epoch Revocation

While each scheme has its own pros and cons, the experiments presented in the previous section, however, evidenced how PKI limits sensibly the throughput of opportunistic retrieval while it does just fine for the distribution of unpopular content that is rarely found at the edge of the network.

At the same time, the experiments showed that unsafe replicas are to be expected. We also know from the previous sections that key-revocation is the main issue of broadcast encryption schemes, while it comes by design with PKI.

In this section we tackle these issues one at the time and propose a protocol based on ABE that is both performant and privacy-aware.

First of all, we tackle the problem of unsafe replicas by proposing a maximum sojourn time for every version of a content in the whole network–rather than a single router.

A producer sends its content to the network specifying a Global Fixed Time of Caching (GFTC) for the content.

Upon receiving the content, say at time t_1, the router keeps track of the time passing and flushes the content whenever the timer expires. Albeit, differently than FTC, whenever the router forwards the content to another router, say at time t_2, it sets $GFTC = t_2 - t_1$.

It is trivial to prove that this prevents stale copies from being copied an unlimited amount

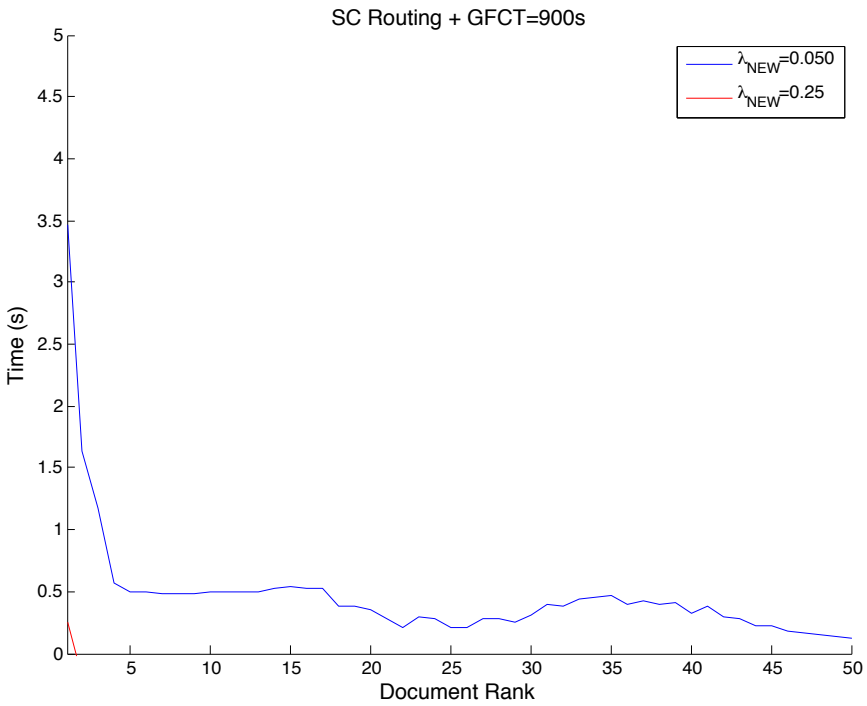


Figure 3.8: Sojourn of unsafe replicas using GFTC

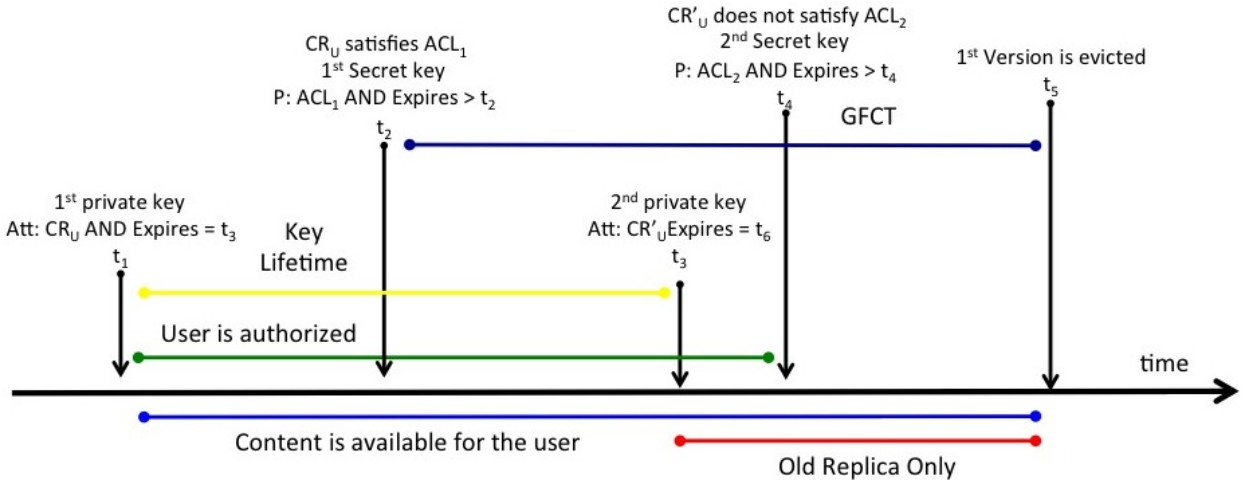


Figure 3.9: Time Sequence of adopting ABE+GFTC

of times even when they are popular in the network. The proof is a direct consequence of the fact that is GTFC is monotonically decreasing over time. The beauty of the algorithm stems from the fact that it does not use any type of coordination between the routers hence introduces no overhead. Its weakness, however, is that it relies on the cooperativeness of the infrastructure. The problem of malicious routers is part of our future work.

GFTC only solves the problem of purging stale content from the network at a given rate but it cannot prevent any new copy from being accessed unless the private key of the client is revoked. By exploiting the fact that GTFC limits the lifetime of a content replica in the network it is possible to use ABE and implement an expiration date for the keys.

Since an attribute can take integer values it can be used to store an expiration date for every private key. Reciprocally, encryption policies can be such to be satisfied solely by keys not expired at the moment of encryption. Fig 3.9 shows this pictorially.

Using standard timestamp, such as UTC, would lead to unbearably large cyphertexts. According to the Table 3.1, it would be of the order of 10KB to 20KB for a 64 bit timestamp. Luckily, It is possible to do better much better with just a few considerations: (1) it is utterly unnecessary to have precision in the order of seconds since routers are not finely synchronized (2) years, decades and so forth are far beyond the expected lifetime of a content in the network thus can be disregarded as well.

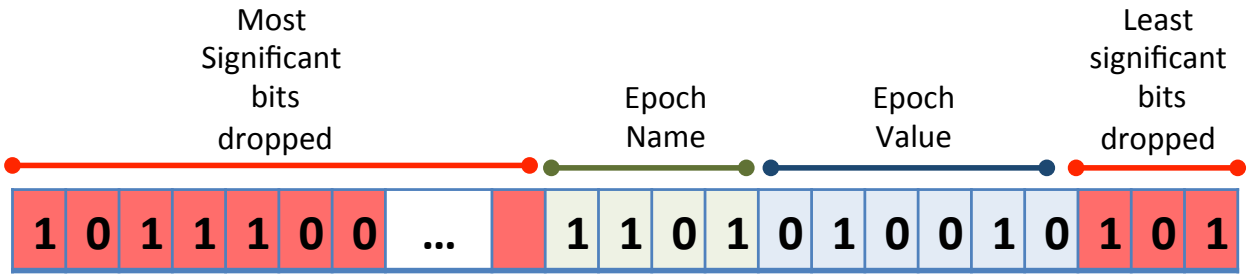


Figure 3.10: Structure of Epochs

Expiration dates can be represented as *epochs*. After dropping the most and the least significant bits of the UTC timestamp, as in Fig. 3.10, the remaining bitmap is split in two parts: a prefix which represents the epoch name and the suffix which represent is current value. Once all the values of the epoch have been used a new one is created at the same way and the keys are redistributed, which is acceptable as long as it happens rarely and in a predictable manner.

Reducing the length of the cyphertext is the real priority since content is downloaded way more frequently than the keys.

Note that the granularity of the epochs remains a parameter of choice, for example: using 8 bits allows both to have a granularity of an hour for 10 days or almost an entire year with a granularity of one day, using 16 bits gives even more freedom while keeping cyphertexts reasonably small, roughly $\sim 600 Bytes$. In general each application will choose what solution best fits.

One last consideration is that using only ABE with expiring keys would not be sufficient to solve the problem. If stale replicas could stay in the network for prolonged time, one could simply use his old private key to access the decryption key and then the content. GFTC makes ABE more suitable for ICN and this was not included in the original ICN design.

3.8 Discussion

In the previous section we intentionally omitted how to handle the case of a cyphertext that stays in the network across two different epochs.

When this happens private keys that contain solely the latest epoch cannot decrypt the content. In this case there are two possible solutions: (1) the server can encrypt using the current epoch as initial threshold and the future epoch as a boolean attribute (which does not cause the same blow up of the cyphertext length) or, more practically, the user that cannot decrypt with the most recent private keys can simply skip the cached versions and force the server to re-encrypt the content with the new credentials.

Another aspect that deserves to be clarified is how to handle *immediate revocation*. In other words, what to do when a content has to be removed immediately. This issue does has not have a universal solution. Setting a proper GFTC is as much as it can be done for the general case. Obviously the lower the GFTC, the higher will be the load on the servers. It must be said that this problem is consistent with every distributed system that relies on replicas.

Something that can be done to prevent highly confidential content from being disclosed by mistake (for instance a compromising picture in a social network), is to initially publish

the content using a very low GFTC and then gradually increase its value as it becomes clear that the user shared the content with the right people. However this aspect has to be taken care of at the application level and not at the network level.

Social Networks are in general challenging for ICN due to the criticality of the content and the lack of control on the replicas. Issues and possible solutions on this topic have been discussed by Angius et al. in [AWG13a].

Ultimately, we owe the reader a justification for choosing ABE rather than another broadcast encryption scheme and proxies in particular. The main reason for choosing ABE is twofold: (1) it is easy in its definition and matches directly the user's perception of access control list, (2) it is already a mature and largely studied technology.

With regard to proxies, while the idea is intriguing it seems to open more problems than the solution it offers.

First of all, there is the problem of location, the nearest proxy to the clients reduces the delay but increases the cost exponentially since one would need proxies at every edge of the network. Moving them far away from the clients would increase a delay that is already the highest among all the frameworks studied this far. Moreover, each proxy should be deployed on an extremely performant machine since the aggregated computational load is expected to be very high. We recall that every encryption key must pass by the proxies first.

Lastly, assuming that each producer has its own proxies does not seem cost-worthy since, if that was the case, then one could as well deploy server replicas near the clients.

3.9 Conclusion and Future Work

In this paper we offered a thorough study of privacy protocols in ICN networks showing the tradeoffs between the already existing solutions.

Unlike previous work, we took into consideration the problem of unsafe replicas that we reproduced in vitro and for which we proposed an elegant and efficient solution.

In the future we plan to extend our analysis to larger topologies and work with real

datasets rather than a synthetic network, this would allow us to fine tune our framework for better performance.

Concurrently, we also intend to study the problem from the software engineering point of view since the developers must be able to focus solely on the application logic rather than the burden of encryption and privacy.

CHAPTER 4

An ICN architecture for Social Networks

Online social networks (OSNs) such as Facebook, Twitter, Google+, Myspace, LinkedIn, etc. have become irreplaceable for staying in touch with friends. The reason behind their success is that they enable forming virtual communities. As a consequence the data shared over an OSN can be very personal and demand immediacy from the provider.

OSNs gather huge amount of data about their users, and are now the real custodians of people's identity. Some OSNs even allow third party's software to use this knowledge[1] in order to improve the user experience. Social data can also be used to infer people's interests in order to win their attention and deliver messages more effectively, such as advertisements [KH10]. The OSN platform can also be used by governments and legal agencies to observe and monitor the user's activities [Ian13].

Despite the wide success of social networking, there are controversial aspects to delegating the ownership of social content to the OSNs providers. Currently OSNs maintain ownership of both the social graph and the shared data [PBB12] and they often share them for profit or for legal interception, sometimes without the explicit consent of the user. Further, some information regarding one user might be subject to the privacy settings of one of his social connections, resulting in privacy leaks.

Recent research efforts [SSN10, PGW08, TL11, DVP12], as well as commercial initiatives [ten], have proposed to decentralize the social network so that users can maintain control over their data. Concurrently, ICN is now mature enough to overcome the network issues that distributed social networking has faced so far. Therefore, we propose a solution that implements all the functionality common to social networks while leaving to the users in full

[1] An example is the protocol Facebook Connect [Fac08].

control of their data. Furthermore, our solution is compatible with the value add of current OSN providers.

In this paper we present the WARP framework, that is: an architecture that allows users to share data, potentially over the infrastructure of an OSN provider, but which prevents the OSN provider to access to the user's data without the explicit consent of the user.

Let us, for instance, consider Facebook. At the moment, the "Facebook application" is a software that runs on their servers in their data center, and the end-user chooses how to interface to it, either via a web-browser or a mobile application. In other words, data is generated at the source by the user, and then moved to Facebook. Then Facebook manages and distributes the data. We want to reverse this scheme in an ICN fashion by having a Facebook application running on the source of the data, i.e. the user device. The application must have the duty of collecting the data from the users, whatever this data is, and pass it to the WARP framework which then takes care of encrypting and distributing the data to the network as a authenticated source.

Note that Facebook can still distribute the data, for instance to improve the performance or offering the distribution service in exchange of showing their advertisement. WARP only guarantees to the user that his data will not be accessible unless he decides to grant to Facebook the right to access it. Our ultimate goal is to show that, with minor changes, ICN does this by design, and that additionally, it is possible to use WARP to build a shared social infrastructure as an overlay of the current Internet that is able to serve both the interests of OSNs and the rights of the users.

The rest of the paper is organized as follows: in the next section, we review the related work for both privacy preserving OSN mechanisms and Information-Centric Networks. In Section 4.2, we provide motivation and technical requirements for our architecture. We then describe our architecture in Section 4.3, with the details of the protocol in Section 6.4. We evaluate the framework in Section 4.5, before concluding the work in Section 4.7.

4.1 Related Work

4.1.1 Information Centric Networking

Information Centric Networking[ADI12] describes network architectures that use data retrieval primitives, i.e. *put/get*, in place of the primitives for machine-to-machine message delivery, i.e. *send/listen*. The final goal is to decouple applications from topology and fetch data from anywhere in the network, including transparent caches. Traffic in the Internet follows a power law distribution where a significant fraction of the traffic comes from a rather restricted number of items. For this reason, as the cost of storage decreases much faster than the cost of bandwidth, inserting storage within the network as proposed by ICN protocols appears to be a valid alternative to the current architecture.

ICNs usually rely on two types of packets, data and requests. Data packets simply contain the content object. The content is uniquely named, signed and, since there is no access control on the network caches, encrypted by their producer for security reasons. Requests, intuitively, are the messages used to fetch data. How to route content requests is a critical issue for ICN protocols. Some architectures, such as [KCC07, DKO13], use name resolvers to locate content. This solution, however, is less responsive to changes and is exposed to attacks, such as DoS. Alternatively, architectures such as [JST09a, SSH07] use routing tables or hybrid schemes. For name resolution, the reader is reminded of [AK13, JBD12].

By design, ICN architectures secure the data instead of the communication channel. However, since the producer has no control over the replicas of its data, their application to social networking is very limited. A trivial example is the following: let us imagine that a content, say A, is initially shared by Alice with all her friends. After a few of her friends have downloaded A, A is replicated on several distributors. If now Alice wants to withdraw the permission of reading A from Bob, she must first ensure that all the copies of A stored in caches are voided and replaced with the new version of A. This is required to avoid that Bob fetches the first version of A that he is able to decrypt. The task is made difficult for Alice since the data could have been replicated several times on caches that are unknown to her.

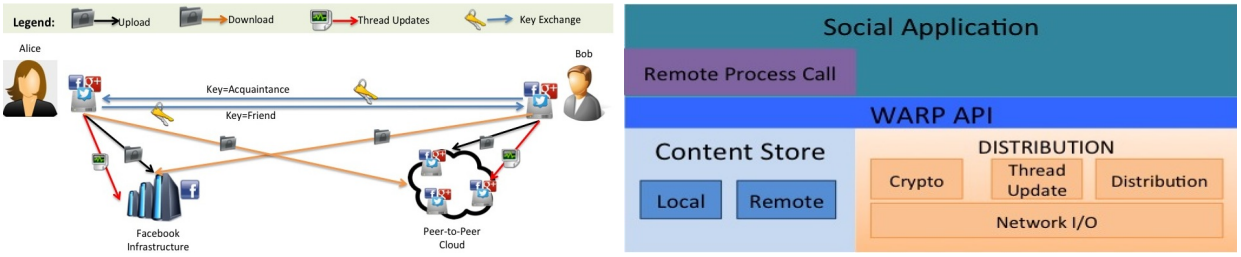
4.1.2 Privacy Preserving Social Content Distribution

PrPl [SSN10] and Musubi [DVP12], have deeply inspired this work since, to the best of our knowledge, these were the first research projects that aimed at decentralizing modern social networks. Despite sharing the same goal, though, their nature is very different from WARP's. In fact, they focus on the application support, and specifically on distributed search indexing. WARP proposes a new network infrastructure for distributing data in an efficient and secure manner. For this reason WARP has to be considered complementary to these projects and integrating all solutions together is the final goal.

Another similar project is Tribler [PGW08]. Tribler, differently from PrPl and Musubi, does take care of the connectivity aspects of distributed OSN although in a differently manner from WARP. It is not a generic architecture and cannot easily interface with the OSN software; moreover it relies on gossiping information across a P2P network and is intrinsically subject to high latencies and provides inadequate real-time notification support.

Cachet [NJM12] addresses a similar problem as WARP. It implements distributed social feeds, policy based encryption and even offline persistency (which WARP does not entirely support at the moment). However, Cachet has three substantial limitations: (1) by admission of its authors, the computational overhead needed to secure access and updates on the DHT is not sustainable; (2) it relies on proxies to re-encode the content and to revoke keys; and (3) it is limited to the news feed type of content whereas WARP allows to develop any kind of software and to enhance it with social functionality. For the sake of precision, Cachet boosts its performance by implementing a gossip-based social caching, although it cannot outperform WARP's structured cache search algorithm.

Other relevant projects include Diaspora [BHG12], PeerSon [BSV09], Safebook [CMS09], LotusNet [AR10] and SCOPE [MNC10]. These projects implements subsets of Cachet's functionality and for this reason will not be discussed further in this paper. The reader is referred to the original papers for the details.



(a) (b)

Figure 4.1: (4.1a) An example of decentralized OSN where Alice chooses to distribute her facebook content using the Facebook infrastructure while Bob chooses a P2P solution using his friends' butlers as distributors. (4.1b) The internals of a butler. Social applications interact locally or remotely via RPC.

4.2 Architectural Requirements

4.2.1 Requirements

Aside from the business incentives issue of protecting the data from the big data analysis of the OSN platform, there are several technical issues in decentralizing the social network. Those corresponds to some requirements that our framework needs to satisfy. We outline these (in no order of importance):

• **Timing:** The Internet is today a real living network. Social content is produced as part of the daily routine and must be delivered in real-time to all its recipients. Real-time feeds, such as the *facebook newsfeed*, usually carry small amounts of data including text, web-links, picture thumbnails, etc. However, they require almost immediate transfer of data, unlike, say, P2P content distribution mechanisms.

• **Visibility and privacy:** People share personal data daily - e.g. opinions, thoughts, political views, pictures, videos etc. - with their friends and acquaintances but not necessarily with everyone. Current solutions rely on authenticated connections and Access Control List (ACL) [SDR13] to discriminate whether a content can be downloaded or not. To implement the same method in a decentralized manner can be computationally unsustainable. However the ICN approach overcomes this issue by securing the data rather than the connection. We

will expand upon this point in the rest of the paper.

• **Scalability:** Social networks sites commonly rely on proprietary large scale data centers for their storage. They also rely on Content Delivery Networks (CDN), such as Akamai, for content delivery. A sample architecture is represented. This solution is highly scalable and fault-tolerant but rarely leaves the user with full control over his data; the users cannot, for instance, decide where unencrypted data are stored, how many times it is replicated and, more importantly, how fast access to data can be revoked. All the previous works minimally use caching: each user is considered responsible for his own content and serves all the requests. In our context, each user is typically a recipient of several hundreds of feeds and also has as many followers. While the current framework uses the OSN servers as aggregators, a decentralized solution opens new scalability issues and a potentially unsustainable overhead. Our solution will have to satisfy this requirement.

4.2.2 High level view of WARP

We propose an open infrastructure to decentralize the distribution of social data whilst enforcing the users' privacy. WARP allows to disseminate social data in an ICN fashion, letting the user decide the right tradeoff between performance, security and control over his own data. The protocol is based on a hierarchical name resolver to facilitate routing without lacking information about users' activities.

Data security is enforced using a combination of symmetric cryptography and Attribute Based Encryption (ABE) similarly to what is done in [JNM11, JMB11] but with the only difference that WARP does not depend on proxies for content updates.

Content updates are taken care of by implementing a two-way agreement between the producer of the original content and the producer of the update. Namely, assuming the case of a comment on a Facebook post, the first user (who owns the post) will agree to link the content update to his Facebook wall as long as the second user gives a signed certificate of what its comment contains and what the name of the data is. By creating this link, the owner of the wall prevents two problems, first: it links only comments that it consider appropriate

and reputable; second, it certifies to his audience that they have not been compromised afterwards; at the same time the content update is bound with its actual producer, who is also be responsible for its distribution.

Instead of implementing a key-revocation protocol, whenever security policies change, the cached objects are deleted, or substituted with a newly encrypted version. This is made possible by using control channels, named *thread updates* that notify content distributors of modifications to their content.

4.2.3 A Motivating Example

With respect to Fig. 4.1a, let us consider the case of two users, Alice and Bob, who want to use a Facebook-like application (FL) built for the WARP architecture.

1. Both the users must have control of a *butler* that will be the guardian and the host of their data. See Section 4.3.2.

2. Independently of the social software Bob and Alice are going to use, they are already able to establish a social relationship as a mere exchange of keys between their butlers. Friendships are defined by assigning a user to one or more categories - e.g. "friend", "colleague" or "family". Let us assume that Alice categorizes Bob as friend whereas Bob categorizes Alice as "colleague".

3. Alice and Bob install on their butler the FL software which has the following duties: offer a user interface, compile the data in the application format, interface with the butler's database to store the data, choose what groups can access to the content created.

4. Alice has no privacy issues; however she does not want to use her own bandwidth to distribute her FL data. Therefore she chooses to distribute everything named under *Alice/facebook/* using the FL infrastructure.

5. Bob is not concerned about bandwidth consumption and does not want to pay a cloud service to distribute his data. Instead he organizes a P2P network with his closest

friends that become distributors of his content. In exchange he does the same for these friends.

6. Alice shares her newsfeed with all her friends and so does Bob. In this case, Bob will be able to follow Alice's feed whereas Alice cannot do the same with Bob's. Moreover Alice will not even be able to see if Bob is writing on his wall because she will not be able to identify what the data objects are.

7. When Bob wants to see Alice's stream, the FL application takes care of identifying all the chunks of the stream, download them, decrypt them and display them on the screen aggregated with all news of Bob's friends.

8. Whenever Alice wants to publish something that Bob should not see she can add to the feed a post with a content specific policy, that excludes Bob from the audience. For instance by replacing the group "friend" with the id of every single friend beside Bob, or creating a new group and distributing the new keys to its members. In this case Bob will be able to see that the network object (because he has access to the stream). However he will not be able to decrypt the post.

9. In order to change the access to a content that has already been published, the user has nothing else to do than simply change the policy on it. The butler will take care of voiding every cached copy. The new copy will be distributed by the butler upon request. Note that when a member is removed from a group, the keys for that group must be redistributed in order to avoid a massive key exchange. For this scenario, we propose a load balancing solution, see section 4.3.3.2.

As we can see in this example, both users select the privacy level they would like to achieve. Users are able to change privacy setting, and to revoke authorization to access content, even though this content has been distributed in the wild.

4.3 Architecture

4.3.1 Data Naming

WARP organizes content in *network objects* that are uniquely named using the following structure:

```
x.y/<folder>/<sub-folder>/.../<appendix>
```

There must be at least two segments divided by a backslash. The first one is always in the form $x.y$ where x is the user name of the producer and y is the Identity Authority (IA) that certifies the identity of the user and his public keys. The remaining part of the name is organized as a file-system, *folders* and *sub-folders* that serve for organization and routing purposes. The last segment, the *appendix*, identifies the network object within its folder. Applications have the duty of obscuring content names, e.g. hashing them, in order to avoid leaking information, as for instance a picture named "Paul and Me in Hawaii" would do.

4.3.2 Butlers and Distributors

A WARP network is a two-tier overlay composed of ***butlers*** and ***distributors***. Every machine in the network must refer to an IA that guarantees for its identity, public keys and network address. Note that the IA does not work as a name resolver for that purpose WARP relies on the DNS service.

Every user in the social network must have a **butler** that serves four fundamental tasks: 1) establishing social links with the other butlers, 2) interfacing with the social applications; 3) storing the user's data; and 4) organizing data distribution. A user exists in the network only via his butler and, from the distribution point of view, all his content is originated there.

Users establish their social relationship directly from the butler interface. The OSN accounts will be dismissed, so that there will be only one global social graph. A relationship is established when the user categorizes the identity of another user. A sketch of a butler's

internals is showed in Fig. 4.1b. At the top of the stack, there is the social application that interfaces with the butler locally or via Remote Process Call. The latter is a strong requirement of our design since content is often generated and consumed from mobile devices. WARP APIs only allow the social application to store their data in the local storage and communicate whether they should be made available to the network.

A butler is always the root source of the user's content. It can distribute the data as a stand-alone server or using *distributors* that behave similarly to ICN caches. A distributor can be: any trustworthy butler, if users want to share their data in a pure peer-to-peer fashion; a paid cloud service, if performance are a concern; or an ad-based service offered, for instance, by the company that developed the social application. WARP distributors differ from ICN caches because they must monitor the updates of the content they store via control channels, named *thread update*, that deliver information about the cached content. Thread updates deliver two types of information: (1) notify security updates, e.g. a newly encrypted version of the content, and (2) notify the creation of new related content in order to allow prefetching.

4.3.3 Cryptography and Credentials

Content is authenticated with signatures generated using any public-key scheme, such as DSA [Inf09] or ECDSA [JMV01], that guarantees confidentiality[2]. At the current state of the art we rely on the IA to authenticate the public keys. However we conjecture that, given the nature of WARP, PGP [Gar95] and a web-of-trust [RAD03] can suffice for the task. In any case we consider the problem of key authentication beyond the scope of this paper.

Privacy is enforced using a combination of symmetric and attribute based encryption (ABE) [GPS06]. ABE is an asymmetric scheme adopted for broadcast encryption [FN94]. It gives fine control over the audience of a private message. The cyphertext is encrypted with a *policy* expressed as a boolean expression. Variables, named *attributes*, are connected by the operators $AND(\wedge)$, $OR(\vee)$ and *k-of-n*. Every user possesses a private-key that is

[2]not all the schemes guarantee confidentiality. For instance, Identity Based Encryption (IBE) [Mar08] does not

associated with some of such attributes and is able to decrypt the cyphertext if these satisfy the encrypting policy. Performance-wise, ABE, as most of asymmetric encryption schemes, is sensibly slower than symmetric cryptography thus WARP secures the data using a secret key (sk) that is encrypted with ABE and distributed together with the cyphertext.

4.3.3.1 Key-distribution

Butlers exchange keys on an as-needed basis, only when some content has to be decrypted and not when the users define their relationship. The reason for doing so is that, as in [PTM06], keys have an expiration date, in the order of a week or a month. If two users do not interact often it is pointless to exchange keys. For the same reason, users should maintain a pool of recent keys in the eventuality that they have to decrypt a content that was encrypted with one of the previous keys.
The butler automatically assigns a random *alias* to every known user so that content can be encrypted for a single user only.

4.3.3.2 Key-Revocation

Key revocation at the scale of a social network is a critical matter due to the fact that users have often hundreds to a thousand friends [BBS09]. Without loss of generality, let us consider the case of a content c encrypted using a single-attribute policy $p = friend$. In order to oust a user from the audience its producer must: (1) create a new version of the attribute, say $friend'$, (2) for each key containing the attribute $friend$ distribute a new one containing the new attribute $friend'$, (3) the butler re-encrypts c with policy $p' = friend'$. Similarly to [JK10], we work around this problem by using aliases of users' identities.
The butler assigns an extra attribute, named ***bucket***, to each known user. A bucket is an integer between 0 and K where K is a constant parameter. When a user, say u_i, is revoked of the attribute, say a, the new policy will be such that the presence of a will be conditioned to the presence of every bucket beside the one assigned to u_i or the individual identity of every other user that shares his bucket with u_i. In practice, let us consider a content C encrypted

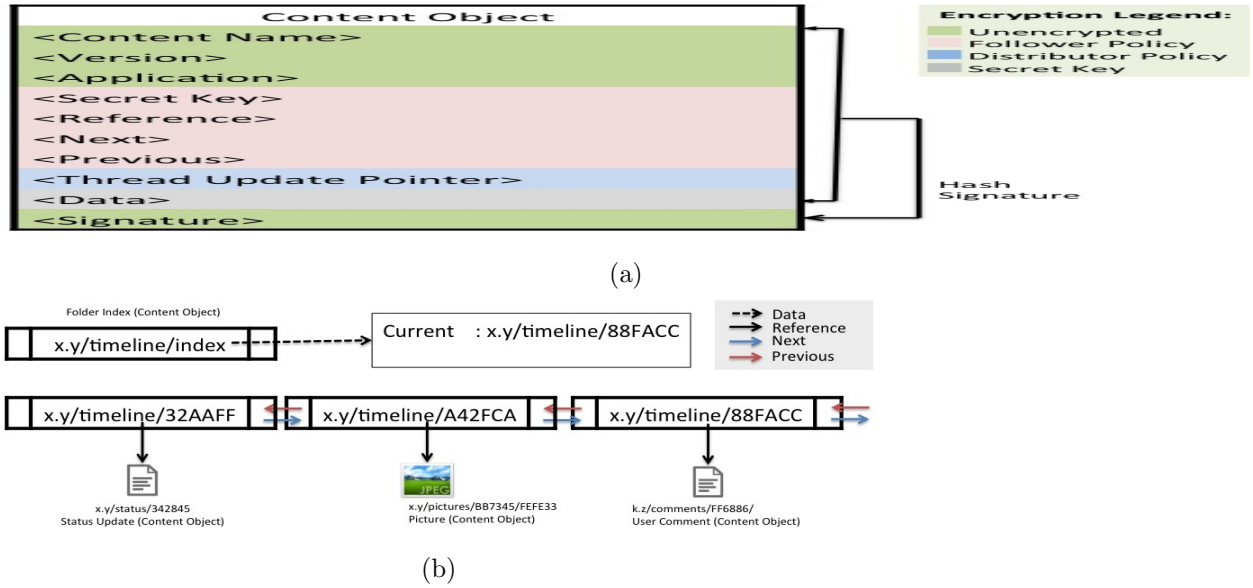


Figure 4.2: (4.2a) A network object (4.2b) A possible organization of network objects to recreate a Facebook-like timeline. The index of the folder `x.y/timeline` keeps track of the latest segment of the news feed which is organized as a WARP feed. Each entry of the timeline reference an external object, from left to right: a status update from the user, a picture from another application, and the comment of another user. The social application "timeline" will take care of aggregating the content and show it to the user.

using a policy $P = att_1$, and four users u_1, u_2, u_3, u_4 distributed into two buckets, respectively $B_1 = \{u_1, u_2\}$ and $B_2 = \{u_3, u_4\}$. In order to prevent the user u_4 from decrypting C this will be re-encrypted using a new policy $P' = (att_1\, AND\, B_1)\, OR\, u_3$. This approach prevents from the redistribution of new keys to all the users whose keys contains att_1 and prevents the number of attributes from growing linearly with the number of users as long as the parameter K is chosen adequately. Moreover, since keys are redistributed periodically, the impaired policy will be enforced only until the new attributes will be generated.

4.3.4 Content

Data in the network is organized as a collection of *network objects*, represented in Fig. 4.2a. An object can contain data, reference another object or both. Each object virtually belongs

to a doubly-linked list thus it maintains a pointer to its successor and predecessor. Note that, while connecting objects in a list is convenient for publishing unbounded streams of data, such as the Facebook Timeline, it is not efficient for random access or to download multiple objects in parallel. One way of solving the problem is to make object names enumerable, for example, given a folder `/x.y/photos/AFAFA/)`, a seed s and a hashing function h the object could be named `/x.y/photos/AFAFA/h(seed,1))`, `/x.y/photos/AFAFA/h(seed,2))` and so forth. This naming convention is more efficient but has the disadvantage that if one wants to ban a member of the audience, making it impossible for him to trace new content, all the names would have to be changed. At this stage of research we want to keep our design as flexible as possible and for this reason we leave to the application developer to choose an appropriate naming convention and wether to facilitate sequential or random access. Public fields contain general information about the object, they are: **Content Name**,**Version** which specifies what decryption key to use, and **Signature** that authenticates the content. Followers fields are accessible only to the audience of the content, i.e. people who are granted a valid key, they are: **Application** that specifies what application created the content, **Secret Key** the key used to encrypt the data, and **Reference** that is used to link another object in the network. **Next** and **Previous** are, respectively, a pointer to the successor and the predecessor of the object in its stream. Ultimately, the field **Thread Update Pointer** is an identifier, readable only by distributors, that specifies at what moment the object was published, distributors use it to monitor future changes to the object or its folder.

4.4 Protocol

4.4.1 Protocol Messages

As discussed in Section 6.2, ICN protocols generally use two types of messages, requests and data. WARP introduces two additional types of messages: ***RESOLVE*** and ***NOTIFY***.

A **RESOLVE** message is used to obtain a list of distributors for a given folder. This type of query can be answered authoritatively only by the producer of the content or a distributor

for a parent folder. In other words, considering the name `x.y/folder1/folder2/name1` a butler that does not know a distributor for `x.y/folder1/folder2/` can obtain a list of distributors by sending a query to the butler `x.y` or any distributor for `x.y/folder1/`. This far we have not mentioned how to resolve machine addresses. Generally speaking since both butlers and distributors rely on the IA to certify their identity, as long as the identity is associated with a domain name the name can be resolved using the normal DNS infrastructure.

NOTIFY messages are one of the core ideas of WARP. They are sent by a butler to another to inform that a content has been created. They are fundamental to implement common functionality - such as comments, re-post, likes and re-tweets - since in WARP a network object cannot be modified by anybody that is not its producer. By design, content aggregation is operated at the application level rather than at the storage level. Namely, assuming the case of Alice adding a post on her Facebook wall and Bob commenting on it, in WARP there would be three distinct objects. Alice's post, Bob's comment and Alice's link to Bob's comment. A notify message contains a content-name, a signed checksum of the related data and an application identifier that says which application has generated the content.

4.4.2 Thread Updates

Thread Updates (TUs) are simple publisher/subscriber channels between a user butler and its distributors. Every TU binds to a folder and delivers messages containing the changes made to that folder and all its sub-folders. Changes made to the butler storage are univocally named using always increasing $identifiers$ so that the evolution of the content can be placed on a timeline. Distributors declare from what point in time the want to receive the messages using the TU pointers in their content. A distributor chooses which channel to subscribe based on the content that it has cached, e.g. if it has stored two objects, say `x.y/folder1/object1` and `x.y/folder2/object2`, it can choose whether to follow their directory exclusively or the whole tree `x.y/`, in any case changes are never forwarded twice

since they are univocally identifiable.

The commands that a distributor receives from a TU are the following:
ADD(X) X has been created. The distributor can decide whether to prefetch X or not.
DELETE(X) X must be purged from the cache and never distributed again.
UPDATE(X,Y) X has been updated with Y. The distributor must purge X, and can decide whether to download Y or not.
CUT(X,Y,X',Y') A feed has been cut from X to Y. All the objects in between must be purged from the cache, the distributor can decide whether to download X' and Y' to merge the remaining pieces.

For our future work we are considering implementing a command that distributes only a differential of an updated object containing renewed version of the secret key and a new signature. However there are security issues to take into account and we do not consider this optimization at the moment.

4.4.3 Fetching of data

Butlers and distributors maintain a routing table to direct their request. Each entry of the routing table contains a folder, a distributor and the expiration of the content. New requests that do not match any entry of the table trigger the butler to resolve the name and add a new entry to the routing table.

4.4.4 Serving Data

In the generic ICN architecture, whenever a cache, or a router, is requested for a content it verifies if there is a copy in its storage. Otherwise it forwards the request to another cache to obtain a copy. Whenever the copy arrives, it can decide whether it should be cached or not. WARP approach is slightly different since it must consider the thread updates. The logic for serving an incoming request for a content R:

1. The distributor verifies whether the content is in the Content Store, if not it continues at

point 5.

2. Consider TU_R the Thread Update of R, if TU_R has been recently read the content is considered updated, then continue at 6.

3. Fetch TU_R.

4. Read TU_R and update the Content Store accordingly, if the content is still updated continue at 7.

5. Fetch R from another distributor

6. Serve Request

7. If R is not a popular content then terminate.

8. If TU_R or any of its parent TUs are balready being followed, add it to the list of threads that must be followed

10. Cache R in the Content Store

4.4.5 Continuous Polling

A possible attack that can tamper the cache control mechanism of WARP is **continuous polling**: a malicious butler generates requests at a high frequency so that it can receive every version of a content before being excluded from the followers. This problem can be solved on the distributor side by implementing a *Request Ban Window* (RBW), for every open connection distributors keep track of when each a butler is allowed to send a new request, if a request arrives before then they do not serve it. Also, whenever they receive a request from the butler they update the RBW to the maximum between its current value and the value of a function:

$$T_{RBW}(t) = \text{current time} + \left\{ \frac{fail(t, \delta)}{1 + succ(t, \delta)} \cdot \delta \right.$$

where $fail(t, delta)$ and $succ(t, \delta)$ respectively are the number of failed and successful requests that a butler has sent in the latest time window of length *delta*. Note that $T_{RBW}(t)$ scales the RBW δ according to the *hit-ratio* of the butlers request, if the requests are mostly successful - i.e. there is a burst of data to download - the butler is allowed to send requests

faster, differently, if the requests are unsuccessful the time window grows. Butlers that respect the suggested inter-request time will never have their requests rejected whereas butlers that do not will delay their opportunity of downloading.

4.4.6 Application

Due to space constraints, we cannot give an extensive discussion on WARP's APIs although in Fig **??** we outline the content design of Facebook-like timeline. The figures show how easily the timeline can be embedded into a WARP's feed to notify the creation of new data objects. The index of the application maintains a reference to the latest portion of the feed so that an application can always request only the latest news. Content is generated by social applications that store it into the butler's DB via APIs, for this purpose WARP APIs offer all the CRUD[3] functionality. Additionally, the application is allowed to compile the index file of its folder, create feeds and map DB content into network objects. Moreover, they can subscribe to be notified whenever a NOTIFY message arrives to the butler. Social applications that run directly on the butler can be prevented from accessing to the network and distribute unencrypted data of the user, their mobile instances however could maliciously read the content storage and transfer the data somewhere else. In order to prevent this problem mobile instances can only read from the database the data that they produced.

4.5 Evaluation

4.5.0.1 Key-Revocation

As opposed to what was done by [NJM12, JNM11, JMB11, GTF08] for social networking and by [XM12] for cloud storage, we decided not to use proxies for the key revocation process. While we do not consider the trustworthiness of proxy a real issue, since all the centralized part of such systems are generally assumed reputable, our concern are centered around scalability. Proxies are intermediaries that take part in every decrypt operation.

[3]common acronym for Create, Read, Update and Delete

This would imply that every data transfer in the network would at one point end in a proxy transaction, and this would include very simple operations such as, for instance, likes on Facebook. Moreover, some proxies need part of the cyphertext in order to complete the transaction which would cause an additional cost in terms of redundant network traffic that we are trying to avoid. In WARP key revocation maps into voiding cached content and re-encoding the content upon request. It is important to remember that interest for social content has generally a very short life-span. Therefore, as long as access rules are properly set when the content is generated, re-encoding might not even be used without the need to pay the cost of a proxy transaction every time the content is accessed.

Another observation that can be made about WARP's revocation procedure is whether the dimension of the encoding policy grows linearly with the number of attributes, as discussed in section 4.3.3.2. Attributes are expanded only and if the content has to be re-encoded during its *popular* period. Since keys are periodically changed, attributes can be shrunk again at the first key renewal. Furthermore we load-balance the problem by uniformly sub-dividing the users into buckets. As long as the number of buckets is properly chosen [AWG13b], the chances of using high-number of attributes are low. As a ground rule we do not consider policy changes to be frequent enough to justify the network and computational overhead of using proxies.

4.5.0.2 Thread-Updates

Every cache should periodically ping a control channel to verify that the content of its cache is still valid. There might be a concern regarding the overhead induced by the TUs. To answer this question, the reader must consider that thread updates can be cached and re-distributed as well. This implies that on the butler side the cost of delivering such information can be very low if the distribution chain is properly set. Moreover, on the distributor side, the cost is still fairly low considered that policy are not updated frequently and that request for updates can be sent with a granularity of choice, not to mention that they have the bandwidth occupation of only a few packets. For instance, a distributor can commit to

enforce new policies only once every 30 minutes.

4.5.0.3 Distribution

Data distribution in the WARP network is intentionally kept as flexible as possible. The solutions mentioned earlier - i.e. centralized, peer-to-peer and multi-tier - are not offered by any previous work. Cachet has the advantage of offering offline data distribution which WARP does not offer because the butler has to take action after receiving a NOTIFY message, although we consider this a minor considered that the whole point of WARP is to prevent personal data to be replicated without control. Beside this aspect though, performance-wise there is not a real comparison between the two protocols. As Cachet is DHT-based content, it requires on average O(logN) steps away to reach the requester. WARP on the other hand can resolve the list of distributors in constant time without regard to the amount N of users that can be in the order of tens of millions to even billions. Their use of social caching, while effective, relies on the number of connections opened and, as shown in their result, is effective only as long as requests are sent to about 40% of the contacts. WARP structured approach never sends a request to more than one distributor.

4.6 Discussion

Misuse of social networks intrinsically leads to lack of privacy, and WARP cannot prevent poor decisions in content sharing. If a user is initially granted access to a content and manages to download it before the permissions have been revoked, little can be done beside trusting that the butler has not been compromised, namely the butler should cache content, for performance reasons, but not maintain a persistent copy of it. WARP, as Facebook and the other OSNs, has a best-effort approach. Once data have been delivered to a follower there is no guarantee that they will not be duplicated by the user. However, if they are not duplicated, subsequent downloads will be denied to access the content. Users can aid by maintaining a web-of-trust and avoiding sharing content with people who are known to use butlers that are not conform to the standard.

While this problem is common to all the solutions mentioned in 6.2 WARP has the advantage of letting the user choose the right tradeoff of privacy, performance and cost. At one extreme, a user may want the highest level of privacy and distributes his own data himself. Such user can be any artist who wants to have a fine-control over who access his material. At the other extreme, a user and his friends can organize a peer-to-peer network because they do not want to support a private company. In the first case, the cost is high but privacy is guaranteed because, not only the content is encrypted but also connections can be authenticated. In the latter case, there is no cost at all but privacy is not guaranteed if one or more users are malicious.

In our belief, most of the users will compromise by distributing their content via a OSN, or cloud storage, because it will allow maintaining the same performance of the current network but without lacking privacy on the distribution side.

4.7 Conclusion and Future Work

In this paper we presented WARP: a scalable ICN architecture that supports social networking with no limitations on both the user and the vendor side. The main contribution of WARP consists in offering a unique solution to the most recent privacy violations and to the current limitations of Information Centric Networking. A first version of the architecture has been implemented on top of CCNx [Par13], working around the limitation of the NDN architecture. While it helped discovering all the issues that we tackled with this work, the implementation is far from the current design of WARP. Future work consists of re-writing the network logic of WARP and to release the first sample applications. Additionally, our priority is also to explore the scientific aspects of ICN and social network, we believe that there is a lot that can be done, especially for caching, leveraging on the information given by the social graph. Ultimately, while bulk data distribution will be most likely operated by third parties, we believe that real-time content such as status updates, tweets and messages can be distributed in a P2P fashion.

CHAPTER 5

Many-To-Many Communication in ICN

This ICN architecture works well for user-to-file or for multicast download communications. However, in many-to-many communications, it runs into some issues. Namely, if multiple users share the same prefix as part of an application, either all the users sharing the prefix must tightly synchronize the content they generate into a single stream, or the end users will not receive all the packets generated within the application.

Many applications involve more than a pair of users. Applications such as swarms in peer-to-peer systems, communities in social networks, large scale video- or voice-conferences, multi-users chat rooms, on-line multi-player games all can be engineered to be natively supported in a CCN architecture. This is the object of this paper: to design the service primitives to efficiently support large scale applications which involve multiple users both as source and destination of a shared content, without requiring an explicit synchronization mechanism.

NDN efficiently support asymmetrical applications, where multiple consumers receive the content generated by a source. As such, it is a solid base for collaborative peer-to-peer applications with a focus on mobile environments such as participatory sensing, social networks, content distribution in social networks, online games etc. Unfortunately, while moving data downstream is the raison d'etre of NDN, making many hosts "talk" to each other in a symmetrical manner is not equally simple. NDN discontinues the former conversational pattern of TCP/IP by balancing data requests and data packets, i.e. a client receives only the packets that it explicitly requested. For this reason a host A can send a message to B only if B knows that such message is going to be generated and what its name (or at least its name prefix) is going to be. In communications involving many users using a shared channel,

such knowledge might not be readily available.

We solve this problem by suggesting a method to initiate a communication between multiple parties. Furthermore, with the aim of using NDN for larger scale collaborative peer-to-peer systems, we raise the issue of sending messages between hosts that are, at least initially, oblivious of each others, which often happens in P2P networks where new peers can join the network in an unpredictable manner. As a first step, we analyze how CCNChat[1] implements many-to-many communication and observe how it leads to a concrete possibility of data-loss due to the unbalance between the time needed to send an interest and the frequency with which messages are generated. The higher the packet rate (or the number of participants), the more messages are lost by the client applications. This paper proposes the Prefix Hopping (PH) algorithm as a solution to this issue. PH allows multiple participant to share a number of prefixes within the same distributed application, so as to increase the rate at which clients can retrieve data from the network without requiring the full name for the data. We validate our design with an implementation which confirms the expected performance improvement in a many-to-many communication application.

The rest of the paper is organized as follows: section 6.2 briefly goes over the NDN protocol and its functionalities,and presents some related works. We then present the system model and analysis in Section 5.2. We then describe our implementation of PH in Section 6.6 and the experimental results are presented there as well. Section 6.8 closes the paper by highlighting some open issues.

5.1 Background

5.1.1 NDN Primer and Distributed protocols in NDN

NDN names data hierarchically so that each packet embeds the name of the content it belongs to. In addition, as a convention, each name should include (1) a **Globally Routable Name** - used to locate the content; (2) an **Organizational Name** - that identifies the content

[1]CCNChat is a sample application included with the official NDN distribution which uses multicast to implement Instant Messaging.

within its site; (3) a **Version** number; and (4) a **Segment** number. Applications fetch data by requesting its name with messages named *Interest*. As the majority of the ICN architectures, packets are also cached on routers for the purpose of serving future requests. This approach has three aims: First, relieving the infrastructure of unnecessary traffic. Second, overcoming the issues of TCP when connectivity is scattering, e.g mobile wireless network. Third, facilitating the development of content-centric services, such as Youtube, by decoupling the application logic from the details of the network topology.

Interest are looked at intermediate router to check if data matching the interest is available in its *Content Store (CS)*. If not, the router looks at its *Pending Interest Table (PIT)* to see if a similar interest is already in flight and the router is already expecting a response. If multiple interests are sent for the same data, then only one is forwarded to retrieve the data. The others are only used to return the data to the consumers, if they are different of each other. If there is no match in the PIT either, then the interest is forwarded towards the source of the named data.

5.1.2 Related Work

Zhu et al. in [**?**] have prototyped a server-less conferencing tool based on NDN while Wei et al. in [**?**] have proposed a hybrid approach. The former discusses a static scenario, where speakers are a foreknown by the conference organizer, whereas the latter uses a *control plane* and leverages existing tools to handle authentication; users join and leave; etc. In addition, similarly to this work Zhu et al., with Chronos [**?**], used hash digests to avoid the consistency issues that are further discussed in this paper. However, Chronos considers the specific applications of instant messaging that has an inherent low rate of updates and has a usually small number of participants. Since it calculates a SHA-1 digests for each generated packet, Chronos is less scalable than PH. Furthermore it requires each user to download the digest of all the other users to keep the participants synchronized whereas PH is meant for scenarios where this condition is not satisfied.

The use of a single shared prefix is justified whenever the network has dynamics that

makes impossible to keep track of all the sources of content. In this paper we consider specifically the case where content updates occur frequently and in an unpredictable manner, a condition often met by collaborative p2p networks where users participate participate to shared activity in a discontinuous manner, moreover this type of application can be deployed on different topology scales, e.g. from Personal Area Network (PAN) to the Urban Area Network (UAN). An immediate target for PH is a twitter-like bulletin board where each hash-tag maps into a content name that users can use as prefix for their messages name. While a single users produces messages at a relatively low rate, cumulatively the rate is more than the current state of the art can handle.

Other large scale projects for content centric networking are [**?**] where Burke et al. discussed their importance for urban planning, public health, creative expression and resource management; [**?**] where Pau et al uses networked cars for pollution monitoring. Differently, Keller et al. in [**?**] implemented cooperative download on mobile devices based on network, this is an example of application that, despite the relatively small topology, cannot name machines in order to maintain its configuration-less nature. NDN is a good match for these scenarios because they are characterized by: (1) temporal locality - the utility of the data last more than a single message exchange - and (2) spatial locality - Data do not need to be reachable from everywhere or by everybody.

5.2 Prefix Hopping: Problem Statement, Model and Analysis

When the router is already waiting for data, it does not forward the same interest twice. This has the implication that from point A to point B the maximum frequency for sending interest is bounded by the inverse of the Round Trip Time (RTT). A graphical explanation is given in figure 5.1. If an interest carries the generic name of a content it is not possible to predict what packet will be retrieved from the routers cache, as any match can be retrieved. Currently, there are no specifications about what should be cached on the routers and what content has priority when replying to a generic interest. For this reason, from now on we will assume that every router answers to an interest with the most recent content that matches

the name in the interest packet as well as all the other fields.

5.2.1 Many-to-Many Communications Over NDN

Many-to-many communication in NDN can be achieved by having multiple entities produce content for a certain prefix, as pictured in fig. 5.1. The advantage of doing so is that the content updates can be requested by monitoring - i.e. sending interest for - one single name instead of many. As an additional scenario, we also consider the case of a dynamic network where participants come and go too frequently for (1) the FIBs to be updated quickly enough and (2) for the participants to know each other. For simplicity, let us consider the simple case of a peer that connects to a network and wants to notify the others of its presence. Obviously, this message must be named independently from its sender since the receiver does not know him, and the sender cannot address any of the participants in the network because they are unknown to him as well. In NDN, any data name can be used as a multicast channel. Given a generic prefix, say */ndn/broadcast/*, any application can create packet named under this prefix and followed by a random identifier or the hash digest of the message, e.g. */ndn/broadcast/12FFAA65CC*. Any other application that generates interest for */ndn/broadcast/* will receive the packets. While using a shared name allows to retrieve data from multiple producers by requesting only one name it has the disadvantage that data delivery is not guaranteed. In fact, messages cannot be safely indexed with a serial number, or a version, because producers are not synchronized. As a consequence a message could be lost for two reasons: (1) there is no way to know its exact name; and (2) there is no way to know that such message exists. In the following sections we provide a solutions to this problem.

As in fig.5.1, let us denote by A, B and C three of the participants and /www.foo.com/activity/ the unique prefix used for exchanging messages. C issues an interest for the prefix, and receive in return the latest chunk created by the participants, in this case B. While the packet is sent back to C, C cannot reach A and B with a new interest. In the mean time, both A and B produce some content under the shared prefix. Upon receiving the response to its

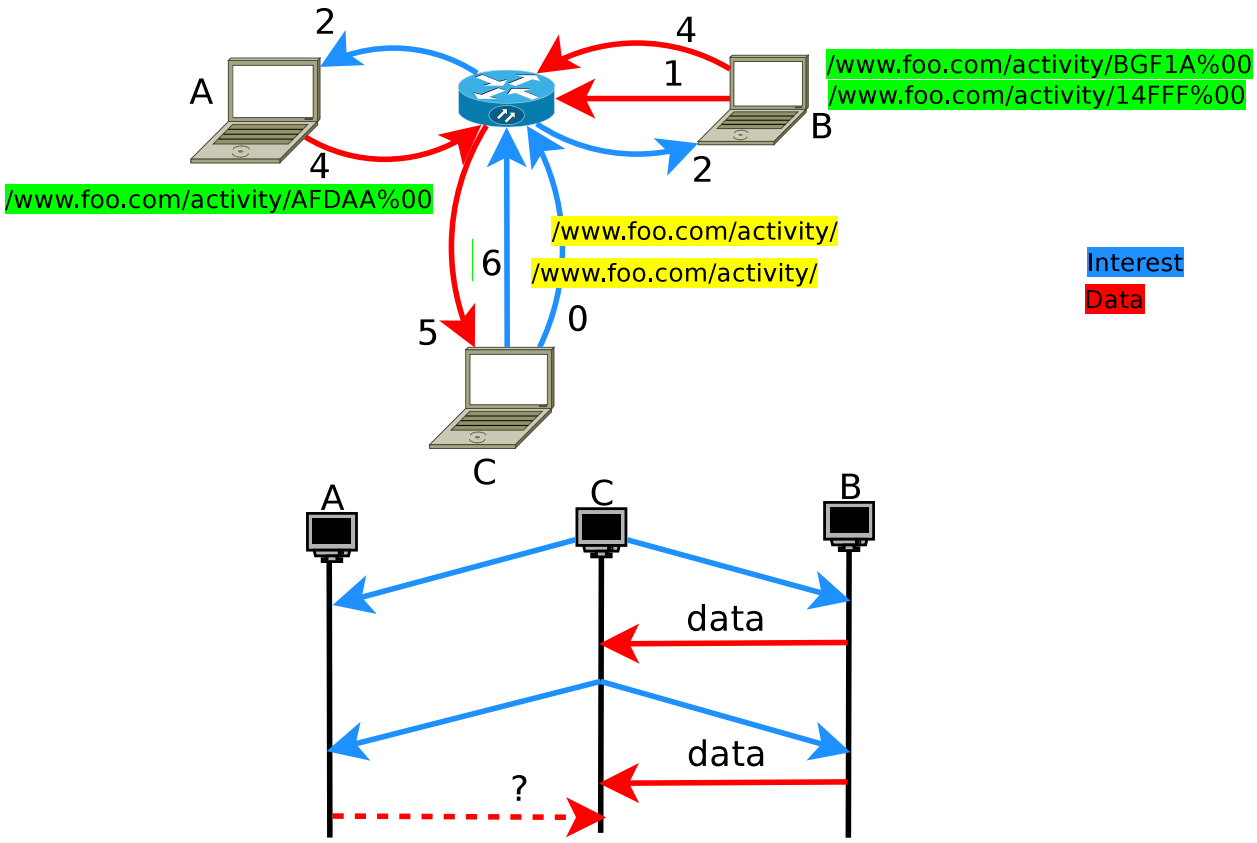


Figure 5.1: The numbers define the timeline of the message exchange. Note that, since A and B produce their data while C cannot forward a new interest (being constrained by its round trip time) the outcome of C's future request (6) cannot be predicted.

request C sends a new interest and attempts to retrieve either A's or B's content. Since packets are not synchronized or indexed, C will retrieve only one of them.

For C to get all the content, it has to generate interests faster than A and B can produce content. However, the number of participants m can be large, and C is rate-limited to generate interests by the minimum of the round-trip time to the participants. Indeed, if C issues multiple interests for the same name, R will discard these interests until it gets data to consume the first interest from C stored by R in its PIT. We discuss possible solutions to this problem below.

5.2.2 Centralized approach

A centralized server could be used to synchronize the content publishers so that messages are distinguished by sequential identifier, e.g. /www.foo.com/activity/seq1, /www.foo.com/activity/seq2/ etc. This requires a central server connected to all the participants that, upon request, tells a peer what sequence number to use to name the next message.

While this solution ensures that any participant can request interests for all the packets in the collaborative stream, it requires a third-party to act as an arbiter. It also requires some registration mechanisms for the participants to identify themselves and register with this centralized server. The centralized server could be one of the participants themselves (say, the host of the collaborative session). This host would then become a single point of failure for the session, and the distributed nature of the communication would not be preserved.

5.2.3 Exclude Filter

Natively NDN offers a solution for multiple name updates[2]. Each interest can carry a Bloom filter [**?**] named exclude field (EF) that specifies what packets the client has already received and therefore should not be sent back as a reply to the interest. However, it cannot retrieve all the data if this are frequently updated during the lapse of a RTT; also packet can be excluded only once they a have been received hence they cannot be downloaded in parallel. Furthermore, BF suffer false positives. In fig. 5.2(a) we plotted the optimal dimension for a EF given the desired precision and the number of packets that the filter excludes. In fig. 5.2(b), instead, we give an estimate of the overhead introduced by using EF to poll a different amount of packets, the reader should notice that in order to maintain the error rate below 1% a client sends out about 60% of the amount of data it receives.

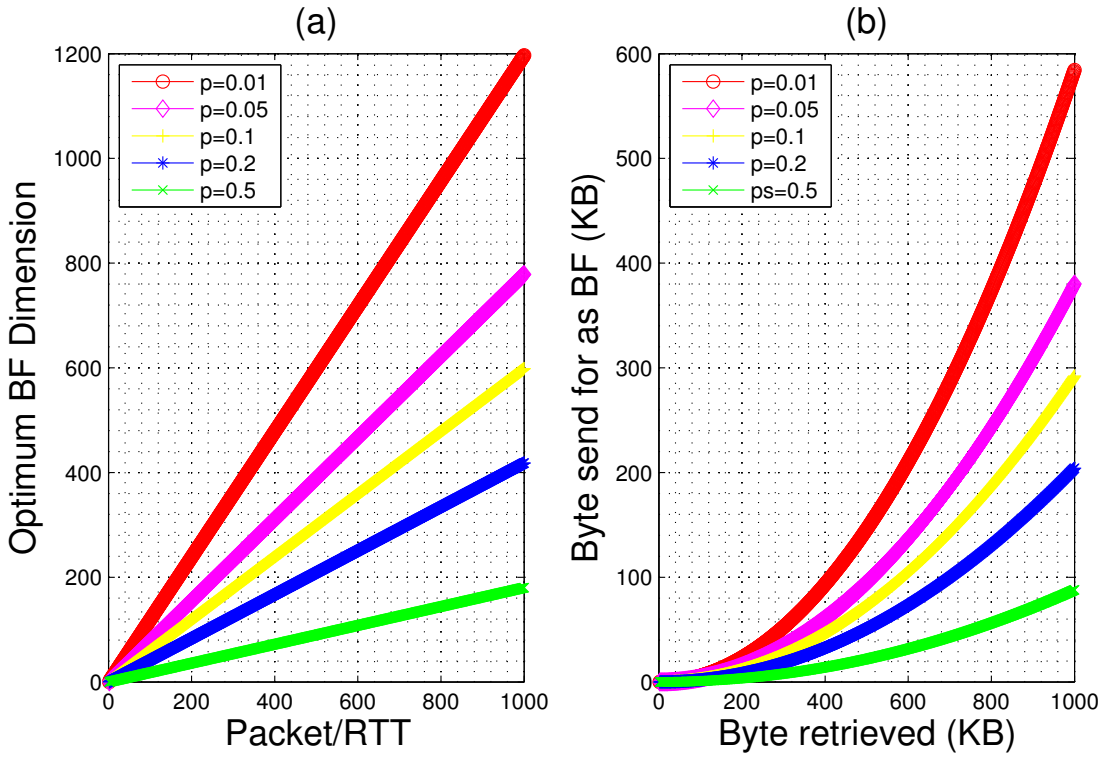


Figure 5.2: (a) the graph shows the optimal dimension of the Bloom Filter against the possible amount of packets that can be generated in a single RTT. Each line represent has a different probability of false positives. (b) The amount of KB needed to request x packets given different level if precision of the Bloom filter.

5.2.4 Prefix Hopping

We now describe Prefix Hopping, a fully distributed solution for content that is frequently updated by each one of the possible producers publishing under a common name.

First, we assume the interests are always sent with the field *Scope* set so that they will skip any cache and forwarded to all participants in the activity. With this premise an host receives all of the updates of a generic data name, say /ndn/broadcast/activity/ if its interests arrive to the producers of the data, before two different updates are posted, fig.5.1. In order to meet this condition, inspired by the frequency hopping algorithm (FHSS) of wireless communications, we propose the prefix hopping algorithm (PH). Briefly, the idea

[2]Although this feature has never been implemented in the CCNx distribution.

consists of splitting one single prefix into multiple sub-prefixes so that, by choosing at random the bin where a host publishes its messages, it is possible to arbitrarily reduce the probability of a collision [**?**].

In detail, let us consider n sub-prefixes, from now on referred to as bins, such as */ndn/broadcast/activity/* */ndn/broadcast/activity/b2/*, etc. Every peer p publishes a stream of packets s_i^p using a sequence of pseudo-random numbers $r_i \in [0...m]$ that respectively place each packet s_i^p into the bin r_i. The likelihood of a collision can be arbitrarily reduced by choosing an appropriate number of bins with respect to the expected number of participants in the network and the frequency of their messages.

Since a collision can still occur, p attaches to each packet s_i^p a pointer to the name of the next message s_{i+1}^p , this so that once s_i^p has been received all the others s_j^p with $j > i$ can be received without losses.With this expedient, every time a peer $p1$ discovers another peer $p2$ by receiving a message s_i^{p2} it can subscribe to $p2$'s stream of messages by setting an interest for s_{i+1}^{p2} s_{i+2}^{p2}. Furthermore, in order to avoid losses the packets for which the name is known are removed using the EF when polling the bins.

This method guarantees that after a packet from another peer is received all the other packets generated by the same peer can be received as well, even if there is collision with the packets of the other peers. The advantage of this method is that, as shown in the analysis, it guarantees low latencies with a small overhead. Note that the number of interest sent by each host is $\Theta(n + u)$ where n is the number of bins and u is the number of users in the network. The packets for which the name is known will be also excluded by the polling for the bins by filling the EF. Note that, what really makes the difference between using only the EF and using PH is that the latter reduces the latency of polling the data, when the content rarely has concurrent updates EF is still a reasonable solution, even though the dimension of the Bloom filter will keep growing if too many updates are created. PH instead performs better when data must be fetched as soon as possible and is subject to frequent concurrent updates. One way of looking at it is that PH embeds the Bloom filter into the naming of data whereas EF adds it to the interest.

5.3 Analysis of PH

We conduct a basic back-of-the-envelope analytical evaluation of the packet loss rate of Prefix Hopping and demonstrate the improvement over using a single shared prefix. Denote by $1/\tau$ the highest rate at which C can issue an interest. τ is thus a function of the round-trip time (RTT) in between the participants, and by λ the rate at which each of m participants generates new chunks in the content stream. We assume this is uniform across the participants for ease of explanation, but the model can naturally be generalized to different λ values for different participants.

Assuming the participants in the stream generate packets according to a deterministic schedule, the number of packets that C will not receive is thus: $m\lambda\tau - 1$. The fraction of lost packets from the stream observed at C is thus:

$$1 - \frac{1}{m\lambda\tau} \tag{5.1}$$

If the time interval between two chunks being generated in the stream by any participant at rate λ is not deterministic, but for instance follows a Poisson distribution, then the fraction of lost chunks increases: denote by p_k the probability of generating k chunks within a time interval τ for a Poisson distribution with rate $m\lambda$. Then the number of packets that C will not observe is:

$$\sum_{k\geq 1} p_k(k-1) \tag{5.2}$$

$$\sum_{k\geq 1} kp_k - (1 - p_0) \tag{5.3}$$

$$m\lambda\tau - (1 - e^{-m\lambda\tau}) \tag{5.4}$$

This increases obviously with the number of participants and for large values it is asymptotic to the deterministic case. In order to alleviate the problem, C needs to issue interests faster than its rate limit. One possible solution is to have the participants use one of n prefixes (say www.foo.com/activity/name/index, where index takes value from $\{1, 2, \ldots, n\}$). C would then issue n interests for each one of the indexes. A participant would publish into one of the streams by randomly selecting the stream each time a new chunk is generated. We

consider the improvement in the reduction of loss observed at one of the participants. If the chunks are generated according to a deterministic process, then in each time window τ, $m\lambda\tau$ packets are generated (by other participants). Those are split into the n different streams. This is akin to tossing balls (new chunks) into bins (indexed streams). The expected number of bins with j balls in it after k chunks are generated is given by $E_n(j,k)$ where:

$$E_n(j,k) = n\binom{k}{j}\left(1-\frac{1}{n}\right)^{k-j}\left(\frac{1}{n}\right)^{j} \tag{5.5}$$

The number of lost packets is thus:

$$L_n(k) = \sum_{j\geq 2} E_n(j,k)(j-1) \tag{5.6}$$

and the number of observed chunks is:

$$O_n(k) = \sum_{j\geq 1} E_n(j,k) = n - E_n(0,k) = n(1-(1-\frac{1}{n})^k) \tag{5.7}$$

Taking $m = n$ (i.e. spreading the interests and content over as many names as participants) and setting $\lambda\tau = 1$ will yield a loss rate of $1 - 1/e$ for large number of participants m. This is of course better than in the single prefix case (where the loss rate was going to 1 for large m). Fig. 5.3 shows the expected loss rate as a function of the number of users and the number of bins.

Keeping $k = o(n)$ gets the number of observed packets arbitrarily close to k, namely the number of chunks generated by all the participants in the stream. However, this incurs an overhead as the number of prefixes to poll is significantly larger than the number of participants.

Those are lower bounds on the loss rate, as having each piece of content announce the next packet in the stream significantly reduces the amount of loss packets, as we will see in the implementation evaluation.

5.4 Implementation and Evaluation

We have implemented our proposed solution using CCNchat [**?**] as our example of distributed collaborative application. The implementation of PH does not require any modification to

the CCNx code[3] hence, in order to validate our analysis, we implemented PH as a thin layer between the final application and the CCN's APIs. At the current state of the art, the java libraries of CCNx support both synchronous and asynchronous data retrieval. We implemented both while remaining compliant with the interfaces distributed with the software. Furthermore, since PH allows to multiplex different streams under the same name name, we are currently adding the feature of demultiplexing the sub streams which might be helpful in the future.

It has not been possible to compare the EF-based solution against PH because EF is not yet implemented in CCNx and requires to modify the codebase. Due to this problem PH has been evaluated against the scenario where multiple producers produce content under the same prefix.

The experiments scenario comprises of two machines, one running a PH-based content fetcher and the latter running several instance of its content producer counterpart. The producers have been run on the same machine in order to have the same roundtrip time, the RTT statistics are reported in fig.5.4. Due to limitation of the software it has not been possible to produce more than ten packet per second, which should not be a challenging rate. However, even at this low rate, we observe significant packet loss and we are able to demonstrate the impact of concurrent updates on the routers cache.

The two top graphs plot the continuity index of two (left) and five (right) sub-streams as they are received on the client side in one of the experiments. While for two producers the data loss is sporadic by increasing the number of producers to five data loss becomes so frequent that one of the 5 peers almost never made it to the client side. The two graphs at the bottom show, instead, how the system behaves using PH. As the reader can notice there is no packet loss and none of the sub streams is penalized in favor of the others. Note that the client received more packets from some producers rather than others because packets were generated adding a random wait in order to avoid the synchronization of the producers. The results obtained can be justified with the following arguments:

Collision Avoidance: PH reduces the probability of a prefix being updated several time

[3]CCNx is the name of the official open-source NDN distribution.

before being requested with an interest, this also helps the CCNx daemon to maintain the cache consistent.

Sub-stream tracking: Each producer adds to every packet a pointer to its follower so that this can be requested individually without need of using the generic prefix of the bin. The statistics of the time needed to discover one sub-stream are reported in fig. 5.4.

5.5 Concluding Remarks

We have identified a challenge in Name Data Networks where multiple users share a common prefix to publish content, as in multiple many-to-many communication implementations. We have offered Prefix Hopping as a solution and provided some basic analysis which demonstrate the benefit of the approach. We also have implemented PH and have shown in the case of a simple chat application that it reduces the packet loss dramatically.

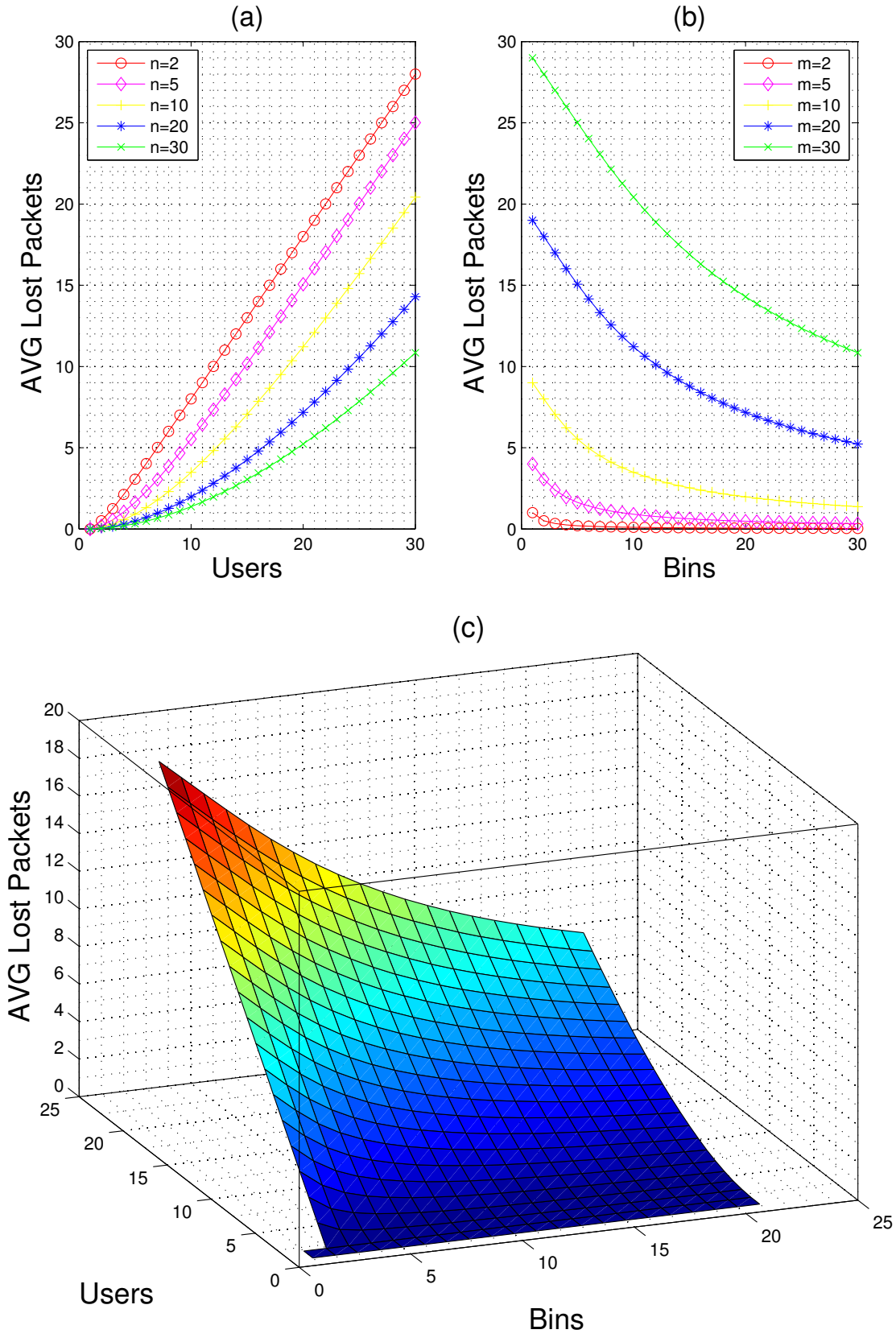
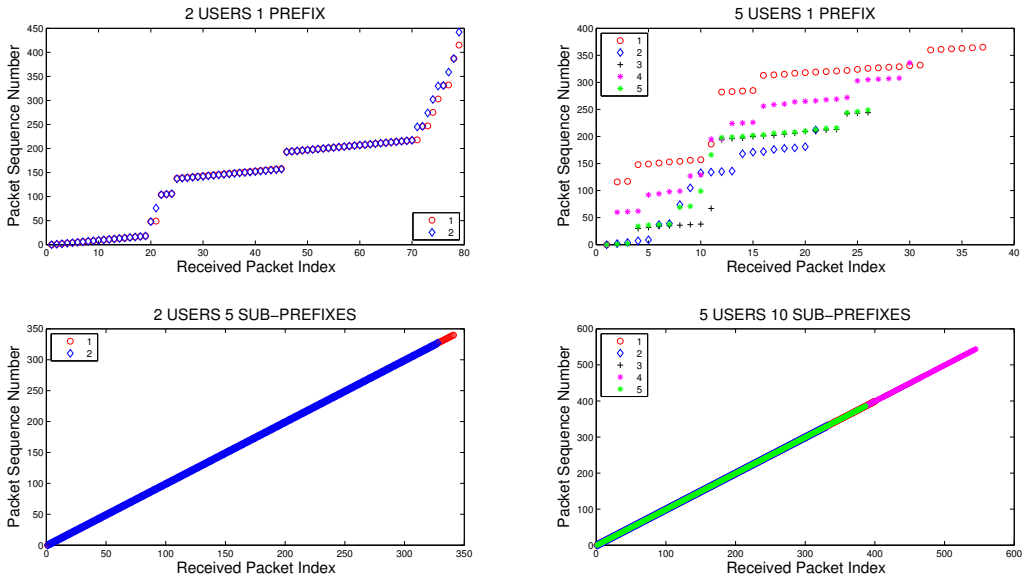


Figure 5.3: (a,b,c) Packet Loss against the number of (a) users (b) bins (c) both. The rate is kept constant to 1 $\frac{packet \cdot user}{RTT}$ in all the graphs



Sub-stream availability [2 producers]: avg=0.6s stddev=0.5s

Sub-stream availability [5 producers]: avg=0.6s stddev=1.8s

RTT: min=14.8ms avg=19ms max=37ms stddev=5ms

Figure 5.4: Continuity index of the sub-stream as received on the client with (bottom row) and without (top row) using PH

CHAPTER 6

ICN for Wireless Networks

6.1 Introduction

Mobile Ad-hoc Networks (MANET) are wireless environments where nodes interact in a peer-to-peer (p2p) fashion and can move arbitrarily across the network. In spite of the efforts spent in the last decades there is still not a widely used architecture for developing consumer software [CB08] in ad-hoc environments, nevertheless, custom solutions are still frequently needed for environmental sensing, military communications and emergency networks [SRK03, VR, CCL03]. The main challenge lies on the scattering connectivity induced by channel interferences and mobility. Initially, researchers pursued the interoperability of Internet and ad-hoc networks but, unfortunately, the nature of the two is deeply different. The former Internet is wired, interference-free and sporadically subject to disruptive topology changes whereas wireless networks commonly suffer packet loss, asymmetric links and frantic changes of topology. In this scenario both IP's routing and TCP's congestion control perform poorly [AS04]. The research work done throughout the years produced numerous routing protocols for MANET, such as OLSR, AODV, DSR and GPSR, that offer different performance on a case-by-case basis. At the very same way many versions of TCP for MANETs have been studied and implemented, some examples are SplitTCP [KKF02] and ATCP [SS99]. Even though IP connectivity leads to undoubtable advantages - i.e. the opportunity of using the same software everywhere - recently researchers moved their attention towards new data-centric architectures that discontinue the former conversational pattern in favor of opportunistic content retrieval, this approach is now commonly known as Information Centric Networking (ICN). Namely in a ICN a client fetches the content from

anywhere this is replicated; by doing so, firstly, it reduces the average distance between the client and the server, secondly, it increases the throughput by augmenting the number of servers and, thirdly, the data transfer is more resilient to packet loss and connection outages - a graphic example is shown in Fig. 6.1. The most prominent ICN projects are Named Data Networking (NDN)[JST09b, JSB09], Haggle [SSH], Mobility First [BR12, SNN11] and DONA [KEC07].

The utility of a network solely based on gross data distribution is justified by the fact that modern mobile devices can manage large arrays of data making the client-server dialogue secondary, while back in the days both storage and computational power resided solely on the server-side.

MADN applies the same concepts but its design is done entirely and specifically for the family 802.11x of wireless protocols which helps overcome the issues of other solutions designed for wired networks. Additionally, MADN is an open platform that can be easily extended to test new solutions.

The rest of the paper is organized as follows: Section 6.2 summarizes previous work and recent advances on this field; Section 6.3 introduces MADN and defines our contribution; Section 6.4 describes the protocol in details; Section 6.6 describes the architecture and the first prototype; Section 6.7 comments the current results; and Section 6.8 concludes with a brief summary and future work.

6.2 Related Work

Because of the vast literature on this topic the following overview intentionally covers only pioneer works and most recent advances on content distribution for MANETs, the reader is reminded to the citations for a broader analysis. The discussion is also restricted to fully distributed protocols where network nodes can only be: clients, relays or both. Furthermore, we will consider only scenarios of limited mobility where clients can move arbitrarily around the network while relays move sporadically, Delay Tolerant Networks (DTN) and Vehicular

Ad-hoc Networks (VANET) are beyond the scope of this paper.

6.2.1 Content Centric MANET

Most of the older project for content centric networking over MANETs somehow rely on the IP protocol to make the nodes dialogue with each other. Taking end-to-end connectivity for granted most of the protocols build an overlay that is able to route content request t where the content is stored. The hardness of the task is due to the fact that can be several replicas of a content and, as time goes, new ones can be added while others are removed. As suggested in [VRL11], the methods used to route content requests fall into three main categories: proactive, reactive and structured. Briefly, proactive routing very much alike IP-routing uses control messages to maintain updated routing tables that will deliver content request to destination, an example is [KLW03]. Reactive routing instead consists of using random walks [PPC06] to crawl the network and locate the content, some examples are [KLW03, CGT05]. Structured routing is a more fine technique wherein, given the destination address, relays use a metric to decide where to forward the request, the drawback is a considerable overhead whenever the network topology changes, relevant works are [RKE02, CCN06].
In spite of the skepticism towards its aggressive use of bandwidth reactive routing and seems to offer the better tradeoff between overhead and accuracy, as in [VRL11]. Moreover a handful of refined gossip algorithms[1] have been proposed throughout the years, among the others[CHP05, LWJ06, YG03]; other works such as [ZMN10, BGJ06, SPR05] are instead designed for disruptive environments such vehicular networks (VANET) and they are relevant to the scope of this paper.
Beside locating the content, another important aspect of content centric MANETs is clearly how this is distributed, specifically how to take advantage from the possibility of downloading from multiple sources, pioneer works on this topic are [JLC07]. A major advance in this field has been the adoption of network coding and erasure coding [Lub02] [Lub98] [Sho06] to distribute the content, as done by Gerla et al. in [LPY06]. The gain of using encoded packets

[1] Gossip Algorithm is the name given to the forwarding methods that rely on message replicas rather than routing tables.

is double, first, packet loss can be overstepped without need of feedback from the client, as done by TCP, and, second, concurrent streams generate less overhead since the packets are equally useful and duplicates are unlikely. One recent project on ad-hoc networking that puts together all the pieces described so far is Microcast of Keller et al. [KLC12].

6.2.2 ICN architectures

ICN comprises all the architectures where data can be addressed without regard of its location in the network. The final goal is, first, to decouple the application logic from the details of the network topology and, second, to reduce the load on the network by caching content on the routers. The general idea consists in the clients fanning out content requests to the network without need of specifying a location. The requests are then forwarded by the routers towards the content replicas, unless the router itself has a cached version of the content in which case the request is served locally. By convention, content always travel backwards on the trace left by the content request.

6.2.2.1 Named Data Networking

Named Data Networking has been proposed by Van Jacobson et al as in [JST09b, JSB09]. The peculiarity of the protocol is the data naming. Each packet is identified by a unique hierarchical name that, by convention, specifies its producer; what content it belongs to; its version; and its sequence number. Packets are requested one by one using special messages named *Interest*, this method is also known as *flow balance* rule [JST09b]. NDN has been designed as an alternative solution for the world wide Internet and it does not quite take into consideration the dynamics of an ad-hoc network, there is although a preliminary work from Meisel et al [MPZ10] that to the best of our knowledge it has never been implemented.

6.2.2.2 Haggle Mobility Framework

Haggle targets heterogeneous mobile wireless networks, in particular Personal Area Networks (PAN). Differently than other protocols, Haggle allows to request content from its metadata

instead of its name. For example, it is possible to request any picture with a certain tag, or any mp3 of a specific rock band. Differently than NDN there is no flow balance between requests and data packets, once the content has been requested a TCP connection is established between the parties for the data exchange.

6.2.2.3 Dona and Mobility First

Both the projects aim at becoming the future Internet architecture leveraging on names resolution to keep track of all the replicas in the network. At the current state of the art, while the architecture takes into consideration mobility and long range wireless communication, ad-hoc networks are not in the picture for both the projects.

6.3 Contribution

MADN takes advantage of the shared nature of the wireless medium using only broadcast packets and never addressing machines directly. Nodes choose a random identifier since they need a unique name only within their range of transmission. Differently than IP, a packet is not routed to destination carrying the name of the initial sender, in MADN a packet always carries the name of the last sender.

Another peculiarity of MADN is that it does not use explicit routes, given a source and a destination there is not a unique string of nodes to define a path between them, Fig. 6.1. The idea is to keep open all the paths so that alternative routes can back-up the shortest *on-the-fly*–i.e. without need of exchanging control messages. To serve this purpose, each relay has its own time-slot to forward a packet, the *best relays* relay first while back-ups wait longer if the formers fail.

MADN uses only three kinds of packets: data requests named *leech*, data packets and beacons. Beacons are the only control message used, MADN dedicates the bandwidth almost entirely to data exchange as will be seen in the evaluation.

The data stored in the network are identified using a two-level address space, namely each *content* has its own id (CID) and each one of its segments, named *stripes*, has its own Stripe

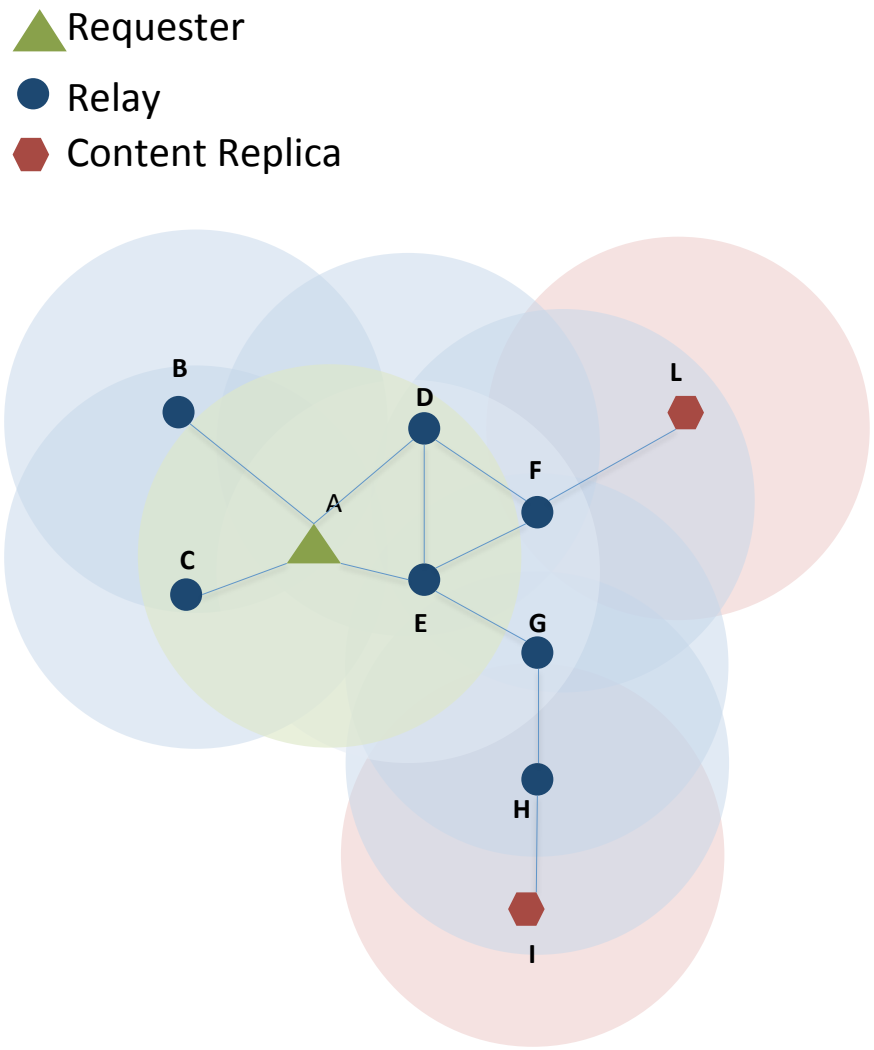
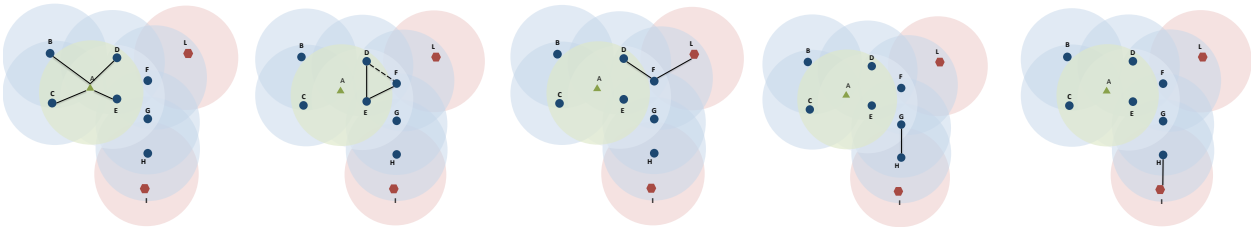


Figure 6.1: A sample topology for a content centric ad-hoc network, a requester node A and two content replicas L and I are connected via multiple paths, namely A-D-F-L, A-E-F-L, and A-E-G-H-I. MADN preferentially uses the last two because avoid a redundant transmission from D.

Id (SID). The decision of not using a full hierarchy, as NDN, stems from the fact that hierarchical names help routing packets only when names reflect the network topology [Cis01]. By design, a stripe is uniquely identified by its DataID (CID + SID) and it is both the smallest and the biggest piece of information that can be addressed with a data request. The reason is double: first of all, in a wireless network it would be inefficient to split the bandwidth between request packets and data packets, as NDN does. Second of all, it is wise to set a



(a) (b) (c) (d) (e)

An example of leech dissemination in MADN, from left to right: the requester sends a request that is forwarded towards the edges of the network. The dotted line between D and F is to show that, while it could, D does not relay the packet to F since E, which can reach both F and G, relays first.

Figure 6.2: Leech Dissemination

limit to the maximum amount of data that a node can request with a single request, this in order to avoid Denial of Service (DoS). Moreover, it is useful to access to a content by segments, for example, considering a real-time video the user will likely request the most recent segments only. Also, since MADN is designed to use network coding it would not be functional to encode the data packets without fragmenting the content.

One innovation that, to the best of our knowledge, none of the previous work has implemented is a a flow control mechanism based on packet counters. A leech packet in fact does not request the entire stripe but only a certain amount of its encoded packets, once the amount of packets has been delivered the requester will send a new request for additional packets and so on until the whole stripe is decoded. This approach not only avoids flooding but also allows to implement flow control mechanisms for both single and multi-source download without needing end-to-end communications as in SCTP [SX01]. Note that this would not be possible without using rateless encoding.

6.4 Protocol

6.4.1 Beacons and BlooGo

Beacons are necessary to the BlooGo algorithm to work properly. In details, the algorithm minimizes the overlapping area of two consecutive transmissions consequently reducing the delay and the bandwidth usage of data dissemination. For example in Fig. 6.1, a packet sent by node A is not relayed by peer B and C because they cannot reach any node that is not in range of A. To enforce this method, each node i periodically sends a beacon that contains: (1) its own identifier and (2) its immediate neighborhood, hereafter written as $N_b(i)$. By doing so, every peer j in range of i can discriminate whether to forward packets for i by verifying if $N_b(j) \not\subset N_b(i)$.

Note that this condition does not quantify how useful this transmission is. In fact, considering Fig. 6.1, D can reach F but if E relays the packet it will reach both F and G in one single transmission. To serve this purpose BlooGo uses the following utility function

$$D_{ij} = |N_b(j) \setminus N_b(i)|$$

that namely is the cardinality of the difference between the neighborhood set of j and the one of i.

Note that other gossip algorithms, such as [CHP05, LWJ06, YG03], use similar approaches and benefit of similar properties, the advantage of using BlooGo is that it encodes the neighborhood set as a bloom filter making beacons considerably more compact. Consequently, beacons can be sent more frequently in order to be more reactive to topology changes. Additionally, two neighborhoods can be compared in constant time instead than linear time. The tradeoff is that bloom filters are a randomized data structure subject to false positives, for a complete analysis the reader is reminded to the original paper [AGP12].

6.4.2 Leech Dissemination

Nodes fan out leech packets to locate and request a content. A leech packet has the following format:

NodeID	DataID	HP	Nonce	PC	FID

In the order: (1) the identifier of the node that sent the packet; (2) the DataID of the requested stripe; (3) a hop counter to limit the flooding; (4) a random nonce to avoid routing loops; (5) the number of packets that must be sent in response; and (6) a flow identifier, it is used whenever the leech targets a specific source for the content.

When a node receives a leech packet it verifies if it can serve the request from its storage, if that is the case it starts serving the request otherwise, if it can, it relays the packet to a new part of the network that has not received it yet. However, as previously mentioned there might be several nodes that can do it, and the idea is choose the one that can cover the most new ground. In order to do so avoiding duplicates MADN does not use control messages but instead schedules retransmissions in time-slots. Specifically, considering a sender i a relay j will delay its retransmission of an amount of time inversely proportional to the utility function D_{ij}, leading to the formula:

$$T_{ij}(r) = \begin{cases} (K - D_{ij})T_s + T_s r & \text{for} \quad D_{ij} \leq K \\ T_s r & otherwise \end{cases}$$

where K is a parameter that defines the maximum number of *time slots* that a packet retransmission can be delayed; T_s is the dimension of each time slot and $r \in [0, 1]$ is a random number that prevents possible ties.

If after waiting a time T_{ij} j does not overhear another node k such that $N_b(j) \subseteq N_b(k)$ then j forwards the packet, otherwise the packet is dropped. The packet will continue being relayed until it hits a relay that cached the content or the producer of the content, Fig. 6.2e.

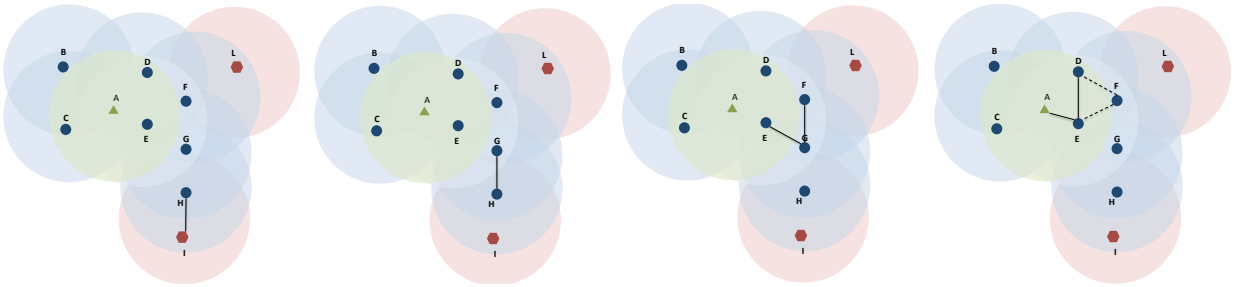
6.4.3 Request Table

Every node that has overheard a leech packet, and not just those that relayed it, keeps track of it using a request table (RT) structured as follows:

From now on we will write *x.y.field-name* to indicate the fields of the x.y entry of the RT.

When a node j receives a leech packet for a DataID $x.y$ it first verifies the nonce to ensure

Field	*Notation*	Description
DataID	$< x.y >$	The stripe identifier
Packet Counter	PC	N. of packets to serve
Requesters	Rq	The nodes that requested the stripe
Flows	Fl	The data flows the pass by the node
Forwarded	Fw	True if the node relayed a leech for $x.y$



(a) (b) (c) (d)

Figure 6.3: An example of how data packets are relayed from a source to the requester. From left to right: a data packet is relayed from I to A. Note that when the packet reaches E and F, it is forwarded by E because it has forwarded the initial request. If the node E fails then F forwards to D that forwards to A.

it is not a duplicate, in which case the packet is discarded. Otherwise, the RT is updated in the following way:

- $x.y.PC = max(l.PC, x.y.PC)$
- $x.y.Rq = x.y.Rq \cup l.NodeID.$
- Iff j relayed the leech packet then $Fw_{x.y} = 1$.

6.4.4 Data Delivery

A node serves a data request for $x.y$ by broadcasting as many encoded data packets as indicated in its RT. A data packet has the following format:

NodeID	DataID	FID	Nonce	PAYLOAD

where: (1) NodeID is the id of the sender; (2) DataID is the stripe the packet belongs to; (3) FID identifies which node encoded the packet; and (4) Nonce is used to avoid routing loops. All The nodes in range with an entry $x.y$ in their RT can relay the packet closer to its destination. As for disseminating requests, each relay j has its own time-slot to forward the packet, and if in the mean time another node k, such that $x.y.Rq \subseteq N_b(k)$, relays the same packet then j withdraw the transmission. We then need a new utility function:

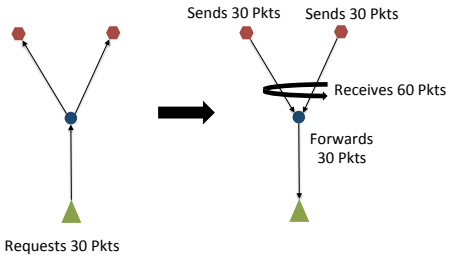
$$D_{ij}(x.y) = |x.y.Rq \setminus N_b(k)|$$

which is equal to the number of nodes that requested x.y to j that are not in the neighborhood of k. The time j will wait before relaying a data packet then becomes:

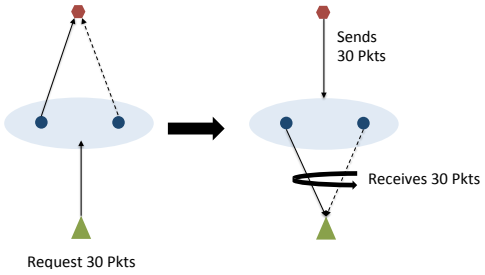
$$T_{x.y}(r) = \begin{cases} T_s \cdot r & \text{if } Fw_{x.y} = 1 \\ (S_r + K - D_{ij}(x.y))T_s + T_s r & \text{if } D_{ij} \leq K \\ S_r T_s + T_s r & otherwise \end{cases}$$

Note that the parameter S_r gives priority to the nodes that a leech packet for x.y, this is done in order to maintain path symmetry between requests and content. An example of how data packets are routed to the requester is given in Fig. 6.3.

We mentioned before that MADN uses packet counters to enforce flow control, an important remark is that the counter is not exclusive between the requester and the node that serves the request. Specifically, each relay that serves the route, both back-up and best nodes, decreases its counter each time it forwards or overhears a data packet for x.y. The reason is explained in Fig. 6.4, whenever a relay merges multiple data flows it acts as a valve that stops the rest of the network to be flooded with unrequested packets. At the same way, if more than one relay can serve the same data flow it is important that all the relays are aware that the whole request has been served and more packets, if they will ever arrive, are no longer needed.



(a) An example of forked path. The intermediate relay acts as a valve relaying to the requester only the exact number of packets it has requested.



(b) An example of single path with mutliple relays. Both the intermediate nodes must count the number of packets delivered by the other in order not to exceed the demand of the requester.

Figure 6.4: Optimizations of packet delivery in forking and joining paths

6.4.5 Data Flows

Controlled flooding, as BlooGo does, is useful only when content replicas still have to be located, after the first round of request-response there is a clear path[2] between the parties, i.e. requester and content replicas, and it would be unwise not to reuse it.

For these purpose, each source labels the data packets for a content x.y with a *flow identifier* (FID) that it chooses randomly the first time that it serves a request for x.y. Relays keep track of what data flows pass on their way in the field x.y.Fl of their RT. At the same way, also the requester knows how many sources can serve its requests, and it can autonomously decide which one to send the next requests to.

The routing scheme then changes so that if a leech packet specifies a FID then it will be forwarded only by the nodes that are on the way for that data flow.

[2]Note that the word *path* is intentionally used instead of *route* which commonly defines a ordered string of machines. MADN instead does not choose a single route but marks the nodes that can relay towards destination.

6.5 BlooGo

We previously anticipated that with BlooGo packets carry the sender's neighborhood encoded as a Bloom filter which from now on we will refer to as Neighborhood Bloom Filter (NBF). In this section we analyze the quality of the approximation due to the use of Bloom Filters which are renownedly a randomized data structure.

6.5.1 Bloom Filters

A bloom filter consists of (1) a bit array B of m bits all initialized to zero and (2) a set of k hash functions $h_j(x) : N \rightarrow \{1...m\}$, $j \in \{1...k\}$. Given a set of values $L = \{x_1, x_2, ..., x_{n-1}, x_n\}$ each x_i with $1 \leq i \leq n$ feeds the k hash functions so that each x_i maps into k positions of the array B, respectively $h_1(x_i), h_2(x_i), ...h_{k-1}(x_i), h_k(x_i)$. All these positions are then set to 1 in the array B. In order to query if an element y was part of the original set this must feed the hash functions in order to obtain k positions in the array B. If all these positions are set to 1 we accept y as part of the original set with a known probability. A peculiarity of bloom filters is that they allow false positives but they do not allow false negatives. If an element is not included in a bloom filter then it was surely not part of the original set. The probability of false positive is known from from Mitzelmacher et al in [Mit04] as equal to: $f_p = ((1 - \frac{1}{m})^{kn})^k \sim (e^{\frac{-kn}{m}})^k$ where m is the dimension of the bloom filter, k is the number of hash functions and n is the number of elements added to the bloom filter.

6.5.2 Message Forwarding

Each time a host receives a packet it only has to compare two bloom filters. This operation consists in verifying if the local NBF_l is completely included into the one inside the incoming packet NBF_r. This can be done by testing if $(NBF_r \ AND \ NBF_l) \ XOR \ NBF_l \ > 0$. Where the AND operator finds which bits the two bloom filters have in common and the

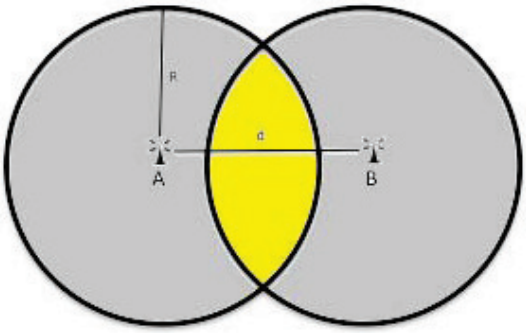


Figure 6.5: Geometry of the network model. The overlapping area in yellow is $S(d)$ whereas the grey area is equal to $I(d)$

Notation	Formula
Φ	πr^2
$Poisson(x, \Delta\delta)$	$\frac{x^{\Delta\delta}}{k!}e^{-\Delta\delta}$
$S(d)$	$2r^2 cos^{-1}\left(\frac{d}{2r}\right) - \frac{d}{2}\sqrt{4r^2 - d^2}$
$I(d)$	$\Phi - S(d)$

Figure 6.6: Table of formulas

final XOR checks if there is at least one host that the local node can reach with one more transmission. This solution, is an approximation prone to error since bloom filters allow false positive a node can mistakenly decide to not transmit, the quality of this approximation will be further discussed. It remains to say how the number of relayed messages can be reduced, i.e. how to avoid that two or more peers relay a packet to the same part of the network. This problem is reduced to the minimum by delaying the transmission proportionally to the number of bits shared by NBF_l and NBF_r, this because there is an high probability that a relay node far from sender can cover a bigger area of the network, as can be seen in **??**. If, while waiting to transmit, a node hears a relayed copy of the message that covers all its neighborhood the packet will not be sent.

6.5.3 Analysis

The analysis is conducted assuming a lossless network with nodes uniformly distributed in a square of side $l = 5$ and density δ. Nodes have a circular transmission range of radius $r = 1$ and area Φ. The average number of nodes in an area of dimension Δ is equal to $\bar{X}(\Delta) = \Delta\delta$; accordingly the probability mass function of the X number of nodes within the area is:

$$f_X(x, \Delta, \delta) \sim Poisson(x, \Delta\delta)$$

where x is the number of peers and $\Delta\delta$ is the rate of the Poisson distribution. This scenario ensures connectivity with high probability when $r = \Theta(\sqrt[2]{log(\bar{n})/\bar{n}})$, as result of [**?**]. Given two nodes h_A and h_B distant d from each other, the dimension of the overlapping area of their range of transmission is written $S(d)$; instead the portion of the range of transmission that the two nodes do not share is written $I(d)$. Table at figure 6.6 shows how this value can be calculated given the parameters and figure 6.5 shows the graphic.

6.5.4 Error with Bloom filters

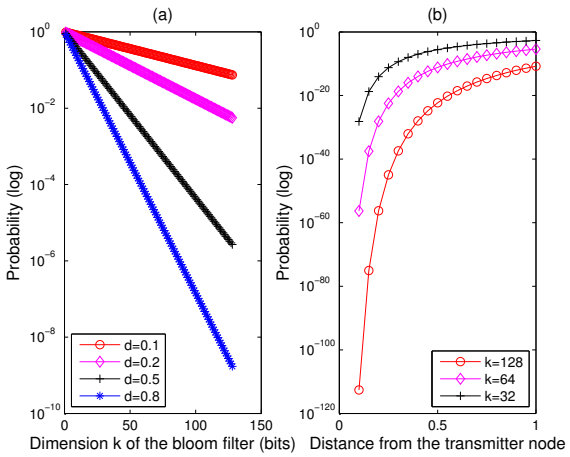


Figure 6.7: (a) probability of error given the dimension in bits of the bloom filter (b) probability of error given the distance from the sender. In both cases the distance d is given as a fraction of the radius $r = 1$

The utility function of BlooGo is prone to error because of the probabilistic nature of bloom filters, i.e. a node can mistakenly decide to not forward a packet when it should. This can be proven by the following claims:

Claim 1 *A forward decision can never be wrong.*

Proof 1 *The claim is proven recalling that bloom filters do not allow false negatives and this is the only condition that can make this happen.*

Claim 2 *Given two nodes h_A and h_B. If h_B receives a packet from h_A the packet will not be forwarded if the NodeIDs of ALL the nodes that are exclusively in range of h_B map into positions of NBF_A that are already set to 1.*

Proof 2 *Considering the expression:*

$$(NBF_A \; AND \; NBF_B) \; XOR \; NBF_B > 0$$

The expression between the parenthesis is equal to the bits that NBF_A shares with NBF_B. Thus XORing the result with NBF_B if this contains at least a bit that is not set to 1 in NBF_A the total must be greater than 0.

In order to derive the quality of the approximation lets consider a node h_B that receives a packet from h_A. On average NBF_A contains $\bar{X}(\Phi)$ NodeIDs. Recalling that a bloom filter with n elements has a false positive probability equal to: $f_p(k, n, m) = (e^{\frac{-kn}{m}})^k$, then $f_p(k, \bar{X}(\Phi), m)$ is the average probability of having one node in range of h_B that maps into positions of NBF_A that are all set to 1, where k is the number of hash functions and m is the dimension of the bloom filter in bits. Although, the packet will not be forwarded only if ALL the nodes that are only in range of h_B map into bits of NBF_A that are set to 1, which leads to:

$$e_{NBF} = (e^{\frac{-k\bar{X}(\Phi)}{m}})^{k\bar{X}(I(d))}$$

Note that, while $\bar{X}(R)$ is function of δ and $\bar{X}(I(d))$ depends on both *delta* and d, e_{NBF} depends only on the distance d between h_B from h_A.The statement can be proven by simplifying the exponent of e_{NBF} given the parameters δ and d, the algebra is omitted because trivial. This property does not hold whenever the network has not a uniform distribution of the nodes, future work will comprise the analysis of more artificial topologies.

6.5.5 Pseudo-Geographic forward

Throughout the paper, and in the previous section, we claimed that BlooGo benefits from implicitly taking into consideration the distance between the sender and the relay. To find

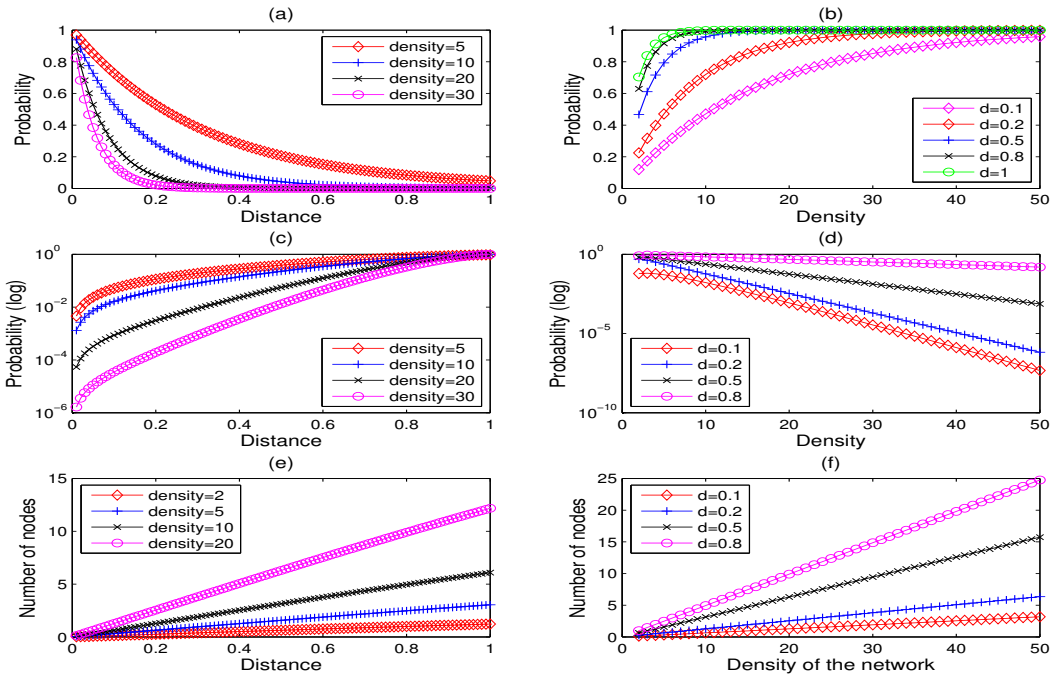


Figure 6.8: (a and b) Show the probability of not being a relay given the distance d from the sender and the density δ. (c and d) Show the probability of being the furthest relay from the sender given the distance d and the density δ (e and f) show the expected number nodes that a relay can reach given its distance d and the network density δ (density is expressed in terms of number of peers per range of transmission)

a close form for the probability that a node h_i is the perfect relay for a message that arrives from h_j is overly complicated, but it is possible to bound the quality of the assumption. A candidate relay for a message is any node h_j that can forward the message to a node h_i outside the range of transmission of the sender h_S. Given the distance d between h_j and h_S and the density δ, the probability of being a candidate relay is equal to $f_r(d, \delta) = 1 - f_X(0, I(d), \delta)$. The best relays h_j are the ones that cover the biggest area outside the range of transmission of the sender h_S. This probability cannot be easily found. Instead, it is possible to bound the probability that a node at distance d from h_S is the furthest candidate relay. In fact, as shown in figure 6.5, any node at distance d from the sender can reach at most a node h_i that is at distance $r + d$ from the sender, where r is the radius of the transmission range.

Any node h_i that is at distance $r < d_i < r + d$ overlaps with the transmission range of h_S with an area equal to $S(d_i)$. The probability of this area being empty can be bounded by Bonferroni's inequality (aka Union Bound) to:

$$f_B(d, \delta) = f_X(0, S(d_i), \delta | r < d_i < r + d) \leq \int_{k=r}^{r+d} f_X(0, S(k), \delta)$$

Thus the probability that a node h_j is the furthest from a sender h_S given its distance d, is equal to

$$f_r(d, \delta) \cdot f_B(d, \delta)$$

Note that the two probabilities can be multiplied together because given the network model they are independent. The results in figure 6.8 show how the probability of being a relay, and the probability of being the furthest relay for a given area, increases accordingly with the distance from the sender.

6.5.6 Simulation

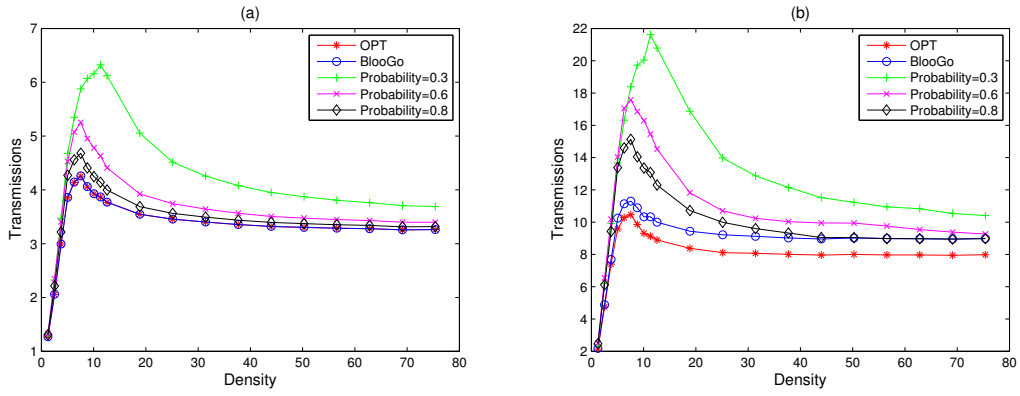


Figure 6.9: (a) shows the average number of transmission used to reach a node from another node. (b) shows the average maximum number of transmission used to reach a node from another. From the plots it is clear how BlooGo outperforms probabilistic routing, especially when the density of the network is low BlooGo is quasi-optimal.

The last results consist in comparing BlooGo's performance against probabilistic forwarding. The experiments are conducted simulating random topologies faithful with the network

model previously discussed. For each set of parameters the simulator generates 100 topologies then for each pair of nodes (h_1, h_2) a message is routed from h_1 to the network until this reaches its destination h_2. The results in figure 6.9 show how BlooGo outperforms probabilistic forwarding reaching the target nodes in a quasi-optimal number of transmissions. Probabilistic forwarding performs poorly, especially when the density of the network is low, because it does not have a real heuristic to forward the packets. BlooGo utility function, instead, tries to cover the network in the fastest way possible.

6.6 Implementation

6.6.1 Architecture

The success of a network architecture strictly depends on how intuitively the application developers can use it. Most of the protocols mentioned so far, with the exception of ICN architectures, are thought as built-in of the final application leaving to the developer the hardship of implementing the network logic. Differently, MADN focuses on building a layered solution and leaves to the developer only to implement the application logic. Additionally, the design of MADN is modular so that the protocol can be changed with minor effort. It comprises four modules:

- Network Layer - handles the network functionalities i.e. epidemic routing, packet relaying, data delivery and flow control.
- Encoder/Decoder - lies in between the network logic and the storage; Upon request, it serves encoded packet to the network layer. Viceversa, it collects the encoded packets arriving from the network and, once the content has been decoded, stores the result into the storage.
- Storage - implements both short-term and long-term persistency. It is mostly a passive component with only two duties: (1) notifies the applications when their data have arrived and (2) deletes the content that is not locally used whenever the cache is full.

- Application - It sends request to the network layer and waits to be notified by the storage of their arrival. Additionally, it can withdraw a request if the content is no longer needed.

Modules are completely decoupled so that they can be replaced without need of modifying the codebase anywhere else, this makes particularly easy evaluating different configurations and solutions. The first prototype of MADN has been built for the Linux operating system making the components dialogue via unix sockets and named pipes.

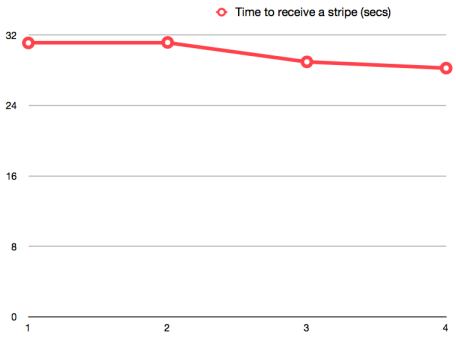
The network layer is written in native C and uses unix raw sockets to fill entirely the 802.11 frame, all the other modules are instead implemented in Java. Final applications can be written in any language, the one used for testing is currently built in java. Applications request stripes directly to the network layer, which takes care fetching them. The current implementation uses a system folder as storage and a small daemon to notify the applications when the content has arrived. The network layer sends the encoded packet to the decoder, written in java, that pushes the decoded stripe to the storage when ready. On the relay nodes content is considered volatile and can be deleted in order to free the system memory, although at the current moment there is no caching discipline implemented.

Currently there are only two encoder/decoder, namely random network coding and coupon collector[3].

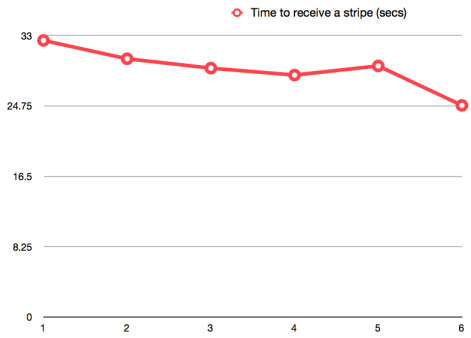
CID and SID are 4 bytes integers but in the near future this will be changed to strings of 16 bytes to make them more user friendly. The payload of each data packet is between 1000 and 1400 bytes depending on the type of encoding used.

At the current moment, the implementation does not include a flow control algorithm which considered the difficulty of the task would be anyway beyond the scope of this paper. Instead a node requests a fixed amount of packets with each request.

[3]Coupon Collector means that the source node chooses which packet of the stripe to send using a uniform distribution. The requester has then to collect them all.



(a) Time to receive a stripe given the number of source nodes.



(b) Time to receive a stripe given the number of requesters.

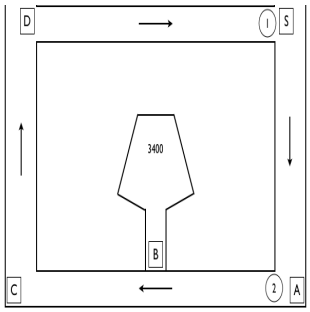


Figure 6.10: 3rd floor of Boelter Hall (UCLA)

6.7 Evaluation

The results presented in this section are obtained using the current implementation of MADN, their main goal is to test correctness and feasibility of the protocol while fine tuning is left to future work. For this same reason, we intentionally used the coupon collector configuration, instead of network coding, to avoid collecting results biased towards the parameters chosen for the data encoder. However, it is common knowledge that by doing so a node will need to receive $\Theta(nlogn)$ packets before downloading the whole stripe, the results must be read keeping this in mind.

6.7.1 Static Nodes

The first set of tests was meant to verify that (1) the nodes could avoid redundant transmissions by sensing their neighbors and (2) that nodes were able to *count* the packets that they overheard for each DataID. The latter task was our main concern because we were afraid that a user space implementation could be too slow or this purpose, instead we observed that the number of duplicates due to a miscount of the relays was negligible. The configuration used for the first experiments consisted of stripes of 700 packets which led to an average of 2000 packets needed to receive the entire stripe.

6.7.2 Variable Senders, Single Receiver

In this experiment, there was a single node requesting a stripe and the number of servers was varied. Fig. 6.10a shows that the time required to receive a stripe is almost constant with respect to the number of servers. The test is considered successful since, without regard to the number of servers, the medium was used at its full bit-rate (1 Mbps) and the servers stopped at the right moment when all the requested packets were sent to the client. Since there is still not flow control it was not possible for the client to selectively chose which servers had to deliver the content. For the same reason, if was not possible to test the case of hidden terminals, meaning that servers can hear the receiver but they cannot hear each others.

6.7.3 Single Sender, Single Receiver with Variable Listeners

This experiment was conducted to verify (1) how smoothly concurrent requests could be served and (2) how smoothly a node could switch from being a client to being a server[4]. It is seen in Fig. 6.10b. that the time remains almost constant, and actually slightly decreases when 6 clients are active. By inspecting the log files we verified that one node had decoded sooner than the others and started serving the requests.

6.7.4 Mobile Nodes

In these tests, nodes were placed in different locations according to the map in Fig. 6.10. The content was initially stored only on the node S, while nodes A, B, C and D acted as relays. A mobile node R moved around the floor in a clockwise manner at normal walking speed, in order to test how smoothly MADN managed the handoff from a relay to another. For this tests we used stripes of 255 packets that are, on average, decoded after receiving about 620 packets. Each leech packet requested 200 packets.

[4]Once a node has received the entire stripe it can start serving requests too.

6.7.4.1 Experiment 1

In this experiment, the starting location was 1, Fig. 6.10. On average, a total of 9 stripes were received and a 10th partially received during one complete round. The node R sent a leech packet every 3sec. From Fig. 6.11b, intuitively when the receiver is closer to the server the stripes are received in a shorter amount of time than those received 2 hops away due to packet loss. Moreover in Fig. 6.11a, half way there was a considerable amount of duplicated packets due to the fact that the network topology was a perfect loop. In fact, when R was half way it was being served from both directions, we are currently working to a clean solution for duplicated paths. In Fig. 6.11e, we show what packets were sent on a network segment over time, the result is reassuring since almost the entire bandwidth capacity is occupied by data packets.

6.7.4.2 Experiment 2

This experiment started from position 2 shown, Fig. 6.10, and results are shown in 6.11d and 6.11c . Requests were sent waiting 1 second from the previous one. An average of six stripes were received during one lap. Analogously to the previous configuration, more duplicates were received when far from node S, Fig. 6.11c, however as we got closer to the server, since requests were more frequent, more packets were sent by the server and overall lesser duplicates were received. In Fig. 6.11f it is possible to notice how the data flow is less fragmented than in Fig. 6.11e, showing again the impact of sending leech packets more frequently. Overall this configuration worked better within 0 to 1 hops from the source but very poorly elsewhere which suggests that most of our effort should go towards a flow control mechanism able to adapt to the network topology.

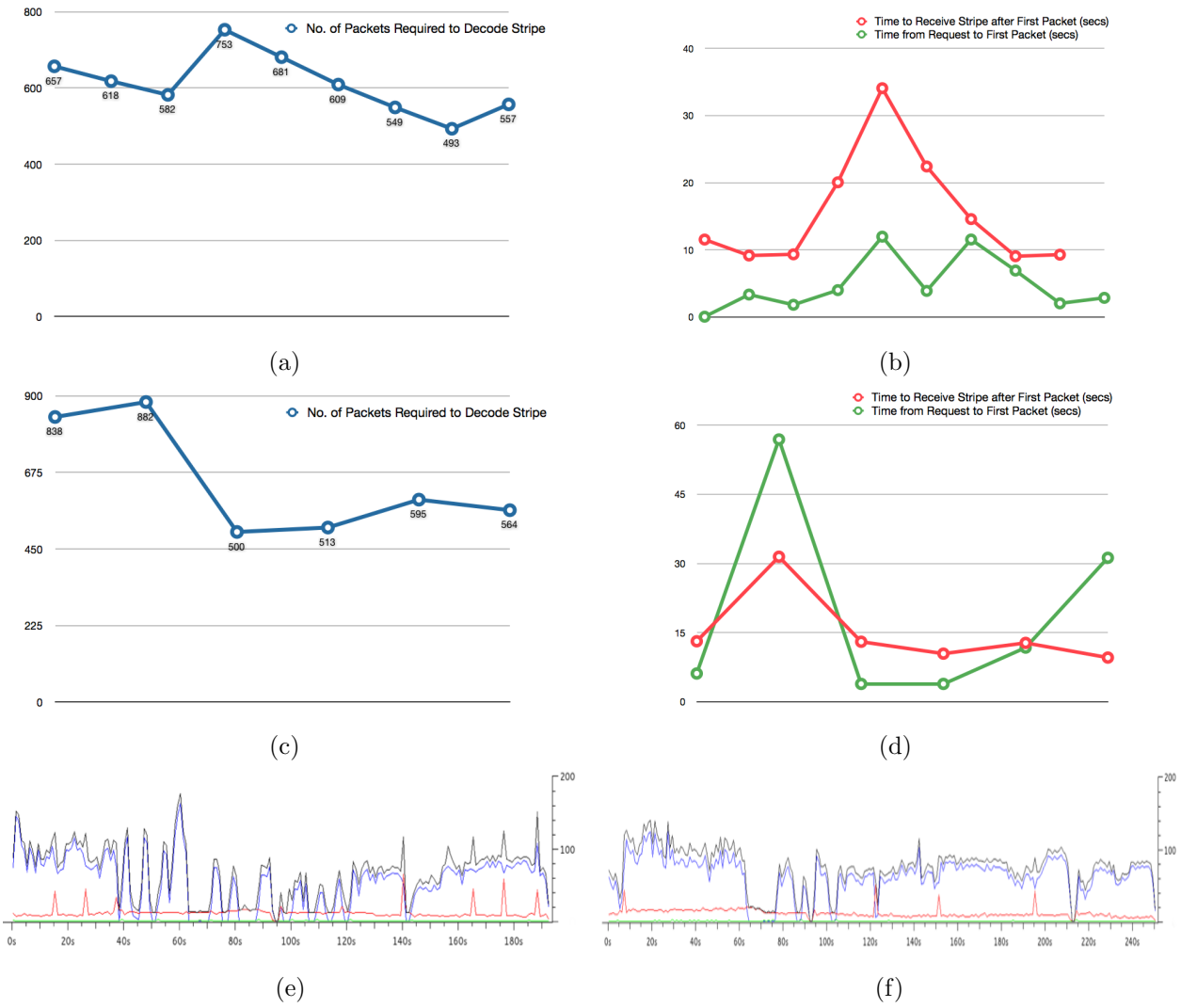
6.8 Conclusions and Future work

We presented MADN a clean-slate Information Centric architecture exclusively made for wireless ad-hoc networks. The goal was to create a solid and intuitive framework able

to make ad-hoc networking more friendly for the application developer, in this paper we presented the preliminary result obtained running our prototype on a testbed of 7 static nodes and 1 mobile nodes. The results evidenced that, while there is still work to do, at the current state of the art MADN allows to fast develop content centric applications in a fully distributed wireless environment; and, to the best of our knowledge, there are no other frameworks that do the same. In addition, it is our believe that MADN will be a useful tool for the research community since its modular design makes particularly easy to implement and test new solutions.

Future work is twofold, we will improve the current core architecture but we will also develop some real-world applications to validate our design choices. For the core system, the priority is to implement a flow control algorithm and a back-push mechanism that can avoid the problem of duplicated paths that we experienced during the experiments. For the latter, multimedia streaming and content-distribution for fast-deployable networks, e.g. emergency or military networks, are two field of application that we consider ideal for testing MADN. Keeping in mind that using MADN does not exclude using TCP/IP we will also explore hybrid solutions.

Currently we are rewriting the network layer in Java in order to facilitate the prototyping of new solutions while the codebase written in C will be maintained as groundwork and it will include only the functionalities that have been fully tested. The source-code is currently available upon request to the authors but will be soon distributed online.



Black: Total number of packets **Blue:** Data Packets **Red:** Beacons **Green:** Leech Packets Result of the experiments with mobility. (Left Column) Requests every 3 secs (Right Column) Requests every 1 sec.

Figure 6.11: Experimental Results

References

[ADI12] Bengt Ahlgren, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher, and Börje Ohlman. "A survey of information-centric networking." *Communications Magazine, IEEE*, **50**(7):26–36, 2012.

[AGH12] Vijay Kumar Adhikari, Yang Guo, Fang Hao, Matteo Varvello, Volker Hilt, Moritz Steiner, and Zhi-Li Zhang. "Unreeling netflix: Understanding and improving multi-CDN movie delivery." In *INFOCOM, 2012 Proceedings IEEE*, pp. 1620–1628. IEEE, 2012.

[AGP12] Fabio Angius, Mario Gerla, and Giovanni Pau. "BLOOGO: BLOOm filter based GOssip algorithm for wireless NDN." In *Proceedings of the 1st ACM workshop on Emerging Name-Oriented Mobile Networking Design - Architecture, Algorithms, and Applications*, NoM '12, pp. 25–30, New York, NY, USA, 2012. ACM.

[AK13] Niels van Adrichem and Fernando Kuipers. "Globally Accessible Names in Named Data Networking." In *Proc IEEE INFOCOM NOMEN'13*, April 2013.

[AR10] Luca M. Aiello and Giancarlo Ruffo. "LotusNet: Tunable privacy for distributed online social network services." *Computer Communications*, December 2010.

[ARS83] Leonard M Adleman, Ronald L Rivest, and Adi Shamir. "Cryptographic communications system and method.", September 20 1983. US Patent 4,405,829.

[AS04] Vaidyanathan Anantharaman and Raghupathy Sivakumar. "TCP Performance over Mobile Ad-hoc Networks - A Quantitative Study." *Wireless Communications and Mobile Computing*, **4**:203–222, 2004.

[AWG13a] Fabio Angius, Cedric Westphal, Mario Gerla, and Giovanni Pau. "WARP: A ICN architecture for social data." In *DySoN - Infocom 2014*. IEEE, 2013.

[AWG13b] Fabio Angius, Cedric Westphal, Mario Gerla, Jun Wei, and Giovanni Pau. "Prefix Hopping: Efficient Many-To-Many Communication Support in Information Centric Networks." *IEEE NOMEN Workshop - Infocom 2013*, 2013.

[BBS09] Randy Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. "Persona: an online social network with user-defined privacy." *ACM SIGCOMM Computer Communication Review*, **39**(4):135–146, 2009.

[BGJ06] John Burgess, Brian Gallagher, David Jensen, and Brian Neil Levine. "MaxProp: Routing for Vehicle-Based Disruption-Tolerant Networks." In *In Proc. IEEE INFOCOM*, 2006.

[BHG12] Ames Bielenberg, Lara Helm, Anthony Gentilucci, Dan Stefanescu, and Honggang Zhang. "The growth of Diaspora - A decentralized online social network in the wild." In *IEEE INFOCOM Workshops*, pp. 13–18, March 2012.

[BKL10] Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, and Peter Vajgel. "Finding a Needle in Haystack: Facebook's Photo Storage." In Remzi H. Arpaci-Dusseau and Brad Chen, editors, *OSDI*, pp. 47–60. USENIX Association, 2010.

[BR12] Akash Baid and Dipankar Raychaudhuri. "Wireless access considerations for the MobilityFirst future Internet architecture." In *Sarnoff Symposium (SARNOFF), 2012 35th IEEE*, pp. 1–5. IEEE, 2012.

[BSV09] Sonja Buchegger, Doris Schiöberg, Le H. Vu, and Anwitaman Datta. "PeerSoN: P2P social networking: early experiences and insights." In *ACM EuroSys Workshop SNS'09*, pp. 46–52, 2009.

[CB08] Justin Collins and Rajive Bagrodia. "Programming in mobile ad hoc networks." In *WICON*, p. 73, 2008.

[CCL03] Imrich Chlamtac, Marco Conti, and Jennifer J.-N. Liu. "Mobile ad hoc networking: imperatives and challenges." *Ad Hoc Networks*, **1**(1):13 – 64, 2003.

[CCN06] Matthew Caesar, Miguel Castro, and Edmund B. Nightingale. "Virtual ring routing: network routing inspired by DHTs." In *In Proc. of ACM SIGCOMM*, pp. 351–362, 2006.

[CGT05] Marco Conti, Enrico Gregori, and Giovanni Turi. "A cross-layer optimization of gnutella for mobile ad hoc networks." In *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, MobiHoc '05, pp. 343–354, New York, NY, USA, 2005. ACM.

[CHP05] Ying Cai, Kien A Hua, and Aaron Phillips. "Leveraging 1-hop neighborhood knowledge for efficient flooding in wireless ad hoc networks." In *Performance, Computing, and Communications Conference, 2005. IPCCC 2005. 24th IEEE International*, pp. 347–354. IEEE, 2005.

[Cis01] Cisco. "The Internet Protocol Journal - Volume 4, Number 4.", 2001. [Online; accessed 25-February-2013].

[CMS09] Leucio Cutillo, Refik Molva, and Thorsten Strufe. "Safebook: A privacy-preserving online social network leveraging on real-life trust." *IEEE Communications Magazine*, **47**(12):94–101, December 2009.

[Cry] CryptoOpp.com. "Cryptographic Algorithms Benchmark." Website.

[CWT01] Hao Che, Zhijun Wang, and Ye Tung. "Analysis and design of hierarchical web caching systems." In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pp. 1416–1424. IEEE, 2001.

[CWY10] Sherman SM Chow, Jian Weng, Yanjiang Yang, and Robert H Deng. "Efficient unidirectional proxy re-encryption." In *Progress in Cryptology–AFRICACRYPT 2010*, pp. 316–332. Springer, 2010.

[DKO13] Christian Dannewitz, Dirk Kutscher, Börje Ohlman, Stephen Farrell, Bengt Ahlgren, and Holger Karl. "Network of Information (NetInf) – An information-centric networking architecture." *Computer Communications*, January 2013.

[DMP02] John Dilley, Bruce Maggs, Jay Parikh, Harald Prokop, Ramesh Sitaraman, and Bill Weihl. "Globally distributed content delivery." *Internet Computing, IEEE*, **6**(5):50–58, 2002.

[DVP12] Ben Dodson, Ian Vo, T. J. Purtell, Aemon Cannon, and Monica Lam. "Musubi: disintermediated interactive social feeds for mobile devices." In *ACM World Wide Web conference WWW'12*, 2012.

[Fac08] Facebook. "Building a Facebook Application, Start to Finish." In *API Developers Guide*, pp. 71–127, 2008.

[FLT13] Seyed Kaveh Fayazbakhsh, Yin Lin, Amin Tootoonchian, Ali Ghodsi, Teemu Koponen, Bruce M. Maggs, K. C. Ng, Vyas Sekar, and Scott Shenker. "Less pain, most of the gain: incrementally deployable ICN." In Dah Ming Chiu, Jia Wang, Paul Barford, and Srinivasan Seshan, editors, *SIGCOMM*, pp. 147–158. ACM, 2013.

[FN94] Amos Fiat and Moni Naor. "Broadcast Encryption." *Lecture Notes in Computer Science*, 1994.

[GAL07] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. "Youtube traffic characterization: a view from the edge." In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pp. 15–28. ACM, 2007.

[Gar95] Simson Garfinkel. *PGP: pretty good privacy*. O'reilly, 1995.

[GPS06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. "Attribute-based encryption for fine-grained access control of encrypted data." In *Proc. ACM CCS*, pp. 89–98, 2006.

[GTF08] Saikat Guha, Kevin Tang, and Paul Francis. "NOYB: privacy in online social networks." In *in ACM WOSN'08*, 2008.

[HBR13] Qi Huang, Ken Birman, Robbert van Renesse, Wyatt Lloyd, Sanjeev Kumar, and Harry C. Li. "An analysis of Facebook photo caching." In Michael Kaminsky and Mike Dahlin, editors, *SOSP*, pp. 167–181. ACM, 2013.

[HHH12] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. "Confused, timid, and unstable: picking a video streaming rate is hard." In *Proceedings of the 2012 ACM conference on Internet measurement conference*, pp. 225–238. ACM, 2012.

[Ian13] Black Ian. "NSA spying scandal: what we have learned.", June 2013.

[Inf09] Federal Information. "DIGITAL SIGNATURE STANDARD (DSS)." In *United States Federal Government standard FIPS*, June 2009.

[JBD12] Van Jacobson, Rebecca L. Braynard, Tim Diebert, Priya Mahadevan, Marc Mosko, Nicholas H. Briggs, Simon Barber, Michael F. Plass, Ignacio Solis, Ersin Uzun, and Others. "Custodian-based information sharing." *Communications Magazine, IEEE*, **50**(7):38–43, 2012.

[JK10] Pascal Junod and Alexandre Karlov. "An efficient public-key attribute-based broadcast encryption scheme allowing arbitrary access policies." In *Proc. ACM DRM'10*, pp. 13–24, 2010.

[JLC07] Sewook Jung, Uichin Lee, Alexander Chang, Dae-Ki Cho, and Mario Gerla. "BlueTorrent: Cooperative content sharing for Bluetooth users." *Pervasive Mob. Comput.*, **3**(6):609–634, December 2007.

[JMB11] Sonia Jahid, Prateek Mittal, and Nikita Borisov. "EASiER: Encryption-based access control in social networks with efficient revocation." In *ACM Symposium on Information, Computer and Communications Security*, pp. 411–415, 2011.

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. "The elliptic curve digital signature algorithm (ECDSA)." *International Journal of Information Security*, **1**(1):36–63, 2001.

[JNM11] Sonia Jahid, Shirin Nilizadeh, Prateek Mittal, Nikita Borisov, and Apu Kapadia. "DECENT: A Decentralized Architecture for Enforcing Privacy in Online Social Networks." In *ArXiv Tech. Rep. 1111.5377*, December 2011.

[JSB09] Van Jacobson, Diana K. Smetters, Nicholas H. Briggs, Michael F. Plass, Paul Stewart, James D. Thornton, and Rebecca L. Braynard. "VoCCN: voice-over content-centric networks." In *Proceedings of the 2009 workshop on Re-architecting the internet*, ReArch '09, pp. 1–6, New York, NY, USA, 2009. ACM.

[JST09a] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. "Networking named content." In *Proc. ACM CoNext'09*, New York, NY, USA, 2009.

[JST09b] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. "Networking named content." In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, pp. 1–12, New York, NY, USA, 2009. ACM.

[KCC07] Teemu Koponen, Mohit Chawla, Byung G. Chun, Andrey Ermolinskiy, Kye H. Kim, Scott Shenker, and Ion Stoica. "A data-oriented (and beyond) network architecture." *SIGCOMM CCR*, **37**(4):181–192, August 2007.

[KEC07] Teemu Koponen, Andrey Ermolinskiy, Mohit Chawla, Kye Hyun Kim, Ion Stoica, Byung gon Chun, and Scott Shenker. "A data-oriented (and beyond) network architecture." In *In SIGCOMM*, 2007.

[KH10] Andreas M. Kaplan and Michael Haenlein. "Users of the world, unite! The challenges and opportunities of Social Media." *Business Horizons*, **53**(1):59–68, January 2010.

[KKF02] S. Kopparty, S.V. Krishnamurthy, M. Faloutsos, and S.K. Tripathi. "Split TCP for mobile ad hoc networks." In *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, volume 1, pp. 138 – 142 vol.1, nov. 2002.

[KLC12] Lorenzo Keller, Anh Le, Blerim Cici, Hulya Seferoglu, Christina Fragouli, and Athina Markopoulou. "MicroCast: cooperative video streaming on smartphones." In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, MobiSys '12, pp. 57–70, New York, NY, USA, 2012. ACM.

[KLW03] A. Klemm, C. Lindemann, and O.P. Waldhorst. "A special-purpose peer-to-peer file sharing system for mobile ad hoc networks." In *Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th*, volume 4, pp. 2758 – 2763 Vol.4, oct. 2003.

[KMV00] Neal Koblitz, Alfred Menezes, and Scott Vanstone. "The state of elliptic curve cryptography." In *Towards a Quarter-Century of Public Key Cryptography*, pp. 103–123. Springer, 2000.

[LPY06] Uichin Lee, Joon-Sang Park, Joseph Yeh, Giovanni Pau, and Mario Gerla. "Code torrent: content distribution using network coding in VANET." In *Proceedings of the 1st international workshop on Decentralized resource sharing in mobile computing and networking*, MobiShare '06, pp. 1–5, New York, NY, USA, 2006. ACM.

[Lub98] Michael Luby. "Tornado Codes: Practical Erasure Codes Based on Random Irregular Graphs." In *Proceedings of the Second International Workshop on Randomization and Approximation Techniques in Computer Science*, RANDOM '98, pp. 171–, London, UK, 1998. Springer-Verlag.

[Lub02] Michael Luby. "LT Codes." *Foundations of Computer Science, Annual IEEE Symposium on*, **0**:271, 2002.

[LWJ06] Hai Liu, Pengjun Wan, Xiaohua Jia, Xinxin Liu, and Frances Yao. "Efficient flooding scheme based on 1-hop information in mobile ad hoc networks." In *Proc. IEEE INFOCOM*, pp. 1–12, 2006.

[Mar08] Luther Martin. "Identity-Based Encryption and Beyond." *IEEE Security & Privacy Magazine*, **6**(5):62–64, September 2008.

[Mit04] M. Mitzenmacher. "Digital fountains: a survey and look forward." In *Information Theory Workshop, 2004. IEEE*, pp. 271 – 276, oct. 2004.

[MNC10] M. Mani, Anh-Minh Nguyen, and N. Crespi. "SCOPE: A prototype for spontaneous P2P social networking." In *IEEE PERCOM'10*, March 2010.

[MPZ10] Michael Meisel, Vasileos Pappas, and Lixia Zhang. "Ad Hoc Networking via Named Data." In *Proceedings of the Fifth ACM Workshop on Mobility in the Evolving Internet Architecture (MobiArch)*, September 2010.

[NJM12] Shirin Nilizadeh, Sonia Jahid, Prateek Mittal, Nikita Borisov, and Apu Kapadia. "Cachet: a decentralized architecture for privacy preserving social networking with caching." In *Proc. ACM CoNext'12*, pp. 337–348, 2012.

[Par13] Parc. "CCNx Distribution.", July 2013.

[PBB12] C. Pushmann, Jean. Burgess, Axel Bruns, and Merja Mahrt. "Data Access, Ownership and Control in Social Web Services: Issues for Twitter Research." In *Meeting of the International Communication Association*, May 2012.

[PGW08] Johan A. Pouwelse, Pawel Garbacki, Jun Wang, Arno Bakker, Jie Yang, Alexandru Iosup, Dick H. J. Epema, Marcel Reinders, Maarten R. Van Steen, and Henk J. Sips. "TRIBLER: a social-based peer-to-peer system." *Concurrency and Computation: Practice and Experience*, **20**(2):127–138, 2008.

[PPC06] Luciana Pelusi, Andrea Passarella, and Marco Conti. "Opportunistic networking: data forwarding in disconnected mobile ad hoc networks." *Communications Magazine, IEEE*, **44**(11):134–141, 2006.

[PTM06] Matthew Pirretti, Patrick Traynor, Patrick McDaniel, and Brent Waters. "Secure Attribute-based Systems." In *Proc. ACM CCS'06*, pp. 99–112, 2006.

[RAD03] Matthew Richardson, Rakesh Agrawal, and Pedro Domingos. "Trust management for the semantic web." In *The Semantic Web-ISWC 2003*, pp. 351–368. Springer, 2003.

[RKE02] Sylvia Ratnasamy, Brad Karp, and Deborah Estrin. "GHT: A geographic hash table for data-centric storage." pp. 78–87. ACM Press, 2002.

[SDR13] Sami M. Shalabi, Cassandra L. Doll, James D. Reilly, and Maurice B. Shore. "Access Control List.", June 2013. US Patent 20,130,145,028.

[SHF99] David Solo, Russell Housley, and Warwick Ford. "Internet X. 509 public key infrastructure certificate and CRL profile." 1999.

[Sho06] Amin Shokrollahi. "Raptor codes." *IEEE/ACM Trans. Netw.*, **14**:2551–2567, June 2006.

[SNN11] Ivan Seskar, Kiran Nagaraja, Sam Nelson, and Dipankar Raychaudhuri. "MobilityFirst future internet architecture project." In *Proceedings of the 7th Asian Internet Engineering Conference*, pp. 1–3. ACM, 2011.

[SPR05] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S. Raghavendra. "Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks.", 2005.

[SRK03] Scott Shenker, Sylvia Ratnasamy, Brad Karp, Ramesh Govindan, and Deborah Estrin. "Data-centric storage in sensornets." *SIGCOMM Comput. Commun. Rev.*, **33**(1):137–142, January 2003.

[SS99] Jian Liu Sun and Suresh Singh. "ATCP: TCP for Mobile Ad Hoc Networks." *IEEE Journal on Selected Areas in Communications*, **19**:1300–1315, 1999.

[SSH] Jing Su, James Scott, Pan Hui, Jon Crowcroft, Eyal De Lara, Christophe Diot, Ashvin Goel, Meng How Lim, and Eben Upton. "Haggle: Seamless Networking for Mobile Applications.".

[SSH07] Jing Su, James Scott, Pan Hui, Jon Crowcroft, Eyal de Lara, Christophe Diot, Ashvin Goel, Meng Lim, and Eben Upton. "Haggle: Seamless Networking for Mobile Applications." In *Proc. UbiComp*, Lecture Notes in Computer Science, pp. 391–408. Springer, September 2007.

[SSN10] Seok-Won Seong, Jiwon Seo, Matthew Nasielski, Debangsu Sengupta, Sudheendra Hangal, Seng K. Teh, Ruven Chu, Ben Dodson, and Monica S. Lam. "PrPl: a decentralized social networking infrastructure." In *ACM Workshop on Mobile Cloud Computing & Services*, 2010.

[SX01] Randall R Stewart and Qiaobing Xie. "Stream Control Transmission Protocol (SCTP)." 2001.

[Sym] Symantec. "Elliptic Curve Cryptography (ECC) Certificates Performance Analysis." whitepaper.

[ten] "Tent.io." http://tent.io. Accessed: 2013-02-30.

[TL11] Ben D. Te-Yuan and Huang M. Lam. "The Junction Protocol for Ad Hoc Peer-to-Peer Mobile Applications." In *Tech. Rep. Stanford University*, 2011.

[VR] N Vetrivelan and Dr. A V Reddy. "Analyzing the Mobility and Protocol Performance in MANET using a Novel Move Stop Deviate Model.".

[VRL11] Matteo Varvello, Ivica Rimac, Uichin Lee, Lloyd Greenwald, and Volker Hilt. "On the design of content-centric MANETs." In *Wireless On-Demand Network Systems and Services (WONS), 2011 Eighth International Conference on*, pp. 1–8. IEEE, 2011.

[WU] Christopher A Wood and Ersin Uzun. "Flexible End-to-End Content Security in CCN.".

[XM12] Zhiqian Xu and K. M. Martin. "Dynamic User Revocation and Key Refreshing for Attribute-Based Encryption in Cloud Storage." In *IEEE TrustCom'12*, pp. 844–849, 2012.

[YG03] Yunjung Yi and Mario Gerla. "Efficient flooding in ad hoc networks: a comparative performance study." In *in Proceedings of the IEEE International Conference on Communications (ICC*, pp. 1059–1063, 2003.

[YWR10] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. "Attribute based data sharing with attribute revocation." In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pp. 261–270. ACM, 2010.

[ZMN10] Hongbo Zhou, Matt W. Mutka, and Lionel M. Ni. "Secure prophet address allocation for MANETs." *Security and Communication Networks*, **3**(1):31–43, 2010.