Epidemic Protocols: From Large Scale to Big Data

Davide Frey

HABILITATION À DIRIGER DES RECHERCHES
UNIVERSITÉ DE RENNES 1

sous le sceau de l'Université Bretagne Loire

Mention : Informatique

École doctorale MathStic

présentée par

Davide FREY

préparée à l'unité de recherche Inria Rennes
Institut National de Recherche en Informatique et Automatique
EPI WIDE

Epidemic Protocols: From Large Scale to Big Data

HDR soutenue à Rennes
le 11/06/2019

devant le jury composé de :

SONIA BEN MOKHTAR
Directeur de Recherche CNRS / *Rapporteur*
GAEL THOMAS
Professeur Telecom SudParis / *Rapporteur*
PASCAL FELBER
Professeur Université de Neuchatel / *Rapporteur*
ANNE-MARIE KERMARREC
CEO Mediego / *Examinateur*
OLIVIER BARAIS
Professeur Université de Rennes 1 / *Examinateur*
DANNY HUGUES
Professeur, KU Leuven / *Examinateur*

Acknowledgments

I would like to start by expressing my gratitude to the three reviewers, Sonia Ben Mokhtar, Pascal Felber, and Gael Thomas, for taking the time to read and and comment on this manuscript despite their busy schedules, as well as to the other members of the jury, Olivier Barais, Danny Hughes, and Anne-Marie Kermarrec.

I am also immensely grateful to Anne-Marie Kermarrec for a number of other reasons: for trusting my motivation when I applied for a post-doc in 2007, for providing me with great opportunities to do interesting research while starting new and fruitful collaborations, and for supporting me throughout my research career at Inria with her mentorship and advice. I would also like to express my gratitude to Rachid Guerraoui, who was also a great mentor during my postdoc at Inria, and with whom I have had the opportunity to co-authors many papers.

I am equally grateful to Francois Taiani for his role as a great team leader, for his advice, and for our many co-authorships, and to Michel Raynal who, even if we only co-authored a couple of papers, has been a constant source of interesting insights and wise counsel.

Among my mentors, my greatest thanks also go to my Phd advisors, Gian Pietro Picco, and Amy Murphy, and to my first post-doc advisor, Catalin Roman, without whom I would not have reached this stage in my career.

My gratitude goes to my many other co-authors and to the students that significantly contributed to my research. It would be too long to name them all, but I would like to express my special thanks at least to (in random order) David Bromberg, George Giakkoupis, Vivien Quema, Maxime Monod, Arnaud Jegou, Antoine Boutet, Heverson Ribeiro, Mathieu Goessens, Raziel Carvajal, Antoine Rault, Stephane Delbruel, Pierre Louis Roman, Pascal Molli, Brice Nedelec, Rhicheek Patra, Jingjing Wang, Tristan Allard, Vincent Leroy, Julien Tanke, Julien Lepiller, Kostas Kloudas, Quentin Dufour, Amaury Bouchra Pilet, and Florestan De Moor. My special thanks also go to Inria's support staff and in particular to Virginie Desroches and Cecile Bouton.

My warmest thanks go to my late parents for nurturing my interest in continuous learning since my early childhood, to my family, Nelly and Alessandra for their loving and continuous support even when work takes away time from family life, and to my sister, Giliola, and my extended family who, even if far, have always supported me even in my decisions to move abroad.

Finally, I would like to apologize and thank all the people that may have forgotten in this list.

Foreword

Since the start of my PhD at Politecnico di Milano, my research interest have focused on the design of solutions and algorithms for large scale systems. My initial work focused on distributed content-based publish subscribe [24] based on deterministic tree topologies [22, 29]. But I soon started focusing on randomized protocols for large-scale systems, first in wireless networks [28, 23], particularly during my from my first postdoc at Washington University in St Louis, and then in wired networks [27, 21, 20, 19] when I joined Inria first as postdoc and later as a permanent researcher. Over the years, this has led my research to explore several major research axes, the main two being epidemic content dissemination and privacy-preserving recommender systems. The rest of this section briefly summarizes my contributions since I joined Inria at the end of 2007.

Scalable Content Dissemination. During my postdoc at Inria, I started working with Maxime Monod, a then PhD student of Rachid Guerraoui at EPFL, on the application of epidemic protocols to video streaming. Our first contribution in this setting consisted of an experimental analysis of gossip for high-bandwidth dissemination that allowed us to fine tune important parameters of gossip protocols such as fanout and refresh rate. In particular, we observed that in real settings fanout cannot be increased arbitrarily as suggested by some theoretical models [21]. Within the same project, we also proposed heuristics that improve high-bandwidth content dissemination [19]. I present these results in chapters 3 and 4.

More recently, I resumed working on a related topic with the beginning of the PhD of Quentin Dufour, whom I co-advise with David Bromberg. Quentin's PhD topic revolves around the design of scalable protocols for large scale distributed systems within the context of the O'Browser ANR project. As a first step of his work, I proposed Quentin to work on the optimization of gossip dissemination by integrating network-coding [69, 113] into state-of-the-art dissemination algorithms [60]. This work, described in Chapter 5 resulted in a paper that was recently accepted at INFOCOM 2019 [6].

While working on Maxime Monod in 2008-2010, I had also proposed *HEAP*, a novel Gossip protocol that targets environments in which nodes have heterogeneous capabilities [20]. I also present this work in Chapter 4. More recently, I worked on a different form of heterogeneity in the context of gossip protocols, as a co-advisor of Pierre-Louis Roman's PhD thesis. In particular, we proposed a gossip protocol that addresses heterogeneity in the requirements of receiving nodes [10]. Specifically, Gossip Primary Secondary (GPS) distinguishes two sets of nodes: primary nodes that need to receive messages fast, and secondary nodes that need to have a consistent view of the system, while waiting a little longer. We applied this protocol to design a hybrid eventual consistency criterion, Update-Query Consistency with Primaries and Secondaries (UPS), which supports these two classes of requirements.

Finally, I also worked on extending the applicability of epidemic protocols to the context of browser-to-browser networking. In 2011, Google introduced WebRTC, an API which is now becoming a standard for browser-to-browser communication. In 2013, together with Stephane Grumbach from the DICE team in Lyon, I submitted an ADT proposal aimed at exploring the use of WebRTC for Web2.0 peer-to-peer applications. Thanks to this funding, I supervised the work of Raziel Carvajal Gomez in the development of WebGC, a library for gossip-based applications running within web-browsers. We successfully demonstrated WebGC at Middleware 2014 [31] and at WISE 2015 [30]. This also bootstrapped a collaboration with the GDD team led by Pascal Molli in Nantes on the design of a peer-sampling protocol that addresses the tendency of Web 2.0 applications to be subject to popularity bursts [2].

Recommenders and Privacy. I also applied gossip to decentralized data-oriented applications like recommendation systems. I started working on this topic in the context of Anne-Marie Kermarrec's Gossple ERC project. My first contribution on this topic was on the main Gossple paper [18], which defined an architecture to automatically infer personalized connections in an Internet-scale decentralized system. As a natural follow-up of Gossple, I worked on the design and the implementation of WhatsUp a decentralized instant-news recommender [32, 14], and on several satellite projects that applied the idea of decentralized recommendation in diverse contexts such as CDNs [13] or distributed marketplaces [17, 5].

A large part of my contributions in the context of recommendation focus on techniques to guarantee privacy to

the users of a recommender system. In the context of the WhatsUp recommender we proposed two obfuscation mechanisms [3, 11] that hide the exact profiles of users without significantly decreasing their utility. I describe these contributions in chapters 8 and 9. In the same context, I also worked on anonymity-based techniques and proposed FreeRec [4, 15], an architecture consisting of three layers of gossip-based overlay protocols.

More recently, I started a line of research that focuses on privacy preservation techniques for decentralized computation. This has led for now to two protocols [25, 35] that apply secret sharing schemes to gossip averaging and I am leveraging this work in the context of the PhD thesis of Amaury Bouchra Pilet, which focuses on decentralized algorithms for privacy-preserving machine learning.

Other Research Contributions. Besides my work on the application of epidemic protocols to various problems, I also addressed a variety of related topics. In the context of recommendation, for example, I worked on HyRec, a semi-decentralized solution to provide a cost-effective personalization platform to web-content editors. Even if partially centralized, HyRec still takes inspiration from our previous work on gossip-based KNN protocols [18, 14] and shows its applicability in server-based settings

As a co-advisor of the PhD thesis of Stephane Delbruel, I explored the use of tags to optimize data placement in distributed storage systems. In a first paper, [8], we carried out an extensive analysis of a YouTube dataset and showed that tags can be used to predict the countries from which videos will be viewed. This allowed us to propose a data placement strategy that can optimize video storage. In a later paper [9], I contributed my experience on decentralized similarity-based overlays [18] to help Stephane define and evaluate a distributed architecture that estimates the aggregated affinity of a new video with all the users in a country.

As another example of scalable systems, in 2012, I collaborated with Kostas Kloudas a then PhD student of Anne-Marie Kermarrec on the design of a cluster-based backup system [16]. Finally, in the context of Pierre-Louis Roman's thesis, I have also been working on Dietcoin, an extension to the Bitcoin protocol that makes it possible for lightweight devices such as smartphones to verify the legitimacy of the transactions they are involved in. We published a preliminary version of Dietcoin at ARM 2016 [26], while the most current version is available as a technical report [33].

Finally, besides the above applied contributions I have also worked on more theoretical problems. In 2014, I picked up an old piece of work that I had started during my PhD, on solution methods for the quadratic shortest path problem [12, 1]. In 2015, I instead collaborated with Michel Raynal and Hicham Lakhlef on the problem of distance-2 coloring of a graph in synchronous broadcast networks [7].

Organization of this manuscript. This manuscript attempts to provide a synthesis of the above research contributions by focusing on the application of epidemic protocols to two problems: data dissemination, and decentralized recommendation.

Contents

Contents iii

1 Introduction 1

I Epidemic Data Dissemination 3

2 Epidemic Content Dissemination 5
- 2.1 State of the Art In Content Dissemination . . . 5
- 2.2 Three-Phase Gossip . . . 6
- 2.3 Contributions in Chapters 3 through 5 . . . 7

3 Analyzing Gossip-Based Streaming 9
- 3.1 Overview . . . 9
- 3.2 Tailoring gossip-based dissemination . . . 9
- 3.3 Evaluation . . . 10
  - 3.3.1 Impact of varying the fanout . . . 11
  - 3.3.2 Proactiveness . . . 13
  - 3.3.3 Performance in the presence of churn . . . 14
- 3.4 Concluding remarks . . . 14

4 Improving Gossip-Based Streaming 17
- 4.1 Overview . . . 17
- 4.2 Problem Statement . . . 18
- 4.3 HEAP . . . 18
  - 4.3.1 Heterogeneity Management . . . 18
    - 4.3.1.1 Fanout Adaptation . . . 19
    - 4.3.1.2 Sample-Based Capability Aggregation . . . 19
  - 4.3.2 Reliability . . . 19
    - 4.3.2.1 Codec . . . 19
    - 4.3.2.2 Claim . . . 20
- 4.4 Experimental Setting . . . 21
  - 4.4.1 Protocol Parameters . . . 21
  - 4.4.2 Bandwidth Limitation . . . 21
  - 4.4.3 Metrics . . . 22
- 4.5 Performance Evaluation . . . 22
  - 4.5.1 HEAP Test Drive: Heterogeneous Bandwidth . . . 22
  - 4.5.2 Understanding Bandwidth Usage . . . 24
  - 4.5.3 Reliability: Need for Codec and Claim . . . 24

4.5.4 Sensitivity Analysis . . . 25
4.5.4.1 RPS Gossip Interval . . . 25
4.5.4.2 View Size . . . 26
4.5.5 Scalability . . . 27
4.5.6 Responsiveness to Churn . . . 28
4.5.7 Cohabitation with External Applications . . . 28
4.6 Concluding Remarks . . . 31

5 Augmenting Gossip with Network Coding **33**
5.1 Overview . . . 33
5.2 Background on network coding and challenges . . . 34
5.3 Contribution . . . 35
5.3.1 System model . . . 35
5.3.2 Solving RLNC challenges . . . 35
5.3.3 CHEPIN . . . 36
5.4 Application to Pulp . . . 38
5.5 Evaluation . . . 39
5.5.1 Experimental setup . . . 39
5.5.2 Configuring the Protocols . . . 39
5.5.3 Bandwidth and delay comparison . . . 39
5.5.4 Adaptiveness optimization . . . 40
5.5.5 Behaviour under network variation . . . 42
5.6 Conclusion . . . 43

II Decentralized Recommenders and Privacy **45**

6 Context and Background **47**
6.1 Recommender Systems . . . 47
6.2 Contributions in Chapters 7 through 9 . . . 48
6.3 Focus on Decentralized KNN . . . 48
6.4 Experimental Setting . . . 50
6.4.1 Metrics . . . 50
6.4.2 Datasets . . . 50
6.4.3 Adversary Models . . . 51

7 WhatsUp: A Decentralized Instant News Recommender **53**
7.1 Overview . . . 53
7.2 WUP . . . 54
7.2.1 News item . . . 54
7.2.2 Profiles . . . 55
7.2.3 Updating profiles . . . 55
7.2.4 Initialization . . . 55
7.2.5 *Profile window* . . . 55
7.3 BEEP . . . 56
7.3.1 Forwarding a disliked item . . . 56
7.3.2 Forwarding a liked item . . . 57
7.4 Experimental setup . . . 57
7.4.1 WhatsUp Competitors . . . 58
7.4.2 Evaluation metrics . . . 58

7.4.3 WHATSUP system parameters . . . 58
7.5 Evaluation . . . 59
7.5.1 Similarity metric . . . 59
7.5.2 Amplification and orientation . . . 60
7.5.3 Implicit nature of WHATSUP . . . 61
7.5.4 Simulation and implementation . . . 63
7.5.5 Message loss . . . 63
7.5.6 Bandwidth consumption . . . 64
7.5.7 Partial information . . . 64
7.5.8 Sociability and popularity . . . 64
7.6 Concluding remarks . . . 65

8 Privacy-Preserving Distributed Collaborative Filtering 67
8.1 Overview . . . 67
8.2 Obfuscation Protocol . . . 68
8.2.1 Overview . . . 68
8.2.2 Profile Updates . . . 69
8.3 Randomized Dissemination . . . 70
8.4 Experimental setup . . . 72
8.4.1 Alternatives . . . 72
8.4.2 Evaluation metrics . . . 73
8.5 Performance evaluation . . . 73
8.5.1 Compacting profiles . . . 73
8.5.2 Filtering sensitive information . . . 74
8.5.3 Randomizing the dissemination . . . 75
8.5.4 Evaluating 2-DP . . . 75
8.5.5 Privacy versus accuracy . . . 76
8.5.6 Resilience to a censorship attack . . . 77
8.5.7 Bandwidth consumption . . . 78
8.6 Concluding Remarks . . . 78

9 Hide & Share: Landmark-based Similarity for Private KNN Computation 79
9.1 Overview . . . 79
9.2 System Model . . . 80
9.2.1 Adversary Model . . . 80
9.2.2 Problem Statement . . . 80
9.3 The Hide & Share Landmark-based Similarity . . . 81
9.3.1 Landmark Generation . . . 82
9.3.1.1 Computation Confidentiality . . . 82
9.3.1.2 Independence of peer profiles . . . 83
9.3.1.3 Fair Landmark generation . . . 83
9.3.1.4 Minimal information release . . . 83
9.3.2 Similarity approximation . . . 84
9.4 Evaluation . . . 84
9.4.1 Methodology . . . 84
9.4.1.1 Simulator . . . 84
9.4.1.2 Datasets . . . 84
9.4.1.3 Evaluation metrics . . . 84
9.4.1.4 Default parameters . . . 85
9.4.2 Recommendation quality . . . 85

9.4.3 Neighborhood quality . . . 86
9.4.4 Privacy . . . 87
9.4.5 Overhead . . . 88
9.4.5.1 Computational overhead . . . 88
9.5 Privacy Guarantee . . . 89
9.5.1 Conditional Entropy as a Measure of Information Leakage . . . 89
9.5.2 Leaked Information and the Landmark Matrix . . . 90
9.5.3 Computation of Expected Information Leakage . . . 91
9.6 Conclusion . . . 91

III Conclusion **93**

10 Conclusion and Perspectives **95**
10.1 Summary of Contributions . . . 95
10.2 Research Perspectives . . . 96
10.2.1 Optimizing Data Dissemination . . . 96
10.2.2 Privacy-Preserving and Decentralized Recommenders . . . 96
10.2.3 Privacy-Preserving Decentralized Data Analytics . . . 97
10.2.4 Blockchain . . . 98

List of Figures **99**

Chapter 1

Introduction

Initially introduced in the context of replicated databases by Demers et al [171], gossip protocols have found application in a variety of settings, the most predominant being that of content dissemination [171, 159, 114, 93, 60, 50]. Their robustness and ability to cope with network dynamics has prompted a number of researchers to investigate their behavior both theoretically [132, 146, 113, 69], and experimentally [93, 60, 50, 141, 130, 110, 158]. The reason for gossip's success lies partly in its simplicity. At a high level, a gossip protocol involves periodic exchanges of messages of bounded size between pairs of nodes, with some form of randomness in the choice of communication partners [102]. Moreover, the inherent redundancy that characterizes gossip protocols allows them to operate in a variety of network environments even when message delivery is unreliable.

From a practical standpoint, almost any gossip protocol can be modeled in terms of a generic protocol consisting of an active cycle and a passive cycle, which in turn execute three main operations: *Peer selection*, *data exchange*, and *data processing* [106]. The *active cycle* executes periodically and starts by randomly selecting one or more peers to communicate with (Peer selection). It then selects some data and sends it to the chosen peer (data exchange). The *passive cycle* executes whenever a peer receives a gossip message and updates the peer's state with the received message (data processing). A key parameter of gossip protocols consists of the fanout, the number of peers contacted at each communication round. In general, increasing the fanout makes it possible to achieve faster and more reliable dissemination even though Chapter 3 shows that this is not always the case. But reliability can also be increased by disseminating the same message for multiple rounds [132], or by combining push and pull dissemination [60].

A key component of gossip-based system lies in a special type of gossip protocol known as peer sampling [106, 2]. Peer sampling provides each node with the abstraction of an oracle that, at any time, can provide it with a set of references to other nodes sampled randomly from the network. Initial protocols for peer sampling operated using a random-walk model [142], but more recent proposals and implementations adopt a shuffling scheme [106, 126, 84, 57] that maintains a continuously changing overlay network whose properties resemble those of a random graph.

The first application of gossip was that of information dissemination [171, 159, 114, 93, 60, 50], a topic that continues to attract the interest of both theoretical [113, 69, 47, 36] and applied researchers [6, 10, 50, 44, 60]. But gossip has been applied in a variety of different contexts. In addition to maintaining random peer-sampling graphs, gossip has been applied to reproduce arbitrary topologies [83, 100], to build distributed hash tables [74] or small-world overlays [99]. It serves as base protocol for content-delivery networks [13, 13], publish-subscribe systems [121, 98], and recommender systems [56, 14]. Gossip finds application in cloud-oriented applications such as key-value stores [104], and more recently in the the context machine learning algorithms [37, 43], and blockchain systems [180, 95].

In this manuscript, I attempt demonstrate the versatility of gossip in areas ranging from large-scale content dissemination to big-data applications. I do so by covering several contributions on data dissemination and decentralized recommendation. Data dissemination, discussed in Part I, represents the most logical application of epidemic protocols and was also the first application I tackled during my research career. The part starts with an introductory Chapter 2, which presents an overview of the state of the art in data dissemination, and defines some basic notions used in the remaining chapters. Then Chapter 3 presents the experimental analysis of gossip for high-bandwidth content

dissemination from [21]. Chapter 4 presents *HEAP* [20] as well the associated heuristics from [19]. Finally, Chapter 5 presents the more recent work on network coding and gossip that we published at INFOCOM 2019 [6]. Part II of this manuscript provides instead a summary of my research on decentralized recommenders and privacy. Like Part I, Part II starts with an introductory Chapter 6, which presents the state of the art and provides some basic notions. Then Chapter 7 presents WhatsUp [14], the decentralized instant-news recommender I worked on in the context of Anne-Marie Kermarrec's Gossple grant. Then Chapter 8 presents a data obfuscation technique designed to protect privacy in the context of WhatsUp with limited impact on recommendation quality. Chapter 9 concludes Part II by presenting a more general privacy-preserving similarity metric that exploits distances from randomly-generated profiles called landmarks [11]. Part III concludes the manuscript by presenting some future research directions and perspectives in Chapter 10.

Part I

Epidemic Data Dissemination

Chapter 2

Epidemic Content Dissemination

The first part of this manuscript explores the application of epidemic protocols (also known as gossip protocols[1]) to the realm of content dissemination. This chapter provides a brief state of the art on epidemic protocols for dissemination in Section 2.1. Then, Section 2.2 focuses on a specific form of three-phase gossip that has been used in high-bandwidth content dissemination and that constitutes a basic building block for the contributions in Chapter 3 and Chapter 4. In this setting, naive push-based gossip cannot offer reasonable performance, and applications require push-pull techniques possibly combined with erasure coding (Chapter 4) or network coding (Chapter 5). Finally, Section 2.3 summarizes how the following three chapters address the limitations of the state of the art.

2.1 State of the Art In Content Dissemination

In their seminal paper [170], Xerox researchers introduced three forms of gossip protocols to disseminate information: push, pull and push-pull. Push protocols work by having informed nodes relay messages to their neighbors. Some protocols are active, as they have a background thread that regularly retransmits received rumors, like balls and bins [132]. Other protocols adopt a reactive approach, where rumors are directly forwarded to the node's neighbors upon reception, like infect-and-die and infect-forever protocols [129]. Push protocols are particularly efficient to quickly reach most of the network, however reaching all the nodes takes more time and involves significant redundancy, and thus bandwidth consumption.

In pull protocols, nodes that miss messages ask other nodes for the missing messages. As a consequence, *pull protocols* more efficiently reach the last nodes of the network, as inherently, they get messages with higher probability. However, they require sending more messages over the network: (i) one to ask for a missing message, and (ii) another one for the reply that contains the missing message. Furthermore a mechanism or a rule is needed to know what are the missing messages to pull, which explains why these protocols are generally used in conjunction with a push phase.

Push-Pull protocols try to retain the best from push and pull protocols by reaching as many nodes as possible with minimal redundancy on the push phase. Then, nodes that have not received a message will send pull requests to other nodes in the network. By ordering messages, Interleave [110] proposes a solution to discover the missing messages in the pull phase, but works only with a single source. Instead of ordering messages, Pulp [60] piggybacks a list of recently received message identifiers in every sent message, allowing multiple sources.

Live Streaming and Gossip. The first protocols addressing peer-to-peer live video streaming exploited tree-based solutions [86]. Single trees however leave a large percentage of nodes underutilized. Thus, protocols like Splitstream [138] or Chunkyspread [120] combine multiple intertwining trees in which the internal nodes of a tree act as leaves in the others.

[1] We will use the two terms interchangeably in the following.

More recently, some authors have focused on the combination of trees with other dissemination structures. For example, [54] uses server replication to improve availability in video-on-demand systems, while [53] addresses the trade-off between bandwidth efficiency and stability by using two trees, respectively optimizing latency and reliability. Bullet [148] follows similar approach using a tree to disseminate most of the packets, and employing a mesh topology to disseminate the remaining ones.

Many mesh-based systems [92, 94, 97, 85] use a generic graph topology to extend the idea of intertwining trees. For example Coolstreaming [92] and Gridmedia [111] build multi-tree structures dynamically on top of an unstructured overlay when nodes perceive they are stably served by their neighbors. Others, like Tribler [73], directly exploit the mesh topology and manage bandwidth capabilities using techniques like tit-for-tat [87].

Gossip-based streaming takes the idea behind multiple trees and mesh topologies to the extreme by establishing a new dissemination tree for each stream packet [21]. While initially introduced to disseminate *small updates* [171, 159, 141, 147], gossip found application in a number of application-level multicast protocols [141, 147, 159]. But this early work omitted all bandwidth considerations. CREW [114] constitutes one of the first gossip protocols to take into account bandwidth, but in the context of file sharing. Nodes simply stop offering data when their bandwidth is exhausted. Smart Gossip [118] adopts a similar approach in wireless networks. Peerstreamer [52, 59], addresses heterogeneity by varying the number of neighbors to which it sends propose messages similar to the solution we present in Chapter 4. But unlike our solution, it bases its control actions on the queuing delays resulting from bandwidth constraints.

Finally, several authors [111, 90, 94, 97, 77] have analyzed the impact of packet-scheduling strategies in gossip-based streaming. In particular, [77] proves the optimality of latest-useful and deadline-based chunk scheduling algorithms. It would be interesting to explore to what extent these techniques could improve HEAP's performance. It would also be interesting to apply HEAP in the context of HTTP streaming. Existing systems [61, 62, 51] rely on servers for operations like membership management, and consider much less constrained bandwidth scenarios.

Network Coding and Gossip. One of the issues that characterizes gossip dissemination consists of its redundancy. While useful in the presence of message loss or node failures, redundancy can generate excessive overhead particularly in the presence of bandwidth constraints. Some authors have proposed the use of erasure coding or Random Linear Network Coding to better manage the inherent redundancy of gossip. Instead of increasing the number of copies of a single message, coding makes it possible to exploit redundancy across multiple messages. This reduces the need to increase parameters like the fanout that fuel gossip's redundancy, and leads to more efficient dissemination.

Existing work has applied these techniques in the context of single-source gossip dissemination [80, 110, 139]. For example, [80] uses fountain codes to eliminate the overly redundant dissemination that happens once a message has reached a large portion of the network. Multi-sender scenarios have also been studied, but, to the best of our knowledge, only from a theoretical perspective [113, 69], with the assumption that messages are previously ordered by some oracle. A practical application of RLNC gossip in a multi-sender scenario, on the other hand, requires organizing messages in generations that are encoded together, and techniques to determine which messages belong to which generation, while representing their linear combinations in a compact manner. We address these challenges in Chapter 5.

2.2 Three-Phase Gossip

Gossip protocols were initially introduced to disseminate small updates in distributed databases [171]. In this setting, the redundancy of gossip provides fault tolerance at a limited cost. But for scenarios like high-bandwidth content dissemination, excessive redundancy may cause unacceptable overhead. For this reason, most gossip protocols for high-bandwidth dissemination follow a push-request-push pattern with an infect-and-die model [119, 93, 67]. This consists of the following three phases also depicted in Figure 2.1.

- *Propose phase.* Every gossip period, a node that has new stream packets available selects f (fanout) other nodes uniformly at random, and sends them proposal messages advertising the available packets. In Figure 2.1, node i

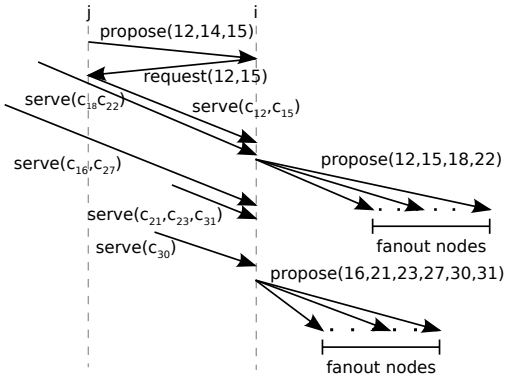


Figure 2.1: Three-phase gossip protocol.

proposes packets 12, 15, 18 and 22 in one gossip period, and packets 16, 22, 23, 27, 30 and 32 in the subsequent one.

- *Request phase.* As soon as a node receives a proposal for a set of packet identifiers, it determines which of the proposed packets it needs and requests them from the sender. In Figure 2.1, node i requests packets 12 and 15 from j.

- *Serving phase.* Finally, when the proposing node receives a request message, it replies with a message containing the packets corresponding to the requested identifiers. Nodes only serve packets that they previously proposed. In Figure 2.1, node j serves i with packets c_{12} and c_{15}.

This three-phase protocol makes gossip-based streaming practical by creating duplicates only on small propose messages and transmitting the actual payload only once to each recipient. Nonetheless, it still exhibits important limitations. First, it naturally distributes dissemination efforts over all participants. This load-balancing aspect makes gossip resilient to disconnections and message loss, but it quickly turns into a burden when operating in heterogeneous environments. For example, corporate nodes generally have plenty of available bandwidth, while upload bandwidth often constitutes the limiting factor in home-based setups. Second, while the three-phase approach effectively avoids useless redundancy in the dissemination of large [SERVE] messages, it also makes dissemination more vulnerable to churn and message loss. We address these limitations in the following chapters.

2.3 Contributions in Chapters 3 through 5

Despite the wide application of gossip in high-bandwidth gossip dissemination [90], until 2009, most of the work on gossip-based streaming had considered ideal settings; e.g., unconstrained bandwidth, no (or uniform) message loss, global knowledge about the state of all nodes. Evaluations conducted through simulation [114, 90, 108] also assumed that key parameters of gossip, such as fanout and gossip rate, could be arbitrarily tuned to improve robustness and to adequately adapt to network dynamics. The exceptions of gossip experiments in real settings assumed infinite bandwidth [93], or considered applications with low bandwidth needs, such as membership maintenance [105].

Through our research, we observed the inadequacies of such studies, which were nonetheless based on theoretical results. For example, theory shows that if nodes disseminate messages with a fanout that is at greater than a threshold of $ln(n) + c$, n being the size of the network and c a constant, then gossip dissemination will reach all nodes with high probability [146]. In our work, we showed (Chapter 3 that fanout cannot be increased arbitrarily, as too large fanout values may saturate bandwidth causing unnecessary message loss and degrading performance. This requires reducing fanout and introducing other forms of redundancy like forward error correct to recover undelivered messages

(Chapter 4). Similarly, we showed that the load balancing nature of gossip may be a disadvantage in heterogeneous settings and proposed a solution to tailor gossip to heterogeneous settings (Chapter 4).

The contributions in chapters 3 and 4 rely on a three-phase gossip-model (see Section 2.2) to limit the overhead associated with the redundancy of gossip. In Chapter 5 we take a radically different approach and design a solution for multi-source dissemination that exploits Random Linear Network Coding (RLNC). As highlighted above, applying this technique to multi-source gossip in a practical setting involves addressing two important challenges: organizing messages into generations and representing messages in linear combinations in a compact manner. Chapter 5 addresses these challenges and integrates them into Pulp, a state-of-the-art push-pull dissemination protocol [60], significantly improving its latency and bandwidth consumption.

Chapter 3

Analyzing Gossip-Based Streaming

This chapter evaluates the effectiveness of the three-phase gossip model for video streaming, and assesses the impact of its parameters on performance. In particular, it shows that the high-bandwidth characteristics of video streaming highlight challenges that were often overlooked prior to our work, particularly in the context of theoretical analyses. I contributed to this work when I was a post-doctoral researcher in the ASAP team at Inria Rennes. I closely collaborated with Maxime Monod who was then a PhD student of Rachid Guerraoui, Vivien Quema, and Anne-Marie Kermarrec. We obtained the results presented in this chapter through extensive experiments on the PlanetLab platform.

The content of this chapter is an adapted excerpt from:
Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, Maxime Monod, Vivien Quéma: Stretching gossip with live streaming. DSN 2009: 259-264.

3.1 Overview

Back in 2009, epidemic protocols were known to be effective for challenging applications like live streaming [90]. Yet, most of the work evaluating gossip had considered ideal settings; e.g., unconstrained bandwidth, no (or uniform) message loss, and global knowledge about the state of all nodes. Evaluations conducted through simulation [114, 90, 108] also assumed that key parameters of gossip, such as fanout and gossip rate, could be arbitrarily tuned to improve robustness and to adequately adapt to network dynamics. The few gossip experiments in real settings assumed infinite bandwidth [93], or considered applications with low bandwidth needs, such as membership maintenance [105].

With this contribution, we evaluated the effectiveness of three-phase gossip in a real setting, with constrained bandwidth and message loss. The experiments that follow show that gossip can be very effective in greedy and capability-constrained applications, but only within small parameter ranges. First, the resulting stream quality is very sensitive to the fanout value. The power of gossip is unleashed when the fanout is slightly larger than $ln(n)$ but degrades drastically with higher fanout values as a result of higher contention. For example, with a bandwidth cap of 700 kbps for a stream rate of 600 kbps, gossip reaches its optimal performance with a fanout of 7 (230 nodes). Second, gossip is most effective when the set of communication partners is continuously changing, particularly in the presence of churn or with tight bandwidth constraints.

3.2 Tailoring gossip-based dissemination

We consider the three-phase gossip protocol presented in Section 2.2 and explore its configuration by considering its two main parameters: fanout and proactiveness.

Fanout. Fanout is defined as the number of communication partners each node contacts in each gossip operation. As already mentioned, theory has shown that a fanout greater than $\ln(n)$ in an infect-and-die model [146] ensures a highly reliable dissemination. Theory also assumes that increasing the fanout results in an even more robust (as the probability to receive an event id increases) and faster dissemination (as the degree of the resulting dissemination tree increases). In practice, however, too high a fanout can negatively impact performance as heavily requested nodes may exceed their capabilities in bandwidth-constrained environments.

Proactiveness. We define proactiveness as the $rate$ at which a node modifies its set of communication partners. We explore two ways of modifying this set. First, the node may locally refresh its set of communication partners and change the output of selectNodes every X calls. In short, when $X = 1$ the gossip partners of the node change at every call to selectNodes (i.e., every gossip period), whereas $X = \infty$ means that the communication partners of a node never change. Second, every Y gossip periods, the node may contact f random partners asking to be inserted in their views. When $Y = 1$, a node A sends a *feed-me* message to f random partners every gossip period asking them to feed it. Each of the random f partners replaces a random node from its current set of f partners with A.

3.3 Evaluation

We evaluate the impact of fanout and proactiveness by deploying a streaming application based on Algorithm 1 over a set of 230 PlanetLab nodes. Our implementation is based on UDP and incorporates retransmission to recover lost stream packets (lines 14, 15, 25 in Algorithm 1).

```
1  Function init():
2      f := ln(n) + c
3      eventsToPropose := eventsDelivered := requestedEvents := ∅
4      start (GossipTimer(gossipPeriod))
   /* Phase 1 - Push event ids
5  Function publish(e):
6      deliverEvent(e)
7      gossip({e.id})
8  Upon (GossipTimer mod gossipPeriod) = 0 do:
9      gossip(eventsToPropose)
10     eventsToPropose := ∅ /* Infect and die
   /* Phase 2 - Request events
11 Upon receive [PROPOSE, eventsProposed] do:
12     wantedEvents := ∅
13     forall e.id ∈ eventsProposed do
14         if (e.id ∉ requestedEvents) then
15             wantedEvents := wantedEvents ∪ e.id
16     requestedEvents := requestedEvents ∪ wantedEvents
17     reply [REQUEST, wantedEvents]
18     if (e requested less than K times) then
19         start(RetTimer(retPeriod, eventsProposed))
   /* Phase 3 - Push payload
20 Upon receive [REQUEST, wantedEvents] do:
21     askedEvents := ∅
22     forall e.id ∈ wantedEvents do
23         askedEvents := askedEvents ∪ getEvent(e.id)
24     reply [SERVE, askedEvents]
25 Upon receive [SERVE, events] do:
26     forall e ∈ events do
27         if e ∉ eventsDelivered then
28             eventsToPropose := eventsToPropose ∪ e.id
29             deliverEvent(e)
*/
30     cancel(RetTimer(retPeriod, events))
   /* Retransmission                                    */
31 Upon (RetTimer(retPeriod, eventsProposed) mod retPeriod) = 0 do:
32     receive [PROPOSE, eventsProposed]
*/
   /* Miscellaneous                                     */
*/33 Function selectNodes(f): Returns a set of nodes
34     return f uniformly random chosen nodes in the set of all nodes
35 Function getEvent(event id): Returns an event
36     return the event corresponding to the id
37 Function deliverEvent(e):
38     deliveredEvents := deliveredEvents ∪ e
39     deliverEvent(e)
40 Function gossip(event ids):
41     communicationPartners := selectNodes(f)
42     forall p ∈ communicationPartners do
*/43         send(p,[PROPOSE, event ids])
```

Algorithm 1: Standard gossip protocol

Bandwidth constraints. PlanetLab nodes benefit from high bandwidth capabilities and therefore are not representative of peers with limited capabilities. We thus artificially constrain the upload bandwidths of nodes with three different

caps: 700 kbps, 1000 kbps and 2000 kbps. To limit message loss resulting from bandwidth bursts, our bandwidth limiter also implements a bandwidth throttling mechanism.

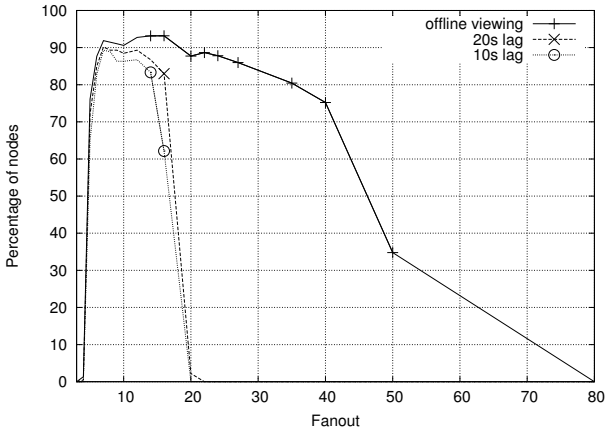
Streaming Configuration. A source node generates a stream of 600 kbps and proposes it to 7 nodes in all experiments. To provide further tolerance to message loss (combined with retransmission), the source groups packets in windows of 110 packets, including 9 *FEC coded packets*. The gossip period is set to 200 ms.

Evaluation metrics. We assess the performance of the streaming protocol along two metrics: *(i)* the *stream lag*, defined as the difference between the time at which the stream is published by the source and the time at which it is actually delivered to the player on the nodes; and *(ii)* the *stream quality*, which represents the percentage of the stream that is viewable. A window is *jittered* if it does not contain enough packets (i.e., strictly less than 101) to be fully reconstructed. A stream with a maximum of 1% jitter means that at least 99% of the windows are complete.[1]

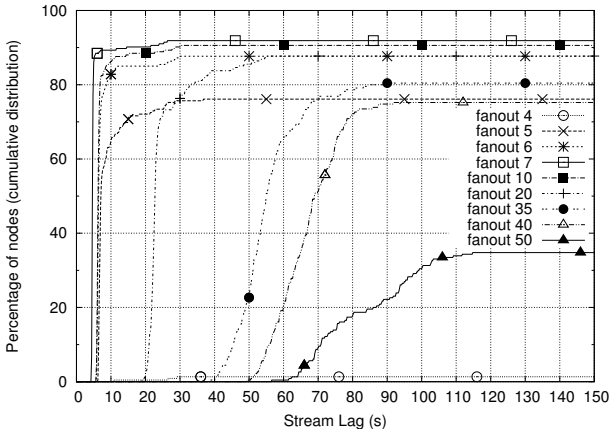
Stream lag and quality are correlated notions: the longer the acceptable lag, the better the quality. We consider stream qualities corresponding to several lag values as well as to an infinite lag, which represents the performance obtainable by a user that downloads the stream for playing at a later stage, e.g., offline viewing. Each experiment was run multiple times. We plotted the most representative one.

3.3.1 Impact of varying the fanout

We start our analysis by measuring how varying the fanout impacts each of the two metrics, with a proactiveness degree of 1 ($X = 1$). Results are depicted in Figures 3.1a–3.2a.



(a) Percentage of nodes viewing the stream with less than 1% of jitter (upload capped at 700 kbps).



(b) Cumulative distribution of stream lag with various fanouts

Figure 3.1: Percentage of nodes viewing 99% of the stream and distribution of the corresponding stream lag (upload capped at 700 kbps).

Optimal fanout range. Figure 3.1a shows the percentage of nodes that can view the stream with less than 1% jitter for various stream lags in a setting where all nodes have an upload capability of 700 kbps. The plot clearly highlights an optimal range of fanout values (from 7 to 15 in this configuration) that gives the best performance independently of lag.

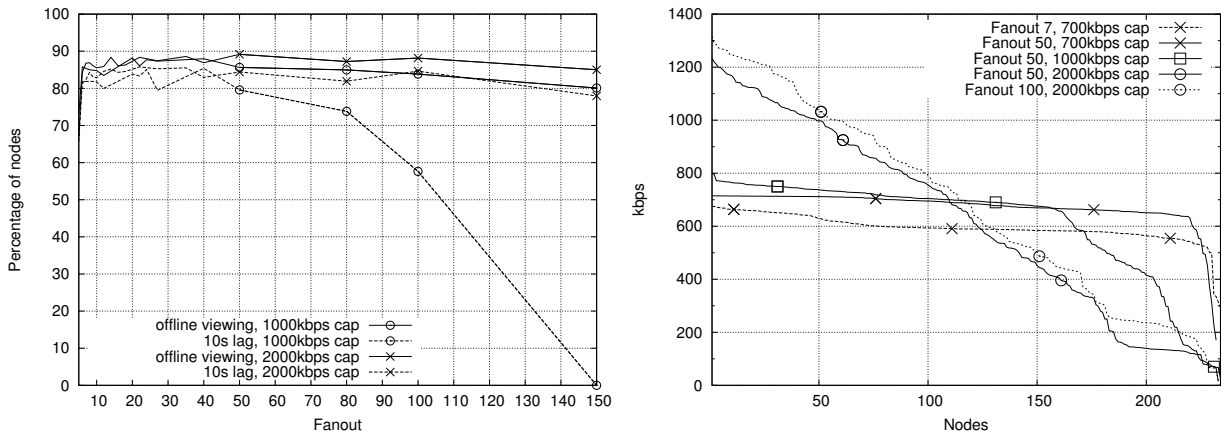
[1] An incomplete window does not mean that the window is unusable. Using systematic coding, a node receiving 100 out of the 101 original packets, experiences a 99% delivery in that window.

Lower fanout values are insufficient to achieve effective dissemination, while larger values generate higher network traffic and congestion, thus decreasing the obtainable stream quality.

The plot also shows that while the lines corresponding to finite lag values have a bell shape without any flat region, the one corresponding to offline viewing does not drop dramatically until a fanout value of 40. The bandwidth throttling mechanism is in fact able to recover from the congestion generated by large fanout values once the source has stopped generating new packets. For fanouts above 40, on the other hand, such recovery does not occur.

Critical lag value. A different view on the same set of data is provided by Figure 3.1b. For each value t, the plot shows the percentage of nodes that can view at least 99% of the stream with a lag shorter than t. A fanout in the optimal range (e.g., 7) causes almost all nodes to receive a high-quality stream after a critical lag value ($t = 5$ s for a fanout of 7). Moderately larger fanout values cause this critical value to increase ($t = 22$ s for a fanout of 20), while for fanouts above 35, no critical value is present. Rather congestion causes significant performance degradation. With a fanout of 40, only 20% of the nodes experience a lag shorter than 60 s, and a lag of 90 s is necessary to reach 75% of the nodes.

Behavior with less tight distributions. The presence of an optimal fanout value clearly results from bandwidth limits. Figure 3.2a complements the picture by showing how fanout affects performance under less critical conditions: 1000 kbps and 2000 kbps of capped bandwidth. As available bandwidth increases, the range of good fanout values clearly becomes larger and larger and tends to move to the right. With an available bandwidth of 1000 kbps, which is more than 1.67 times the stream rate, it is still possible to identify a clear region outside of which performance degrades significantly. With 2000 kbps of available bandwidth, both the 10 s-lag and the offline performance figures appear to remain high even with very large fanout values. This behavior may be better understood by examining how bandwidth is actually used by the PlanetLab nodes involved in the dissemination.



(a) Percentage of nodes viewing the stream with less than 1% of jitter.

(b) Distribution of bandwidth usage among nodes.

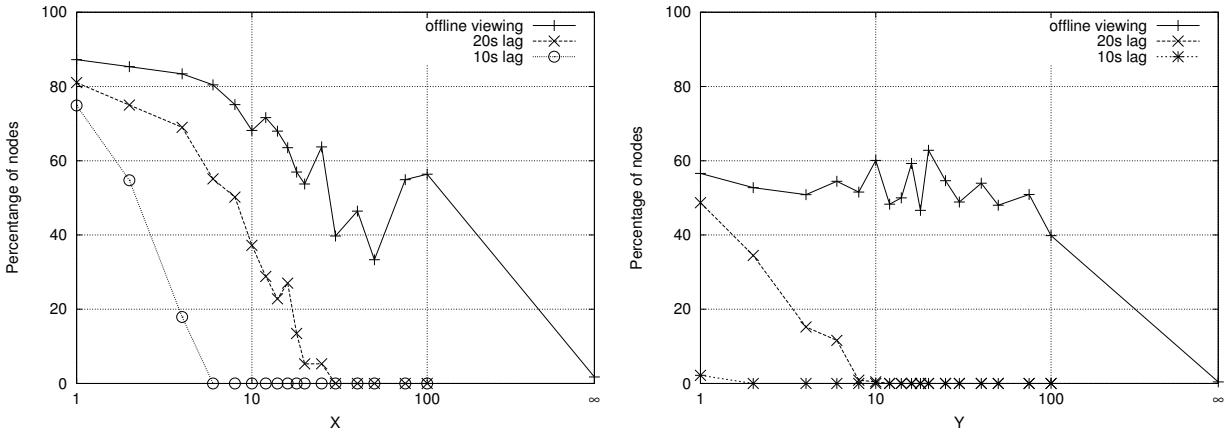
Figure 3.2: Percentage of nodes viewing 99% of the stream and bandwidth usage distribution with different fanout values and upload caps.

Bandwidth usage with different fanouts values. Figure 3.2b shows the distribution of bandwidth utilization over all the nodes involved in the experiments sorted from the one contributing the most to the one contributing the least. The plot immediately highlights an interesting property: even though all the considered scenarios have a homogeneous bandwidth cap, the distribution of utilized bandwidth is highly heterogeneous. This behavior is a direct result of the three-phase protocol employed for disseminating large content.

According to Algorithm 1, all nodes contribute to the gossip dissemination by sending their proposal messages to the same number of nodes. However, the contribution of a node in terms of serve messages depends on the probability that its proposal messages are accepted by other nodes. In general, nodes with low-latency and reliable connections (i.e., *good* nodes) have higher probabilities to see their proposals accepted. This is confirmed by Figure 3.2b. The plot also shows that the heterogeneity in bandwidth utilization increases with the amount of available bandwidth. For example the lines for the 700 kbps bandwidth cap show an almost homogeneous distribution apart from a small set of *bad* nodes. The latencies exhibited by good nodes when their bandwidth utilization is close to the limit causes other nodes to work more, thus equalizing bandwidth consumption. On the other hand, with a fanout of 50 in the 1000 kbps and 2000 kbps scenarios and with a fanout of 100 in the 2000 kbps scenario, good nodes have enough spare capacity to operate without saturating their bandwidths. As a result, the contribution of nodes remains highly heterogeneous.

3.3.2 Proactiveness

Next, we present our analysis of gossip proactiveness by showing how refreshing the set of communication partners affects performance. Figure 3.3a presents the results obtained by varying the *view refresh rate*, X, in a scenario with a 700 kbps bandwidth cap. The plot shows three lines corresponding to stream lags of 10 s and 20 s as well as to offline viewing. In all cases, the protocol obtains the best performance when varying the set of gossip partners at every communication round. If on the other hand, the set of communication partners remains constant for long periods of time, a small set of nodes end up having the responsibility of feeding large numbers of nodes for as long as they keep being selected early in the dissemination process. This means that their upload rates remain constantly higher than their allowed bandwidth limits, ultimately resulting in high levels of congestion, huge latencies, and message loss.



(a) Refresh rate.

(b) Request rate.

Figure 3.3: Percentage of nodes viewing the stream with at most 1% jitter as a function of the refresh rate X and request rate Y.

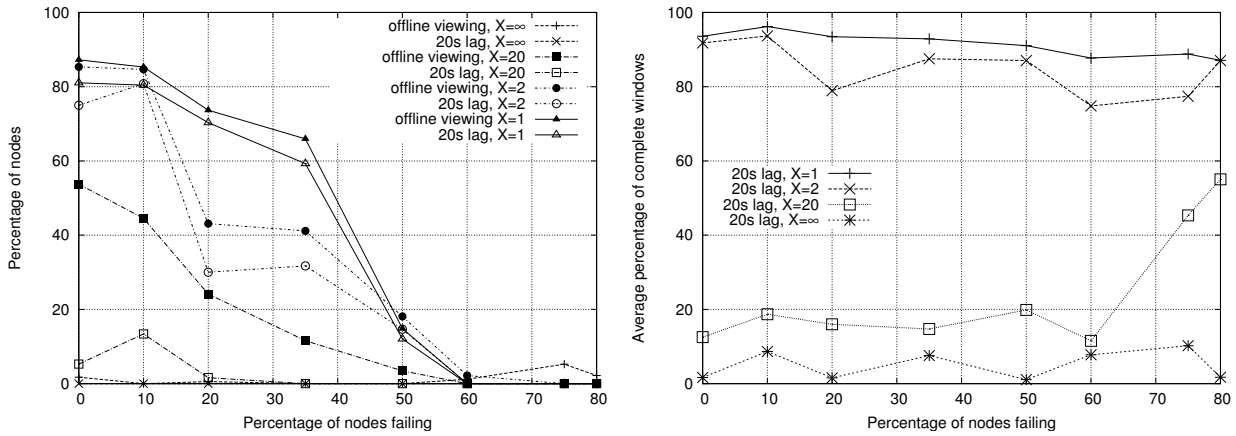
In accordance with these observations, Figure 3.3a shows that the slope at which the curves decrease with X is most negative for a lag of 10s. This is because longer values of lag allow the bandwidth throttling mechanism more time to recover from the bursts generated by a constant set of communication partners. Nonetheless, a completely static dissemination mesh invariably yields bad performance even for offline viewing as the load becomes then concentrated on a very small set of nodes for the entire experiment.

Requesting nodes to update their views. A second way to modify the proactive behavior of the considered streaming protocol is for nodes to periodically request a new set of nodes to feed them, i.e., every Y dissemination rounds. In

Figure 3.3b we show the results obtained with different values of Y. Results show that this technique remains inferior to the simpler approach of choosing a *view refresh rate* of $X = 1$, as discussed above. The additional messages used by this approach may in fact be lost or delayed while the node is congested, resulting in a larger Y than planned.

3.3.3 Performance in the presence of churn

Finally, we evaluate the impact of proactiveness in the presence of churn with $Y = \infty$. Results are depicted in Figures 3.4a and 3.4b. The experiments consist in randomly picking a percentage of nodes and making them fail simultaneously. We compare the baseline results (e.g., when no nodes fail) with those with increasing percentages, from 10% to 80%, of failing nodes. Figure 3.4a shows the percentage of remaining nodes experiencing less than 1% jitter after the catastrophic failure, for values of X of 1, 2, 20 and ∞.



(a) Percentage experiencing less than 1% jitter for different values of X.

(b) Average percentage of complete windows.

Figure 3.4: Surviving nodes: percentage viewing 99% of the stream, and average percentage of complete windows.

Figure 3.4a highlights that a completely dynamic mesh offers the best performance in terms of ability to withstand churn. With 35% churn, a proactiveness of $X = 1$ is able to deliver an unaffected stream to 60% of the remaining nodes, while the percentage drops to 32% for $X = 2$ and a stream lag of 20 seconds. The results obtained with large values of X and in particular when $X = \infty$ show very high degrees of variability from experiment to experiment. The resulting static or semi-static random graph may become completely unable to disseminate the stream if failing nodes are close to the source or it may appear extremely resilient to churn if failing nodes are located at the edge of the network. On average, performance therefore favors completely dynamic graphs ($X = 1$).

Figure 3.4a only shows how many nodes manage to remain completely unaware of the churn event. To characterize the extent of the performance decrease experienced by all surviving nodes, Figure 3.4b shows the average percentage of decoded windows over the total number of windows streamed by the source. With $X = 1$, the protocol is almost unaffected by churn, and nodes correctly receive over 90% of the windows for all churn percentages lower than 80%. In addition, almost all the missing windows turn out to be concentrated in a time frame of 5 s to 10 s around the churn event when 20% and 80% of nodes fail (not shown in the plot).

3.4 Concluding remarks

This chapter highlighted two important aspects of the application of three-phase gossip to video streaming and other bandwidth-intensive applications. Message loss and limited bandwidth significantly restrict the range of parameter values in which gossip can successfully operate. First, the fanout cannot be increased arbitrarily to improve reliability

and latency, but must remain small enough to prevent bandwidth saturation. Second, the set of communication partners should change frequently, at every communication round, in order to minimize congestion and provide an effective response to churn. These results challenged previous analyses carried out mostly by simulation in close-to-ideal settings, and provide a starting point for the protocols we present in the following two chapters.

Chapter 4

Improving Gossip-Based Streaming

The previous chapter showed that while three-phase gossip can be an effective tool for video streaming, its performance strictly depends on parameters like fanout and proactiveness. This chapter builds on these results and considers a further challenge: heterogeneous-bandwidth. It first shows that heterogeneous bandwidth can seriously hamper the ability of gossip to provide effective content dissemination. Then it introduces HEAP *HEterogeneity-Aware gossip Protocol*, a peer-to-peer streaming protocol targeting heterogeneous bandwidth scenarios. First, HEAP includes a fanout-adaptation scheme that tunes the contributions of nodes based on their bandwidth capabilities. Second, HEAP comprises heuristics that significantly improve reliability. Like for Chapter 3, the research presented in this chapter results from a collaboration with colleagues at EPFL and at Inria. While the content is essentially based on the two publications mentioned below, this chapter also presents some additional experiments that are further detailed in a technical report [34].

> The content of this chapter is an adapted excerpt from the following set of papers:
> Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, Boris Koldehofe, Martin Mogensen, Maxime Monod, Vivien Quéma: Heterogeneous Gossip. Middleware 2009: 42-61
> Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, Maxime Monod: Boosting Gossip for Live Streaming. Peer-to-Peer Computing 2010: 1-10

4.1 Overview

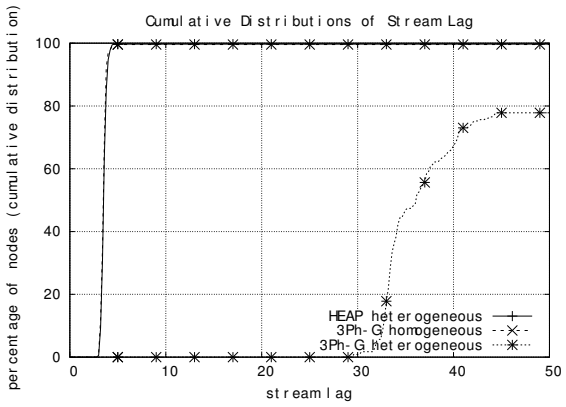
The gossip protocol we presented in the previous chapters is inherently *load balancing* in that it asks all nodes to contribute equally to the streaming process. This sharply contrasts with the highly heterogeneous nature of large-scale systems. The presence of nodes with highly diversified available bandwidths, CPU capabilities, and connection delays leads standard gossip protocols to inefficient resource utilization. Highly capable nodes end up being underutilized, while resource-constrained nodes can be overloaded to the point that they provoke system-wide performance degradation. Finally, resource availability is inherently dynamic. Consider a user watching a live stream of the football world-cup final. In the middle of the game, his son starts using another computer on the same network to upload some pictures and make a Skype call. This dramatically changes the amount of bandwidth available for the video stream thereby disrupting performance.

This chapter evaluates the impact of this heterogeneity and presents *HEAP*, a gossip-based streaming protocol, designed to operate in large-scale heterogeneous environments. *HEAP* achieves efficient dissemination by adapting resource consumption to heterogeneous resource availability, while supporting significant levels of message loss, and operation in the presence of other bandwidth consumers.

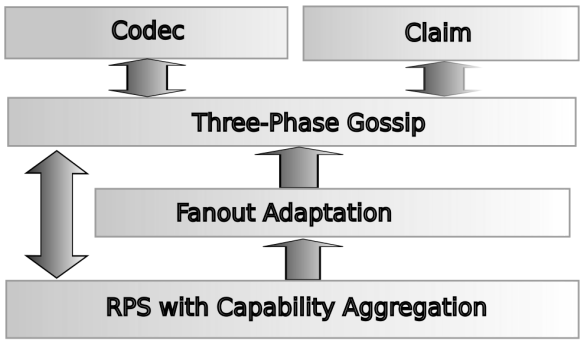
4.2 Problem Statement

The previous chapter showed the effectiveness of three-phase gossip in the presence of constrained bandwidth provided that (i) fanout remains within a limited range and (ii) that nodes change their communication partners as often as possible. Yet, all the experiments presented in Chapter 3 considered a uniform setting in which all nodes have the same upload-bandwidth capability. Reality often differs from this assumption, so this chapter explores what happens in the presence of heterogeneous capabilities.

Figure 4.1a shows performance in terms of stream lag of the three-phase gossip protocol in two scenarios. A uniform one like those considered in Chapter 2, and a heterogeneous one. Both scenarios have the same average upload-bandwidth capability of 691kbps, but the second is skewed so that only 60% of the node have a bandwidth greater than the stream rate. The difference between the two scenarios is striking. In the homogeneous setting, three-phase gossip effectively delivers a clear stream to all nodes within less than 4s. In the heterogeneous case, only 79% of the nodes manage to ever receive a clear stream, and with a lag ranging between 29s and 45s.



(a) Three-phase gossip performance



(b) Overall Architecture of *HEAP*

Figure 4.1: Three-phase gossip performance in homogeneous and heterogeneous settings (left); and architecture of *HEAP* (right)

The reason for this drop in performance lies in the contrast between the load-balancing nature of gossip protocols and the intrinsic heterogeneous nature of large-scale distributed systems. In the rest of this chapter, we present a protocol that eliminates this contrast by providing, in a heterogeneous setting, the same stream lag as the standard protocol in the homogeneous case.

4.3 HEAP

HEAP, augments three-phase gossip with four novel mechanisms, as depicted in Figure 4.1b: *Fanout Adaptation*, *Capability Aggregation*, *Codec*, and *Claim*. In the following we describe each of these mechanisms by considering their two main goals: *heterogeneity management* (*Fanout Adaptation*, *Capability Aggregation*) and *reliability* (*Codec*, *Claim*).

4.3.1 Heterogeneity Management

To address bandwidth heterogeneity, HEAP tunes the contribution of each node to the protocol so that it matches the corresponding upload capability. The key knob in this adaptation process consists of the *fanout*, the number of partners contacted by a node at each communication round.

4.3.1.1 Fanout Adaptation

A node's fanout not only controls the number of [PROPOSE] messages sent during the first phase of the protocol; it also impacts the overall bandwidth consumption. As explained in Section 2.2, [SERVE] messages comprise the vast majority of a node's traffic because of their larger size and of the high frequency at which they are sent. This allows us to approximate upload bandwidth by counting the number of [SERVE] messages.

Let us consider a node, i, that sends a [PROPOSE] message for a stream packet to f_i other nodes. In a non-congested setting, we can assume that all the f_i recipients have the same probability p of accepting it. Thus, node i will send pf_i [SERVE] messages as a result of its proposal. More generally, the number of [SERVE] messages sent by a node per unit of time is directly proportional to its fanout. Nodes can therefore control their employed upload bandwidths by selecting their fanout values proportionally to their upload capabilities. However, simply increasing or decreasing the fanout according to capability values may lead to undesired consequences. For example, our analysis in Chapter 3 shows that in a homogeneous setting, performance drops with too large or two low fanout values.

To avoid these two extremes, we recall that gossip dissemination remains reliable as long as the arithmetic mean—from now on *average*—of all fanouts is at least $\ln(n) + C$ regardless of the fanout distribution across nodes, given a source fanout ≥ 1, and $C > 0$ being a constant [130]. This allows us to express the desired fanout of a node, n, with capability u_n in terms of the desired average fanout, $\overline{f}$, and of the average bandwidth capability, $\overline{u}$.

$$f_n = \frac{u_n}{\overline{u}} \cdot \overline{f} \tag{4.1}$$

4.3.1.2 Sample-Based Capability Aggregation

To apply Equation (4.1), each node, n, can compute its upload bandwidth, u_n, prior to the start of the streaming process by means of mechanisms such as those in [144, 151]. In addition, each node needs to obtain information about the average bandwidth capability of the system. To this end, *HEAP* employs a *sample-based capability-aggregation scheme* that exploits a fundamental component of gossip-based protocols: the Random Peer-Sampling (RPS) Service [106].

The RPS service provides each node with a continuously changing sample of the network, the *view*, from which to choose communication partners. Nodes communicate periodically by exchanging subsets of their views and mixing them like one would do when shuffling a deck of cards. After a sufficient number of iterations, extracting a communication partner from a node's view approximates extracting one randomly from the entire network. [106, 84, 57, 79].

HEAP augments the RPS service by bundling capability information together with the IP addresses, and timestamps of the nodes in the RPS views. Each node then uses the average capability of the entries in its view as as an estimator for the average capability of the network. Our experiments reveal that a view size $v_{\mathrm{RPS}} = 160$ nodes provides an accurate estimation in networks of up to 100k nodes.

4.3.2 Reliability

To address reliability, HEAP comprises two novel mechanisms: *Codec* and *Claim*. The former boosts the delivery rate of the propose phase in the presence of low fanout values. The latter introduces a multi-source strategy to recover from message loss in the [REQUEST] and [SERVE] phases.

4.3.2.1 Codec

Gossip's redundancy allows [PROPOSE] messages to withstand some level of message loss with a large enough fanout. However, our results in Chapter 3 show that fanout values cannot grow indefinitely in the presence of constrained bandwidth. *Codec* reduces the impact of message loss with low fanout values by employing a *block-based* deterministic Forward-Error-Correction (FEC) scheme [165]. Figure 4.2 summarizes its operation. The source of the stream creates, for each group of k source packets, e additional encoded ones. A node receiving at least k random packets from the

$k + e$ possible ones decodes them and forwards them to the video player (step (i) in Figure 4.2). If it receives fewer, the node can still forward the $j < k$ source packets it received to the player, thereby rendering a *jittered* fragment.

The FEC technique employed by *Codec* has negligible CPU costs [165], but it incurs some overhead to disseminate the e additional packets in each group. The key feature of *Codec* lies in its ability to decrease this overhead by feeding information back into the gossip protocol. Specifically, a node can stop requesting packets for a given group of $k + e$ as soon as it has received any k packets, not necessarily the original ones (step (ii) in Figure 4.2). The decoding process will provide the remaining source packets required to play the stream, thereby saving bandwidth. Finally, the use of a deterministic encoding allows *Codec* to re-inject these decoded packets into the protocol (step (iii) in Figure 4.2).

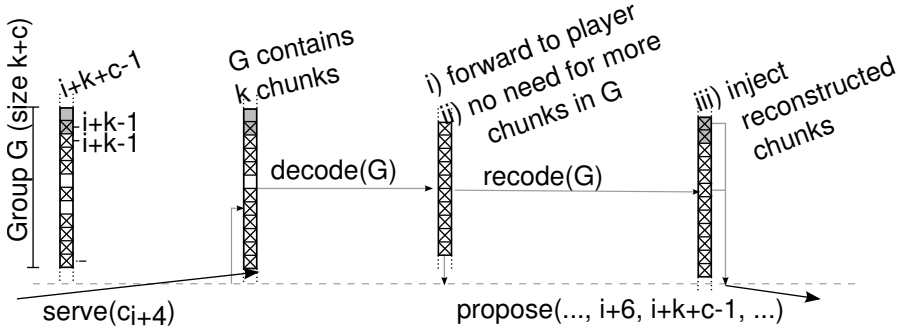


Figure 4.2: *Codec*: a node decodes group G to reconstruct the k source packets and sends them to the player (step (i)). The node then signals the protocol not to request any more packets in G (step (ii)). Finally (step (iii)), it re-encodes the k source packets and re-injects them into the protocol.

4.3.2.2 Claim

Codec significantly improves the performance of three-phase gossip. Nonetheless, when used alone, it does not suffice to provide a satisfactory streaming experience. For this reason, *Claim* introduces a retransmission mechanism that leverages gossip duplication to diversify the retransmission sources of missing information. *Claim* allows nodes to re-request missing content by recontacting the nodes from which they received advertisements for the corresponding packets, leveraging the duplicates created by gossip. Instead of stubbornly requesting the same sender, the requesting node re-requests nodes in the set of proposing nodes in a round-robin manner as presented in Figure 4.3.

Nodes can emit up to $r = 5$ re-requests for each packet. To determine how to time their re-requests, nodes keep track of the past response times for delivered packets. We define response time as the time elapsed between the moment a node sends a [REQUEST] and the moment it receives the corresponding [SERVE] message. When requesting a packet, the node schedules the first re-request after a timeout $t_{r(0)}$, equal to the 99.9th percentile of previous response times. To minimize variability in the percentile value, nodes only apply this rule if they have received at least 500 packets. If they have not, they use a default initial value $t_{r(0),\text{init}}$. Nodes bound this first re-request interval between a minimum of $t_{r(0),\min}$ and a maximum of $t_{r(0),\max}$. To schedule further re-requests, if needed, nodes keep halving the initial timeout so long as its value is at least $t_{r,\min}$. For example if the ith re-request's timeout is $t_{r(i)} > t_{r,\min}$, then $t_{r(i+1)} = \frac{t_{r(i)}}{2}$; otherwise $t_{r(i+1)} = t_{r(i)}$.

Our experiments show that *Claim* alone cannot guarantee reliable dissemination of all the streaming data to all nodes. On the other hand, its combination with *Codec* provides all nodes with a clear stream even in tight bandwidth scenarios, and in the presence of crashes.

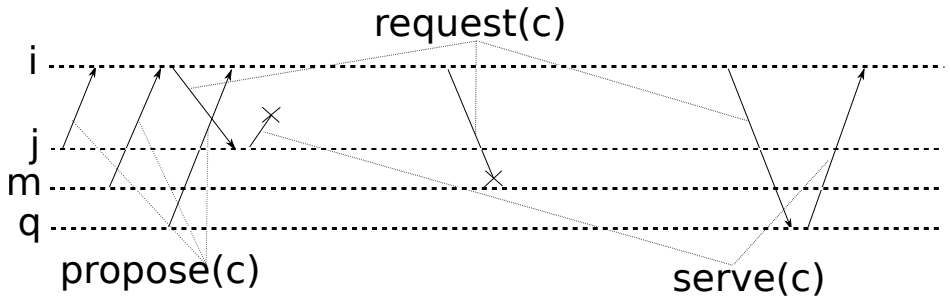


Figure 4.3: *Claim*: node i requests packet c from j, but j's serve message gets lost. Instead of re-requesting j, i now requests m. This time, the request gets lost. So i requests another node that proposed c, node q, which finally serves i.

4.4 Experimental Setting

We evaluated HEAP on 200 nodes, deployed over a cluster of 19 Grid5000 [55] machines and we evaluated the scalability of its capability-aggregation scheme by simulation in networks of up to 100k nodes. Cluster experiments (section 4.5.1 through 4.5.4 and 4.5.6 through 4.5.7) provide a controlled and reproducible setting, while simulation (Section 4.5.5) allows us to evaluate scalability on very large networks.

4.4.1 Protocol Parameters

The source serves a 3-minute video consisting of 1397-byte packets, sent at a rate of 55 per second for an average stream rate of 600 kbps. We set the gossiping period of each node to 200 ms, yielding an average of 11.26 packet ids per propose message. Where not otherwise specified, we set the average fanout across all nodes to 7. We configure *Codec* to use a default window size of 100, with 10% of redundancy for a total coded size of 110 packets. At the Random Peer Sampling layer, nodes maintain a view of size $v_{\text{RPS}} = 50$, and exchange gossip messages containing $g_{\text{RPS}} = 25$ entries from their views. We vary the parameters of the RPS in Section 4.5.4. Finally, we configure *Claim* as follows. We use an initial re-request timeout, $t_{r(0)} = 10\,\text{s}$, a minimum initial timeout, $t_{r(0),\text{min}} = 2\,\text{s}$, and a maximum initial timeout, $t_{r(0),\text{max}} = 15\,\text{s}$. *Claim* keeps halving timeouts as long as they exceed the threshold of $t_{r,\text{min}} = 5\,\text{s}$ as described in Section 4.3.2.2.

4.4.2 Bandwidth Limitation

To evaluate the performance of HEAP in constrained-bandwidth scenarios, we consider a *token bucket* bandwidth limiter (also known as leaky bucket as a meter) [154]. This limiter constrains the average bandwidth to the specified limit but it also allows for short off-limit bursts. We also experimented with a leaky bucket, but for space reasons we omit the results from this manuscript. The interested reader can find them in [34].

In all our experiments, we give the source enough bandwidth to serve 7 nodes in parallel and limit the upload bandwidth of each of the other nodes according to one of four scenarios. Our first scenario consists of a homogeneous setting, homo-691, in which all nodes have the same upload capability of 691 kbps. The three remaining scenarios reflect heterogeneous bandwidth distributions inspired by those in [111]: ref-724, ref-691, and ms-691.

Table 4.1 summarizes the composition of the heterogeneous scenarios. The *capability supply ratio* (CSR, as defined in [111]) represents the ratio of the average upload bandwidth to the stream rate. In all the considered settings, the average upload bandwidth suffices to sustain the stream rate. But in the heterogeneous scenarios, some of the nodes' upload bandwidths are well below the limit. In the most skewed scenario we consider (ms691), most nodes are in the *poorest* category and only 15% of the nodes have an upload capability larger than the stream rate.

In all the considered scenarios, bandwidth limiters introduce delays in the dissemination of messages. However, data sent over the Internet is also subject to bandwidth-independent delays (e.g., propagation delays). To model this, we

also associate each message with an additional uniformly distributed delay between 50ms and 250ms.

			Fraction of nodes		
Name	CSR	Average	2 Mbps	768 kbps	256 kbps
ref-691	1.15	691 kbps	0.1	0.5	0.4
ref-724	1.20	724 kbps	0.15	0.39	0.46
Name	CSR	Average	3 Mbps	1 Mbps	512 kbps
ms-691	1.15	691 kbps	0.05	0.1	0.85

Table 4.1: Heterogeneous bandwidth scenarios.

4.4.3 Metrics

We evaluate HEAP according to two metrics. First, we define the *stream quality* as the percentage of the stream that is viewable without jitter at a given delay. We consider a FEC-encoded window to be *jittered* as soon as it does not contain enough packets (i.e., at least 100 with our default parameters) to be fully decoded. A X%-jittered stream therefore means that X% of all the windows are jittered. Note that, as explained in Section 4.3.2.1, a jittered window is not entirely lost. Because *Codec* employs systematic coding, a node may still receive 99 out of the 100 original stream packets, resulting in a 99% delivery ratio in a given window. We use the expression *clear stream* to denote a jitter-free stream, that is one with a quality of 100%. Second, we define the *stream lag* as the delay required by the player in order to play the stream with a specified stream quality or level of jitter. When not otherwise specified, we consider the stream lag to obtain a clear stream. This represents the difference between the time the stream is published by the source and the time it can be delivered without glitches or interruptions to the players on the nodes. Equivalently, it can be defined as the largest delay experienced by any packet on any node in the experiment.

4.5 Performance Evaluation

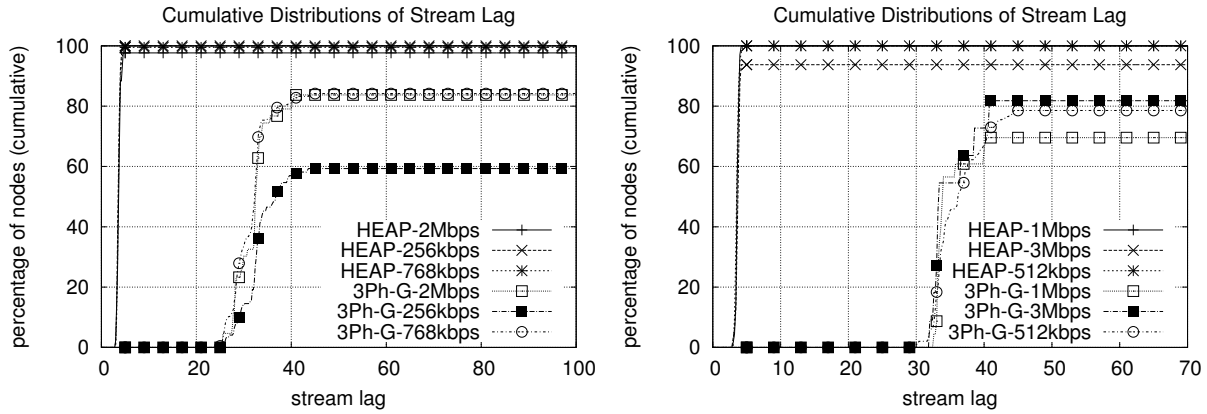
We start our evaluation by examining the performance of HEAP in the presence of heterogeneous bandwidth constraints, and by comparing it with that of the standard three-phase gossip protocol described in Section 2.2. Then, we analyze the reasons for its good performance by examining different bandwidth scenarios, bandwidth limiters, and configurations of the RPS protocol. Finally, we evaluate the impact of HEAP on external applications. To ensure a fair comparison, we augment the standard three-phase gossip protocol with the same error correction and retransmission mechanisms as HEAP, namely *Codec* and *Claim*.

4.5.1 HEAP Test Drive: Heterogeneous Bandwidth

Figure 4.4 compares the stream lags required by HEAP and standard three-phase gossip to obtain a non-jittered stream in the two most constrained scenarios in Table 4.1: ref-691 and ms-691. In all cases, HEAP drastically reduces the stream lag for all capability classes. Moreover, as shown in Figure 4.4b the positive effect of HEAP significantly increases with the skewness of the distribution.

The plots depict the cumulative distribution of the nodes that manage to view a jitter-free stream as a function of the stream lag, for each capability class and for each of the two protocols. In all cases, HEAP provides a clear stream to almost all nodes with a lag of less than 5s, while standard three-phase gossip induces much higher lags (an order of magnitude higher) to reach much fewer nodes.

Figure 4.4a presents the results for ref-691. Here, HEAP provides a perfectly clear stream to 100% of the low-capability nodes within 4.4s, to 99.5% of the mid-capability ones within 5s, and to 97.7% of the high-capability ones within 4.6s. Standard three-phase gossip, on the other hand, cannot provide a clear stream to any node within less than 24s and it only reaches a much lower fraction of nodes (60% to 84%) with delays that exceed 40s, as shown in Table 4.2.



(a) ref-691 (b) ms-691

Figure 4.4: Cumulative distribution of stream lag in the ref-691 and ms-691 scenarios for HEAP and standard 3-ph gossip.

The results for ms-691 (Figure 4.4b) have a similar flavor with two minor differences. First, standard three-phase gossip turns out to be even less efficient. Second, HEAP's performance decreases slightly for high-bandwidth nodes, but it remains at very high levels: 93.75% of them receive a perfect stream within 4.2s.

	HEAP			Standard three-phase Gossip		
bw	low	mid	high	low	mid	high
ref-691	100% / 4.4s	99.5% / 5s	97.7% / 4.6s	59.3% /43.6s	84%/ 43.8s	83.7%/ 40.6s
ms-691	100% / 4.6s	100% / 4.2s	93.8% / 4.2s	78.5% /44.4s	69.6%/ 40.2s	81.8%/ 40.8s
ref-724	99% / 4.2s	100% / 4.4s	98.4% / 4.0s	45.4% /48.2s	73.2%/ 48.0s	82.8%/ 44.8s

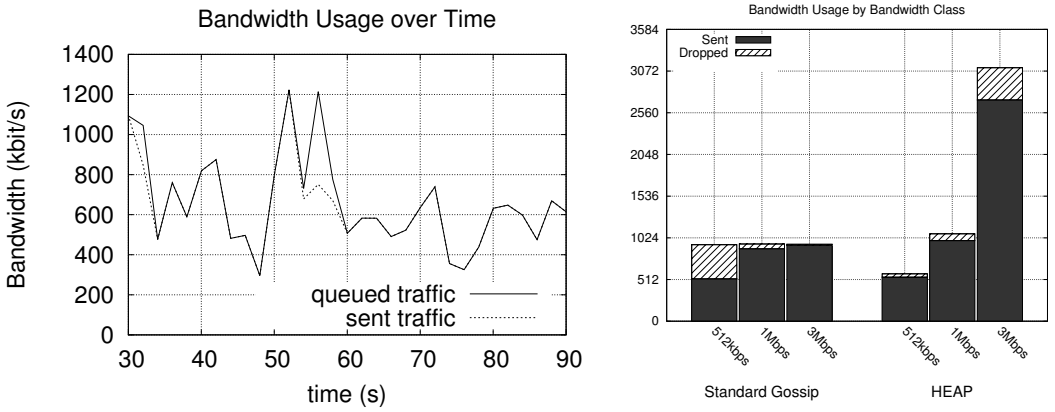
Table 4.2: Pct. of nodes that receive 100% of the stream and corresponding maximum lag.

	HEAP			Standard three-phase Gossip		
bw	low	mid	high	low	mid	high
ref-691	100% / 3.4s	100% / 3.4s	100% / 3.4s	94.1% /40.6s	99.5%/ 35.6s	100%/ 33.0s
ms-691	100% / 3.2s	100% / 2.8s	100% / 2.8s	98.5% /39.6s	100%/ 35.4s	100%/ 33.0s
ref-724	100% / 2.8s	100% / 3.0s	100% / 3.0s	85.9% /47.6s	99.4%/ 42.2s	100%/ 35.4s

Table 4.3: Pct. of nodes that receive 99.9% of the stream and corresponding maximum lag.

Table 4.3 completes these results by adding the percentage of nodes that receive 99.9% of the stream and the corresponding maximum lag. Even with this other metric, HEAP consistently exhibits higher node percentages and shorter lags than standard three-phase gossip. In the most constrained scenario, ms-691, HEAP provides 99.9% of the stream to 100% of the nodes within 3.2s, while standard three-phase gossip requires 25s to provide the same percentage of stream to the first few percent of the nodes (not shown in the table).

To summarize, HEAP consistently provides a clear stream to a larger number of nodes than standard three-phase gossip and with a lag that is one order of magnitude lower. Moreover, we observe that the lag induced by HEAP is in fact indistinguishable from the lag induced in a homogeneous bandwidth setting.



(a) Bandwidth Usage over Time

(b) Bandwidth consumption by capability

Figure 4.5: Bandwidth usage over time and by capability class with no message loss and slow retransmission.

4.5.2 Understanding Bandwidth Usage

We now analyze how nodes exploit their bandwidth capabilities. We start by considering bandwidth consumption over time. Figure 4.5a plots the number of bits per second submitted to the bandwidth limiter, as well as the number of bits that get effectively sent. We obtained the plots by counting the amount of data queued and sent in every 2s interval.

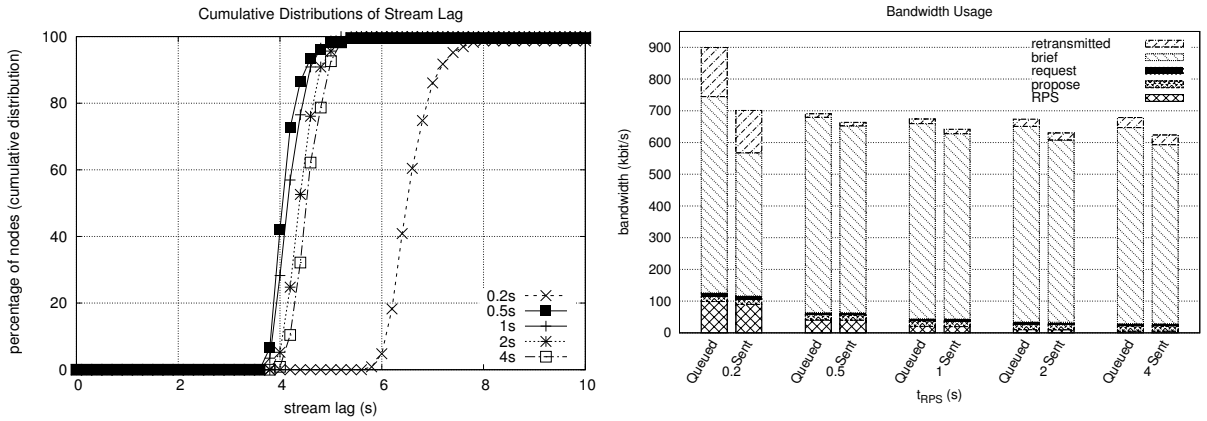
The plot highlights the spiky nature of gossip traffic resulting from its random nature. At times nodes receive a large number of request messages, and thus end up serving many other nodes. At other times, they receive fewer requests and therefore end up serving fewer nodes. This represents one of the main challenges in using gossip-based protocols for high bandwidth dissemination. Chapter 5 presents a solution to it.

Figure 4.5b depicts the breakdown of bandwidth consumption among the three classes of nodes in the ms-691 scenario. The total height of each vertical bar indicates the average bandwidth that nodes in the corresponding capability class attempt to use. The black part shows the portion of this bandwidth that results in successfully sent messages, while the striped part corresponds to messages that are dropped by the bandwidth limiter. For example, the first bar for standard three-phase gossip shows that nodes with an upload capability of 512 kbps attempt to send 940 kbps but manage to send only 512 kbps (black portion), the remaining striped portion being dropped.

Figure 4.5b clearly shows the difference in bandwidth usage between HEAP and standard three-phase gossip. HEAP nodes attempt to use an amount of bandwidth that is very close to their actual capabilities. Standard-gossip nodes, on the other hand, attempt to send approximately 1 Mbps of data regardless of their capability class. The black portions of the vertical bars (successfully sent data) show that standard three-phase gossip is completely unable to exploit high capability nodes and therefore ends up saturating lower capability ones. HEAP, on the other hand, effectively exploits all the available bandwidth of all node classes thereby limiting the number messages dropped by the token bucket.

4.5.3 Reliability: Need for Codec and Claim

The experiments in Sections 4.5.1 and 4.5.2 already demonstrated the reliability of HEAP in a variety of settings. Here we discuss the importance of both its reliability components. To evaluate the need for *Codec*, we tested a protocol version without it with a fanout of 7 in a homo-691 scenario with and without message loss. In either case, none of the tested *Claim* configurations managed to provide a clear stream to any of the nodes with either bandwidth limiter with queue/burst sizes of up to 200kB. We also evaluated a version with *Codec* but without *Claim* in the same configurations. Without any message loss and a token bucket, none of the nodes managed to view a clear stream with a 50kB burst size, and only 8% could when we increased the burst size to $200kB$. With message loss, none could view a clear stream with



(a) homo-691

(b) bandwidth breakdown in homo-691

Figure 4.6: Stream Lag and bandwidth usage with the RPS running at various frequencies—different values of t_{RPS}—in homo-691.

any burst size. Results were slightly better with a leaky bucket: without message loss 95% of the nodes could view a clear stream with a 200kB queue but none with a 50kB one. With 1.5% of message loss, only 8% of the nodes could view a clear stream and only with a 200kB queue. These results highlight that neither *Codec* or *Claim* alone suffices, but that together they enable HEAP to achieve the good performance demonstrated above.

4.5.4 Sensitivity Analysis

A HEAP node continuously needs to advertise stream packets by sending a propose message to f other nodes every 200ms. For the protocol to work correctly, the node must choose these f other nodes as randomly as possible. As discussed in Section 4.3.1.2, HEAP achieves this by relying on an augmented RPS protocol. In this section, we evaluate how two important parameters of this protocol impact the performance of HEAP.

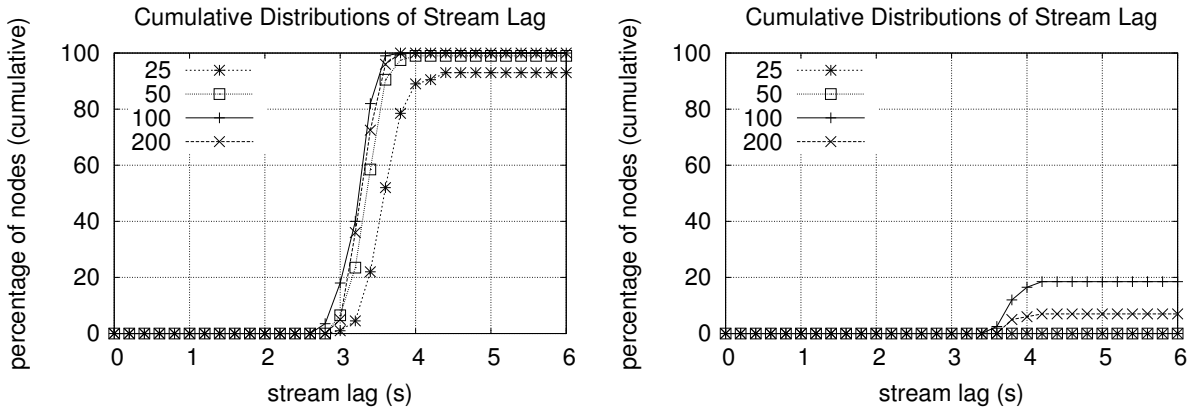
4.5.4.1 RPS Gossip Interval

The RPS gossip interval determines the frequency at which the nodes in the RPS view get refreshed. The higher this frequency, the lower the correlation between the two sets of f nodes selected at two subsequent rounds. To minimize correlation, nodes should ideally refresh their views multiple times between two dissemination actions [106], an almost impossible feat with 5 dissemination actions per second. We therefore seek to identify the best tradeoff between the randomness of peer selection and the need to maintain a reasonable gossip frequency.

To achieve this, Figure 4.6 plots the stream-lag distribution and bandwidth breakdown obtained by HEAP in homo-691 with the RPS running at various frequencies[1]. Figure 4.6a shows that the best results correspond to RPS gossip intervals (T_{RPS}) of 500ms and 1s. Intervals of 2s and 4s perform slightly worse, while an interval of 200ms exhibits the worst performance by adding approximately 2s to the stream lag of the other configurations.

To explain these results, we analyze how the various messages employed by the protocol contribute to bandwidth consumption. Figure 4.6b shows a stacked histogram with the number of bits/s that are queued into the token-bucket and those that are actually sent for each of the RPS configurations in Figure 4.6a. The two bars for $T_{\mathrm{RPS}} = 200ms$ show that the poor performance associated with this gossip frequency results from the very high amount of bandwidth consumed by the RPS protocol in this setting ($100kbps$). This huge bandwidth consumption causes a significant amount

[1] We also ran experiments on ms-691 but we have to omit them for space reasons. They are available in [34]



(a) homo-691 (b) homo-691 - no Claim

Figure 4.7: Stream Lag with various view sizes (v_{RPS}) for the random peer sampling protocol for standard HEAP and for a variant without retransmission.

of message loss, and of consequent retransmission operations. With a T_{RPS} of 500ms, the number of retransmission operations appears a lot more reasonable. Yet, the RPS protocol still consumes as much as $50kbps$, for only a marginal improvement with respect to $T_{\mathrm{RPS}} = 1s$.

The bars for frequencies below one gossip per second highlight what might seem like a contradiction. The average sent bandwidth is less than the average queued bandwidth—some messages are being dropped—even if the latter is lower than the bandwidth limit. The reason for this behavior lies in the bursty nature of gossip traffic discussed in Section 4.5.2. We also observe that starting from $T_{\mathrm{RPS}} = 1s$, the number of retransmission actions slightly increases with T_{RPS}. As expected, slower gossip frequencies cause higher correlation between the nodes chosen at subsequent rounds. This exacerbates the congestion bursts described above resulting in additional message loss and thus in a greater number of retransmission actions.

4.5.4.2 View Size

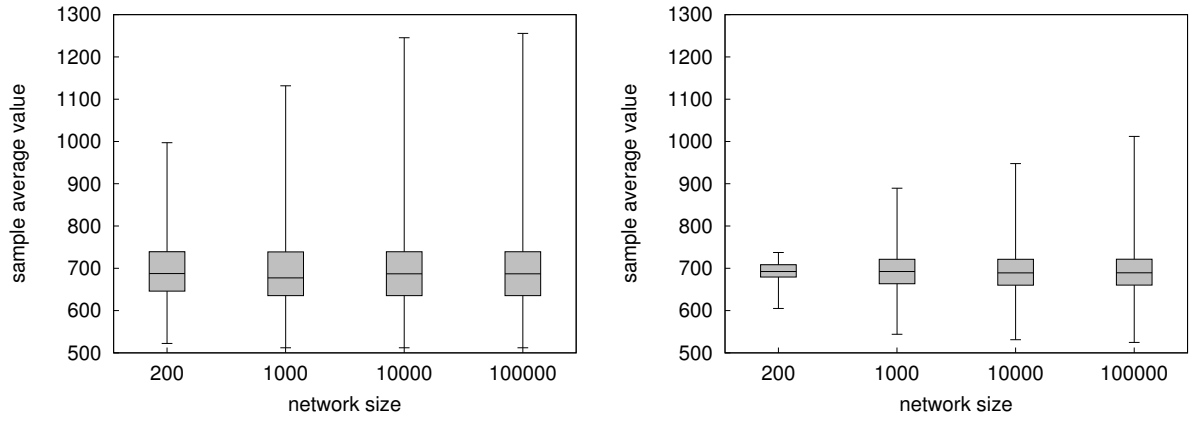
A second way to minimize the correlation between two subsequent samples of f nodes consists in increasing the size of the population from which they are extracted. In the case of the RPS, we can achieve this by increasing its view size. To evaluate the effect of this parameter, Figure 4.7a shows the impact of different view sizes on streaming performance, while maintaining a gossip interval of T_{RPS} =1s.

The plot shows that HEAP achieves the best performance with view sizes of at least 50 nodes. With a fanout of 7, and a T_{RPS} of 1s, nodes choose 35 nodes per second. A view size of 25 therefore necessarily results in highly correlated views, while a view size of 50 provides sufficient randomness. Larger view sizes provide only minimal improvements, which justifies our choice of a default view size of 50.

To appreciate the impact of larger view sizes, we also consider a variant of HEAP in which we disable Claim, the retransmission mechanism described in Section 4.3.2.2. With a view size of 50, this variant is unable to provide a clear stream to any node. Yet, a larger view size of 100 nodes manages to provide a clear stream to almost 20 nodes with less than $4s$ of lag. Albeit not good enough to get rid of Claim, this result shows that the more uniform peer selection provided by a larger view boosts the performance of HEAP. Yet, setting the view size to 200 decreases, rather than increases, performance. The improvement brought about by larger view sizes is in fact limited by the associated bandwidth consumption—nodes exchange subviews that contain half-a-view-size nodes [106].

v_{RPS}	Variance ratios by network size			
	200	1k	10k	100k
50	78.52	58.75	51.23	50.10
80	161.28	95.44	82.38	80.26
160	820.92	208.58	167.08	160.25

Table 4.4: Variance Ratios in the ms-691 scenario.



(a) $v_{\text{RPS}} = 50$ (b) $v_{\text{RPS}} = 160$

Figure 4.8: Whisker plots for sample average obtained from the RPS view with increasing network sizes.

4.5.5 Scalability

To study the scalability of HEAP, we start by observing that the local nature of three-phase dissemination naturally supports arbitrarily large numbers of nodes as demonstrated by existing real-world deployments [92]. Similarly, existing research on RPS protocols has shown their effectiveness in very large networks with hundreds of thousands of nodes [106]. As a result, the only doubts about the scalability of HEAP may arise from the use of the RPS for the estimation of the average bandwidth of the system.

As discussed in Section 4.3.1.2, HEAP estimates the average bandwidth of the network as the mean of a sample. A well known result states that the ratio between the variance of the initial distribution and that of a sample mean is equal to the size of the sample. So with an RPS view of 50, a node should get an estimation of the average with a variance that is 50 times smaller than that of the bandwidth distribution across the network. However, this theoretical result holds in the presence of purely random samples, while the RPS only approximates ideal randomness.

To evaluate whether this affects the accuracy of bandwidth estimation, we simulated the RPS-based averaging component in networks ranging from 200 to 100k nodes. We configured the RPS to use a fixed view size regardless of network size, and considered three configurations with v_{RPS} values of 50, 80, and 160. A configuration with a network size of 200 and $v_{\text{RPS}} = 50$ corresponds to that of our Grid5000 experiments. For space reasons, we only show the results for ms691.

Table 4.4 shows the ratios between the variance of the sample average and the variance of the bandwidth distribution in the entire network. For very large networks of 10k nodes and above, the ratio between the variance of the bandwidth distribution and that of the estimated average indeed follows theoretical predictions despite the imperfect randomness of the RPS. But as expected, for smaller networks, the ratio is even higher as the RPS view constitutes a larger proportion of the network. While this suggests that a view of 50 cannot give the same performance in a network of 100k nodes as in one of 200, the table also shows that a view of 80 yields approximately the same variance ratio in large networks as that of a view of 50 with 200 nodes. This suggest that a relatively small increase in view size can offset an increase in

network size of three orders of magnitude.

To evaluate this hypothesis, Figure 4.8 plots the distribution of sample-average values for $v_{\mathrm{RPS}} = 50$ and $v_{\mathrm{RPS}} = 160$. For each, configuration, the box extends from the first to the third quartiles, while the whiskers show the minimum and the maximum values of the sample average. The figure shows that while the distribution spreads out when increasing the size of the network, a 3-fold increase in view size more than offsets a three-order-of-magnitude increase in network size. One might wonder whether this increase in view size would not lead to excessive bandwidth consumption by the RPS protocol. But Figure 4.7a already demonstrated that increasing the view size from 50 to 200 has almost no impact on the full HEAP protocol. Overall, our analysis confirms the scalability of our average-bandwidth estimation and thus that of our streaming solution.

4.5.6 Responsiveness to Churn

To evaluate HEAP's ability to respond to node failures, we consider a worst-case scenario in which a subset of the nodes instantaneously fail and thereby stop contributing to the protocol. Such a catastrophic event impacts the protocol in two major ways. First, nodes that fail may have proposed stream packets to other nodes, and may have received requests that they can no longer honor. Second, their identifiers remain lingering in the RPS views of other nodes that may therefore continue to propose stream packets to them, thereby wasting time and bandwidth resources.

To evaluate the overall effect of these two issues, we ran several experiments in the ref-691 scenario with various values of T_{RPS} and observed the behavior of HEAP after the failure of 20% and 50% of the nodes. Figure 4.9a shows that HEAP recovers from the failure of 20% of the nodes almost instantly regardless of the value of T_{RPS}. The plots show the percentage of nodes receiving each stream packet against the time at which the packet was generated at the source. We see that approximately 15% of the non failed nodes experience a short video/audio glitch (they lose a few stream packets) when the failure occurs. Figure 4.9b presents a close-up view of this glitch for the most unfavorable setting ($T_{\mathrm{RPS}} = 4s$). The plot shows that the glitch actually extends for as little as $2.25s$ and that most of the nodes recover even faster. The glitch starts before $t = 60s$ because of the stream lag of about $2s$—the plot shows stream time and not absolute time.

Figure 4.9a also shows that a small number of nodes also experience shorter glitches later during the experiment, resulting from the presence of stale references in RPS views. These glitches continue longer for larger values of T_{RPS} and become less and less frequent over time.

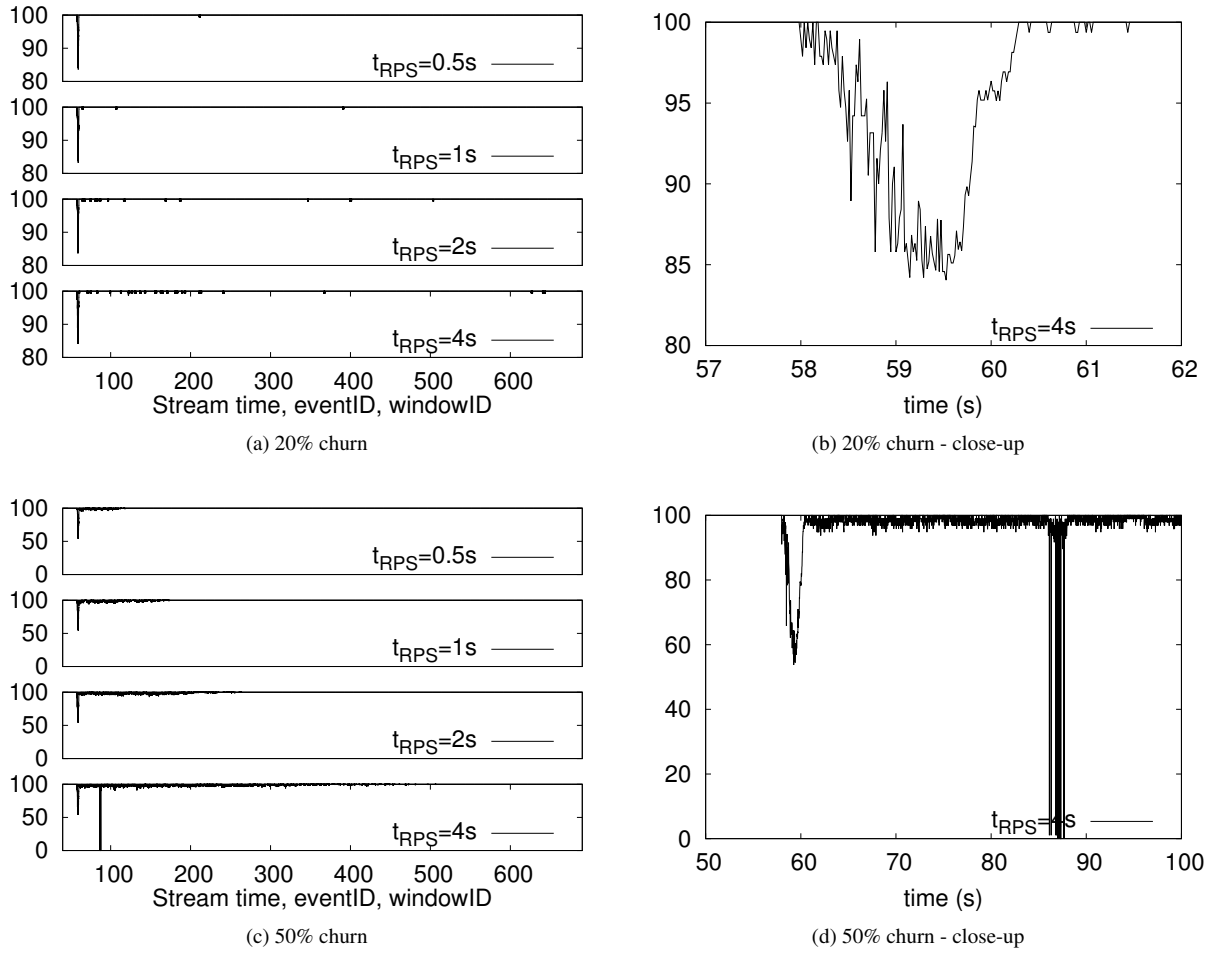
Figure 4.9c shows instead the results for the failure of 50% of the nodes. In this case, up to 50% of the nodes experience a glitch in their stream when the failure occurs and shorter glitches later on. With $T_{\mathrm{RPS}} = 4s$, we observe not only short glitches but also a second major glitch at around $t = 84s$. The close-up view in Figure 4.9d shows that this results from a few packets being completely missed by all of the nodes. Like for shorter glitches, this results from the presence of stale references in the RPS views of nodes. With $T_{\mathrm{RPS}} = 4s$, $t = 84s$ is only 6 gossip cycles after the failure, and completely purging failed nodes from a view of size 50 may take up to 50 cycles in the worst case.

If we join this analysis on churn with our previous analysis on bandwidth consumption (Figure 4.6b), we can observe that a value of $T_{\mathrm{RPS}} = 1s$ strikes a good balance between responsiveness—50% failure completely recovered in less than 2 mins—and bandwidth cost—virtually identical to that of $T_{\mathrm{RPS}} = 4s$.

4.5.7 Cohabitation with External Applications

Next, we evaluate the ability of our streaming protocol to operate in the presence of applications that compete for the use of upload bandwidth. To simulate the presence of an external bandwidth-intensive application, each node periodically sends UDP messages of size m, one every T_{app}. This results in an average bandwidth consumption of $B_{\mathsf{app}} = \frac{m}{T_{\mathsf{app}}}$. To simulate interactive usage such as uploading files to a website, or sending emails with attachments, nodes run the above bandwidth-consumption task with an on-off pattern. The task runs for t_{on}, and then pauses for t_{off}. We refer to the ratio $d_c = \frac{t_{\mathrm{on}}}{t_{\mathrm{off}}+t_{\mathrm{on}}}$ as the application's *duty cycle*.

Figures 4.10a and 4.10b show the results with a 50% duty cycle ($t_{\mathrm{on}} = t_{\mathrm{off}} = 10s$) and respectively $T_{\mathsf{app}} = 200$ms and $T_{\mathsf{app}} = 100$ms. For each T_{app}, we tested various values of B_{app} by setting the message size to $m = T_{\mathsf{app}} \cdot B_{\mathsf{app}}$.



(a) 20% churn

(b) 20% churn - close-up

(c) 50% churn

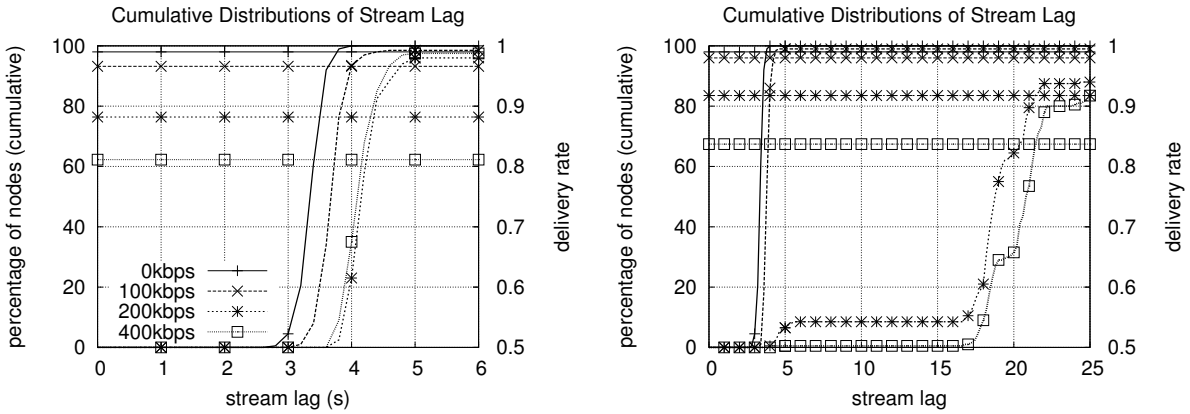
(d) 50% churn - close-up

Figure 4.9: Percentage of nodes receiving a clear stream over time with a catastrophic failure of 20% or 50% of the nodes happening at t=60s.

Each plot shows two curves for each B_{app}. The S-shaped line depicts the cumulative distribution of stream lag in the considered configuration. The horizontal line depicts instead the delivery rate achieved by the external application.

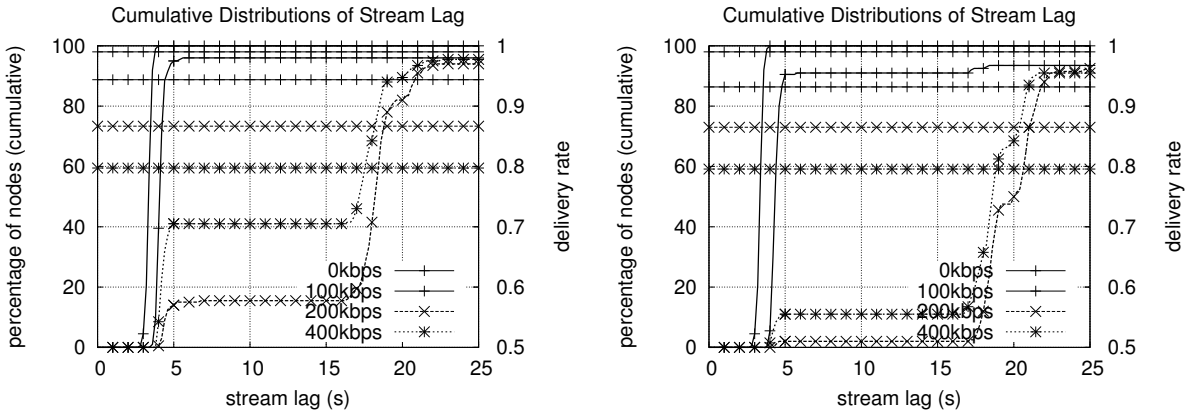
With $T_{\mathsf{app}} = 200$ms, HEAP is almost unaffected by the external application and only experiences small increases in stream lag. With $B_{\mathsf{app}} = 100$kbps—14% of the available bandwidth—HEAP experiences a lag increase of less than half a second while the external application retains a very high delivery rate of almost 97%. When B_{app} increases ($B_{\mathsf{app}} = 200\,\text{kbps}, 400\,\text{kbps}$), HEAP still performs well, albeit with a lag increase of around 1s. This may seem surprising, but it comes at the cost of lower delivery rates for the external application, which scores at 88% for $B_{\mathsf{app}} = 200\,\text{kbps}$, and at 81% for $B_{\mathsf{app}} = 400\,\text{kbps}$.

With $T_{\mathsf{app}} = 100$ms, both HEAP and the external application keep performing well when the external application consumes $100\,\text{kbps}$ during its active intervals. HEAP experiences a delay of less than 1s over the baseline, while the external application maintains a delivery rate of 98%. When B_{app} increases, however, the performance of HEAP decreases more drastically than with $T_{\mathsf{app}} = 200$ms. The average lag to receive a clear stream jumps to around 20s, and less than 10% of the nodes can still view a clear stream with a 5s lag with $B_{\mathsf{app}} = 200\,\text{kbps}$, while none can with $B_{\mathsf{app}} = 300\,\text{kbps}$. Yet, the external application achieves even higher delivery rates than with $T_{\mathsf{app}} = 200$ms: 92% for



(a) $T_{\text{app}} = 200$ ms (b) $T_{\text{app}} = 100$ ms

Figure 4.10: Stream Lag for HEAP and delivery rate for the external application for $T_{\text{app}} = 200ms$ (left) and $T_{\text{app}} = 100ms$ (right) with an application duty cycle of 50%. The key in Fig. 4.10a applies to both plots.



(a) $d_c = 66.6\%$ (b) $d_c = 75\%$

Figure 4.11: Stream Lag for HEAP and delivery rate for the external application for higher duty cycles (d_c) and several values of B_{app} (different lines) with $T_{\text{app}} = 200ms$.

$B_{\text{app}} = 200$ kbps and 87% for $B_{\text{app}} = 300$ kbps.

From this analysis, we can observe that smaller and more frequent messages tend to favor the external application, while larger and less frequent ones tend to favor HEAP. More extreme values of T_{app} (400ms and 50ms)—plots available in [34]—confirm this observation.

Figure 4.11 complements these results with those obtained in scenarios with higher duty cycles for the external application. In particular, Figure 4.11a depicts a duty cycle of 66%, and Figure 4.11b a duty cycle of 75% for $T_{\text{app}} = 200$ms. Clearly, increasing the duty cycle increases the average bandwidth consumption of the external application, thereby increasing its impact on HEAP. Yet, the performance for $B_{\text{app}} = 100$ kbps remains very good, with 97% of the nodes receiving a clear stream within less than $5s$ with a 66% duty cycle, and 94% of the nodes achieving the same result with a 75% duty cycle. Performance for higher values of B_{app} drops, but even with $B_{\text{app}} = 400$ kbps,

HEAP manages to yield a clear stream to over 95% of the users albeit with a pretty large lag: $21s$ with a duty cycle of 66%, and $22s$ with a duty cycle of 75%. The corresponding delivery rates for the external application remain above 80% in all cases.

We conclude this section by observing that with $T_{\mathsf{app}} = 400ms$ (plot not shown), HEAP provides good performance with a lag of less than $5s$ even with a 75% duty cycle and a B_{app} value of 400 kbps. The external application achieves a corresponding delivery rate of 78% in the worst case.

4.6 Concluding Remarks

This chapter presented HEAP, a gossip-based video streaming solution designed to operate in heterogeneous and bandwidth-constrained environments. HEAP preserves the simplicity and proactive (churn adaptation) nature of traditional homogeneous gossip, while significantly improving its effectiveness. Through its four components, HEAP is able to adapt the fanouts of nodes to their bandwidth and delay characteristics while, at the same time, implementing effective reliability measures through FEC encoding and retransmission. Experimental results show that HEAP significantly improves stream quality over a standard homogeneous gossip protocol, while scaling to very large networks.

Chapter 5

Augmenting Gossip with Network Coding

In the previous chapters, we explored how gossip-based protocols can support live video streaming. This required us to make a few adjustments. First, we employed a three-phase model in order to limit the overhead associated with the redundancy of gossip dissemination. Second, our experiments showed that while some nodes may receive a given message multiple times, other nodes may fail to receive it. For this reason we augmented three-phase gossip with *Codec* and *Claim* in Section 4.3.2.1. In short, we used three-phase gossip to limit the impact of gossip's redundancy, but we then added two techniques that introduce additional redundancy.

In this chapter, we change perspective and consider redundancy as a key advantage rather than as a shortcoming. Instead of limiting redundancy, we leverage its presence to improve the dissemination of messages to nodes that would not otherwise receive them. We achieve this by leveraging Random Linear Network Coding (RNLC), a well-known techniques that spreads out redundancy over multiple messages, ultimately increasing delivery rates. The work in this chapter consists of the first contribution in the PhD of Quentin Dufour whom I am currently co-advising with David Bromberg.

> The content of this chapter is an adapted excerpt from the following paper:
> Yérom-David Bromberg and Quentin Dufour and Davide A Frey (Univ Rennes, Inria, CNRS, IRISA, France); Multisource Rumor Spreading with Network Coding. INFOCOM 2019

5.1 Overview

In Section 4.5.2, we highlighted the bursty nature of gossip traffic. Figure 4.5a showed that a given node keeps having bursts of bandwidth that exceed the limit, interleaved with low-bandwidth periods in which it sends fewer messages. The previous chapters also highlighted that at the fanout levels imposed by the bandwidth limit, gossip does not achieve complete dissemination. This required us to integrate our solution with *Claim*, a forward error correction mechanism, which adds further redundancy.

In this chapter, we take a different approach and instead of adding redundancy to gossip, we re-balance the already existing redundancy across multiple messages. To achieve this, we employ Random Linear Network Coding (RLNC). Other authors have already proposed its application to gossip in a theoretical setting [113, 69], or in single source dissemination [80]. But applying RNLC to multi-source gossip protocols still presents major challenges. First, existing approaches suppose that a vector, where each index identifies a message with its associated coefficient as a value, is disseminated. This approach implies a small namespace. Second, to reduce the complexity of the decoding process, messages are split in groups named generations. Existing rules to create generations require having only one sender, which is impractical in the context of multiple sources. Third, the use of RNLC implies linear combinations of multiple messages. This leads to potential partial knowledge of messages received, making precise message pull requests useless and breaks pull frequency adjustment based on missing-message counts.

We address these issues by introducing CHEPIN, a CHeaper EPidemic dissemINation approach for multi-source gossip protocols. CHEPIN solves the problems associated with the size of the identifier namespace by using sparse vectors. It creates generations for messages from multiple sources by leveraging on Lamport timestamps. All messages sharing the same clock are in the same generation whatever its source. It overcomes the issue of partial message knowledge by providing an adaptation of push, pull, push-pull gossip protocols. It pulls specific generations instead of specific messages. The chapter embodies these contributions in updated algorithms that make CHEPIN applicable to the current state of the art of multi-source gossip protocols.

5.2 Background on network coding and challenges

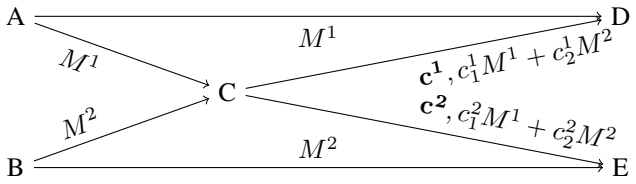


Figure 5.1: With RLNC, C can send useful information to D and E without knowing what they have received

RLNC[116] provides a way to combine different messages on a network to improve their dissemination speed by increasing the chance that receiving nodes learn something new. In Figure 5.1, node C cannot know what D and E have received. By sending a linear combination of M^1 and M^2, nodes D and E can respectively recover M^2 and M^1 with the help of the plain message they also received. Without RLNC, node C would have to send both M^1 and M^2 to D and E involving two more messages. Every message must have the same size, defined as L bits thereafter. To handle messages of different size, it is possible to split or pad the message to have a final size of L bits.

The message content has to be split as symbols over a field $\mathbb{F}_{2^n}$. The $\mathbb{F}_{2^8}$ field has interesting properties when doing RLNC. First, a byte can be represented as a symbol in this field. Thereafter, this field is small enough to speed up some computing with discrete logarithm tables and at the same time sufficiently large to guarantee linear independence of the random coefficients with very high probability.

Encoded messages are linear combinations over $\mathbb{F}_{2^n}$ of multiple messages. This linear combination is not a concatenation: if the original messages are of size L, the encoded messages will be of size L too. An encoded message carries a part of the information of all the original messages, but not enough to recover any original message. After receiving enough encoded messages, the original messages will be decodable.

To perform the encoding, the sources must know n original messages defined as $M^1, ..., M^n$. Each time a source want to create an encoded message, it randomly chooses a sequence of coefficients $c_1, ..., c_n$, and computes the encoded message X as follows: $X = \sum_{i=1}^{n} c_i M^i$. An encoded message thus consists of a sequence of coefficients and the encoded information: (c, X).

Every participating node can recursively encode new messages from the one they received, including messages that have not been decoded. A node that received $(c^1, X^1), ..., (c^m, X^m)$ encoded messages, can encode a new message (c', X') encoded by choosing a random set of coefficients $d_1, ..., d_m$, computing the new encoded information $X' = \sum_{j=1}^{m} d_j X^j$ and computing the new sequence of coefficients $c'_i = \sum_{j=1}^{m} d_j c_i^j$.

An original message M^i can be considered as an encoded message by creating a coefficient vector $0, ..., 1, .., 0$ where 1 is at the ith position. The encoding of a message can therefore be considered as a subset of the recursive encoding technique.

Even if there is no theoretical limit on the number n of messages that can be encoded together, there are two reasons to limit it. First, Gauss-Jordan elimination has a $O(n^3)$ complexity, which becomes rapidly too expensive to compute. Then, the more the messages encoded together, the bigger the sequence of coefficients while the encoded information remains stable. In extreme cases this can result in sending mainly coefficients on the network instead of information. To

encode more data, splitting messages in groups named generations solves the previous problems, as only messages in the same generation are encoded together.

However applying network coding to epidemic dissemination raises several challenges.

Assigning a message to a generation Generations often consist of integers attached to a message. Messages with the same generation value are considered in the same generation and can be encoded between them. The value must be assigned in such a way than enough messages are in the same generation to benefit from RLNC properties but not too many to keep the decoding complexity sufficiently low and limit the size of the coefficients sent on the network. In a single source scenario, the size of the generation is a parameter of the protocol, and is determined by counting the number of messages sent in a given generation. However, with multiple sources, there is no way to know how many messages have been sent in a given generation.

Sending coefficients on the network Coefficients are generally sent under the form of a dense vector over the network. Each value in the vector is linked to a message. On a single source scenario, that is not a problem, the source knows by advance how many messages it will send and can assign each message a position in the vector and start creating random linear combinations. In the case of multiple sources, the number of message is not available, and waiting to have enough messages to create a generation could delay message delivery and above all, the network traffic required to order the messages in the dense vector would ruin the benefits of network coding.

Pulling with RLNC When doing pull-based rumor mongering, a node must have a way to ask what rumors it needs. Without network coding, it simply sends a message identifier to ask for a message. But sending a message identifier in the case of network coding raises several questions: does the node answer only if it has decoded the message? Or if it can generate a linear combination containing this message?

Estimating the number of missing packets Some algorithms need to estimating the number of missing messages. Without network coding, the number of missing packets corresponds to the number of missing messages. But with RLNC, it is possible to have some linear combination for a given set of messages but without being able to decode them. All the messages are considered as missing but one packet could be enough to decode everything.

5.3 Contribution

5.3.1 System model

Our model consists of a network of n nodes that all run the same program. These nodes communicate over a unicast unreliable and fully connected medium, such as UDP over Internet. Nodes can join and leave at any moment, as no graceful leave is needed, crashes are handled like departures. We consider that each nodes can obtain the address of some other nodes of the service via a Random Peer Sampling service [106]. There is no central authority to coordinates nodes, all operations are fully decentralized and all exchanges are asynchronous. We use the term message to denote the payload that must be disseminated to every node of the network, and the term packet to denote the exchanged content between two nodes. We consider the exchange of multiple messages and any node of the network can inject messages in the network, without any prior coordination between nodes (messages are not pre-labeled).

5.3.2 Solving RLNC challenges

Due to our multiple independent source model and our goal to cover push and pull protocols, we propose new solutions to the previously stated challenges:

Assigning generations with Lamport timestamps With multiple senders, we need to find a rule that is applicable only with the knowledge of a single node when assigning a generation as relying on network would be costly, slow and unreliable. We chose Lamport timestamps[175] to delimit generations by grouping every messages with the same clock on the same generation. This method doesn't involve sending new packets as the clock is piggybacked on every network coded packet. When a node wants to disseminate a message, it appends its local clock to the packet and update it, when it receives a message, it uses its local clock and the message clock to update its clock. This efficiently associated messages that are disseminated at the same time into the same generation.

Sending coefficients in sparse vectors When multiple nodes can send independent messages, they have no clue on which identifiers are assigned by the other nodes. Consequently, they can only rely on their local knowledge to choose their identifiers. Choosing identifiers randomly on a namespace where all possible identifiers are used would lead to conflicts. That is why we decided to use a bigger namespace, where conflict probabilities are negligible when identifiers are chosen randomly. On a namespace of this size, it is impossible to send a dense vector over the network, however we can send a sparse vector: instead of sending a vector $c^1, ..., c^n$, we send m tuples, corresponding to the known messages, containing the message id and their coefficient: $(id(M^{i_1}), c^{i_1}), ..., (id(M^{i_m}), c^{i_m})$.

Pulling generations instead of messages A node sends a list of generations that it has not fully decoded to its neighbors. The target node answers with one of the generation it knows. To determine if the information will be redundant, there is no other solution than asking the target node to generate a linear combination and try to add it to the node's local matrix. During our tests, it appears that blindly asking generations didn't increase the number of redundant packets compared to a traditional Push-Pull algorithm asking for a message identifier list while greatly decreasing message sizes.

Count needed independent linear combinations To provide adaptiveness, estimating the number of useful packets needed to receive all the missing messages is needed by some protocols. Without network coding, the number of needed packets corresponds to the number of missing messages. With network coding, partially decoded packets are also considered as missing messages, but to decode them we need fewer packets than missing messages. In this case, the number of useful packets needed corresponds to the number of independent linear combinations needed.

5.3.3 CHEPIN

To ease the integration of RLNC in gossip-based dissemination algorithms, we encapsulated some common logic in algorithm 2. We represent a network-coded packet by a triplet: $\langle g, c, e \rangle$, where g is the generation number, c an ordered set containing the network coding coefficients and e the encoded payload.

We define 3 global sets: rcv, ids and dlv. rcv contains a list of network coded packets as described before that are modified each time a new one is received to stay linear independent until all messages are decoded. ids contains a list of known messages identifiers under the form $\langle g, gid \rangle$ where g is the generation and gid is the identifier of the message inside the generation. By using this tuple as unique identifier, we can reduce the number of bytes of gid as the probability of collision inside a generation is lower than the one in the whole system. Finally dlv contains a list of message identifiers similar to ids, but contains only identifiers of decoded messages.

The presented procedure relies on some primitives. RANK returns the rank of the generation, by counting the number of packets associated to the given generation. SOLVE returns a new list of packets after applying a Gaussian elimination on the given generation and removing redundant packets. DELIVER is called to notify a node of a message (if the same message is received multiple time, it is delivered only once).

This procedure updates the 3 global sets previously defined, delivers decoded messages and return the usefulness of the given packet. Internally, the node starts by adding the packet to the matrix and do a Gaussian elimination on the packet's generation (line 10), if decoding the packet didn't increase the matrix rank, the packet was useless and the processing stops here. Otherwise, the node must add unknown message identifiers from the packet coefficient list to the

```
1  Function init():
2      g ← 0 /* Encoding generation                                   */
3      rcv ← Set() /* Received packets                               */
4      ids ← OrderedSet() /* Known message identifiers               */
5      dlv ← OrderedSet() /* Delivered message identifiers           */
6  Function ProcessPacket(p):
7      if p = ∅ then
8          return False
9      ⟨g¹, c¹, _⟩ ← p
10     oldRank ← Rank(g¹, rcv)
11     rcv ← Solve(g¹, rcv ∪ {p})
12     if oldRank = Rank(g¹, rcv) then
13         return False /* Packet was useless                        */
14     forall ⟨id, _⟩ ∈ c¹ do
15         ids ← ids ∪ {⟨g¹, id⟩} /* Register new identifiers        */
16     forall ⟨g², c², e⟩ ∈ rcv do
17         ⟨id, _⟩ ← c²[0]
18         if g¹ = g² ∧ len(c²) = 1 ∧ ⟨g², id⟩ ∉ dlv then
19             dlv ← dlv ∪ {⟨g², id⟩}
20             Deliver(e) /* New decoded message                     */
21     if g¹ > g ∨ Rank(g, rcv) ≥ 1 then
22         g ← max(g¹, g)+1 /* Update Lamport Clock                  */
23     return True /* Packet was useful                             */
```

Algorithm 2: Process RLNC Packets

known identifiers set. After that, the node delivers all decoded messages thanks to the received packet and stores their identifiers in dlv. Finally, the node checks if the clock must be updated.

Algorithms 3 and 4 show how the above procedures can be used to implement push and pull gossip protocols. For push, we do not directly forward the received packet, but instead forward a linear combination of the received packet's generation after adding it to our received packet list. For Pull, we request generations instead of messages. Like existing protocols, we keep a $rotation$ variable that rotates the set of missing identifiers, allowing missing generations to be generated in a different order on the next execution of the code block.

```
1  Function init():
2      k, ittl ← ... /* Push fanout and initial TTL   */
3      dottl, dodie ← ... /* Push strategy            */
4  Function SendToNeighbors(h, headers):
5      for k times do
6          p ← Recode(h, rcv)
7          Send(PUSH, p, headers)
8  Function Broadcast(m, headers):
9      id ← UniqueID()
10     p ← ⟨g, {⟨id, 1⟩}, m⟩
11     dlv ← dlv ∪ {⟨g, id⟩}
12     ProcessPacket(p)
13     headers.ttl ← ittl
14     SendToNeighbors(g, headers)
15 Function NCPush(p, headers):
16     ⟨h, _, _⟩ ← p
17     if ProcessPacket(p) ∨ ¬dodie then
18         if dottl ∧ headers.ttl ≤ 0 then
19             return
20         if dottl then
21             headers.ttl ← headers.ttl − 1
22         SendToNeighbors(h, headers)
```

Algorithm 3: Push

```
1  Function init():
2      rotation ← 0 /* Rotation position                            */
3  Function NCPullThread(headers):
4      ask ← OrderedSet()
5      rotation ← rotation + 1 mod |ids \ dlv|
6      forall m ∈ Rotate(ids \ dlv, rotation) do
7          ask ← ask ∪ Gen(m, rcv)
8      Send(PULL, ask, headers)
9  Function NCPull(asked, headers):
10     p ← ∅
11     if ∃ g ∈ asked, Rank(g, rcv) > 0 then
12         p ← Recode(g, rcv)
13     Send(PULLREPLY, p, headers)
14 Function NCPullReply(p):
15     ProcessPacket(p)
```

Algorithm 4: Pull

5.4 Application to Pulp

To apply our network-coding approach to a concrete use case, we design CHEPIN-Pulp, a protocol inspired by Pulp [60]. Pulp achieves cost-effective dissemination by optimizing the combination of push-based and pull-based gossip. In particular, nodes disseminate each message through a push-phase with little redundancy due to a fanout and a TTL configured to reach only a small portion of the network. As the push-phase doesn't provide a complete dissemination, the message will be retrieved by the rest of the network during a pull phase. To this end, each node periodically sends its list of missing messages to a random node. The target node answers with the first message it knows. To discover missing messages, nodes piggyback the list of recently received messages on every packet exchange. To improve reactivity and reduce delays, Pulp provides a pulling frequency-adaptation mechanism based on the node's estimation of missing messages and usefulness of its pull requests.

```
1  Function init():
2      ts, sm ← ... /* Trading window size and margin                          */
3      Δ_adjust, Δ_pull_min, Δ_pull_max ← ... /* Periods config.              */
4      dottl, dodie ← True, True /* Set push strategyy                        */
5      Δ_pull ← Δ_adjust
6  Function GetHeaders():
7      start ←max(0, |ids| − sm − tm) end ←max(0, |ids| − sm) return {tw : ids[start : end]}
8  Upon Receive Push(p, headers) do:
9      ids ← ids ∪ headers.tw
10     NCPush(p,GetHeaders())
11 Upon Receive Pull(asked, headers) do:
12     ids ← ids ∪ headers.tw
13     NCPull(asked,GetHeaders())
14 Upon Receive PullReply(p, headers) do:
15     ids ← ids ∪ headers.tw
16     NCPullReply(p)
17 Thread: Every Δ_pull run:
18     NCPullThread(GetHeaders())
19 Thread: Every Δ_adjust run:
20     missingSize ← |ids| − |rcv|
21     if missingSize > prevMissingSize then
22         Δ_pull = Δ_adjust / (missingSize − prevMissingSize + prev_useful)
23     else if missingSize > 0 ∧ prev_useless ≤ prev_useful then
24         Δ_pull ← Δ_pull × 0.9
25     else
26         Δ_pull ← Δ_pull × 1.1
27     Δ_pull ←max(Δ_pull, Δ_pull_min)
28     Δ_pull ←min(Δ_pull, Δ_pull_max)
29     prev_useless ← 0
30     prev_useful ← 0
31     prevMissingSize ← missingSize
```

Algorithm 5: Pulp NC

On top of the two previously defined algorithms 3 and 4, we propose a push-pull algorithm inspired by Pulp (Algorithm 5. First, we must convert the Pulp message discovery mechanism which consists on exchanging recent message history via a trading window. The trading window is generated by the GETHEADERS function, which will be added to every packets. On reception, the trading window will be retrieved and its new identifiers will be added to the ids set. The major difference with Pulp is that we don't trade identifiers of delivered messages but any identifiers we know, even if we compare both approaches in Section 5.5.

The adaptation mechanism is the second feature of Pulp, the pull frequency is adapted according to the number of missing packets and the usefulness of the pull requests. Our only modification is made on how to compute the number of missing packets, as we retained the number of needed independent linear combinations instead of the number of missing messages. To do so, we compute the difference between the number of messages identifiers and the number of independent linear combinations we have.

5.5 Evaluation

We evaluated our solution in the Omnet++ simulator, using traces from PlanetLab and Overnet to simulate respectively the latency and the churn. To assess the effectiveness of CHEPIN, we implemented a modified version of Pulp as described in Section 5.4, and compare it with the original Pulp protocol.

5.5.1 Experimental setup

We run our experiments with 1 000 nodes, sending 1 000 messages at a rate of 150 messages per second. Each message weighs 1KB and has a unique identifier. For Pulp, it is simply an integer encoded on 8 bytes. In the case of CHEPIN-Pulp, the unique identifier is composed of its generation encoded on 4 bytes and a unique identifier in this generation, also encoded on 4 bytes. Δ_{adapt} is set to 125 ms.

In order to accurately simulate latency, we use a PlanetLab trace[45]. Latency averages at 147 ms for a maximum of almost 3 seconds. Most of the values (5th percentile and 95th percentile) are between 20 ms and 325 ms. Finally we have a long tail of values between 325 ms and the maximum value.

5.5.2 Configuring the Protocols

To configure protocols, we chose an experimental approach. First, we selected a suitable value for the size of the trading window. As explained in Section 5.4, too small values of this parameter result in wasted pull requests, and missing messages, while too large ones lead to wasted bandwidth. We therefore tested the ability of the original Pulp, and of our solution to achieve complete dissemination (i.e. all messages reach all nodes) with different trading window sizes, and a safety margin of 10. Results, not shown for space reasons, show that our solutions reaches complete dissemination with trading window sizes of at least 6, while Pulp requires trading-window sizes of at least 9. For the rest of our analysis, we therefore considered a trading-window size of 9, and a safety margin of 10. Nonetheless, this first experiment already hints at the better efficiency of our network-coding-based solution.

Next, we selected values for fanout and TTL. Figure 5.2 reports the delivery delays and bandwidth consumption of the two protocols with several values of these two parameters. To measure bandwidth consumption, we consider the ratio between the average amount of bandwidth consumed by the protocol, and the lower bound represented by the bandwidth required for the same task in a tree structure in a stable network. First, we observe that in terms of delays and bandwidth used, our network coding variant is more stable than the original Pulp. That is, with low values of fanout and ttl, the original algorithm deteriorates faster.

Next, we see that our network coding variant performs better or similarly for every combination of fanout and ttl both in terms of sent bandwidth and delay. The best configuration in term of sent data for Pulp corresponds to the configuration $k = 6, ttl = 4$ with 2.12 KB for 1KB of useful data and an average of 0.67 seconds to disseminate a message. Our network-coding solution reduces delay to 0.55, with a bandwidth consumption of 1.83KB/msg. With a fanout of 5 our solution further decreases consumed bandwidth to 1.66 KB/msg but with a slight increase in delay (0.83 s). Clearly, to achieve the minimum delays, the best strategy consists in boosting the push phase by increasing the TTL, but this defeats the bandwidth-saving goal of Pulp and our approach. As a result, we use the configuration with $k = 6, ttl = 4$ for both protocols in the rest of our comparison.

5.5.3 Bandwidth and delay comparison

We evaluate how our algorithm performs over time in Figure 5.3. First, we logged the number of packets sent per second for the three types of packets: push, pull and pull reply. As we configured the two protocols with the same fanout and TTL, we would expect seeing almost the same number of push packets. But our network-coded variant sends 12% more push packets. Pulp stops forwarding a push packet if the corresponding message is already known. But since our variant can use a large number of linear combinations, our algorithm manages to exploit the push-phase better, thereby reducing the number of packets sent in the pull phase: 33% fewer pull and pull reply packets. This strategy enables us to have a packet ratio of only 2.27 instead of 2.70.

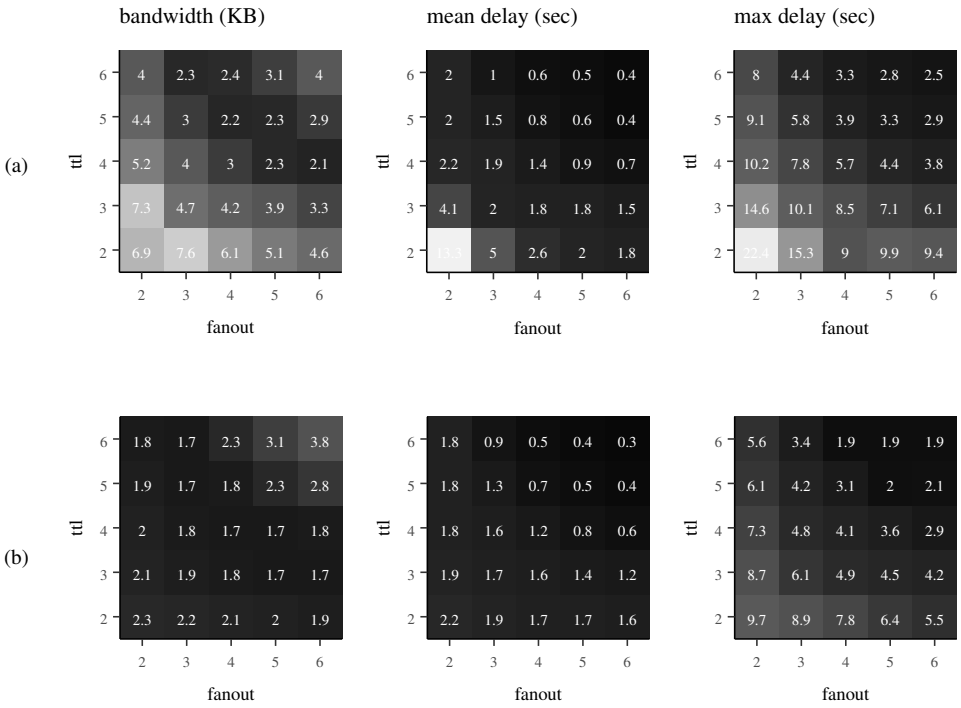


Figure 5.2: Pulp (5.2a) and our algorithm (5.2b) behavior under various configuration of the protocol (fanout and time to live)

As network coded packets include a sparse vector containing messages identifiers and values, our the algorithm has larger pull and pull reply packets than Pulp. Considering push packets, we also send more of them, which explains why we send 17% more data for these packets. However, as our pull phase is way smaller than the one from Pulp, the pull reply packets of our algorithm consume 28% less data than the ones from Pulp. We can also notice that the pull packets consume less data than the two others. Indeed, these packets never contain the 1 KB payload. However, we can notice that our algorithm still consumes less data than Pulp as we transmit generation id instead of each message id. With 150 messages/second, at a given time time, every nodes will be aware of a huge list of missing messages and ask it to their peers. Generally, this list will contain messages sent approximately at the same time, so probably in the same generation, which explain why it is way more efficient to ask for generation identifiers, and why pull packets will be smaller for our algorithm. These two facts enable us to have a data ratio 1.84 instead 2.12.

Finally, we study the distribution delay of each message. As our algorithm has a longer push phase, delays are smaller on average. We see a downward slope pattern on our algorithm's delays, especially on the maximum delays part. This pattern can be explained by the fact that decoding occurs at the end of each generation, so messages that are sent earlier wait for longer than the most recent ones.

5.5.4 Adaptiveness optimization

We now carry out a sensitivity analysis to understand the reasons for our improved performance. To this end, Figure 5.4 compares our algorithm with to Pulp and with two intermediate variants.

The first variant corresponds to a modification in the GetTradingWindow function of algorithm 2. Instead of using the message identifiers contained in the ids variable, we use the message identifiers contained in the dlv variable

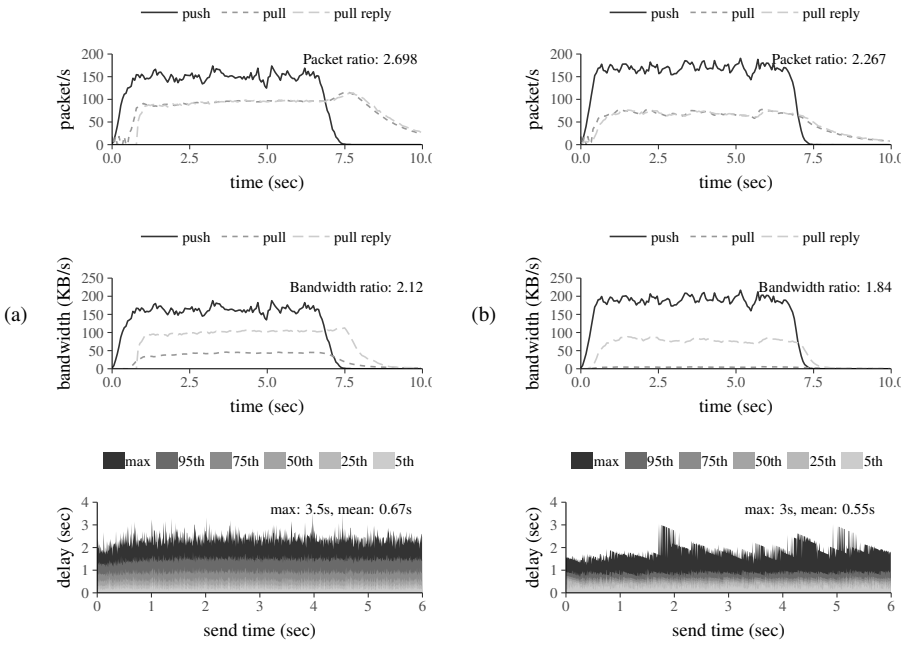


Figure 5.3: Comparison of exchanged packet rate, used bandwidth and message delay for 5.3a pulp original ($k = 6, ttl = 4$) and 5.3b our algorithm ($k = 5, ttl = 4$)

like in the case of the standard Pulp protocol. In other words, we disseminate the identifiers of messages we have decoded and not those we are aware of.

The second variant is a modification on how we count the number of missing messages at line 19 in algorithm 5. For this variant, we do $missingSize \leftarrow |missing|$ like in the original pulp. We thus evaluate the number of missing messages by counting all the message identifiers we have not yet decoded, without taking into account the progress of the decoding in our generations.

The two variants perform worse than our solution both in terms of delay and bandwidth. Variant 1 does not manage to achieve complete dissemination with a fanout of 6 and a TTL of 4, while Variant 2 achieves complete dissemination but at a higher cost ratio: 2.4 instead of 1.83 for our solution. This shows the importance of the modifications we made to the Pulp protocol.

To understand how these modifications behave, Figure 5.4 shows the different between the number of useful and useless packets. We see that our algorithm and Variant 1 perform similarly. The Pulp algorithm has a more efficient pull strategy than ours, possibly because there are more messages to pull, we receive redundant linear combination for the requested generation or the frequency adaptation is not optimal in our case. However, we see that we obtain way better results than the second variant, which mainly pull useless messages. At the end, we see lot of useless messages, as we don't inject useful messages anymore. The adaptive part of the algorithm therefore decreases the number of pull per second, to reduce useless messages.

When we look at the pull period, we see that Variant 1 and our algorithm are quite similar, with the biggest pull period as they have less messages to pull. Pulp has a smaller pull period, but has it has more messages to pull, it is not surprising. Our second variant pull period is the smallest even though this variant hasn't more messages to pull than our algorithm or our first variant, and explain why we have many useless messages: we pull to frequently.

Finally, when we study the evolution of our missing message estimation, it appears that the first variant has the lowest estimation of missing messages. It is due to the fact that we relay only decoded messages via the trading window, which adds a delay and improve the chances to receive the information via network coding coefficients.

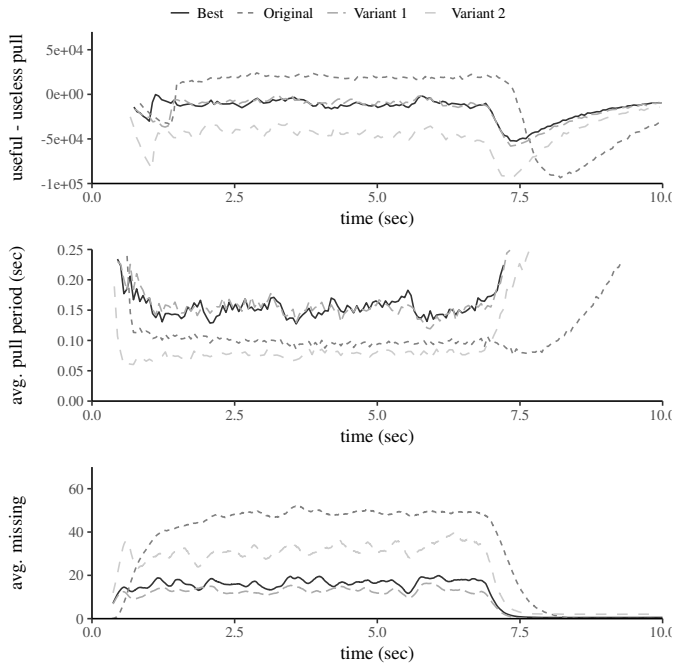


Figure 5.4: How adaptiveness algorithms impact the protocol efficiency

However, it appears that this method is not efficient to disseminate message identifiers as it fails to achieve a complete dissemination with high probability. Indeed, if a message is sent at the end of a generation, its content and existence will only be forwarded on its push phase and not by other messages of the generation.

5.5.5 Behaviour under network variation

Figure 5.5a shows how our algorithm performs under different network configurations. We see that when the number of messages per second decreases, the difference in term of data sent between the algorithms decreases too. This can be explained by the fact that our generations become smaller when the number of messages per second decreases, and so less useful.

At an extreme, when we have only one message per generation, we have the same model as Pulp: asking for a list of generation identifiers is similar to asking a list of message identifiers in Pulp. The network coded packets will contain a generation identifier, a sparse vector containing only one message identifier and one coefficient and the message. As a generation identifier plus a message identifier of our algorithm has the same size of a message identifier in Pulp, the only overhead is the one byte coefficient value.

We use an Overnet trace to simulate churn[137]. The trace contains more than 600 active nodes over 900 with continuous churn—around 0.14143 events per second.

We use this trace replayed at different speed to evaluate the impact of churn on the delivery delays of our messages, as plotted on Figure 5.5b. We chose to re-execute the trace at different speed: 500, 1000 and 2000 times faster for respectively 71, 141 and 283 churn events per second. We see that the original Pulp algorithm is not affected by churn, as the average and maximum delivery delays stay stable and similar to those without churn. Considering the average delay, it's also the case for our algorithm, where the average delay didn't evolve. The maximum delay doesn't evolve significantly either. However we can see huge differences in the shape of the maximum delay for each individual messages. Indeed, the decoding order and the generation delimitation are affected by the churn, but limited impact on message dissemination.

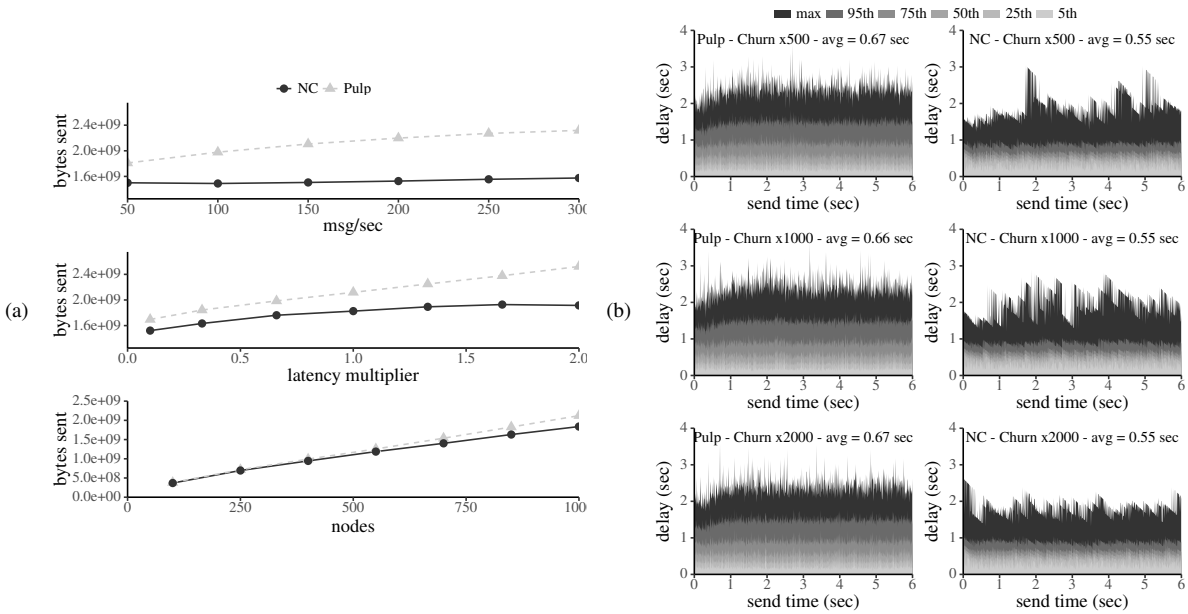


Figure 5.5: Delay in dynamic networks: varying network configurations (a) and churn (b).

5.6 Conclusion

Distributed systems become more and more complex and dynamic with time. Due to their properties and resistance to failure, data dissemination protocols are the cornerstone of any distributed system. However, these advantages come at the cost of bandwidth spent by the embedded redundancy in the protocol. This chapter introduced CHEPIN, a multi source gossip protocol that uses Lamport clocks to create generations, and sparse vectors to exchange coefficients. We demonstrated that is possible to apply RLNC for push and pull algorithm by thoroughly evaluating our approach. At worst, CHEPIN performs like the state of the art. At best, CHEPIN significantly improves both delay and bandwidth consumption.

Part II

Decentralized Recommenders and Privacy

Chapter 6

Context and Background

The second part of this manuscript, discusses the the application of epidemic protocols to decentralized recommender systems. This covers a line of research that started with my contribution to the Gossple project, and that is still ongoing. Gossple was an ERC grant awarded to Anne-Marie Kermarrec which funded part of my postdoctoral research at Inria. At that time, we started working on the application of gossip protocols to personalization and soon moved to their application to decentralized recommenders. This chapter starts by presenting a brief overview of the state of the art of decentralized recommender systems and by highlighting the contributions of this part of the manuscript. It then discusses a basic protocol for decentralized KNN which is employed and extended by our contributions. Finally, it discusses the main experimental settings we used in evaluating our protocols.

6.1 Recommender Systems

Recommender systems have become one of the primary means to navigate information in the digital space. Initially proposed in the early 1990s [169, 167], they have become widespread with the explosion of online retailers [149]. Content-based systems [109, 112, 78, 68] operate by analyzing the characteristics of the items being recommended. Collaborative filtering [149], on the other hand, ignores the content of the items and concentrates on the fact that users that have made like-minded choices in the past, will probably make like-minded choices in the future.

Several forms of collaborative filtering exist, the main distinction being between memory-based and model-based techniques. Memory based techniques store the user-item matrix in memory and compute similarities between users (user-based systems) [166] or between items (item-based ones) [149]. Model-based systems, on the other hand build intensive models that make it possible to achieve better scalability. Memory-based systems were the first to be implemented [167]. But in 2006, the Netflix Prize [101] popularized model based techniques like matrix factorization [117].

Even if most recommender systems maintain a centralized or at least data-center-based architecture, several authors have started proposing decentralized recommendation solutions. Rather than collecting all ratings at a central point, decentralized recommenders compute recommendations collaboratively and divide computation cost among user machines [66, 14, 133, 182, 103, 131]. For example, [131] proposes a Chord-based CF system to decentralize the recommendation database on a P2P infrastructure.

In addition to popularizing matrix factorization, the Netflix prize also highlighted the privacy issues associated with the technology after researchers de-anonymized a user from the provided dataset [96]. This prompted researchers to focus on alternative ways to provide recommendations, in particular by exploring decentralized systems [66, 14]. The interest of decentralized recommendation is twofold. On the one hand, decentralization provides an effective manner to improve the scalability of recommender systems. But most importantly a decentralized recommender system eliminates the need to collect information at a central entity, thereby reducing the damage that can be done by an attacker with a single attack. However, decentralization also increases the surface of attack. The decentralized KNN algorithms

at the basis of most peer-to-peer recommenders [14, 56] require peers to exchange their interest profiles with other peers in order to compute similarity values. In doing so, they do not simply *risk* to share sensitive information; they systematically require users to share personal data with other random users. This makes it very easy for an attacker to learn about the interests of a large number of victims.

Several collaborative-filtering approaches [150, 123] have tried to preserve the privacy of sensitive data by applying randomized masking and distortion techniques to user profiles. Other decentralized approaches such as [152, 153, 133] exploit homomorphic encryption in a P2P environment. [70], in turn, addresses privacy by trust where participants exchange information and profile only across trusted content producer/consumer pairs. [58] proposes a differentially private protocol to measure the similarity between profiles. While differential privacy provides a strong notion of privacy, [71] highlights its important trade-off between privacy and accuracy.

A number of authors have proposed to address privacy by means of anonymity. Some, like [89] achieve receiver anonymity using group communication primitives like broadcasting, multicasting, and flooding. Others [128] focus on sender anonymity and relay messages from a node along a single anonymous path formed by nodes within the infrastructure. Some authors have already suggested the integration of gossip and anonymous services. The work in [63] uses gossip protocols to improve the robustness of trust-based overlays to provide privacy-preserving data dissemination. More precisely, it creates and maintains additional anonymous links on top of an existing social overlay.

Similarly, [72] relies on gossip protocols to support confidential communications and private group membership. This solution leverages existing multi-hops paths to circumvent network limitations such as NAT and firewalls to form anonymous channel. Other authors investigated accountability [82, 75, 46] and spam resilience [76]. Only a few, however, combined privacy and anonymity with personalization [39, 40, 43].

6.2 Contributions in Chapters 7 through 9

In the following chapters we present a decentralized recommender system for news items in a twitter-like setting [14] (Chapter 7) as well as two techniques for preserving privacy in decentralized recommendation [3, 11] (chapters 8 and 9).

WhatsUp, the contribution in Chapter 7, represents one of the first examples of decentralized recommenders, and to the best of our knowledge was the first to deal with dynamic streams of news items in a decentralized setting. It also introduces a similarity metric that outperforms cosine similarity, which generally provides the best empirical results [124].

The privacy-preserving technique in Chapter 8 works by adding non-random noise to user profiles and by applying differential privacy to item dissemination. Existing work has in fact shown that random noise can easily be separated from sensitive information [122, 145]. As a result, our contribution in Chapter 8 uses noise that depends on the interests of dynamic communities of users. This limits the amount of information exchanged between users to coarse-grained user profiles that only reveal the least sensitive information. Finally, the contribution in Chapter 9 presents a technique for privacy-preserving similarity computation and proves its efficacy in the context of recommendation.

The contributions in the following chapters are also closely related to several other contributions that did not fit in this manuscript. Gossple [18], my first piece of work in the context of personalized systems, includes a gossip-on-behalf protocol that hides the association between a user and her profile. Freerec [4] offers an alternative solution to securing the operation of WhatsUp. It employs TOR-like proxy chains to add anonymity to the operation of gossip-based decentralized recommenders.

6.3 Focus on Decentralized KNN

The following chapters design a decentralized recommender system and then address its major privacy issues. To this end, they build on a decentralized KNN protocol framework that was initially proposed by [127], and that we later exploited in the context of the Gossple Anonymous Social Network [18]. Computing KNN graphs constitutes a fundamental operation in a variety of data-mining applications. In our case, we use KNN to implement user-based

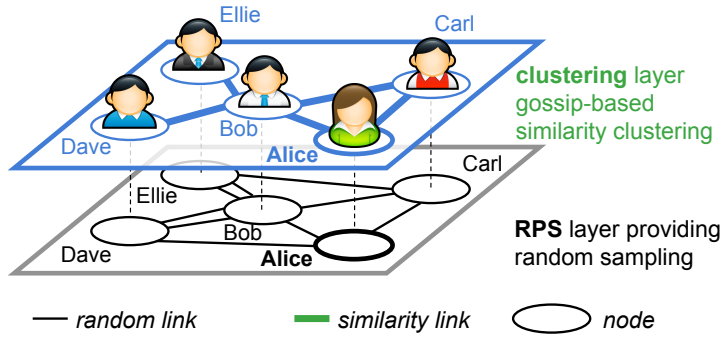


Figure 6.1: Gossip-based distributed KNN

collaborative filtering. This technique provides recommendations by identifying the items appreciated by the closest neighbors of a target user.

The framework consists of two protocols: a random-peer sampling (RPS) and a clustering protocol. The former maintains a continuously changing topology, while the latter converges to the KNN graph. Both protocols follow the same high-level behavior. In each protocol, each peer maintains a data structure, called $view$, consisting of a list of references to other peers: the peer's current neighbors in the corresponding protocol. Periodically, a peer p contacts another peer q from this list and sends it a subset of its own view—half of its view in the RPS protocol, and its entire view in the clustering protocol. Upon receiving such a subset, q merges the received subset with its own view. In the case of the RPS, it keeps c random entries from the union of the two views. In the case of the clustering protocol, it keeps the c entries whose profiles are most similar to its own after combining its own clustering view, its own RPS view and the received clustering view. Then q replies by sending to p a subset of its view before the update, and p updates its view analogously. Several choices are available to p for the selection of the peer q contacted at each round [106]: where not otherwise specified, we select the peer with the oldest timestamp.

The clustering protocol provides each peer with a view that converges to its KNN. The RPS provides resilience to churn and partitions and ensures that the process does not get stuck into a local minimum.

Figure 6.2 exemplifies the operation of the clustering protocol. Alice and Bob are interested in hearts, though Bob prefers diamonds. After exchanging their respective lists of neighbors, they keep the users which are closest to their interests. In this example, Alice replaces Ellie with Carl who likes hearts, and Bob replaces Alice with Ellie who likes diamonds. After a few cycles of this protocol, each peer's neighborhood view contains the corresponding KNN.

The system uses $asynchronous\ rounds$ that are executed periodically by each peer. In each round, each peer

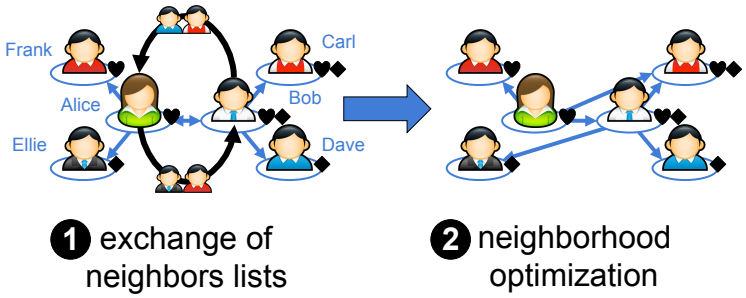


Figure 6.2: Clustering mechanism for convergence to an optimal neighborhood. In this example, after exchanging their profiles, Alice and Bob modify their neighbors in order to be connected with the users who share the most their interests.

attempts to select a better set of similar other nodes (its *neighbors*) according to some similarity metric: a widely used metric is for example cosine similarity [174]. It considers profiles as high-dimensional vectors in which each unique item is a dimension and values for each dimension correspond to ratings. It then evaluates the similarity of two profiles as the cosine of the angle between the two corresponding vectors.

$$cos(u_1, u_2) = \frac{u_1 \cdot u_2}{\|u_1\|\|u_2\|} \tag{6.1}$$

6.4 Experimental Setting

The following chapters present a variety of experiments that demonstrate the effectiveness of our contributions. In the following, we discuss the most common parameters we used in our experiments. We provide further more specific details in each chapter.

6.4.1 Metrics

We evaluate the effectiveness of our recommendation solutions using classical metrics used in information-retrieval and in recommender systems: *recall* (*i.e.* completeness), *precision* (*i.e.* accuracy), and *f-score*.

All measures are in $[0, 1]$. For an item, a recall of 1 means that all interested users have received the item as a recommendation. The problem with recall is that it does not account for spam. A trivial way to ensure maximum recall consists in recommending all items to all users. Precision tackles this problem: a precision of 1 for an item means that the item has been recommended to all the users that are interested in it.

In general, changing system parameters makes it possible to improve one metric while worsening the other. For this reason, it is important to provide a good trade-off between these two metrics. We express this trade-off by means of the F1-Score, defined as the harmonic mean of precision and recall [174].

$$Precision = \frac{|\{interested\ \ users\} \cap \{reached\ \ users\}|}{|\{reached\ \ users\}|}$$

$$Recall = \frac{|\{interested\ \ users\} \cap \{reached\ \ users\}|}{|\{interested\ \ users\}|}$$

$$F1 - Score = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

In addition, to recommendation quality, we also consider systems metrics that evaluate our contributions from a systems perspective. In particular, we consider the network traffic they generate. For simulations, we compute the total number of *sent messages*. For our implementations or where otherwise relevant, we also measure the average consumed *bandwidth*.

6.4.2 Datasets

We ran our experiments using the following datasets, summarized in Table 6.1.

ArXiv synthetic dataset. To obtain an objective evaluation without the artifacts of real datasets, we identified distinct groups among the 5242 users in the arXiv dataset (covering scientific collaborations between authors [181]) using a community-detection algorithm [134]. This allows us to deal with clearly defined communities of interest, thus enabling the evaluation of our recommender system in Chapter 7 in a clearly identified topology. The resulting dataset contains 21 communities ranging in size from 31 to 1036, for a total of 3703 users. For each community, we use a random subset of nodes as sources that generate 120 news items (for a total of about 2000). We use this dataset in Chapter 7.

	# users	# items	Chapters
arXiv synthetic	3,180	2,000	Ch 7
Digg	750	2,500	Ch 7
Survey	480	1,000	Ch 7, Ch 8
ML-100k	943	1,682	Ch 9
ML-1M	6,040	3,900	Ch 9
Jester-1-1	24,983	100	Ch 9

Table 6.1: Datasets and chapters in which they are used.

Digg dataset. Digg is a centralized social-news website designed to help users discover and share content. It disseminates news items along the edges of an explicit social network (an approach known as cascading). Relying on explicitly declared friends, as in Digg, is known to limit the content that can be received [136] by substantially influencing decision making [164]. Basically, users are only exposed to the content forwarded by their friends, while other items may be of interest to them. To remove this bias, we extracted for each user, u, the categories of the news items she generates. We then defined user u's interests by including all the news items associated with these categories. We collected traces from Digg over 3 weeks in 2010. The resulting dataset consists of 750 users and 2500 news from 40 categories. We use this dataset in Chapter 7.

Dataset from a user survey. We conducted a survey on 200 news items involving 120 colleagues and relatives. We selected news randomly from a set of RSS feeds illustrating various topics (culture, politics, people, sports, ...). We exposed this list to our test users and gathered their reactions (like/dislike) to each news item. This provided us with a small but *real* dataset of users exposed to exactly the same news items. To scale our system, we generated 4 instances of each user and news item in the experiments. This results in a bias that nonetheless affects both our systems and the state-of-the-art solutions we compare them with. We use this dataset in chapters 7 and 8.

MovieLens datasets. MovieLens consists of a group of datasets [49] extracted from the MovieLens movie-recommendation web site [161].[1] In our experiments, we consider two of the datasets: ML-100k and ML-1M, containing respectively 100,000 and 1,000,000 ratings ranging from 1 to 5. Since the recommenders discussed in the following only consider binary ratings, we make ratings binary by considering an item as liked when its rating is greater than or equal to 3. We use these datasets in Chapter 9.

Jester dataset. Jester consists of a dataset from for the Jester [156] online joke-recommendation service.[2] In our experiments, we use a subset of the dataset consisting of the first third of the ratings in dataset Jester-1. Ratings in Jester range from -10 to 10. Like for MovieLens, we make them binary by consider an item as liked when its rating is greater than or equal to 0.0. We use this dataset in Chapter 9.

6.4.3 Adversary Models

When evaluating our privacy-preserving techniques, we generally consider honest but curious adversaries that participate in the protocol correctly but that seek to learn information about user profiles. In specific cases, we consider more powerful adversaries. In Section 8.5.6 we consider a colluding coalition of adversaries that attempt to censor the dissemination of news items. In Chapter 9 we consider an adversary that can take a limited set of active actions: tapping unencrypted communications; attempting to bias multi-party computations; computing her similarity with her target as many times as she want. In each chapter, we also define specific metrics to measure the effectiveness of privacy protection.

[1] MovieLens datasets are available at: http://grouplens.org/datasets/movielens/
[2] Jester datasets are available at: http://eigentaste.berkeley.edu/dataset/

Chapter 7

WHATSUP: A Decentralized Instant News Recommender

This chapter introduces WHATSUP, a collaborative-filtering system for disseminating news items in a large-scale dynamic setting with no central authority. WHATSUP leverages the decentralized KNN protocol presented in Chapter 6 to cluster users based on their opinions (*like-dislike*) about the news items they receive. WHATSUP embodies two main contributions: WUP, and BEEP. WUP consists of a profile-management mechanism that takes into account long-standing and emerging (dis)interests while seeking to minimize spam through a novel similarity metric. BEEP consists of a novel heterogeneous gossip protocol that (1) biases the orientation of its targets towards those with similar interests, and (2) amplifies dissemination based on the level of interest in every news item. The work in this chapter is at the core of the PhD thesis of Antoine Boutet, a former PhD student of Anne-Marie Kermarrec with whom I closely collaborated. Moreover it involves contributions by Arnaud Jegou who was the first PhD student I formally co-advised with Anne-Marie Kermarrec.

> The content of this chapter is an adapted excerpt from the following paper:
> Antoine Boutet, Davide Frey, Rachid Guerraoui, Arnaud Jégou, Anne-Marie Kermarrec:
> WHATSUP: A Decentralized Instant News Recommender. IPDPS 2013: 741-752

7.1 Overview

The continuous stream of news items we are exposed to calls for automatic techniques to filter the right content and alleviate the need to spend a substantial amount of time browsing information online. Explicit subscription-based approaches (*e.g.* RSS, pub/sub, online social networks) often filter too much or not enough. *Personalized news recommender systems*, based on collaborative filtering (CF) [88], are much more appropriate for they operate in a dynamic and fine-grained manner to automate the celebrated *word-of-mouth* pattern by which people recommend useful items to each other. However, CF approaches require the maintenance of huge amounts of information as well as significant computation resources, especially in the context of continuous streams of news items that must be instantly delivered to users that potentially change interests over time.

This chapter proposes WHATSUP a completely decentralized instant news system based on CF. Intuitively, a P2P approach is attractive because it naturally scales and circumvents a central entity that controls all user profiles potentially exploiting them for commercial purposes. Yet, the absence of a central authority with global knowledge makes the filtering very challenging and calls for CF schemes that need to cope with partial and dynamic interest profiles. Like any CF scheme [88], WHATSUP assumes that users who have exhibited similar tastes in the past are likely to be interested in the same news items in the future. To implement this paradigm, in a decentralized setting WHATSUP relies on two distributed protocols, WUP and BEEP (Figure 7.1), that together provide an *implicit publish-subscribe* abstraction.

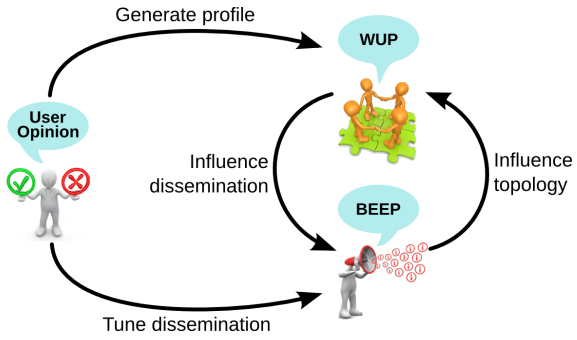


Figure 7.1: *Interactions between (1) user opinion; (2)* WUP*: implicit social network; (3)* BEEP*: news dissemination protocol*

They allow users to receive published items without having to specify explicit subscription filters. WHATSUP's user interface captures the opinions of users on the news items they receive through a simple *like/dislike* button. A user *profile* collects the resulting implicit interests in a vector associating news items with user opinions. This provides the driving information for the operation of WUP and BEEP.

WUP manages user profiles and relies on the decentralized protocol presented in Chapter 6 to maintain a directed graph linking nodes (reflecting users) with similar interests. WUP augments the KNN protocol by introducing a novel similarity metric that seeks to minimize spam while selecting users with potentially interesting items.

BEEP (Biased EpidEmic Protocol) consists of novel heterogeneous epidemic protocol that disseminates news items using WUP's topology. BEEP obeys the *explore-and-exploit* principle. It biases dissemination towards nodes that are likely to have similar tastes (*exploit*), while introducing enough randomness and serendipity (ability of making fortunate discoveries while looking for something unrelated) to tolerate the inherent unreliability of the underlying network as well as to prevent interesting news items from being isolated within specific parts of the network (*explore*).

7.2 WUP

WUP builds and maintains an implicit social network reflecting the interests of users. To this end, WUP augments the algorithm described in Section 6.3 with a novel similarity metric, consisting of a variation of the well-known cosine similarity [125], and with profile management mechanisms. The WUP metric seeks to maximize the number of items that were liked in both profiles being compared. It also strives to minimize spam by discouraging a node, n, with profile P_n, from selecting a neighbor, c, with profile P_c, that explicitly dislikes the items that n likes. We achieve this by dividing the number of liked items in common between the two profiles by the number of items liked by n on which c expressed an opinion. We define $sub(P_n, P_c)$ as the subset of the scores in P_n associated with the items that are present in P_c. By further dividing by the number of items liked by c (as in cosine similarity), we then favor neighbors that have more restrictive tastes. The asymmetric structure of this metric is particularly suited to push dissemination (*i.e.* users choose the next hops of news items but have no control on who sends items to them) and improves cold start with respect to cosine similarity as explained in Section 7.5.3.

$$Similarity(n, c) = \frac{sub(P_n, P_c) \cdot P_c}{\|sub(P_n, P_c)\| \, \|P_c\|}$$

7.2.1 News item

A news item consists of a title, a short description, and a link to further information. The source of an item (the user publishing it) associates it with a timestamp indicating its creation time and a dislike-counter field initialized to zero

that sums the number of dislikes obtained by the item. The WUP algorithm also uses an 8-byte hash as the identifier of the news item. This hash is not transmitted but computed by nodes when they receive the item.

7.2.2 Profiles

WUP records information about interests for items in profile data structures. A profile is a set of triplets: identifier, timestamp, and score; $P \in \{< id, t, s > | id \in \mathcal{N}, t \in T, s \in [0, 1]\}$. Identifier and timestamp are defined as above, and each profile contains only a single entry for a given identifier. The score, instead, represents the level of interest for an item: 1 meaning interesting, and 0 not interesting.

WUP associates each node with a profile, the *user profile* ($\tilde{P}$), which contains information about the node's own interests. The scores associated with this profile are integer values (like-dislike). To disseminate news items, nodes employ an additional profile structure, the *item profile* (P^N). Unlike a user profile, the item profile is associated with a news item. Its score values are real numbers and are obtained through the aggregation of the profiles of the users that liked the item along its dissemination path. As a result, two copies of the same item along two different paths will have different profiles. This causes an item profile to reflect the interests of the portion of the network it has traversed. The item profile can also be viewed as a community profile expressing the interests of an implicit social network of nodes.

7.2.3 Updating profiles

Updating user profiles ($\tilde{P}$). A node updates its profile whenever it expresses its opinion on a news item either by clicking the *like* or the *dislike* button (line 5 or 7 in Algorithm 6), or when generating a new item (line 14). In either case, the node inserts a new tuple containing the news item's identifier, its timestamp, and a score value of 1 if it liked the item and 0 otherwise.

Updating item profiles (P^N). The item profile of an item I records the interests of the users who like I by aggregating their profiles along I's path. This works as follows. Let n be a node that likes I. When n receives I for the first time, it first updates its own user profile, $\tilde{P}$, as described above. Then, it iterates through all the tuples in $\tilde{P}$ (line 3). Let id be the identifier of one such tuple and let s^n be its score. Node n checks if I's item profile already contains a tuple for id (addToNewsProfilefunction). If so (line 20), let s be the tuple's score value in I's item profile; n replaces s with the average between s and s^n—the score in n's user profile. This averaging gives the same weight to both scores, s and s^n: it thus personalizes I's item profile according to n's interests. If I's item profile contains no tuple for id, node n inserts the tuple from its own user profile into I's item profile (line 22). When a new item is generated (function generateNewsItemin Algorithm 6), the source initializes the corresponding item profile by integrating its own user profile (line 15).

7.2.4 Initialization

A node, n, that is joining the system for the first time (*cold start*) contacts a random node, and inherits its RPS and WUP views. It then builds a fresh profile by selecting and rating the 3 most popular news items from the profiles of the nodes in its the selected RPS view. This process results in a profile and in a WUP view that are very unlikely to match n's interests. However, it provides n with a way to enter the WUP social network. Because the WUP metric takes into account the size of user profiles, nodes with very small profiles containing popular items such as joining nodes are more likely to be part of the WUP views of other nodes and quickly receive additional news items. This allows them to fill their profiles with more relevant content, thereby acquiring closer neighbors.

7.2.5 *Profile window*

The information stream is continuously evolving. In order to take into account only the current interests of users and to dynamically connect similar users, all profiles are cleaned of old items. Specifically, each node periodically purges its user profile of all the tuples whose timestamps are older than a profile window. Similarly, nodes purge item profiles of

```
1  on receive (item < id^I, t^I >, profile P^N, dislike counter d^I) do
2      if iLike(id^I) then
3          for all < id, t, s > ∈ P̃
4              addToNewsProfile(< id, t, s >, P^N)
5          add < id^I, t^I, 1 > to P̃
6      else
7          add < id^I, t^I, 0 > to P̃
8      for all < id, t, s > ∈ P^N
9          if t older than profile window then
10             remove < id, t, s > from P^N
11     BEEP.forward((< id^I, t^I >, P^N, d^I))
12 function generateNewsItem(item id^I)
13     P^N ← ∅;  d^I ← 0;  t^I ← currentTime
14     add < id^I, t^I, 1 > to P̃
15     for all < id, t, s > ∈ P̃
16         addToNewsProfile(< id, t, s >, P^N)
17     BEEP.forward((< id^I, t^I >, P^N, d^I))
18 function addToNewsProfile(< id, t, s^n >, P^N)
19     if ∃s| < id, ∗, s > ∈ P^N then
20         s ← (s + s^n)/2
21     else
22         P^N ← < id, t, s^n >
```

Algorithm 6: WUP: receiving / generating an item.

non-recent items before forwarding items to BEEP for dissemination (lines 8 to 10). The value of this profile window defines the reactivity of the system with respect to user interests as discussed in Section 7.4.3.

It is important to note that the profile window also causes inactive users who have not provided ratings during the current window to have empty profiles, thus being considered as new nodes. Yet, as in the case of initialization, the WUP metric allows these users to reintegrate quickly as soon as they connect and resume receiving news items.

7.3 BEEP

BEEP consists of a novel gossip-based dissemination protocol embodying two mechanisms: *orientation* and *amplification*, both triggered by the opinions of users on news items. Orientation leverages the information provided by WUP to direct news items towards the nodes that are most likely to be interested in them. Amplification varies the number of dissemination targets according to the probability of performing a useful forwarding action. Orientation and amplification make BEEP the first user-driven gossip protocol to provide heterogeneity in the choice as well as in the number of dissemination targets, achieving differentiated delivery. BEEP follows the well-known SIR (Susceptible, Infected, Removed) [81] model. A node receiving a news item for the first time updates the item's profile as described in Section 7.2.3. Then, it forwards the item to fanout (f) other nodes chosen according to its opinion on the item, as described in the following. A node receiving an item it has already received simply drops it.

7.3.1 Forwarding a disliked item

With reference to Algorithm 7 and Figure 7.2, consider Bob, who does not like item I sent by Carlos. BEEP first verifies if the dislike-counter field of the item has already reached the prescribed TTL (line 3). If it has, it drops the item. Otherwise it increments its value, and achieves orientation by identifying the node from Bob's RPS view whose user profile is closest to the item's profile (line 5) and forwards the item to it (line 12). The item profile allows BEEP's orientation mechanism to identify a target that is reasonably close to someone who liked the item, even if its topic falls outside Bob's interests. The use of a fanout of 1, instead, accounts for unexpected interests and addresses serendipity by

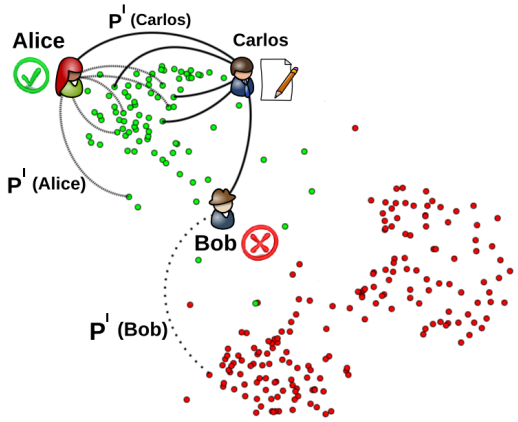


Figure 7.2: *Orientation and amplification mechanisms of Beep*

giving news items the chance to visit portions of the overlay where more interested nodes are present. At the same time, it also prevents non-interesting items from invading too many users.

```
1  function forward((< id^I, t^I >, profile P^N, dislike counter d^I))
2      if ¬ iLike(id^I) then
3          if d^I < TTL then
4              d^I ← d^I + 1
5              N ← selectMostSimilarNode(P^N, RPS)
6          else
7              N ← ∅
8      else
9          N ← selectRandomSubsetOfSize(WUP, f_LIKE)
10     if N ≠ ∅ then
11         for all n ∈ N
12             send < id^I, t^I > with associated P^N and d^I to n
```

Algorithm 7: BEEP: forwarding a news item.

7.3.2 Forwarding a liked item

Consider now Alice (Figure 7.2), who instead finds item I interesting. BEEP achieves orientation by selecting dissemination targets from her social network (WUP view). Unlike the profiles in the RPS view, those in the WUP view are relatively similar to each other. However, to avoid forming too clustered a topology by selecting only the closest neighbors, BEEP selects its targets randomly from the WUP view (line 9 in Algorithm 7). Moreover, since the targets' interests are expected to be similar to those of the node, BEEP amplifies I by selecting a relatively large subset of f_{LIKE} (*like fanout*) nodes instead of only one node, thus giving I the ability to reach more interested nodes.

7.4 Experimental setup

In this section, we present the experimental setup of WHATSUP's evaluation. We used three of the datasets presented in Section 6.4.2: *(i)* the 3180-user synthetic trace derived from Arxiv, *(ii)* the Digg dataset, and *(iii)* the survey. In the following, we present the remaining experimental settings: the competitors we compared against, the configurations for WHATSUP's parameters, and the evaluation metrics we use to assess the performance of WHATSUP.

7.4.1 WHATSUP Competitors

In order to demonstrate the effectiveness of WHATSUP, we evaluate it against the following alternatives:

Explicit cascading Cascading is a dissemination approach followed by several social applications, e.g., Twitter, Digg. Whenever a node *likes* (tweets in Twitter and diggs in Digg) a news item, it forwards it to all of its explicit social neighbors. We compare WHATSUP against cascading in the only dataset for which an explicit social network is available, namely Digg.

Complete explicit pub/sub WHATSUP can be seen as an *implicit publish/subscribe* (pub/sub) system turning interests into implicit subscriptions. Typically, pub/sub systems are explicit: users explicitly choose specific topics [140, 103]. Here, we compare WHATSUP against C-Pub/Sub, a centralized topic-based pub/sub system achieving complete dissemination. C-Pub/Sub guarantees that all the nodes subscribed to a topic receive all the associated items. C-Pub/Sub is also ideal in terms of message complexity as it disseminates news items along trees that span all and only their subscribers. For the sake of our comparison, we extract explicit topics from keywords associated with the RSS feeds in our survey. Then we subscribe a user to a topic if she likes at least one item associated with that topic.

Decentralized collaborative filtering In a decentralized CF scheme based on nearest-neighbor technique, when a node receives a news item it likes, it forwards it to its k closest neighbors according to some similarity metric. We implemented two versions of this scheme: one relying on the same metric as WHATSUP (CF-WUP) and one relying on cosine similarity [125] (CF-Cos). While it is decentralized, this scheme does not benefit from the orientation and amplification mechanisms provided by BEEP. More specifically, it takes no action when a node does not like a news item.

Centralized version of WHATSUP We also compare WHATSUP with a centralized system (C-WHATSUP) gathering the *global knowledge* of all the profiles of its users and news items. C-WHATSUP leverages this global information (vs a restricted sample of the network) to boost precision using complete search. When a user likes a news item, the server delivers it to the f_{LIKE} closest users according to the cosine similarity metric. In addition, it also provides the item to the f_{LIKE} users with the highest correlation with the *item's profile*. When a user does not like an item, the server presents it to the f_{DISLIKE} nodes whose profiles are most similar to the item's profile (up to TTL times).

7.4.2 Evaluation metrics

We consider the recommendation-quality and systems metrics we presented in Section 6.4.1. Throughout our evaluation, we examine results obtained over a wide range of fanout values by plotting the F1-Score against the fanout, and against the number of generated messages. The F1-Score for corresponding fanout values makes it possible to understand and compare the behavior of WHATSUP and its competitors under similar conditions. The F1-Score for corresponding numbers of messages, instead, gives a clearer picture about the trade-offs between recommendation quality and cost. Two different protocols operating at the same fanout, in fact, do not necessarily generate the same amount of traffic.

7.4.3 WHATSUP system parameters

The operation of WHATSUP is controlled by a number of system parameters. The first two parameters we consider are the WUP view size (WUP_{vs}) and the BEEP-I-like fanout (f_{LIKE}). Clearly, the former must be at least as large as the latter. As a node forwards a liked news item to random neighbors among its WUP view, a WUP_{vs} close to f_{LIKE} boosts precision while a large WUP_{vs} compared to f_{LIKE} increases recall. We set the value of WUP_{vs} to the double of f_{LIKE} as experiments provide the best trade-off between precision and recall for these values.

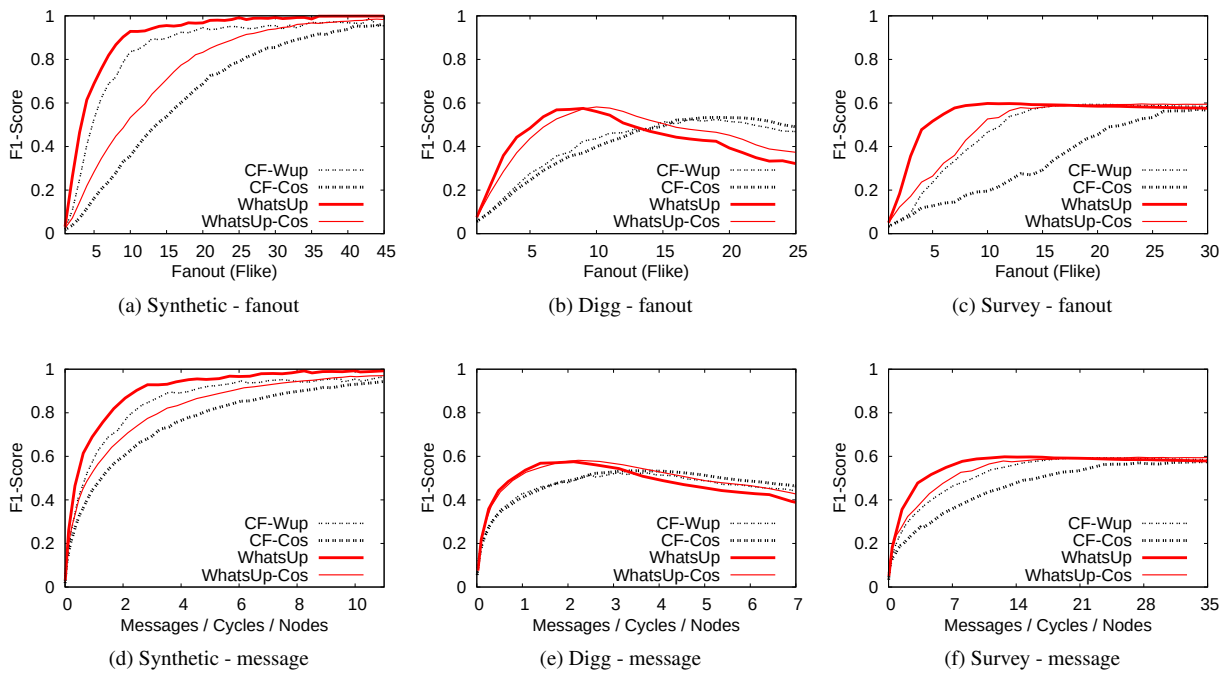
The third important parameter is the RPS view size. It directly impacts the potential of WUP to discover new nodes. We set its value to 30 to strike a balance between the need to discover information about nodes, the cost of gossiping,

and the need to retain some randomness in the selection of WUP neighbors. Too large values would lead the WUP view to converge too fast, hampering the ability to address non-foreseeable interests (serendipity). Nonetheless, we verified that our protocol provides good performance with values between 20 and 40 in the considered traces.

The BEEP TTL controls WHATSUP's *serendipity*, but it should not be too large in order not to hamper precision. We therefore set it to 4, and examine its impact in Section 7.5.2.

Finally, the size of the profile window determines WHATSUP's ability to adapt to dynamic and emerging interests of users. We set its value to 13 gossip cycles, corresponding to $1/5$ of the experiment duration, according to an analysis of its influence on the F1-Score. A size between $1/5$ and $2/5$ of the whole period gives the best F1-Score, while smaller or larger values make WHATSUP either too dynamic or not enough. For practical reasons, our simulations use the duration of a gossip cycle as a time unit to represent the length of the profile window. Yet, the actual duration of a gossip cycle is important and determines the dynamic response of our system. We discuss this parameter and its impact when evaluating our deployment (Section 7.5.6).

7.5 Evaluation



(a) Synthetic - fanout (b) Digg - fanout (c) Survey - fanout

(d) Synthetic - message (e) Digg - message (f) Survey - message

Figure 7.3: *F1-Score depending on the fanout and message cost*

We carried out an extensive evaluation of WHATSUP by simulation and by deploying its implementation on PlanetLab and on a ModelNet-based [155] cluster. All parameters, based on observations on a wide range of experiments on all datasets, are summarized in Table 7.1. We present the results by highlighting each important feature of WHATSUP.

7.5.1 Similarity metric

We start by evaluating the effectiveness of the WUP metric. Figures 7.3a-7.3f compare two CF approaches and two versions of WHATSUP based, respectively, on cosine similarity (CF-Cos and WHATSUP-Cos) and our WUP metric

Parameter	Description	value
RPS_{vs}	Size of the random sample	30
RPS_{f}	Frequency of gossip in the RPS	1h
WUP_{vs}	Size of the social network	$2f_{\text{LIKE}}$
Profile window	News item TTL	13 cycles
BEEP TTL	Dissemination TTL for dislike	4

Table 7.1: WHATSUP *parameters - on each node*

Algorithm	Precision	Recall	F1-Score	Mess./User
Gossip ($f = 4$)	0.35	0.99	0.51	4.6k
CF-Cos ($k = 29$)	0.50	0.65	0.57	5.9k
CF-Wup ($k = 19$)	0.45	0.85	0.59	4.7k
WHATSUP-Cos ($f_{\text{LIKE}} = 24$)	0.51	0.72	0.60	4.3k
WHATSUP ($f_{\text{LIKE}} = 10$)	**0.47**	**0.83**	**0.60**	**2.4k**

Table 7.2: *Survey: best performance of each approach*

(CF-WUP and WHATSUP). Our metric consistently outperforms cosine similarity in all datasets. Table 7.2 conveys the fact that it achieves this by improving recall over cosine similarity (by 30% for CF approaches and 15% for WHATSUP in the survey dataset with lower message cost in both cases). Moreover the relatively high precision of cosine similarity is partly an artifact of its low recall values resulting from highly clustered topologies. As a result, approaches using cosine similarity require a much larger fanout and message cost to provide the same quality of recommendation. The WUP metric generates instead topologies with a lower clustering coefficient by avoiding node concentration around hubs (an average clustering coefficient of 0.15 for WUP metric compared to 0.40 for cosine similarity in the survey dataset). In addition, the WUP metric avoids fragmenting the topology into several disconnected parts. Figure 7.4 shows the fraction of nodes that belong to the largest strongly connected component (LSCC) with increasing fanout values. Once all users are part of the same connected component, news items can be spread through any user and are not restricted to a subpart of the network. This corresponds to the plateaus in the F1-Score values visible in Figure 7.3c. The WUP metric reaches this state with fanout values around 10 both in CF-WUP and WHATSUP. This is a lot earlier than cosine similarity, which only reaches a strongly connected topology with fanout values above 15. Additional results, not on the plot, show that the fragmentation induced by the WUP metric is consistently lower than that associated with cosine similarity even for smaller fanout values. With a fanout of 3, for instance, WHATSUP's and CF-WUP's topologies contain respectively an average of 1.6 and 2.6 components, while WHATSUP-Cos's and CF-Cos's contain respectively an average of 12.4 and 14.3.

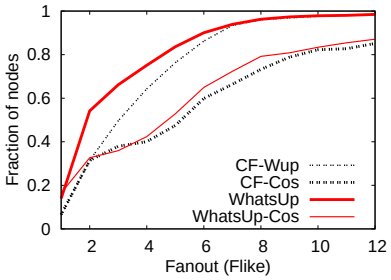


Figure 7.4: *Survey: Size of the LSCC depending on the approach*

7.5.2 Amplification and orientation

Comparing WHATSUP with CF schemes allows us to evaluate the impact of amplification and orientation. The results in Figures 7.3a-7.3f show that WHATSUP consistently outperforms CF, reaching higher F1-Score values with lower fanouts and message costs. Table 7.2 shows that it achieves recall values much higher than those of CF, with less than two thirds the message cost. This is a direct result of the amplification and dislike features, which allow an item to reach interested nodes even after hitting uninterested ones. This observation is confirmed by comparing Figure 7.3c

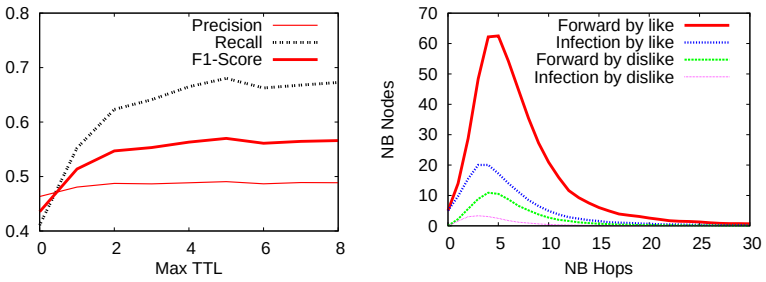
with Figure 7.4. Even if approaches adopting the same metric result in similar topologies as conveyed by Figure 7.4, the performance of those that employ amplification and dislike is consistently higher for corresponding fanout values.

Table 7.3 further illustrates the impact of the *dislike* feature by showing, for each news item received by a node that likes it, the number of times it was forwarded by nodes that did not like it. For instance, we can see that 31% of the news items liked by nodes were forwarded exactly once by nodes that did not like them. This conveys the benefit of the dislike feature and the importance of (negative) feedback from users in giving items a chance to reach interested nodes across the entire network.

Number of dislikes	0	1	2	3	4
Fraction of news	54%	31%	10%	3%	2%

Table 7.3: News received and liked via dislike

Figure 7.5a shows the impact of the TTL value on the performances. Too low a TTL mostly impacts recall; yet values of TTL over 4 do not improve the quality of dissemination. Finally, Table 7.2 also includes the performance of a standard homogeneous gossip protocol, which achieves the worst F1-Score value of 0.51 with almost twice as many messages as WHATSUP.



(a) *Impact of the dislike feature* (b) *Impact of amplification ($f_{\text{LIKE}} = 5$)*

Figure 7.5: [*Dislike and Amplification in* BEEP *on the Survey dataset.*

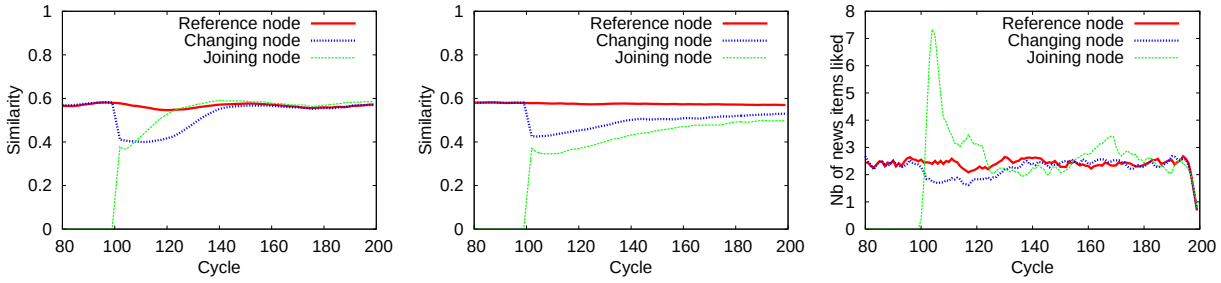
Figure 7.5b shows hows nodes at increasing distances from the source of a news item contribute to dissemination. We observe from the bell-shaped curve that most dissemination actions are carried out within a few hops of the source, with an average around 5. This is highly beneficial because a small number of hops leads to news items being disseminated faster. Finally, the plot also confirms the effectiveness of the dislike mechanism with a non-negligible number of infections being due to dislike operations.

7.5.3 Implicit nature of WHATSUP

Next, we evaluate WHATSUP's reliance on implicit acquaintances by comparing it with two forms of *explicit* filtering: *cascading* over explicit social links, and the ideal pub/sub system, C-Pub/Sub.

The first set of results in Table 7.4 shows that WHATSUP achieves a higher F1-Score with respect to cascading. More specifically, while both approaches provide almost the same level of precision, WHATSUP outperforms (by more than six times) cascading in terms of recall. The very low recall of cascading highlights the fact that the explicit social network does not necessarily connect all the nodes interested in a given topic. The low number of messages of cascading is a result of its small recall. The network traffic per infected user generated by WHATSUP is, in fact, 50% less than that of cascading (2.57K messages vs 5.27K).

The second set of results in the table compares WHATSUP with C-Pub/Sub. As discussed in Section 7.4.1, C-Pub/Sub disseminates news items to all subscribers with a minimal number of messages. Its recall is therefore 1 while



(a) Similarity in WUP view (WHATSUP) (b) Similarity in WUP view (WHATSUP-Cos) (c) Reception of liked news items (WHATSUP)

Figure 7.6: *Cold start and dynamics in* WHATSUP

Dataset	Approach	Precision	Recall	F1-Score	Messages
Digg	Cascade	0.57	0.09	0.16	228k
	WHATSUP	**0.56**	**0.57**	**0.57**	**705k**
Survey	C-Pub/Sub	0.40	1.0	0.58	470k
	WHATSUP	**0.47**	**0.83**	**0.60**	**1.1M**

Table 7.4: WHATSUP *vs C-Pub/Sub and Cascading*

its precision is only limited by the granularity of its topics. In spite of this, WHATSUP improves C-Pub/Sub's accuracy by 12% in the survey dataset with a little more than three times as many messages while conserving a good recall. This results in a better trade-off between accuracy and completeness as indicated by its higher F1-Score.

Another important advantage of WUP's implicit approach is its ability to cope with interest dynamics. To measure this, we evaluate the time required by a new node joining the network and a node changing of interests to converge to a view matching its interests both in WHATSUP (Figure 7.6a) and in WHATSUP-Cos (Figure 7.6b).

For the joining node, we select a reference node and introduce a new joining node with an identical set of interests. We then compute the average similarity between the reference node and the members of its WUP view and compare it to the same measure applied to the joining node. We repeated the experiment by randomly choosing 100 joining nodes and averaged the results. The WUP metric significantly reduces the number of cycles required by the joining node to rebuild a WUP view that is as good as that of the reference node (20 cycles for WHATSUP vs over 100 for WHATSUP-Cos).

Yet, the node starts receiving news quickly as shown in Figure 7.6c with the peak in the number of interesting news received as soon as the node joins. This is a result of both our cold start mechanism (Section 7.2.4) and our metric's ability to favor nodes with small profiles. Once the node's profile gets larger, the number of received news per cycle stabilizes to values comparable to those of the reference node. Nonetheless, the joining node reaches 80% of the reference node's precision after only a few cycles.

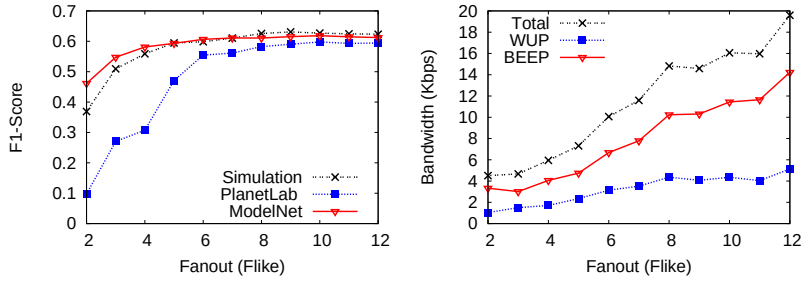
For the changing node, we select a pair of random nodes from the survey dataset and, at 100 cycles into the simulation, we switch their interests and start measuring the time it takes them to rebuild their WUP views. Figure 7.6 displays results obtained by averaging 100 experiments. Again, the WUP metric causes the views to converge faster than cosine similarity: 40 cycles as opposed to over 100. Moreover, the values of recall and precision for the nodes involved in the change of interests never decrease below 80% of the reference node's values. These results are clearly tied to the length of the profile window, set to about 40 cycles in these experiments. Shorter windows would in fact lead to an even more responsive behavior. We are currently evaluating this aspect on the current WHATSUP prototype. Moreover, while it may seem surprising that switching interests takes longer than joining a network from scratch, this experiment is an unlikely situation that provides an upper bound on the impact of more gradual interest changes.

Finally, the implicit nature of WHATSUP and the push nature of BEEP also make WHATSUP resilient to basic forms of content bombing. Unless a spammer node has enough resources to contact directly a large number of nodes, it will be unable to flood the network with fake news. The dislike mechanism, with its small fanout and TTL values will,

in fact, limit the dissemination of clearly identified spam to a small subset of the network.

7.5.4 Simulation and implementation

We also evaluate the performance obtained by our implementation in two settings: *(i)* a 170 PlanetLab node testbed with 245 users, and *(ii)* an emulated network of 245 nodes (machines and users) deployed on a 25-node cluster equipped with the ModelNet network emulator. For practical reasons we consider a shorter trace and very fast gossip and news-generation cycles of 30sec, with 5 news items per cycle. These gossip frequencies are higher than those we use in our prototype, but they were chosen to be able to run a large number of experiments in reasonable time. We also use a profile window of 4min, compatible with the duration of our experiments (1 to 2 hours each).



(a) Survey: F1-Score (b) Bandwidth in planetlab

Figure 7.7: *Implementation: bandwidth and performance*

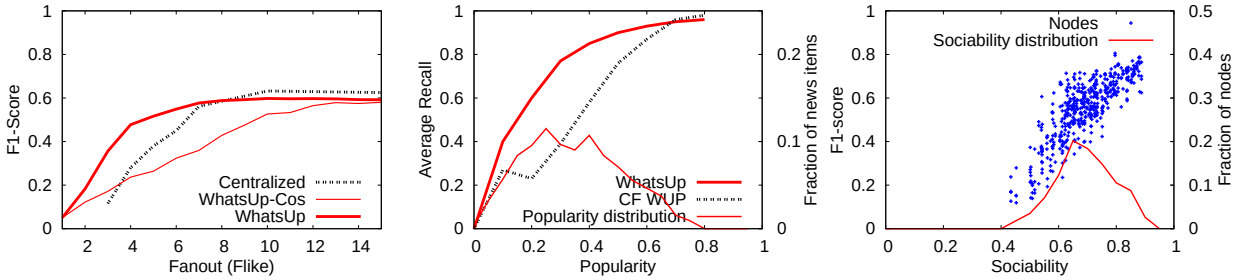
Figure 7.7a shows the corresponding results obtained on the survey and compares them to those obtained through simulation on the same 245-user dataset with increasing fanout values. ModelNet results confirm the accuracy of our simulations. The corresponding curves closely match each other except from some fluctuations with small fanout values. PlanetLab results, on the other hand, exhibit a clear decrease in performance with small fanouts. To understand this behavior, we can observe that in simulation and ModelNet, recall reaches scores above 0.50 with fanout values as small as 3. In PlanetLab, it only achieves a value of 0.18 with a fanout of 3, and goes above 0.50 only with fanouts of at least 6. The difference in recall with small fanout values can be easily explained if we observe the message-loss rates in the PlanetLab setting. With a fanout of 3, we recorded that nodes do not receive up to 30% of the news that are correctly sent to them. This is due to network-level losses and to the high load of some PlanetLab nodes, which causes congestion of incoming message queues. The impact of these losses becomes smaller when the fanout increases because BEEP is able to produce enough redundancy to recover from the missing messages.

7.5.5 Message loss

To understand the impact of lost messages, we experiment in the ModelNet network emulator with increasing loss rates affecting both BEEP and WUP messages and ranging from 0 to a huge value of 50%. Table 7.5 shows that both protocols preserve the reliability properties of gossip-based dissemination. With a fanout of 6, the performance in terms of F1-Score is virtually unchanged with up to 20% of message loss, while it drops only from 0.60 to 0.45 when half of the messages are lost by the network layer. With a fanout of 3, the impact of message loss is clearly more important due to the smaller amount of redundancy. 20% of message loss is sufficient to cause the F1-Score to drop from 0.54 to 0.47. This explains the differences between PlanetLab and ModelNet in Figure 7.7a. These drops are almost uniquely determined by the corresponding recall. With a fanout of 3 and a loss rate of 50%, recall drops to 0.07, causing an artificial increase in precision, and yielding an F1-Score of 0.12, against the 0.45 with a fanout of 6.

Loss Rate	0%		5%		20%		50%	
Fanout	3	6	3	6	3	6	3	6
Recall	0.63	0.82	0.61	0.82	0.46	0.80	0.07	0.45
Precision	0.47	0.48	0.47	0.47	0.47	0.46	0.55	0.44

Table 7.5: *Survey: Performance versus message-loss rate*



(a) *Centralized vs decentralized* (b) *recall vs popularity* (c) *F1-Score vs sociability*

Figure 7.8: Survey dataset: centralized vs decentralized, recall vs popularity, and F1-Score vs sociability.

7.5.6 Bandwidth consumption

Increasing fanout has a cost, which is highlighted by our bandwidth analysis in Figure 7.7. The number of times each news item is forwarded increases linearly with fanout values, causing an equally linear increase in the bandwidth consumption of BEEP. The bandwidth used by WUP also shows a slight increase with fanout due to the corresponding increase in the sizes of the WUP social networks. Nonetheless, the cost of the protocol is dominated by news. This highlights the efficiency of our implicit social-network maintenance. These experiments on a very fast trace with a gossip cycle every 30sec lead to a bandwidth consumption of about 4Kbps for WUP's view management. Our prototype is characterized by significantly lower gossip frequencies, on the order of 5min per gossip cycle. This results in a much lower average bandwidth consumption of about 0.4Kbps.

7.5.7 Partial information

To understand the impact of decentralization, we compare WHATSUP with a centralized variant, C-WHATSUP, that exploits global knowledge to instantaneously update node and item profiles. Figure 7.8a shows that WHATSUP provides a very good approximation of this variant (a 5% decrease of the F1-Score). More precisely, global knowledge yields better precision (17%) but slightly lower recall (14%).

7.5.8 Sociability and popularity

An additional interesting aspect is the impact of the popularity of items and the sociability of users. Figure 7.8b depicts the distribution of news-item popularity in the survey dataset together with the corresponding recall for WHATSUP and CF-WUP. WHATSUP performs better across most of the spectrum. Nonetheless, its improvement is particularly marked for unpopular items (0 to 0.5). This is highly desirable as popular content is typically much easier to manage than niche content. Recall values appear to converge for very popular items. However, each point in the plot represents an average over several items. An analysis of the data distribution (not shown for space reasons), instead, highlights how CF-WUP exhibits much higher variance leaving some items almost completely out of the dissemination. WHATSUP provides instead good recall values across all items thanks to the effectiveness of its *dislike* feature.

Figure 7.8c instead examines how the F1-Score varies according to the sociability of users in the survey dataset. We define sociability as the ability of a node to exhibit a profile that is close to others, and compute it as the node's

average similarity with respect to the 15 nodes that are most similar to it. Results confirm the expectations. WHATSUP leverages the similarity of interests between users and provides relevant results for users with alter-egos in the system. The more sociable a node the more it is exposed only to relevant content (improving both recall and precision). This acts as an incentive: the more a user exhibits a consistent behavior, the more she will benefit from the system.

7.6 Concluding remarks

This chapter presented WHATSUP, the result of our research on the feasibility of a fully decentralized collaborative filtering instant-news recommender. While relying only on partial knowledge, WHATSUP achieves a good trade-off between the accuracy and completeness of dissemination. For simplicity, this contribution, as well as the ones presented in later chapters focus solely on user-based collaborative filtering. Leveraging content-based filtering or decentralizing other collaborative filtering techniques may however provide interesting improvements as discussed in Section 10.2.2. Finally, this chapter purposely left out privacy concerns. The following chapters present two solutions that can enable WHATSUP or other collaborative-filtering systems [56] to protect user profiles from curious users or observers.

Chapter 8

Privacy-Preserving Distributed Collaborative Filtering

The distributed CF system defined in Chapter 7 does not protect the privacy of users. In this chapter, we aim to protect a decentralized item recommender from malicious nodes that extract information in two ways: (i) from the profiles they exchange with other nodes; and (ii) from the items they receive in the dissemination process. We do so by introducing two novel mechanisms. The first consists of an obfuscation scheme that hides the exact profiles of users without significantly decreasing their utility. The second consists of a randomized dissemination protocol that ensures differential privacy during the dissemination process. The work in this chapter is part of the PhD thesis of Antoine Boutet, a former PhD student of Anne-Marie Kermarrec with whom I closely collaborated. But it also involves significant contributions by Arnaud Jegou who was the first PhD student I formally co-advised with Anne-Marie Kermarrec.

> The content of this chapter is an adapted excerpt from the following paper:
> Antoine Boutet, Davide Frey, Rachid Guerraoui, Arnaud Jégou, Anne-Marie Kermarrec: Privacy-preserving distributed collaborative filtering. Computing 98(8): 827-846 (2016)

8.1 Overview

While decentralization removes the prying eyes of *Big-Brother* companies, it leaves those of curious users who might want to discover the personal tastes of others. In WHATSUP, we can distinguish two mains privacy leaks. First, as described in Section 7.2, nodes exchange profile information for maintaining the interest-based overlay. Profiles contains precise information about the interests of a user, which are potentially sensitive. Second, the dissemination protocol described in Section 7.3 is driven by the interests of users. As a consequence, a node a that receives an item from another node n can conclude that n *liked* that item. The predictive nature of this dissemination protocol thus also constitutes a leak of sensitive information. This chapter presents a solution that addresses each of these two vulnerabilities by means of two dedicated components. Section 8.2 describes the first: a profile obfuscation mechanism that prevents curious users from ascertaining the tastes of the user associated with an exchanged profile. Section 8.3 presents the second: a randomized dissemination protocol that ensures differential privacy by hiding the preferences of nodes during the item forwarding process.

We consider adversaries following the Honest-But-Curious model [143] where malicious nodes can collude to extract information and predict interests from received profiles but are not able to cheat in the protocol. In Section 8.5.6, we also consider adversaries that can modify their obfuscated profiles to control their location in the interest-based topology ($i.e.$ their clustering views).

8.2 Obfuscation Protocol

Our first contribution consists of an obfuscation protocol that protects user profiles by (i) aggregating their interests with those of similar users, and (ii) revealing only the least sensitive information to other users. By tuning these two mechanisms, system designers can manage the trade-off between disclosed information and recommendation quality [71]. An excessively obfuscated profile that reveals very little information is difficult to compromise, but it also provides poor recommendation performance. Conversely, a highly accurate profile yields better recommendations, but does not protect privacy-sensitive information effectively. As we show in Section 8.5, our obfuscation mechanism provides good recommendation while protecting privacy.

For clarity, this Section describes a simplified version of our obfuscation protocol. Section 8.3 completes this description with features required by our differentially-private dissemination scheme. Figure 8.1 gives an overview of the complete protocol.

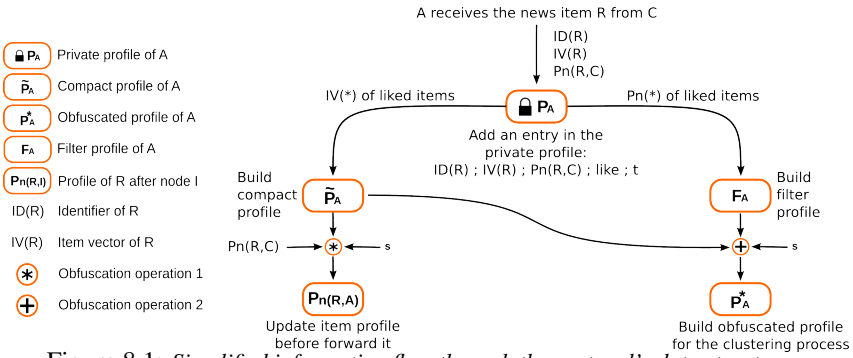


Figure 8.1: *Simplified information flow through the protocol's data structures.*

8.2.1 Overview

Our protocol relies on random indexing, an incremental dimension reduction technique [65, 157]. To apply it in our context, we associate each item with an *item vector*, a random signature generated by its source node. An *item vector* consists of a sparse d-dimensional bit array. To generate it, the source of an item randomly chooses $b << d$ distinct array positions and sets the corresponding bits to 1. It then attaches the *item vector* to the item before disseminating it.

Nodes use item vectors when recording information about items in their obfuscated profiles. Let us consider a node A that receives an item R from another node C as depicted in Figure 8.1. Node A records whether it likes or dislikes the item in its *private profile*. A node never shares its private profile. It only uses it as a basis to build an *obfuscated profile* whenever it must share interest information with other nodes in the clustering process. Nodes remove the items whose timestamps are outside the latest *time window*. This ensures that all profiles reflect the current interests of the corresponding nodes.

Upon receiving an item R that she likes, user A first updates the item profile of R and then forwards it (Figure 8.1). To this end, A combines the item vectors of the liked items in its *private profile* and obtains a *compact profile* consisting of a bit map. This dimension reduction introduces some uncertainty because different sets of liked items may result in the same *compact profile* as described in Section 8.2.2. Then A updates the *item profile* of R: a bitmap that aggregates the compact profiles of the nodes that liked an item. To update it, A combines its own *compact profile* and R's old *item profile*. This aggregation amplifies the uncertainty that already exists in *compact profiles* and makes R's *item profile* an obfuscated summary of the interests of the nodes that like R.

Before sharing interest information with other nodes, A must build its *obfuscated profile*. First, it creates a *filter profile* that aggregates the information contained in the *item profiles* of the items it liked. Then, it uses this filter to identify the bits from its *compact profile* that will appear in its *obfuscated profile*. The *filter profile* allows A to select the bit positions that are most popular among the nodes that liked the same items as it did. This has two advantages.

```
1  on receive (item < id^N, t^N >, item vector S^N, item profile P^N) do
2      if iLike(id^N) then
3          P ← < id^N, t^N, 1, S^N, P^N >
4          buildCompactProfile(S^N)
5          updateItemProfile(P^N)
6          forward(< id^N, t^N >, S^N, P^N)
7      else
8          P ← < id^N, t^N, 0 >
9  function buildCompactProfile()
10     for all < id, t, 1, S, P^N > ∈ P
11         P̃[i] = S[i] OR P̃[i]
12 function updateItemProfile(item vector P^N)
13     for all i ∈ P^N
14         Sum[i] = Integer(P̃[i]) + Integer(P^N[i])
15     for all i ∈ the s highest values in Sum
16         P^N[i] = 1
17 function forward(< id^R, t^R >, item vector S^N, item profile P^N)
18     for all n ∈ Neighbors
19         send < id^R.t^R > with associated S^N and P^N to n
```

Algorithm 8: Receiving an item.

First, using the most popular bits makes A's *obfuscated profile* likely to overlap with those of similar nodes. Second, these bits carry less information than less popular ones, which makes them preferable in terms of privacy.

8.2.2 Profile Updates

Private Profile A node updates its private profile whenever it generates a new item or receives an item it likes (lines 3 and 8 in Algorithm 8). In either case, the node inserts a new tuple into its private profile. This tuple contains the item identifier, its timestamp (indicating when the item was generated) and a score value (1 if the node liked the item, 0 otherwise). For liked items, the tuple also contains two additional fields: the item vector, and the item profile upon receipt.

Compact Profile. Unlike private profiles, which contain item identifiers and their associated scores, the compact profile stores liked items in the form of a d-dimensional bit array. As shown in Figure 8.1, and on lines 14 of Algorithm 8 and 5 of Algorithm 9, a node uses the compact profile both to update the item profile of an item it likes and to compute its obfuscated profile when exchanging clustering information with other nodes. In each of these two cases, the node computes a fresh compact profile as the bitwise OR of the item vectors of all the liked items in its private profile (line 11 of Algorithm 8).

This on demand computation allows the compact profile to take into account only the items associated with the current *time window*. It is in fact impossible to remove an item from an existing compact profile. The reason is that compact profile provides a first basic form of obfuscation of the interests of a user through bit collisions: a bit with value 1 in the compact profile of a node may in fact result from any of the liked items whose vectors have the corresponding bit set.

Compact profiles bring two clear benefits. First, the presence of bit collisions makes it harder for attackers to identify the items in a given profile. Second, the fixed and small size of bit vectors limits the size of the messages exchanged by the nodes in the system. As evaluated, in Section 8.5.7, this drastically reduces the bandwidth cost of our protocol.

Item Profile. A node never reveals its compact profile. Instead, it injects part of it in the item profiles of the items it likes. Consequently, the item profile of an item aggregates the interests of the users that liked the item along its

```
1 on demend do
2     Algorithm1.buildCompactProfile()
3     buildFilterProfile()
4     for all i ∈ the s highest values in F
5         P*[i] = P̃[i]
6 function buildFilterProfile()
7     for all < id, t, 1, S, P^N >∈ P in the current time window
8         F[i] = F[i] + Integer(P^N[i])
```

Algorithm 9: Building obfuscated profile.

dissemination path. A parameter s controls how much information from the compact profile nodes include in the item profile.

Let n be a node that liked an item R. When receiving R for the first time, n computes its compact profile as described above. Then, n builds an integer vector as the bit-by-bit sum of the item profile and its own compact profile (line 14 in Algorithm 8). Each entry in this vector has a value in $\{0, 1, 2\}$: node n chooses the s vector positions with the highest values, breaking ties randomly, and creates a fresh profile for item R by setting the corresponding bits to 1 and the remaining ones to 0. Finally, when n generates the profile for a new item, (line 16 in Algorithm 8), it simply sets to 1 the values of s bits from those that are set in its compact profile. This update process ensures that each item profile always contains s bits with value 1.

Filter Profile. Nodes compute their filter profiles whenever they need to exchange clustering information with other nodes (line 3 in Algorithm 9). Unlike the other profiles associated with nodes, this profile consists of a vector of integer values and does not represent the interests of a user. Rather it captures the interests of the community of users that have liked similar items. A node computes the value at each position in its filter profile by summing the values of the bits in the corresponding position in the profiles of the items it liked (line 8 in Algorithm 9) in the latest $time\ window$. This causes the filter profile to record the popularity of each bit within a community of nodes that liked similar items.

Obfuscated Profiles. As shown in Figure 8.1, a node computes its obfuscated profile whenever it needs to exchange it with other nodes as part of the clustering protocol. As shown in Figure 8.1, it achieves this by filtering the contents of its compact profile using its filter profile: this yields a bit vector that captures the most popular bits in the node's community and thus hides its most specific and unique tastes. The fine-grained information contained in the node's private and compact profiles remains instead secret throughout the system's operation.

As shown on line 2 and line 3 of Algorithm 9, a node n computes its obfuscated profile by first generating its compact and filter profiles as described above. Then it selects the s positions that have the highest values in the filter profile, breaking ties randomly, and sets the corresponding bits in the obfuscated profile to the values they have in its compact profile. It then sets all the remaining bits in the obfuscated profile to 0.

The resulting profile has s bits (set at 0 or 1) that reflect the node's compact profile and provide a coarse-grained digest of user interests. Through the value of s, the system designer can control the amount of information that can filter from the compact to the obfuscated profile, and can therefore tune the trade-off between privacy and recommendation quality. It is important to note that the positions of the bits whose value is 1 in the obfuscated profile depend on the filter profile and thus do not suffice to identify the item vectors that contributed to the corresponding compact profile. This prevents isolated attackers from precisely understanding which news items the node liked as shown in Section 8.5.5.

8.3 Randomized Dissemination

An attacker can discover the opinions of a user by observing the items she forwards (Section 8.1). We address this vulnerability through our second contribution: a differentially-private randomized dissemination protocol.

The key idea of our protocol is to randomize the forwarding decision: a node that likes an item drops it with probability pf, while a node that does not like it forwards it with the same pf. This prevents an attacker from acquiring

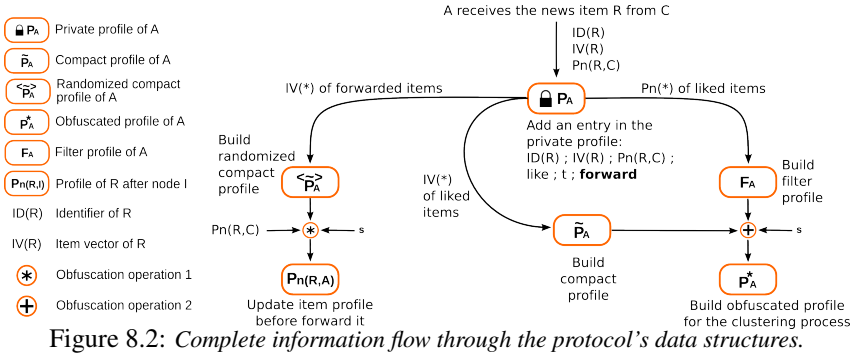


Figure 8.2: *Complete information flow through the protocol's data structures.*

certainties about a user's interests by observing which items she forwards. However, the attacker could still learn something from the content of the associated *item profiles* (modification of the item profile only when the user likes it). To ensure that the whole dissemination protocol does not expose any non-differentially-private information, we therefore randomize not only forwarding actions, but also the *item profiles* associated with forwarded items. This requires us to modify the protocol described in Section 8.2 as follows.

First, we introduce a new field in the *private profile*: the *randomized decision*. In addition to record whether the node liked or disliked an item, we use this new field to store the corresponding forwarding decision taken as a result of the randomization process (1 for forward and 0 for drop).

We then introduce a new *randomized compact profile* (as shown in Figure 8.2). The node fills this profile analogously to the compact profile but it uses the *randomized decision* instead of its actual opinion on the item. The node iterates through all the items for which the randomized decision is 1 and integrates their signatures into the *randomized compact profile* using the same operations described for the non-randomized one.

Finally, the node updates the *item profile* of an item when it decides to forward it as a result of randomization, regardless of whether it likes it or not. Moreover, the node performs this update as described in Section 8.2.2 except that the node uses its *randomized compact profile* instead of its *compact profile*.

Nodes still use their non-randomized *compact profile* when choosing their neighbors. In this case, they compare their *compact profile* with the *obfuscated profiles* of candidate neighbors. However, the above modifications guarantee that the actual content of the *compact profile* never leaks during dissemination. This guarantees that our dissemination protocol is *differentially private* [91].

A randomized algorithm $\mathcal{A}$ is ϵ-differentially private if it produces approximately the same output when applied to two neighboring datasets (*i.e.* which differ on a single element). In the context of dissemination, the datasets that need to be randomized are vectors of user opinions. Given two neighboring vectors of opinions (*i.e.* differing on a single opinion) $o1 \in \mathcal{D}^n$ and $o2 \in \mathcal{D}^n$, we define differential privacy as follows.

Differential privacy [115] A randomized function $\mathcal{F} : \mathcal{D}^n \rightarrow \mathcal{D}^n$ is ϵ-differentially private, if for any pair of neighboring opinion vectors $\mathbf{o1}, \mathbf{o2} \in \mathcal{D}^n$ and for all $\mathbf{t} \in \mathcal{D}^n$:

$$\Pr[\mathcal{F}(\mathbf{o1}) = \mathbf{t}] \leq e^{\epsilon} \cdot \Pr[\mathcal{F}(\mathbf{o2}) = \mathbf{t}]$$

This probability is taken over the randomness of $\mathcal{F}$, while e is the base of the natural logarithm.

In the case of our algorithm, we toss a coin each time the user expresses her opinion about an item in order to decide whether the item should be forwarded. This scheme is known as randomized response [177]: instead of randomizing the output of a function f, we randomize each of its inputs independently. Because these inputs as well as the output values are binary $\in \{0, 1\}$, we can rewrite the above equation as follows.

$$\Pr[f(o) = b] \leq e^{\epsilon} \cdot \Pr[f(1 - o) = b]$$

Our randomization function f flips the opinion o and produces the output $1 - o$ with probability pf. In order to achieve ϵ-differential privacy the value of pf must be such that:

$$1/(e^{\epsilon}+1) \leq pf \leq 1/2$$

For space reasons, we omit the details of the reasoning leading to this result, as well as the proof of the equivalence between randomized response and Definition 8.3. Nonetheless they are similar to those in [58].

This algorithm bounds the amount of information an observer gets when receiving an item from a user. Instead of knowing with certainty that the user liked the item, the observer knows that the user liked it with probability $1 - pf$. However, this does not make our solution fully differentially private, but only the dissemination component. In addition, it can only ensures ϵ-differential privacy when a user expresses her opinion about an item she received, not when she generates a new one. In the latter case, the user always forwards the item.

8.4 Experimental setup

We implemented and extensively evaluated our approach on the survey dataset presented in Section 6.4.2. We compare our solution with two baselines: a cleartext solution with no privacy mechanism, where profiles are exchanged in clear, and a solution that applies a differentially private mechanism both when generating the profiles that users exchange and upon dissemination. We refer to our solution as OPRD (Obfuscated Profile and Randomized Dissemination) in the following.

8.4.1 Alternatives

We compare our approach with the two following alternatives.

Cleartext profile (CT). This baseline approach implements the decentralized CF solution presented in Chapter 7 where user and item profiles are exchanged in clear during the clustering and dissemination processes. This solution does not provide any privacy mechanism.

Differentially private approach (2-DP). This alternative, denoted by 2-DP in the following, applies randomization both when generating user profiles and during dissemination. Every time a user expresses an opinion about an item, the algorithm inverses it with probability pd: this results in a differentially private clustering protocol and a differentially private dissemination protocol. The latter is similar to our randomized dissemination. However, unlike our solution, 2-DP also applies randomness when generating user profiles. When a user dislikes an item, 2-DP considers this item as liked with a probability pd, thus integrating it in the profile of the user and disseminating it to her neighbors. Conversely, when a user likes an item, 2-DP considers it as disliked with probability pd. In this case, it silently drops it without including it in the user's profile.

2-DP builds user profiles that are structurally similar to our compact profiles. However, they gather the item vectors of the items identified as liked after the randomization of user opinions. This extends the privacy guarantee associated with our dissemination protocol to the profiles of users. This represents a contribution in its own right. For space reasons, we do not include the associated proof. However, it follows a similar intuition than the one presented in Section 8.3.

As user profiles change over time and are impacted by the dissemination of items, applying a randomization function on cleartext profiles as in [58] is not enough. Iteratively probing the profiles of a user and analyzing the dissemination process could be enough to weaken the privacy guarantee. Instead, 2-DP does not randomize profiles, but it randomizes the opinion of a user on the items she is exposed to. Moreover, it does so independently of the user's opinion on other items.

2-DP uses the output of its randomization function to build user profiles and drive the dissemination. In particular, users use the resulting randomized profiles to compute their clustering views. We show in Section 8.5.4 that this introduces a weakness in the context of the decentralized CF scheme considered in these chapters. Moreover, section 8.5.6 shows that 2-DP remains more vulnerable to censorship attacks than our solution.

8.4.2 Evaluation metrics

We evaluate recommendation by considering recall, precision, and f-score as defined in Section 6.4.1. In addition, we define additional metrics for overhead and privacy.

Overhead. We measure overhead in terms of number of messages and bandwidth as described in Section 6.4.1. A key parameter that determines network traffic is the $fanout$ of the dissemination protocol, $i.e.$ the number of neighbors from the interest-based overlay to which nodes forward each item.

Privacy. We define privacy as the ability of a system to hide the profile of a user from other users. We measure it by means of two metrics. The first evaluates to what extent the obfuscated profile is close to the real one by measuring the similarity between the two. We consider the Jaccard index [174] to measure the similarity between a compact profile and the corresponding obfuscated one. The second measures the fraction of items present in a compact profile out of those that can be predicted by analyzing the presence of item vectors in the corresponding obfuscated profile. As item vectors are public, a malicious user can leverage them to guess the contents of the obfuscated profiles of other users, thereby inferring their interests.

8.5 Performance evaluation

In this section, we evaluate the ability of our solution to achieve efficient information dissemination while protecting the profiles of its users. First, we show that compacting user profiles, filtering sensitive information, and randomizing dissemination do not significantly affect the accuracy of dissemination when compared to CT, yielding slightly better results than 2-DP. Then we analyze the trade-off between accuracy and privacy and show the clear advantage of our solution in protecting user profiles in the context of a censorship attack. Finally, we show the benefits of our solution in term of network cost. We conducted an extensive evaluation through simulations, and through a real implementation deployed on PlanelLab. In both cases, we randomly select the source of each item among all users. We refer to our solution as OPRD (Obfuscation Profile and Randomized Dissemination) in the following.

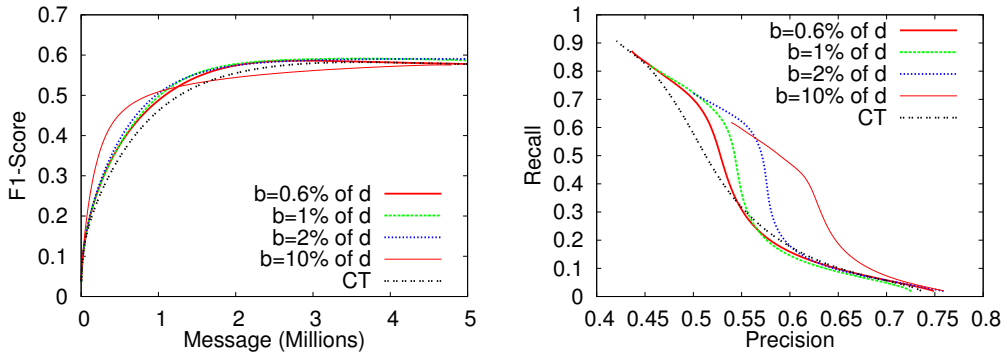
8.5.1 Compacting profiles

As explained in Section 8.2.2, our solution associates each item with a (sparse) item vector containing b 1's out of d possible positions. When a user likes an item, we add the corresponding item vector to her compact profile by performing a bitwise OR with the current profile. The ratio between b and d affects the probability of having two items sharing bits at 1 in their vectors, which in turn affects the accuracy of the similarity computation between users. Figure 8.3 evaluates its effect on performance.

Figure 8.3a shows the values of the F1-Score depending on network traffic for various values of the b-to-d ratio. The points in each curve correspond to a range of fanout values, the fanout being the number of neighbors to which a user forwards an item she likes: the larger the fanout the higher the load on the network. Figure 8.3b shows instead the corresponding precision-recall curve. Again, each curve reflects a range of fanout values: the larger the fanout, the higher the recall, and the lower the precision.

Interestingly, the larger the b-to-d ratio, the bigger the difference between our solution and CT. With a low b-to-d ratio, it is unlikely for any two item vectors to contain common bits at 1. As a result, the performance of our solution closely mimics that of CT. When the b-to-d ratio increases, the number of collisions between item vectors—cases in which two distinct item vectors have common bits at 1—also increases. This has two interesting effects on performance.

The first is that the F1-Score increases faster with the fanout and thus with the number of messages: the $b = 10\%$ curve climbs to an F1-Score of 0.4 with less than $400k$ messages. The curve on Figure 8.3b shows that this results from a higher recall for corresponding precision values (bump in the $b = 10\%$ curve). The high probability of collisions between item vectors results in some user profiles being similar even though they do not contain many common items.



(a) F1-Score vs messages

(b) Precision-recall curve

Figure 8.3: *Impact of compacting the profiles (various b-to-d ratios)*

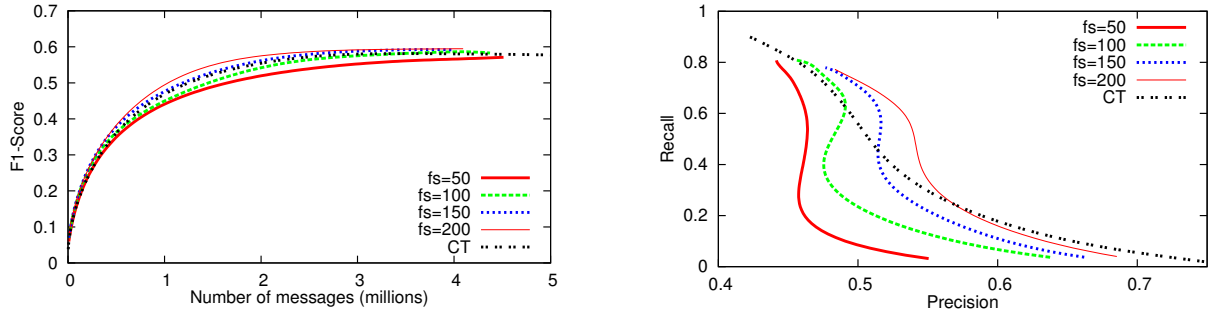
This leads to a topology in which users are less clearly clustered, and in which the items can be disseminated more easily, which explains the high recall value.

The second effect is that the maximum F1-Score attained by the protocol with a large b-to-d ratio (to the right of Figure 8.3a) stabilizes at lower values. Figure 8.3b clarifies that this results from a lower maximum recall, as indicated by the left endpoints of the curves corresponding to high values of b. The artificial similarities caused by a large b—advantageous with small fanout values (small number of messages)—also create false clusters that ultimately inhibit the dissemination of items to large populations of users. This effect is even more prominent with values of b that set a vast majority of the bits in compact profiles to 1 (not shown in the plot).

In the following, we set d to 500 and b to 5 for our evaluations. The values assigned to b and d should be computed depending on the expected number of items per user profile. The rationale for choosing these value is similar to those that relate the number of hash functions and the size of a bloom filter [64].

8.5.2 Filtering sensitive information

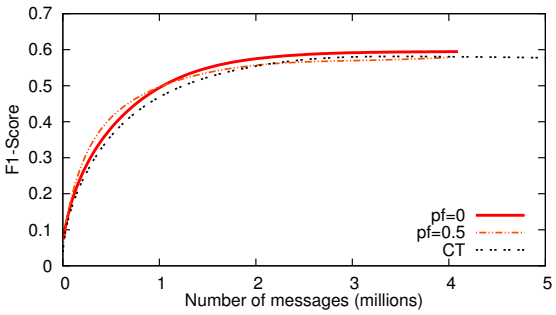
In our solution, the size of the filter defines how much information from the compact profile appears in the obfuscated profile. The larger the filter, the more the revealed information. Figure 8.4a depicts the F1-Score as a function of the number of messages. The performance increases with the size of the filter. Figure 8.4b shows that this variation comes from the fact that precision strongly decreases when the filter size decreases. The important aspect is that both plots highlight that a filter of 200 bits (*e.g.* 40% of the compact profile) achieves performance values similar to those of a system using full profiles.



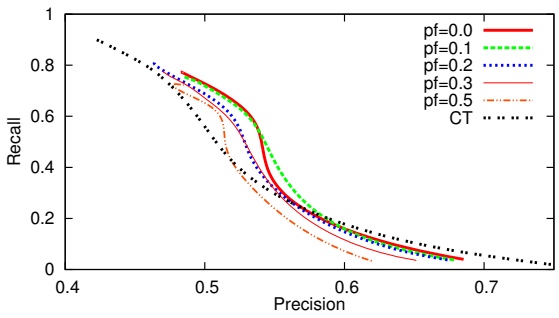
(a) F1-Score vs messages

(b) Precision-recall curve

Figure 8.4: *Impact of filtering sensitive information (various filter sizes, fs)*



(a) F1-Score vs messages for various pf

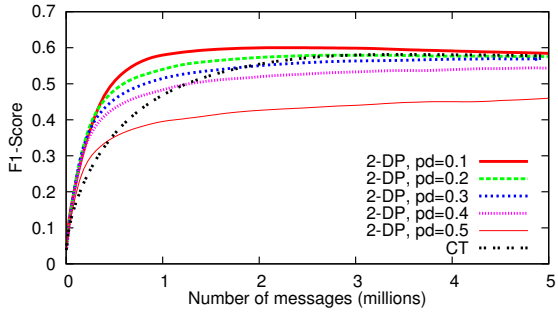


(b) Precision-recall curve for various pf

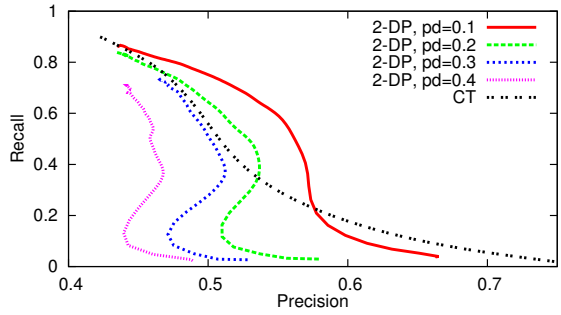
Figure 8.5: *Impact of obfuscating profiles and randomizing dissemination ($fs = 200$)*

8.5.3 Randomizing the dissemination

We now evaluate the impact of randomizing the dissemination process in addition to the obfuscation protocol evaluated above (the previous results were obtained without randomization). Figure 8.5a shows the F1-Score for our solution using a filter size of 200 and several values for pf. Performance decreases slightly as we increase the amount of randomness (for clarity, we only show $pf = 0$ and $pf = 0.5$, the other curves being in between). Figure 8.5b shows that increasing pf results mostly in a decrease in precision.



(a) F1-Score vs messages for various pd



(b) Precision-recall curve for various pd

Figure 8.6: *Impact of the randomization for 2-DP*

8.5.4 Evaluating 2-DP

In this section, we evaluate the 2-DP alternative defined in Section 8.4.1. 2-DP reverses the opinions of users with a probability, pd, that affects both the construction of user profiles and the dissemination process. This differs from our solution in which only the dissemination is randomized.

Figure 8.6a shows the F1-Score of 2-DP versus network traffic for various values of pd. Performance strongly increases at low fanout values for $dp = 0.1$, but decreases for larger values. A small amount of randomness proves beneficial and allows the protocol to disseminate items more effectively with a low fanout. This effect, however, disappears when the number of messages increases at high fanouts. Too much randomness, on the other hand, causes a drastic decrease in the F1-Score. Figure 8.6b shows that randomness induces an increase in recall with respect to CT and a decrease in precision. The former dominates with low values of pd while the latter dominates for high values.

Figure 8.7 compares the F1-Score of OPRD using a filter of size of 200 and a pf value of 0.3, with that of CT and 2-DP using a pd of 0.3. We observe that above $2M$ messages, our solution provides slightly better F1-Score values

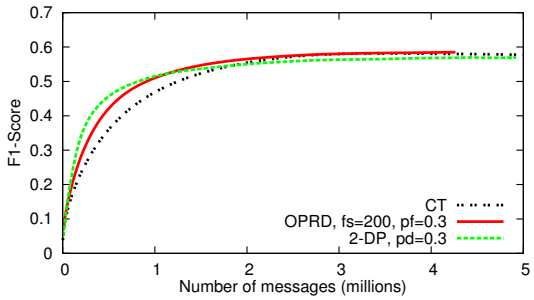
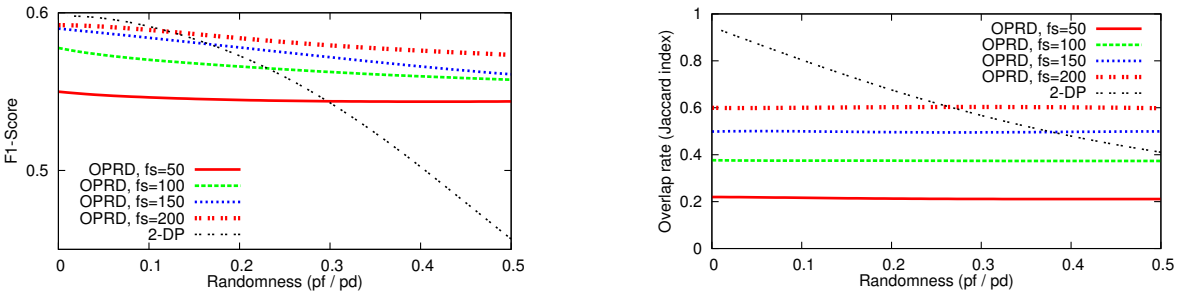


Figure 8.7: *OPRD vs 2-DP: F1-Score vs number of messages*



(a) Accuracy: OPRD vs 2-DP

(b) Public and real profiles overlap

Figure 8.8: *Randomness vs performance and level of privacy*

than 2-DP. Overall, however, the best performances of the two approaches are comparable. In the following, we show that this is not the case for their ability to protect user profiles.

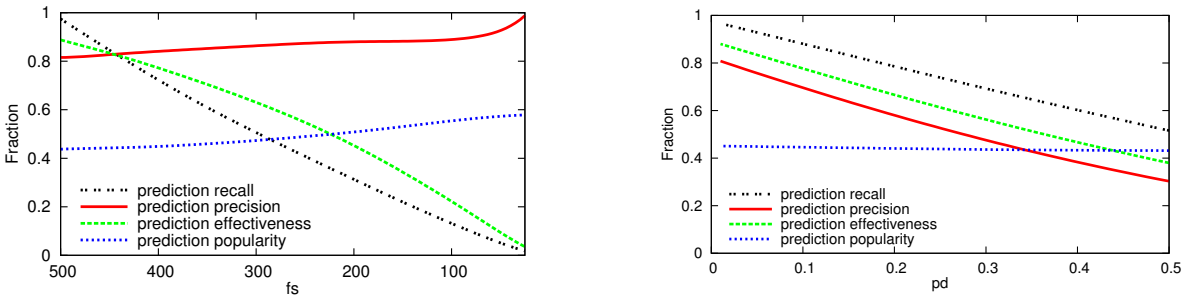
8.5.5 Privacy versus accuracy

We evaluate the trade-off between privacy, measured as the ability to conceal the exact profiles of users, and accuracy for both OPRD and 2-DP. OPRD controls this trade-off with two parameters: the size of the filter, and the probability pf. 2-DP controls this trade-off by tuning the probability pd to switch the opinion of the user, impacting both profile generation and the dissemination process.

Figure 8.8a compares their recommendation performance by measuring the F1-Score values for various filter sizes. The x-axis represents the evolution of the probabilities pf, for our solution, and pd, for 2-DP. We show that the F1-Score of 2-DP decreases faster than ours. The F1-Score of 2-DP with a pd of at least 0.2 is smaller than that of our solution with a filter size greater than 100. In addition, revealing the most popular 10% of the compact profile ($fs = 50$) yields similar performance as 2-DP with $pd \geq 0.3$.

Figure 8.8b measures the level of privacy as the overlap rate (computed with the Jaccard index) between the compact profile and the obfuscated profile: lower overlap rate implies more privacy. As our randomized dissemination protocol hardly impacts the obfuscated profile, our results are almost independent of pf. 2-DP sees instead its similarity decrease with increasing pd. With $pd = 0.3$, 2-DP yields an overlap rate of about 0.55 with an F1-Score (from Figure 8.8a) of 0.55. Our approach, on the other hand yields the same overlap rate with a filter size between $150 < fs < 200$, which corresponds to an F1-Score value of about 0.57.

Figure 8.9, instead, assesses privacy by measuring if the items in a user's real profile can be predicted by an attacker that analyzes the user's public profile. Note that in 2-DP, the real profile is the one that would exist without random perturbations. We evaluate this aspect by measuring the recall and the precision of predictions. Prediction recall measures the fraction of correctly predicted items out of those in the compact profile. Prediction precision measures the



(a) Prediction with OPRD, $pf = 0.2$

(b) Prediction with 2-DP

Figure 8.9: *Profile prediction*

fraction of correct predictions out of all the prediction attempts. For our solution, in Figure 8.9a, we use a $pf = 0.2$ to control the randomized dissemination, and vary the filter size. For 2-DP (Figure 8.9b), we instead vary pd.

The plots show that while our approach is subject to fairly precise predictions, these cover only a small fraction of the compact profile with reasonable values of fs. With $fs = 200$, the prediction recall is of about 30% In contrast, 2-DP exposes a higher number of items from the compact profile. With $pd = 0.2$ the prediction recall is 0.8 with a prediction precision of 0.6. The curves for prediction effectiveness, computed as the harmonic mean of recall and precision, further highlight our approach's ability to strike an advantageous balance between privacy and recommendation performance.

The two plots also show the average popularity of the predicted items. We observe that when the filter size decreases, the correctly predicted items are among the most popular ones, which are arguably the least private.

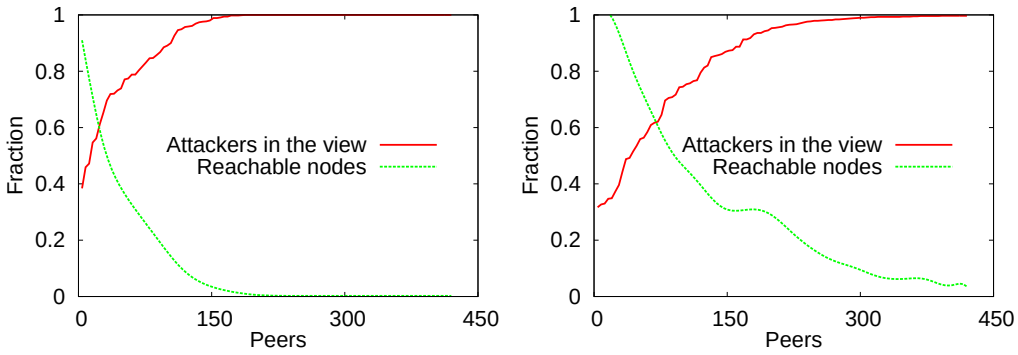
Finally, we also observe that the compact profile itself provides a small protection to the prediction of items due to its inherent collision rate. With a filter of size 500 ($e.g.$ with no difference between the compact and the public profile), the error rate is equal to 0.15.

8.5.6 Resilience to a censorship attack

We illustrate the resilience of our obfuscation protocol against censorship by implementing a simple eclipse attack [135]. A coalition of censors mirrors the (obfuscated) profile of a target node in order to populate its clustering view. This is turn isolates it from the remaining nodes since its only neighbors are all censors. If the user profiles are exposed in clear, the profile of the censors matches exactly that of the target node: this gives censors a very high probability to enter its view. Once the censors have fully populated the target node's view, they simply intercept all the messages sent by the target node, preventing their dissemination. We evaluate the efficiency of this attack with two metrics: the poisoning rate of the target's clustering view by attackers; and the fraction of honest nodes ($e.g.$ not censors) reachable by the target when it sends an item.

We ran this attack for each user in the dataset. The x-axis represents the users in the experiment sorted by their sensitivity to the attack. Figure 8.10a and Figure 8.10b depict the results obtained with a cluster size of 50, and 50 censors (we observe similar results independently of the cluster size). In addition, this experiment uses a filter of 125 and $pf = 0.2$ for our solution, and $pd = 0.2$ for 2-DP. We can clearly see that 2-DP is not effective in preventing censorship attacks: only 150 nodes have a poisoning rate lower than 1. This is because 2-DP computes similarities using the randomized compact profile, which it also shares with other users. Therefore 2-DP exhibits exactly the same vulnerability as CT. The censors can trivially match the profile of the target node.

Our approach is more resilient to this censorship attack. It is difficult for censors to intercept all messages sent by the target and only a third of the nodes have a fully poisoned clustering view. The obfuscated profile only reveals the least sensitive information to other nodes: censors only mirror a coarse-grained sub part of the target node's profile. Consequently, their profiles are more likely to resemble those of users with correlated interests than to match the target profile. Figure 8.8b confirms this observation by showing the overlap between obfuscated and compact profiles. The resilience of OPRD is driven by the size of the obfuscation filter, the smaller the filter, the more resilient the protocol.



(a) 2-DP, $pd = 0.2$ (b) OPRD, $fs = 125$, $pf = 0.2$

Figure 8.10: *Resilience to censorship*

Fanout	2	4	6	8	10	15	20
CT	1.8	3.0	4.4	6.5	8.2	12	14
OPRD	0.8	1.1	1.5	1.7	2.7	2.8	4.1

Table 8.1: *Bandwidth usage in kbps per node in PlanetLab*

8.5.7 Bandwidth consumption

We also conducted experiments using our prototype with 215 users running on approximately 110 PlanetLab nodes in order to evaluate the reduction of network cost resulting from the compactness of our profiles. The results in terms of F1-Score, recall, and precision closely mimic those obtained with our simulations and are therefore omitted. Table 8.1 shows the bandwidth cost of our protocols in terms of bandwidth: our obfuscation protocol is effective in reducing the bandwidth consumption of decentralized collaborative filtering. The cost associated with our obfuscated solution is about one third of that of the solution based on cleartext profiles.

8.6 Concluding Remarks

This chapter presented a mechanism to preserve privacy in the context of decentralized collaborative filtering. Our proposal relies on two components: (i) an original obfuscation scheme revealing only the least sensitive information in the profiles of users, and (ii) a randomization-based dissemination protocol ensuring differential privacy during the dissemination process. We showed the viability of our mechanism by comparing it with both a non-private as well as a fully (differentially) private alternative.

Chapter 9

Hide & Share: Landmark-based Similarity for Private KNN Computation

In this chapter we focus on one of the operations carried out by a decentralized user-based recommender like WHATSUP: k-nearest-neighbor computation. As highlighted in Chapter 8, KNN computation presents an important privacy risk in the context of decentralized collaborative filtering in that users need to exchange their profiles with one another in order to compute their similarities. In this chapter, we address this vulnerability by proposing *H&S* (Hide & Share), a landmark-based similarity mechanism for decentralized KNN computation. Landmarks allow users (and the associated peers) to estimate how close they lay to one another without disclosing their individual profiles. The work in this chapter lies at the core of the PhD thesis of Antoine Rault, whom I co-advised with Anne-Marie Kermarrec.

The content of this chapter is an adapted excerpt from the following paper:
Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, Antoine Rault, François Taïani, Jingjing Wang: Hide & Share: Landmark-Based Similarity for Private KNN Computation. DSN 2015: 263-274

9.1 Overview

H&S consists of a similarity computation mechanism that makes it possible to compute the KNN graph without requiring users to share their profile information with anyone else. *H&S* relies on a simple observation: user-centric KNN applications such as recommendations do not require perfect knowledge. We illustrate this observation in Figure 9.1. The plot depicts recommendation quality (quality increases towards the top and the right) with varying level of randomness injected into user profiles. Randomness levels of up to 75% do not significantly hamper recommendation quality. Based on this observation *H&S* trades-off precision in the computation of similarity for privacy. This allows it to gain significant protection in terms of privacy with a minimal impact on applications like recommendation. This makes *H&S* a perfect fit for decentralized CF systems like the one we discussed in the previous chapters.

H&S's key contributions lie in a novel *landmark-based* approximation technique as well as in a fair landmark-generation protocol. These contributions allow two users to indirectly measure their similarity by comparing their own profiles with a set of randomly generated profiles (the landmarks). The similarity between a user's profile and a landmark acts as a coordinate in a coordinate system. Users then exchange vectors of landmark-based coordinates and compute an approximation of their actual similarity. This preserves user privacy as users do not exchange their full profiles and landmark coordinates only reveal a limited amount of information about a user.

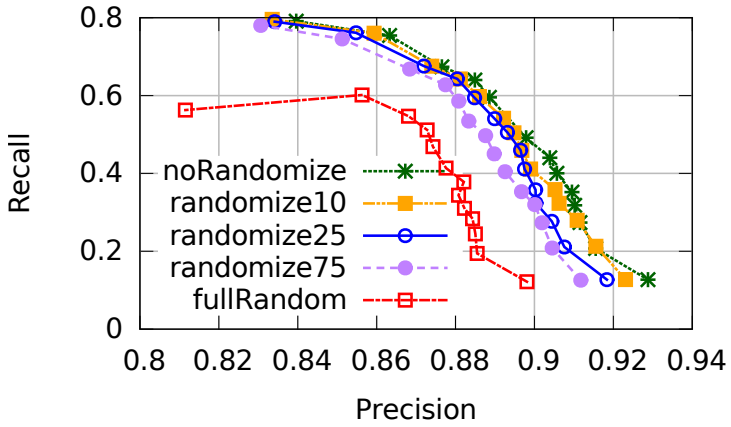


Figure 9.1: Recommendation quality with different levels of randomization of user profiles. This quality is not significantly hampered by levels of randomization of up to 75%.

9.2 System Model

We present *H&S* in the context of a slightly different recommender system than the one we considered in the previous chapters. We consider a decentralized system with binary ratings, but we assume a cosine similarity (Equation (6.1)) metric as a starting point rather than the WUP metric defined in Section 7.2.

In addition, WHATSUP operates in a push-based model: peers forward items to their neighbors as part of the recommendation process. Here, we consider instead a pull-based decentralized recommender: peers use their neighbors as sources of recommendations as proposed by [56]. The KNN process is oblivious to this difference. As a result, even if we describe *H&S* in the context of a pull-based recommender, our solution can easily be applied in the context of push-based recommenders based on cosine similarity.

9.2.1 Adversary Model

W consider a *curious adversary* model that can only take a limited set of active actions to reach her goal, and can otherwise passively gather information. The adversary aims to discover the profile of a chosen user (target) by a profile reconstruction attack, using information obtained during similarity computation. The adversary only controls one peer, i.e we assume there is no collusion between adversaries, and our adversary cannot forge peer identities (no sybil capacity). The adversary also has no *a priori* knowledge regarding her target's interests. The active actions the adversary can take are: tapping unencrypted communications; attempting to bias multi-party computations; computing her similarity with her target as many times as she want.

9.2.2 Problem Statement

As suggested above, computing similarity constitutes a major privacy risk. Before convergence, both the RPS and the clustering protocol (see Section 6.3) require peers to communicate with a large number of other peers, even with non similar ones. This means that a malicious non-similar peer can easily copy the profile of a target peer in order to forcibly enter its clustering view. This chapter addresses this privacy threat by presenting *H&S*, a novel similarity mechanism that does not require peers to exchange their profile information.

Thanks to *H&S*, peers can identify their KNN without having to disclose any personal details to other peers. Once they identified their KNN, they do share their profile information with neighbors that are sufficiently stable to compute recommendations. However, this does not constitute a significant privacy risk because peers identified as KNN already know that they have similar profiles. Learning the details of each other's profiles therefore does not add much to this

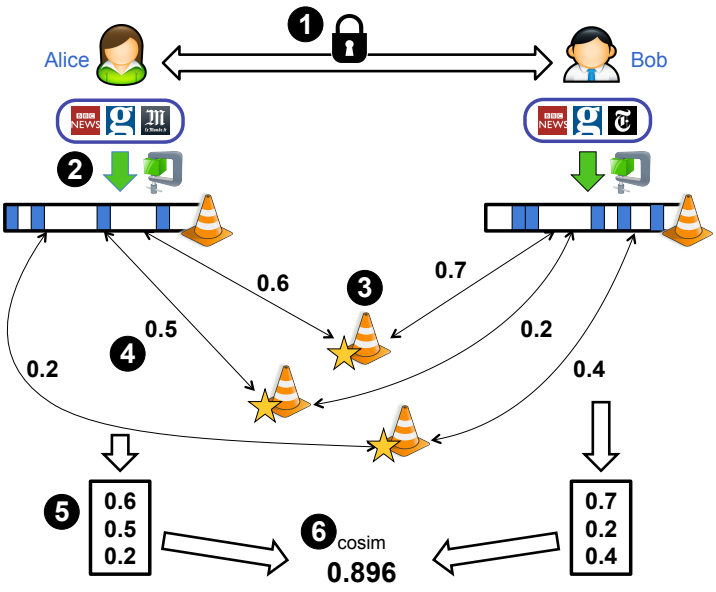


Figure 9.2: Overview of the *H&S* similarity computation mechanism

knowledge. Conversely, a malicious peer that wanted to become a neighbor of a target node would not be able to clone the corresponding profile without being already similar to the target peer.

9.3 The Hide & Share Landmark-based Similarity

H&S relies on a simple observation: good recommendations do not require perfect neighborhoods. *H&S* therefore relaxes the precision of similarity computation by exploiting randomly selected intermediate profiles (*landmarks*) with respect to which each peer positions itself. This allows peers to compute similarity scores they can exploit without exchanging clear-text profiles.

H&S landmarks take inspiration from reference points in geo-localization systems. For instance, two-dimensional geographic locations usually refer to *the Equator* and *the Greenwich meridian*: two landmarks that define their latitude and longitude. However, our landmarks also exhibit two important differences with respect to this geographic analogy. First our landmarks are not fixed and set for the whole system; rather, each pair of peers randomly generates its own set of landmarks. This prevents cross-pair comparisons. Second, we use far fewer landmarks than there are dimensions in our system. This prevents a precise reverse computation of each peer's clear-text coordinates (i.e. its profile) from its landmark coordinates. Thanks to these differences, users can safely exchange their landmarks because they do not characterize their interests in any specific topic.

Figure 9.2 presents an overview of the operation of *H&S* by means of an example. Alice and Bob need to compute their similarity with each other. In a traditional system like the one described in Section 6.3, Bob would send his profile to Alice and Alice would send hers to Bob. Each of them would then compute the similarity by applying Equation (6.1). With *H&S*, none of this happens. Rather, Alice and Bob follow these 6 steps. (1) They create a secure communication channel. (2) They derive a compact version (Bloom filter) of their profile. (3) They agree on a set of random landmarks. (4) They compute the similarity of their compact profile with each landmark. (5) They gather these similarity values in a similarity vector. (6) They exchange each other's similarity vector and compute their final similarity estimate. From a practical perspective, this translates into to two main components: a landmark generation mechanism, and a similarity estimation protocol. In the following we detail each of these two contributions.

```
1  session_key ← AK(key_p1, pub_key_p2);
2  secure_channel ← connect(p2, session_key);
3  if p2 is known then
4      s ← load_seed(p2);
5      if s is not older than th_L then
6          seed ← s;
7          goto 15;
8  for all i s.t. 0 ≤ i < 32
9      r ← rand_bit();
10     seed[i] ← coin_flip(r, secure_channel);
11 save_seed(p2, seed, timestamp(now));
12 prng ← init_prng(seed);
13 for all i s.t. 0 ≤ i < L
14     M_i ← generate_lm(prng);
15 for all i in 0 ≤ i < L
16     v_p1[i] ← cosine(c_p1, M_i);
17 send(v_p1, secure_channel);
18 v_p2 ← receive(secure_channel);
19 similarity ← cosine(v_p1, v_p2);
20 return similarity;
```

Algorithm 10: *H&S* landmark-based similarity computation protocol between peers $p1$ and $p2$, as executed by $p1$

9.3.1 Landmark Generation

H&S uses landmarks to estimate the similarity between two peers without requiring them to exchange their profiles with each other. To prevent adversaries from reconstructing profile information from these landmarks, the landmark generation mechanism must satisfy a set of requirements.

i *Computation confidentiality*: Only the two peers participating in the similarity computation may access the data they exchange. This includes landmark and similarity values.

ii *Independence of peer profiles*: Landmarks must be random and independent of the profiles of the peers that generate them.

iii *Fair landmark generation*: The choice of the landmarks must be fair. Neither of the two participating peers may bias the generated landmarks.

iv *Minimal information release*: An attacker should not be able to reconstruct a target profile by combining information from multiple landmark similarities, or by repeatedly computing its *H&S* similarity with the target.

In the following, we present our landmark generation mechanism by focusing on how it addresses each of these requirements. We detail the various steps in lines 1 through 14 of Algorithm 10.

9.3.1.1 Computation Confidentiality

Requirement (i) states that third-party peers should not be able to eavesdrop any communication between peers that are computing their similarity. To achieve this, *H&S* encrypts all the communication between two peers, including that relative to landmark generation. Specifically, each peer maintains a public/private key pair. Peers exchange their public keys with each other by attaching them to the information transferred through the RPS and clustering protocols, similar to what was is done in [4]. In addition, we assume that peers may verify the authenticity of a public key by means of a certification authority or a web of trust [107, 17].

Peers use their key pairs to establish a secure communication channel whenever they need to evaluate their similarity. To this end, they exploit an authenticated key agreement (AK) protocol [160] as shown in lines 1 and 2. A possible AK protocol consists of an authenticated variation of the elliptic curve Diffie-Hellman key agreement such as the one available in the NaCl cryptographic library [179].

9.3.1.2 Independence of peer profiles

Requirement (ii) states that landmarks consist of *randomly generated* profiles that are independent of the profiles or of the choices of participating peers. However, as we discussed in Section 9.2, profiles consist of lists of item-score pairs, where the items belong to an unbounded or at least very large universe. This would make it difficult, if not impossible to generate random landmarks. To circumvent this problem, *H&S* replaces traditional profiles with *compact profiles* (step 2 in Figure 9.2).

A *compact profile* consists of a Bloom filter [176] and contains only the items considered as liked by the corresponding peer. A Bloom filter provides a compact representation of a set in the form of an array of n bits. To add an item to the set, the bloom filter applies h hash functions to the item to obtain h bit positions in the array and sets these positions to 1. To query for the presence of an item, the filter uses the same hash functions and checks if all the bits at the h indexes have a value of 1.

Compact profiles carry slightly less information than full profiles. First, Bloom filters can return false positives even though they never return false negatives. Second, compact profiles cannot distinguish between disliked items and items to which the user has not been exposed. This does not constitute a problem: the like status of items proves sufficient to describe the interests of peers, and the effect of false positives may actually be beneficial in terms of privacy. Compact profiles also reduce Equation (6.1) to counting the number of common bits between the two bloom filters.

Given a user or peer, $p \in \{1, 2, \ldots, N\}$, we denote her compact profile as $\vec{c}_p \in \mathbb{Z}_2^n$. Lines 8 through 14 of Algorithm 10 show how peers use compact profiles to generate random landmarks. Let L be a system parameter specifying the number of landmarks to generate and let PRNG be a pseudo-random number generator whose code is available to all peers (for example MRG32k3a [162] or Mersenne Twister [163]). Two peers, say $p1$ and $p2$, may generate a set of landmarks by first generating a common random seed (lines 8 to 10 in Algorithm 10). Then, each of them saves this seed (line 11), along with a timestamp, and uses it to initialize the PRNG (line 12). Finally Each of the two peers independently uses the PRNG to generate the L landmarks: $\{M_i\}$ with $i \in \{0, 1, \ldots, L\}$ (lines 13-14). Each generated landmark consists of a vector of bits of the same size as a compact profile, with a few random bits (around 5%) set to 1, while other bits are set to 0. This proportion of set bits mimics that of compact profiles, which are usually sparse.

9.3.1.3 Fair Landmark generation

Requirement (iii) states that the choice of the landmarks must be fair. To achieve this, peers agree on their common seed using a bit-commitment scheme like Blum's coin-flipping protocol [173]. Blum's protocol operates as follows. Both $p1$ and $p2$ flip a coin. They set the output of the protocol to 1 if they obtain the same result, and to 0 otherwise. To exchange their coin-flip results without cheating, $p1$ and $p2$ employ a bit-commitment scheme. After flipping its coin, $p1$ sends $p2$ a commitment on its result ($f(concatenate(result, nonce))$). Then $p2$ reveals its result to $p1$, and $p1$ reveals its result to $p2$. $p2$ cannot cheat because it is the first to send its result. $p1$ cannot cheat because $p2$ can then check its result against the initial commitment.

Blum's protocol does not provide an unbiased coin, which is impossible in the two-party case [172], but a weaker fairness guarantee that suffices for our application. This guarantee holds as long as a malicious party does not abort the protocol before it ends. Since the two peers in our protocol use a secure channel, if $p2$ aborts, $p1$ can deduce that $p2$ is trying to bias the result.

9.3.1.4 Minimal information release

Requirement (iv) states that attackers should not be able to reconstruct a target profile by combining information from multiple landmarks or by repeatedly computing their similarity with the target. To satisfy the first part of this requirement, *H&S* similarity uses a small number of landmarks with respect to what would be required to reconstruct the original profile. In Section 9.4.4, we show that this does not significantly impact the ability to provide good recommendations.

To satisfy the second part of this requirement, $H\&S$ peers do not generate new landmarks each time they meet. Rather they only do so if their latest common set of landmarks is older than a threshold, th_L. To achieve this, they verify the timestamp associated with their latest saved common seed. If the timestamp is newer than the threshold, then they reuse the seed, otherwise they generate a new random seed.

9.3.2 Similarity approximation

We conclude the description of our protocol by presenting how $H\&S$ approximates the similarity between two peers using its randomly generated landmarks. Let $\{M_1, \ldots, M_L\}$ be a set of common landmarks known to peers $p1$ and $p2$. First, each of the two peers independently computes its similarity with each of these landmarks (step 4 in Figure 9.2 and lines 15-16 in Algorithm 10). This consists in applying Equation (6.1) to its own profile and each of the landmarks. Both $p1$ and $p2$ then store the results of these computations in a similarity vector (respectively $\vec{v}_{p1}$ and $\vec{v}_{p2}$) as shown in step 5 in Figure 9.2 and on line 16 in Algorithm 10. Second, $p1$ and $p2$ exchange their similarity vectors with each other. This consists of lines 17 and 18 in Algorithm 10. Finally (step 6 and line 19), $p1$ and $p2$ compute their $H\&S$ similarity by applying Equation (6.1) to their own similarity vector and to the one they have received (note that $cos(\vec{A}, \vec{B}) = cos(\vec{B}, \vec{A})$).

9.4 Evaluation

We evaluate $H\&S$ by applying it in the context of a gossip-based decentralized recommendation system. Using publicly available traces, we evaluate the quality of its recommendations, its ability to protect privacy, and the overhead it implies.

9.4.1 Methodology

9.4.1.1 Simulator

We use our own simulator written in Java. The simulator takes as input a trace from a recommendation system, consisting of user-item matrix of ratings, split into a training set and a test set. The training set (80% of the ratings) allows peer neighborhoods to converge, while the test set (the remaining 20%) provides the ground truth to evaluate the relevance of recommendations. The simulator operates in two steps. First it uses the training set to simulate the convergence of the clustered overlay, then it generates r recommendations for each peer using the converged overlay and compares the results with the ratings in the test set.

9.4.1.2 Datasets

We run our experiments using three of the datasets described in Section 6.4.2: ML-100k, ML-1M, and Jester-1-1.

9.4.1.3 Evaluation metrics

We evaluate recommendation quality in terms of precision and recall as defined in Section 6.4.1. Peers recommend the r most liked item in their neighborhoods, not including those they have already rated. We check whether a recommended item is liked by looking at the rating given to this item by the recipient in the test set.

To evaluate $H\&S$'s ability to protect privacy we consider both neighborhood quality, and a privacy metric. Neighborhood quality evaluates how much the neighborhoods provided by $H\&S$ resemble the optimal neighborhoods, that is those obtained with the standard cosine similarity metric. Specifically, for each user we measure the average of the cosine similarities with all the peers in its $H\&S$ view, and we normalize it by the average cosine similarity with the peers in the optimal neighborhood obtained using an exhaustive search procedure. Let u be a user with full profile, $profile_u$, and let n_u and N_u be respectively u's $H\&S$ neighborhood and u's optimal neighborhood. Then we compute u's neighborhood quality as follows.

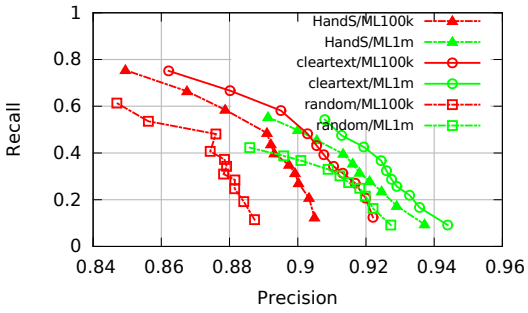


Figure 9.3: Recommendation quality expressed as precision and recall with a varying number of recommendations r, using the MovieLens datasets.

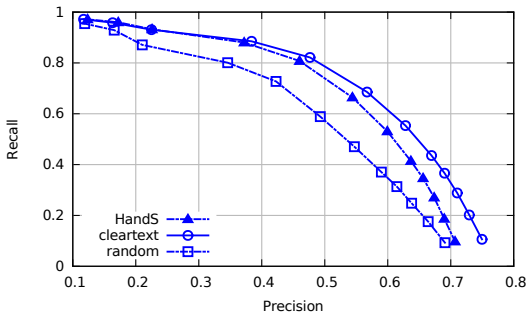


Figure 9.4: Recommendation quality expressed as precision and recall with a varying number of recommendations r, using the Jester dataset.

$$quality(u) = \frac{\frac{1}{k}\sum_{p \in n_u} cos(profile_u, profile_p)}{\frac{1}{k}\sum_{p \in N_u} cos(profile_u, profile_p)}$$

Neighborhood quality provides a first indication of privacy: lower quality implying better privacy. To obtain a more precise privacy evaluation, we also define *set score*. This metric measures the success rate of the adversary in the context of a profile reconstruction attack. Let G be the set of items that the adversary guesses as liked by the target, and let P be the set of items actually liked by the target. We then define set score as follows, with $\triangle$ as the symmetric difference of two sets.

$$setScore(G, P) = \frac{|G \triangle P| - |G \cap P|}{|G \cup P|}$$

A set score of 1 (adversary's failure) indicates that all the guessed items are wrong (highest privacy), while a set score of -1 (adversary's success) indicates the adversary guessed exactly the target's liked items (no privacy).

Finally, we evaluate overhead by comparing the bandwidth consumption and the storage space required by a *H&S*-based recommendation system with those required by a standard implementation like that of the reference model described in Section 9.2.

9.4.1.4 Default parameters

The subsequent results correspond to simulations using neighborhood and RPS view sizes of 10 peers. Compact profile sizes depend on the dataset used: 660 and 1473 bits for ML-100k and ML-1M respectively (roughly 40% of the number of items), and 99 bits for Jester. When the number of landmarks is not explicitly mentioned, *H&S* uses 50 landmarks. This represents a good trade-off between recommendation quality and privacy. For all the metrics except set score, we plot values averaged over all the peers.

9.4.2 Recommendation quality

Figures 9.3 and 9.4 show precision and recall values for several values of r. The former shows the results with the MovieLens datasets, and the latter shows the results with the Jester dataset. For each dataset, we compare the results of the *H&S*-based system (triangle-shaped) with a lower bound (square-shaped) and a cleartext baseline (circle-shaped). The lower bound consists of a CF system that uses completely random neighbors. The baseline consists of the reference model with full profiles in cleartext, as described in Section 9.2.

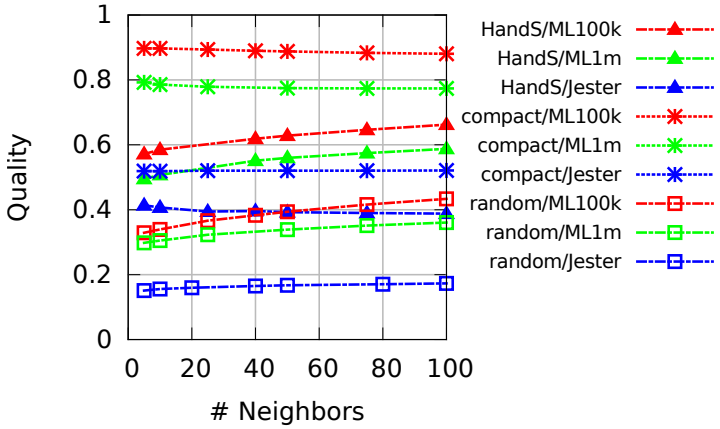


Figure 9.5: Effect of compact profiles and the *H&S* similarity on neighborhood quality. The *H&S* similarity is the main source of perturbation of neighborhood quality.

Figure 9.3 shows consistent results by the $H\&S$-based system across the two MovieLens datasets. $H\&S$ provides a reasonable quality of recommendations: it never suffers from a degradation of more than 50% with respect to the cleartext baseline. Moreover the higher the value of r, the closer the quality remains to that of the cleartext baseline.

Figure 9.4 shows a similar behavior of the $H\&S$-based system with the Jester datastet. Recall reaches almost a value of 1 because the dataset only contains 100 items. This characteristic is also the cause of the maximum precision values being lower than those of the MovieLens datasets. As the test set does not contain many items, we consider that a recommended item without rating in this set is disliked by the recipient, instead of ignoring it as done otherwise. Although this approach is pessimistic, it allows us to make a sufficient number of recommendations.

We showed that $H\&S$ preserves the quality of recommendation, being only slightly worse than the cleartext baseline. In the following, we show that it achieves this while protecting the privacy of users.

9.4.3 Neighborhood quality

In order to evaluate the extent to which neighborhoods are different from the optimal neighborhoods, we use the neighborhood quality measure as defined in Section 9.4.1.3.

Figure 9.5 shows the evolution of neighborhood quality with the size of neighborhoods. For each dataset, it compares the $H\&S$-based system (triangle-shaped) with a CF system using random neighbors as a lower bound (square-shaped) and a variant of our system model using compact profiles (star-shaped). Our reference model from Section 9.2 by definition achieves a neighborhood quality of 1 and compact profiles provide neighborhoods that are almost identical in the ML datasets. In the case of Jester, they lower neighborhood quality by 50% because the Jester dataset contains only a few items. This makes it more sensitive to the collisions in the Bloom filters.

$H\&S$ similarity has a more significant impact on neighborhood quality than compact profiles. Yet, $H\&S$'s neighborhood still retain their utility in terms of recommendation as we showed in Section 9.4.2. Because landmarks are randomly generated, some of them might be "far" from the two users comparing themselves, thus giving little information about the users' similarity. Moreover, a set of landmarks is not necessarily linearly independent. The lower quality of $H\&S$-generated neighborhoods is in fact an asset in terms of privacy. Because of this mix of neighbors with various levels of similarity, the adversary cannot infer her target's interests just by looking at her target's neighbors.

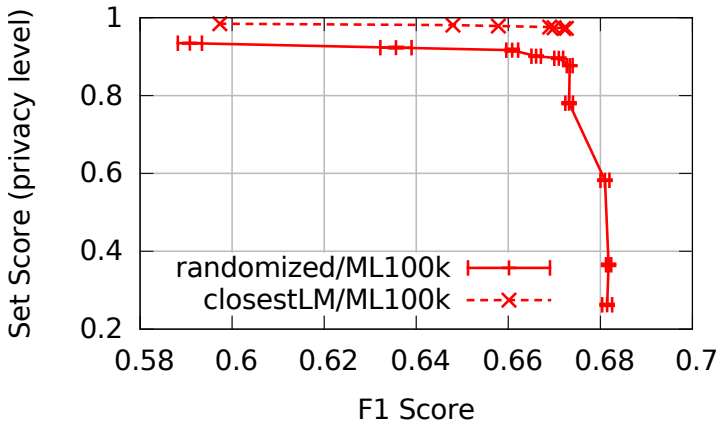


Figure 9.6: Trade-off between recommendation quality and privacy for the *H&S*-based system and a system with perturbation-based privacy.

9.4.4 Privacy

We evaluate the privacy offered by $H\&S$ by running a profile reconstruction attack against it. This attack consists in trying to discover the liked items in a targeted peer's profile using information obtained during similarity computation. We quantify the resilience of $H\&S$ to such attacks with the set score defined in Section 9.4.1.3.

The adversary makes her guess in two steps: *(1)* she tries to infer her target's compact profile, then *(2)* she tries to deduce the items forming this profile. We consider for *(1)* that the adversary uses the closest landmark to her target as her guessed profile. For *(2)*, we consider that the adversary knows all the items in the system, so she includes in her guessed set all the items matching the guessed profile.

We compare our $H\&S$-based system with a perturbation-based privacy technique. When using this technique, peers compute their similarity by the usual profile exchange, but they add random noise to their profile to protect their privacy. For the sake of comparison, peers implement this technique by using compact profiles and randomizing a certain percentage of bits in the profile.

Figure 9.6 compares our $H\&S$-based system and a recommendation system using randomized compact profiles, in terms of the trade-off between recommendation quality and privacy. We use set score for the latter and F1 score, the harmonic mean of precision and recall, for the former. We obtain different values of this trade-off by varying the number of landmarks from 2 to 100 for the $H\&S$ system, and by varying the number of randomized bits in profiles from 5% to 100% for the perturbation-based system. Set score values are averages over 100 different adversaries and 200 different targets, i.e. 20,000 different sets of landmarks. F1 score values correspond to $r = 30$ recommendations.

We observe that the $H\&S$-based system provides an excellent level of privacy in any case. It also provides a recommendation quality on par with the best values of the other system, starting from 25 landmarks. However, the increase in recommendation quality does not grow as fast as increase in the number of landmarks.

The recommendation system using randomized compact profiles preserves an almost optimal recommendation quality with up to 75% of randomized bits. Although it achieves reasonable privacy ($setScore = 0.8$ approximately) starting from 50% of randomized bits, it never reaches the privacy levels offered by the $H\&S$-based system.

With these basic strategies for the profile reconstruction attack, we showed that $H\&S$ provides improved privacy to users without sacrificing recommendation quality, and without obvious flaws.

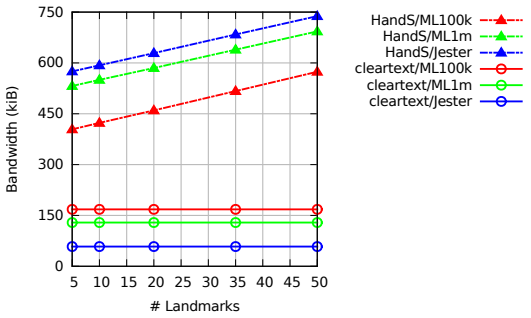


Figure 9.7: Average bandwidth consumption of a peer per gossip cycle. The *H&S*-based system consumes roughly twice to seven times more bandwidth than our system model with 5 to 50 landmarks.

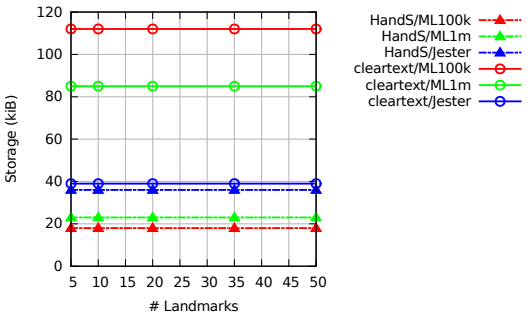


Figure 9.8: Average storage space needed for a peer. The *H&S*-based system needs less storage space because peers only store the seed used to generate landmarks.

9.4.5 Overhead

We evaluate the overhead caused by $H\&S$ to peers in terms of bandwidth consumption and storage space. Overall, $H\&S$ incurs the most part of its overhead when two peers compute their similarity for the first time because they have to generate a seed using the bit commitment scheme and store this seed. So we measure in our simulations the average number of similarity computations with new peers. The other parameters influencing $H\&S$'s overhead are the sizes of the RPS and neighborhood views, and the number of landmarks.

The main factors impacting bandwidth consumption are the exchange of coordinate vectors and the bit commitment scheme. The main factor impacting storage space is the need to store seeds.

Figure 9.7 compares the $H\&S$-based system (triangle-shaped) and the reference model (circle-shaped) in terms of the average bandwidth consumption of a peer per gossip cycle. Bandwidth consumption of the $H\&S$-based system increases linearly with the number of landmarks used. It consumes roughly twice to seven times more bandwidth than the reference model, but the absolute values remain reasonable (up to 700KiB per cycle). Moreover, it can probably be improved as a bit commitment protocol with $O(1)$ bits of communication per committed bit exists [168].

Figure 9.8 compares the $H\&S$-based system and the reference model in terms of the average storage space needed by a peer. The $H\&S$-based system needs less storage space because peers only store the seed used to generate landmarks instead of storing the profiles of peers in their neighborhood and RPS views as done by standard systems. Still, we observe that the required storage space is tiny compared to the storage capacity of modern devices (computers, smartphones, tablets, etc).

9.4.5.1 Computational overhead

We observe that the computational overhead of $H\&S$ is negligible from the point of view of peers. The most computationally intensive elements of the $H\&S$ similarity are *(1)* the authenticated key agreement (AK) protocol, *(2)* the generation of random bits (bit commitment scheme and mostly landmark generation), *(3)* the cosine similarity computations with landmarks.

(1) Executing cryptographic primitives incurs negligible cost on modern devices. It is similar to accessing a website with HTTPS: the end user does not perceive a difference between accesses over HTTP and HTTPS. *(2)* Efficient PRNGs such as Mersenne Twister can generate millions of bit/second on modern devices and $H\&S$ only needs a few thousands of bits to generate landmarks during a gossip cycle, which lasts several seconds at least. *(3)* Cosine similarity is cheap to compute on binary vectors such as landmarks and compact profiles. This confirms the applicability of our approach.

9.5 Privacy Guarantee

We now analyze $H\&S$ from an information theoretical viewpoint and evaluate the amount of information leaked during our landmark-based similarity computation. We carry out our analysis from the point of view of an attacking peer, a, that seeks to obtain information about another peer, p.

During the protocol, a, and p share three pieces of information: the common seed they agree upon, the landmarks, $\{M_1, \ldots, M_L\}$, they generate using this seed, and the similarity vectors containing their similarity with respect to $\{M_1, \ldots, M_L\}$. The first two of these items do not depend on the profile of p and thus do not disclose any information. So we concentrate our analysis on the information that $\vec{v}_p$ may leak about the corresponding compact profiles.

9.5.1 Conditional Entropy as a Measure of Information Leakage

We start our analysis by obtaining a first expression for the amount of information leaked by our landmark-based similarity computation. Let $\vec{c}$ be the compact profile of p. We define W as the random variable for p's normalized compact profile, with realization $\vec{w} = \frac{\vec{c}}{||\vec{c}||}$. We also define V as the random variable for the corresponding similarity vector with realization $\vec{v}$, and Mt as the random variable for the landmark matrix with realization M. We can then express the uncertainty about W given V and Mt through the conditional entropy $H(W|V, Mt)$. Such uncertainty corresponds to the amount of information protected from the adversary. According to the definition of conditional entropy, we have:

$$H(W|V, Mt) = \sum_{\vec{w},\vec{v},M} p(\vec{w}, \vec{v}, M) \log \frac{p(\vec{v}, M)}{p(\vec{w}, \vec{v}, M)}. \tag{9.1}$$

We can then express $p(\vec{w}, \vec{v}, M)$ as follows.

$$\begin{aligned} p(\vec{w}, \vec{v}, M) &= p(\vec{v}|\vec{w}, M)p(\vec{w}, M) \\ &= \begin{cases} 1 \cdot p(\vec{w}, M) & \text{if } \vec{v} = \vec{w}M \\ 0 & \text{if } \vec{v} \neq \vec{w}M \end{cases} \end{aligned} \tag{9.2}$$

This allows us to rewrite Equation (9.1).

$$\begin{aligned} H(W|V, Mt) &= \sum_{\vec{w},\vec{v},M,\ \text{s.t.}\ p(\vec{w},\vec{v},M)\neq 0} p(\vec{w})p(M) \log \frac{p(\vec{v}, M)}{p(\vec{w})p(M)} \\ &= \sum_{\vec{w},\vec{v},M,\ \text{s.t.}\ p(\vec{w},\vec{v},M)\neq 0} p(\vec{w})p(M) \log \frac{p(\vec{v}|M)}{p(\vec{w})} \\ &= \sum_{M} p(M) \sum_{\vec{w}} p(\vec{w}) \sum_{\vec{v}=\vec{w}M} \log \frac{p(\vec{v}|M)}{p(\vec{w})}. \end{aligned} \tag{9.3}$$

We can split Equation (9.3) into two parts, using the fact that $log(\frac{a}{b}) = log(a) - log(b)$. Let $H(W|V, Mt) = L + K$, we have

$$L = \sum_{M} p(M) \sum_{\vec{w}} p(\vec{w}) \sum_{\vec{v}=\vec{w}M} \log p(\vec{v}|M) \tag{9.4}$$

$$K = \sum_{M} p(M) \sum_{\vec{w}} p(\vec{w}) \sum_{\vec{v}=\vec{w}M} -\log p(\vec{w}) \tag{9.5}$$

Because there is only one $\vec{v}$ such that $\vec{v} = \vec{w}M$, we can write K as

$$\begin{aligned} K &= \sum_{M} p(M) \cdot \sum_{\vec{w}} p(\vec{w})(1 \cdot (-\log p(\vec{w}))) \\ &= -\sum_{\vec{w}} p(\vec{w}) \log p(\vec{w}) \\ &= H(W). \end{aligned} \tag{9.6}$$

So we have $H(W|V, Mt) = H(W) - L$. The quantity $-L$ represents the amount of leaked information, that is the amount of information that the adversary, a, can learn about p's compact profile. Equation (9.4) provides a first expression for this amount of information. In the following, we refine this expression and provide a way to compute it.

9.5.2 Leaked Information and the Landmark Matrix

We now identify a relationship between the amount of leaked information and the number of non-zero rows in the landmark matrix, M. We start by taking a closer look at the term $p(\vec{v}|M)$ from Equation (9.4). We expand it as follows.

$$\begin{aligned} p(\vec{v}|M) &= \sum_{\vec{w} \text{ s.t. } p(\vec{v},\vec{w}|M)\neq 0} p(\vec{v}, \vec{w}|M) \\ &= \sum_{\vec{w} \text{ s.t. } p(\vec{v},\vec{w}|M)\neq 0} p(\vec{v}|\vec{w}, M)p(\vec{w}) \\ &= \sum_{\vec{w} \text{ s.t. } p(\vec{v},\vec{w}|M)\neq 0} p(\vec{w}) \\ &= \sum_{\vec{w} \text{ s.t. } \vec{v}=\vec{w}M} p(\vec{w}). \end{aligned} \tag{9.7}$$

The first line follows from the law of total probability while the third and fourth result from the same observations on $p(\vec{v}|\vec{w}, M)$ as in Equation (9.2).

Let us now define $S(\vec{v}, M) = \{\vec{u}|\vec{v} = \vec{u}M, \vec{u} \in \mathbb{Z}_2^n\}$ as the set of all compact profiles that have the same similarity vector given a set of landmarks. This definition implies that: $\forall \vec{u}, \vec{w}(\vec{u}, \vec{w} \in S(\vec{v}, M) \iff \vec{0} = (\vec{u} - \vec{w})M)$. For this condition to hold, all the non-zero rows of M must have zero coefficients (i.e. $M_i \neq \vec{0} \implies [\vec{u} - \vec{w}]_i = 0$), while the remaining ones can have arbitrary coefficients. Let $D(M)$ be the number of non-zero rows in M, then, since $M \in \mathbb{Z}_2^{n\times m}$, we can compute the cardinality of $S(\vec{v}, M)$ as follows.

$$|S(\vec{v}, M)| = 2^{n-D(M)} \tag{9.8}$$

Then, because we assume that all compact profiles are equally likely ($p(\vec{w}) = \frac{1}{2^n}$), we can simplify Equation (9.7) into Equation (9.9).

$$p(\vec{v}|M) = p(\vec{w})|S(\vec{v}, M)| = \frac{1}{2^n} 2^{n-D(M)} = \frac{1}{2^{D(M)}} \tag{9.9}$$

This allows us to compute the amount of leaked information.

$$\begin{aligned} -L &= -\sum_{M} p(M) \sum_{\vec{w}} \frac{1}{2^n} \log \frac{1}{2^{D(M)}} \\ &= \sum_{M} p(M) D(M) \end{aligned} \tag{9.10}$$

	n	ρ_1	$-L$	F1 score
ML-100k, $m = 10$	660	0.4013	265	0.6578
ML-100k, $m = 25$	660	0.7226	477	0.6690
ML-100k, $m = 50$	660	0.9230	609	0.6699

Table 9.1: F1 score and leaked information in the configurations of Section 9.4.

To further simplify $-L$, let $d \in [0, nm]$ be the number of 1's in the matrix M, and let $N(d, D(M))$ be the number of M matrices with d 1's and $D(M)$ non-zero rows. Finally, let $p(M_d)$ be the probability of a matrix with d 1's. Then Equation (9.11) decomposes the summation in the last line of Equation (9.10) as follows. The outer sum considers all the matrices with i non-zero rows. The inner sum considers all the matrices with d 1's (at least i and no more than mi, m being the number of columns).

$$\begin{aligned} -L &= \sum_{i=1}^{n} \sum_{d=i}^{im} N(d,i)p(M_d)i \\ &= \sum_{i=1}^{n} i \sum_{d=i}^{im} N(d,i)p(M_d) \\ &= \sum_{i=1}^{n} ip(D(M) = i) \\ &= E[D(M)] \end{aligned} \tag{9.11}$$

The last two lines follow because $\sum_{d=i}^{im} N(d,i)p(M_d) = p(D(M) = i)$ is the probability of having a matrix with i non-zero rows.

9.5.3 Computation of Expected Information Leakage

We conclude our analysis by computing the expected information leakage in the test configurations of Section 9.4. To this end, let η be the probability of an element in the M matrix's being 1, and let $\vec{r}$ be a row vector in matrix M. We can compute the probability of having a non-zero row vector in M as follows.

$$\rho_1 = p(\vec{r} \neq \vec{0}) = 1 - (1 - \eta)^m \tag{9.12}$$

This allows us to rewrite Equation (9.11) to obtain:

$$-L = E[D(M)] = \rho_1 n. \tag{9.13}$$

Equation (9.13) tells us that the adversary will have access to a fraction ρ_1 of the bits in p's compact profile. For the configuration of Section 9.4, we obtain $\eta = 0.05$. Depending on the dataset and number of landmarks, this leads us to the values of F1 score and leaked information shown in Table 9.1. The results show that 10 landmarks allow *H&S* to provide a good similarity score while only leaking 40% of the information in the compact profile.

9.6 Conclusion

This chapter presented *Hide & Share* (*H&S* for short), a peer-to-peer similarity computation protocol for decentralized KNN computation. We demonstrated both formally and experimentally that *H&S* protects the privacy of users in the context of a peer-to-peer recommender. This protection is provided while preserving the system's effectiveness. *H&S* introduces *landmarks*, random data structures that are shared between peers comparing their profiles. It leverages a combinations of cryptography and security mechanisms to ensure these landmarks cannot be diverted to attack the privacy of users.

Part III

Conclusion

Chapter 10

Conclusion and Perspectives

The work presented in this manuscript shows the applicability of gossip protocols to a wide range of applications by covering two large areas of the research I carried out in the past 10 years. The contributions presented in the various chapters result from collaborations with with PhD students, post-docs, interns as well as with senior researchers. In the following, we discuss a summary of the main contributions of this manuscript, as well as some future research directions.

10.1 Summary of Contributions

Part I presented contribution in the context of data dissemination. This is a topic that I had already started exploring during my PhD thesis on publish subscribe systems [24, 28], and that I focused on during my time as a post-doc the ASAP team of Inria. The contributions I presented in chapters 3 and 4 were the results of work in which I contributed both experimentation and design. Chapter 3 presented an experimental analysis of gossip-based video streaming, and highlighted two important observations that hold in the presence of bandwidth cosntraints. First, gossip fanout cannot be increased arbitrarily. Second, nodes need to refresh their communication partners as often as possible. Chapter 4 complemented these results by presenting *HEAP*, a protocol for video streaming in heterogeneous bandwidth settings, as well as heuristics that improve the performance of gossip-based streaming.

Chapter 5 presented more recent results that I obtained as a supervisor of the PhD thesis of Quentin Dufour. Here my contributions were more in the realm of design and in the planning of the experimental study. The chapter presents the application of existing theoretical research that proposed the application of network coding to multi-source gossip dissemination. Our work addressed several challenges that were left open by theoretical studies and demonstrates that network coding can effectively solve or at least mitigate some of the issues that characterize gossip dissemination, like the bursty nature of gossip traffic.

Part II of this manuscript focused instead on the application of gossip protocols to decentralized recommenders and privacy. The contributions I presented combine both experimental work I did in collaboration with Antoine Boutet when he was a PhD student, and supervision work in the context of the PhD theses of Arnaud Jegou and Antoine Rault. With these colleagues, and several others, I also had the opportunity to work on several other topics that I could not include in this manuscript, but that are not less important.

Chapter 7 presented WhatsUp, a solution for decentralized instant-news recommender. WhatsUp combines a decentralized KNN protocol with a novel similarity metric, and a biased epidemic protocol that manages to balance the need to achieve accurate and serendipitous recommendations. Chapters 8 and 9 presented instead two mechanisms that make it possible to preserve privacy in the context of WhatsUp and of other decentralized recommender solutions. The first consists of an obfuscation mechanism that uses information extracted from similar users to obfuscate a user's profile. This makes it hard for an attacker to pinpoint the precise interests of a use, as it should limit the possibility of using signal-analysis techniques in statistical attacks. The mechanism presented in Chapter 9 consists instead of

technique that replaces similarity computation between profiles, by a similarity between vectors of distances between commonly agreed random profiles. Unlike the contribution of Chapter 8, this techniques results in a noticeable loss in recommendation quality, but it offers more wide applicability and provable bounds in terms of information leakage.

10.2 Research Perspectives

Like most research, the results presented in this manuscript offer avenues for several improvements. In the context of dissemination, for example, it would have been interesting to apply the proposed protocols in larger and more realistic settings with real deployments. At the same time, it would also be interesting to apply the results of Chapter 5 to the protocols presented in the previous chapters. It is very likely, that the use of network coding can allow those or other protocols to achieve better trade offs and performance.

In the context of recommendation, it would also have been nice to investigate the performance of the proposed approaches in more realistic settings. For the solutions in Chapters 8 and 9, we considered relatively simple attacks, but it would be nice to extend our analysis to more challenging settings. In addition, it would be nice to support the protocols from Chapter 8 with theoretical results such as those we obtained in Section 9.5.

10.2.1 Optimizing Data Dissemination

As I mentioned above, a first outcome of the work I presented in Chapter 5 could consist in its application to the protocols presented in the previous chapters. While this could definitely be interesting. I think that the most value could come from a systematic analysis of gossip-based data dissemination solutions. In particular, in Chapter 5 we observed that push-pull solutions like the one in [60] probably provide the best trade-offs between dissemination completeness, latency, and bandwidth consumption. Nonetheless, the various optimization heuristics employed in the state of the art make it difficult to directly compare existing solutions. A concrete avenue to consolidate this line of research would therefore consist in comparing different protocols in greater detail, first by taking into account existing theoretical and experimental results, and then by integrating them with novel analyses and experiments.

Another direction for improvement consists in studying the guarantees that can be offered by the protocol we proposed in Chapter 5. A recent paper [50] proposes the use of gossip to achieve totally-ordered dissemination in large-scale system, a topic that is more and more important if we consider the current success of blockchain-based applications. However, [50] bases its results on balls-and-bins gossip [132] a model that, while easy to analyze, provides very loose bounds. Chapter 5 provided preliminary evidence that network-coding augmented gossip could achieve the same results as [50] for a fraction of the network cost. The challenge remains therefore in providing equivalent theoretical guarantees.

I also plan to apply the work described in Chapter 5 in practical application settings. In the short term, we are considering applying the protocol in the context of micro-blogging applications to federate Mastodon [178] servers in order to enable system-wide search facilities. More in the long term, it would be interesting to see how blockchain protocols can benefit from this and other improvement in data dissemination. One of the major issues of current blockchain architectures lies in their limited scalability. Protocols like the one in Chapter 5 could provide at least partial improvements with respect to the state of the art.

Finally, we also plan to optimize our existing protocol by making it capable to adapt to varying frequencies of message generation. In particular, a key parameter of our protocol is the size of the generations of messages that it encodes together. By making this size self-configuring, we can give greater freedom to application developers.

10.2.2 Privacy-Preserving and Decentralized Recommenders

With respect to accuracy in decentralized recommenders, my work has so far concentrated on user-based collaborative filtering, as this technique turns out to be particularly amenable to decentralization. However, the best performing recommender systems employ a combination of techniques that include model-based methods like matrix factorization. For this reason, I am currently starting to investigate decentralized solutions to perform matrix factorization. In

collaboration with Francois Taiani and Romaric Gaudel, I have started playing with the LLORMA recommender platform to evaluate its ability to operate in a decentralized setting. Initial experiments suggest the need for smarter protocols to address a decentralized environment. We therefore plan to find a student who can work full-time on this topic.

With respect to scalability, I have started to work on the application of biclustering algorithms to recommenders with Antonio Mucherino and Florestan de Moor who is currently applying for PhD funding under my supervision. Florestan is carrying out an experimental analysis of existing biclustering approaches for recommendation. Our goal is to understand to what extent biclustering can help improve accuracy, and responsiveness to dynamic item sets and user preferences.

For his PhD, we are planning to work in collaboration with Mediego the start-up company on recommendation founded by Anne-Marie Kermarrec. Mediego's main challenges stem from its small size and from its set of customers which mostly consists of news outlets. In particular, the cost for computing recommendation represents a significant portion of Mediego's expenses, so we plan to collaborate on the design of computationally efficient algorithms that can leverage approximate computations. To this end, we will leverage the results achieved by Anne-Marie's student, Olivier Ruas, who recently graduated.

Finally, with respect to the trade-off between privacy and quality, I have so far focused on obfuscation and random perturbation techniques for recommendation. In the near future, I plan to explore the use of Trusted execution environments like Intel's SGX to design privacy preserving recommenders that limit the need for obfuscation or randomization techniques. In this respect, I still would like to underline that randomization is still needed to ensure properties like differential privacy, but the use of hardware-level encryption promises to decrease the amount of required randomization and thus its impact on utility.

10.2.3 Privacy-Preserving Decentralized Data Analytics

One of my research goals consists in designing algorithms that will enable data users to perform complex data analytics task without requiring direct access to data repositories. In the context of the PhD thesis of Amaury-Bouchra Pilet, we have started to explore this avenue by building decentralized solutions for machine-learning problems that go beyond recommendation. In particular, we are currently working on a decentralized architecture for multi-task personalized learning with neural networks. Each node in the decentralized network can minimize the effort of learning a specific function by collaborating with other nodes that are learning similar functions. To this end nodes maintain partial models can can be global (i.e. shared with the entire network), semi-local (i.e shared with no one else), or local (not shared with anyone). Although this work is already ongoing, we already see some new challenges that I plan to address in the relatively near future, either the thesis of Amaury, or in that of Florestan de Moor.

First, the algorithm Amaury is working on, assumes that the similarity between the learning targets of different nodes is known in advance. An interesting development of this work therefore consists in studying how to identify similarities between different learning targets, and in how to best organize the partial models of nodes according to these similarities. To achieve this we will take inspiration from our previous work on decentralized KNN but will have to go further. In particular, we will have take into account the different dimensions identified by the different partial models used by a user. For example, two different users may be similar with respect to one partial model and very different with respect to another.

A second challenge lies in the fact that the algorithms being currently developed by Amaury operates by exchanging models between nodes in a gossip-based averaging protocol. Although this is already more private than exchanging training data, research has highlighted [48] techniques to extract sensitive data from trained models. A simple solution to this problem consists of using a privacy preserving averaging protocol like the one Amaury worked on during his thesis. But doing this would add some computational and network overhead at each averaging round. It will therefore be interesting to explore protocols that integrate the privacy-preserving steps with the generation of the model, thereby minimizing overhead without losing privacy guarantees. Like in the case of recommenders, one possibility consists in exploring the benefits that TEEs like Intel's SGX can bring into the design of privacy-preserving data-analytics platforms.

In this same context, I am currently applying for funding the context of two H2020 topics. In both cases, my plans focus on privacy-preserving data analytics solutions, but with different applications settings: the energy market in one, and personal data spaces in the other. I am also considering submitting a proposal as a coordinator in 2020, leveraging previous contacts with companies such as AnswareTech, BMW, and FlandersMake with which I had built a consortium for the ICT-13b call in 2018.

Finally, in the context of the O'Browser project, I have started discussing a collaboration with OrangeLabs Lannion, on a decentralized architecture for network diagnostics. More specifically we plan to record the web browsing performance of mobile devices in specific locations and for specific websites in order to identify possible reasons for performance issues. By aggregating this data over time, we will be able to pinpoint which components of the network infrastructure need updating or intervention. In particular, the devices that experience performance issues will be able to identify the most likely causes, thereby facilitating maintenance actions.

10.2.4 Blockchain

Since the introduction of Bitcoin in 2009, the interest in blockchain-based systems has been steadily increasing. A blockchain consists of decentralized ledger based on a decentralized append-only block-based data structure combined with a protocol that guarantees that the contents of the block respect some consistency rules.

In Section , I already mentioned the work in Dietcoin done in the context of the PhD thesis of Pierre-Louis Roman, who defended in December 2018. In the near future , I plan to focus more on the inner workings of blockchain protocols. A key aspect in the design of a blockchain system lies in the choice of the consensus protocol that controls how blocks are being updated. Although the first blockchain applications all relied on some variant of proof-of-work consensus—a computation-intensive protocol to achieve probabilistic agreement over the content of the ledger—recent systems have been exploring a variety of consensus variants both with probabilistic guarantees[42] and with deterministic ones[41]. In this context, as I mentioned above, I am interested in exploring solutions that can benefit from advances in epidemic dissemination protocols.

In particular, Avalanche [180] promises to build a scalable blockchain solution by exploiting gossip-based techniques for probabilistic agreement. However, the current Avalanche proposal faces important challenges. First, it requires a mechanism to prevent Sybil attacks on the peer-sampling protocol. Second, its gossip-based protocol requires a large number of exchanged messages. I plan to address these two aspects in the near future. To address the first challenge, I plan to apply a Sybil-resilient peer-sampling protocol developed by one of my students, Amaury-Bouchra Pilet, to the Avalanche use case. In particular, I plan to evaluate its effectiveness when compared to Avalanche's current not-so-scalable solution, maintaining global knowledge about the system. To address the second challenge, as I mentioned above, I am considering applying the results in Chapter 5 to optimize message dissemination.

Finally, recent research in blockchain has shown that consensus is not even necessary for the most prominent blockchain application: cryptocurrency [38]. This leads to interesting opportunities for the design of more efficient and more trustworthy cryptocurrency applications. In this context, I plan to work on the design of scalable protocols that can support cryptocurrency and other blockchain applications by relying on causal-broadcast primitives. This also opens opportunities for the design of more scalable protocols that can operate efficiently on low-power devices such as smartphones or IoT devices.

Finally, I plan to investigate the use of blockchain technology in combination with the work on privacy-preserving decentralized analytics I mentioned before. In particular, I would like to integrate a decentralized marketplace that can keep track of which data is shared with whom or for which computation and that can automatically attribute rewards to data providers. This is one of the ideas I put forward my current project submission and that I plan to carry on independently of the funding.

List of Figures

2.1 Three-phase gossip protocol. . . . 7

3.1 Percentage of nodes viewing 99% of the stream and distribution of the corresponding stream lag (upload capped at 700 kbps). . . . 11
3.2 Percentage of nodes viewing 99% of the stream and bandwidth usage distribution with different fanout values and upload caps. . . . 12
3.3 Percentage of nodes viewing the stream with at most 1% jitter as a function of the refresh rate X and request rate Y. . . . 13
3.4 Surviving nodes: percentage viewing 99% of the stream, and average percentage of complete windows. 14

4.1 Three-phase gossip performance in homogeneous and heterogeneous settings (left); and architecture of *HEAP* (right) . . . 18
4.2 *Codec* details. . . . 20
4.3 *Claim* details. . . . 21
4.4 Cumulative distribution of stream lag in the ref-691 and ms-691 scenarios for HEAP and standard 3-ph gossip. . . . 23
4.5 Bandwidth usage over time and by capability class with no message loss and slow retransmission. . . . 24
4.6 Stream Lag and bandwidth usage with the RPS running at various frequencies—different values of t_{RPS}—in homo-691. . . . 25
4.7 Stream Lag with various view sizes (v_{RPS}) for the random peer sampling protocol for standard HEAP and for a variant without retransmission. . . . 26
4.8 Whisker plots for sample average obtained from the RPS view with increasing network sizes. . . . 27
4.9 Percentage of nodes receiving a clear stream over time with a catastrophic failure of 20% or 50% of the nodes happening at t=60s. . . . 29
4.10 Stream Lag for HEAP and delivery rate for the external application for $T_{\text{app}} = 200ms$ (left) and $T_{\text{app}} = 100ms$ (right) with an application duty cycle of 50%. The key in Fig. 4.10a applies to both plots. 30
4.11 Stream Lag for HEAP and delivery rate for the external application for higher duty cycles (d_c) and several values of B_{app} (different lines) with $T_{\text{app}} = 200ms$. . . . 30

5.1 With RLNC, C can send useful information to D and E without knowing what they have received . . . 34
5.2 Pulp (5.2a) and our algorithm (5.2b) behavior under various configuration of the protocol (fanout and time to live) . . . 40
5.3 Comparison of exchanged packet rate, used bandwidth and message delay for 5.3a pulp original ($k = 6, ttl = 4$) and 5.3b our algorithm ($k = 5, ttl = 4$) . . . 41
5.4 How adaptiveness algorithms impact the protocol efficiency . . . 42
5.5 Delay in dynamic networks: varying network configurations (a) and churn (b). . . . 43

6.1 Gossip-based distributed KNN . . . 49

6.2 Clustering mechanism for convergence to an optimal neighborhood. In this example, after exchanging their profiles, Alice and Bob modify their neighbors in order to be connected with the users who share the most their interests. . . . 49

7.1 *Interactions between (1) user opinion; (2)* WUP*: implicit social network; (3)* BEEP*: news dissemination protocol* 54
7.2 *Orientation and amplification mechanisms of Beep* . . . 57
7.3 *F1-Score depending on the fanout and message cost* . . . 59
7.4 *Survey: Size of the LSCC depending on the approach* . . . 60
7.5 [*Dislike and Amplification in* BEEP *on the Survey dataset.* . . . 61
7.6 *Cold start and dynamics in* WHATSUP . . . 62
7.7 *Implementation: bandwidth and performance* . . . 63
7.8 Survey dataset: centralized vs decentralized, recall vs popularity, and F1-Score vs sociability. . . . 64

8.1 *Simplified information flow through the protocol's data structures.* . . . 68
8.2 *Complete information flow through the protocol's data structures.* . . . 71
8.3 *Impact of compacting the profiles (various b-to-d ratios)* . . . 74
8.4 *Impact of filtering sensitive information (various filter sizes, fs)* . . . 74
8.5 *Impact of obfuscating profiles and randomizing dissemination ($fs = 200$)* . . . 75
8.6 *Impact of the randomization for 2-DP* . . . 75
8.7 *OPRD vs 2-DP: F1-Score vs number of messages* . . . 76
8.8 *Randomness vs performance and level of privacy* . . . 76
8.9 *Profile prediction* . . . 77
8.10 *Resilience to censorship* . . . 78

9.1 Recommendation quality with different levels of randomization of user profiles. This quality is not significantly hampered by levels of randomization of up to 75%. . . . 80
9.2 Overview of the *H&S* similarity computation mechanism . . . 81
9.3 Recommendation quality expressed as precision and recall with a varying number of recommendations r, using the MovieLens datasets. . . . 85
9.4 Recommendation quality expressed as precision and recall with a varying number of recommendations r, using the Jester dataset. . . . 85
9.5 Effect of compact profiles and the *H&S* similarity on neighborhood quality. The *H&S* similarity is the main source of perturbation of neighborhood quality. . . . 86
9.6 Trade-off between recommendation quality and privacy for the *H&S*-based system and a system with perturbation-based privacy. . . . 87
9.7 Average bandwidth consumption of a peer per gossip cycle. The *H&S*-based system consumes roughly twice to seven times more bandwidth than our system model with 5 to 50 landmarks. . . . 88
9.8 Average storage space needed for a peer. The *H&S*-based system needs less storage space because peers only store the seed used to generate landmarks. . . . 88

Cited Publications by the Author

Each entry highlights my contribution using the following code.
SC: scientific guidance, CT: core technical contribution, EX: experiments, WT: writing
As per the guidelines, the list below highlights my name and those of the students of which I am a main supervisor.

Awards

Best Paper Award at IC2E Stéphane Delbruel, Davide Frey, François Taïani: Exploring the Use of Tags for Georeplicated Content Placement. IC2E 2016: 172-181 [8]

Journal articles

[1] Borzou Rostami, André Chassein, Michael Hopf, **Davide Frey**, Christoph Buchheim, Federico Malucelli, and Marc Goerigk. "The quadratic shortest path problem: complexity, approximability, and solution methods". In: *European Journal of Operational Research* 268.2 (July 2018), pp. 473 –485. DOI: 10.1016/j.ejor.2018.01.054. URL: https://hal.inria.fr/hal-01781605.
Rating: A
Contribution: CT,EX,WT

[2] Brice Nédelec, Julian Tanke, Pascal Molli, Achour Mostefaoui, and **Davide Frey**. "An Adaptive Peer-Sampling Protocol for Building Networks of Browsers". In: *World Wide Web* 25 (2017), p. 1678. DOI: 10.1007/s11280-017-0478-5. URL: https://hal.inria.fr/hal-01619906.
Rating: A
Contribution: SC,WT

[3] Antoine Boutet, **Davide Frey**, Rachid Guerraoui, **Arnaud Jégou**, and Anne-Marie Kermarrec. "Privacy-Preserving Distributed Collaborative Filtering". In: *Computing*. Special Issue on NETYS 2014 98.8 (Aug. 2016), pp. 827–846. DOI: 10.1007/s00607-015-0451-z. URL: https://hal.inria.fr/hal-01251314.
Rating: A (Special Issue)
Contribution: SC,CT,WT

[4] Antoine Boutet, **Davide Frey**, **Arnaud Jégou**, Anne-Marie Kermarrec, and **Heverson Ribeiro**. "FreeRec: an Anonymous and Distributed Personalization Architecture". In: *Computing* (Dec. 2015). URL: https://hal.inria.fr/hal-00909127.
Rating: A (Special Issue)
Contribution: SC,CT,WT,EX

[5] **Davide Frey**, **Arnaud Jégou**, Anne-Marie Kermarrec, Michel Raynal, and Julien Stainer. "Trust-Aware Peer Sampling: Performance and Privacy Tradeoffs". In: *Theoretical Computer Science* (Feb. 2013). URL: https://hal.inria.fr/hal-00872996.
Rating: A (Special Issue)
Contribution: SC,CT,WT

Conferences

[6] Yérom-David Bromberg, **Quentin Dufour**, and **Davide Frey**. "Multisource Rumor Spreading with Network Coding". In: *INFOCOM 2019*. Paris, France, Apr. 2019. URL: https://hal.inria.fr/hal-01946632.
Rating: A*
Contribution: SC,CT,WT

[7] Hicham Lakhlef, **Davide Frey**, and Michel Raynal. “Optimal Collision/Conflict-Free Distance-2 Coloring in Wireless Synchronous Broadcast/Receive Tree Networks”. In: *45th International Conference on Parallel Processing*. Philadelphia, PA, United States, Aug. 2016, pp. 350–359. DOI: 10.1109/ICPP.2016.47. URL: https://hal.archives-ouvertes.fr/hal-01396940.
Rating: A
Contribution: SC,WT

[8] **Stéphane Delbruel**, **Davide Frey**, and François Taïani. “Exploring The Use of Tags for Georeplicated Content Placement”. In: *IEEE IC2E'16*. Best paper award. Berlin, Germany, Apr. 2016, pp. 172–181. DOI: 10.1109/IC2E.2016.37. URL: https://hal.inria.fr/hal-01257939.
Contribution: SC,WT

[9] **Stéphane Delbruel**, **Davide Frey**, and François Taïani. “Mignon: A Fast Decentralized Content Consumption Estimation in Large-Scale Distributed Systems”. In: *16th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS)*. Ed. by Márk Jelasity and Evangelia Kalyvianaki. Vol. LNCS-9687. Distributed Applications and Interoperable Systems. Heraklion, Greece: Springer, June 2016, pp. 32–46. DOI: 10.1007/978-3-319-39577-7_3. URL: https://hal.inria.fr/hal-01301230.
Rating: B
Contribution: SC,CT,WT

[10] **Davide Frey**, Achour Mostefaoui, Matthieu Perrin, **Pierre-Louis Roman**, and François Taïani. “Speed for the elite, consistency for the masses: differentiating eventual consistency in large-scale distributed systems”. In: *Proceedings of the 2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS 2016)*. Budapest, Hungary: IEEE Computer Society, Sept. 2016, pp. 197–206. DOI: 10.1109/SRDS.2016.032. URL: https://hal.inria.fr/hal-01344138.
Rating: A
Contribution: SC,CT,WT

[11] Antoine Boutet, **Davide Frey**, Rachid Guerraoui, Anne-Marie Kermarrec, **Antoine Rault**, François Taïani, and Jingjing Wang. “Hide & Share: Landmark-based Similarity for Private KNN Computation”. In: *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. Rio de Janeiro, Brazil, June 2015, pp. 263–274. DOI: 10.1109/DSN.2015.60. URL: https://hal.archives-ouvertes.fr/hal-01171492.
Rating: A
Contribution: SC,CT,WT

[12] Borzou Rostami, Federico Malucelli, **Davide Frey**, and Christoph Buchheim. “On the Quadratic Shortest Path Problem”. In: *14th International Symposium on Experimental Algorithms*. 14th International Symposium on Experimental Algorithms. Paris, France, June 2015. DOI: 10.1007/978-3-319-20086-6_29. URL: https://hal.inria.fr/hal-01251438.
Rating: B
Contribution: CT,EX,WT

[13] **Davide Frey**, **Mathieu Goessens**, and Anne-Marie Kermarrec. “Behave: Behavioral Cache for Web Content”. In: *4th International Conference on Distributed Applications and Interoperable Systems (DAIS)*. Ed. by Kostas Magoutis and Peter Pietzuch. Vol. LNCS 8460. Lecture Notes in Computer Science. Berlin, Germany: Springer, June 2014, pp. 89–103. DOI: 10.1007/978-3-662-43352-2_8. URL: https://hal.inria.fr/hal-01079976.
Rating: B
Contribution: SC,CT,WT

[14] Antoine Boutet, **Davide Frey**, Rachid Guerraoui, **Arnaud Jégou**, and Anne-Marie Kermarrec. “WhatsUp Decentralized Instant News Recommender”. In: *IPDPS 2013*. Boston, United States, May 2013. URL: https://hal.inria.fr/hal-00769291.

Rating: A
Contribution: CT,WT,EX

[15] Antoine Boutet, **Davide Frey**, **Arnaud Jégou**, Anne-Marie Kermarrec, and **Heverson Borba Ribeiro**. "FreeRec: an Anonymous and Distributed Personalization Architecture". In: *NETYS*. Marrakesh, Morocco, May 2013. URL: https://hal.inria.fr/hal-00820377.
Contribution: SC,CT,WT,EX

[16] **Davide Frey**, Anne-Marie Kermarrec, and Konstantinos Kloudas. "Probabilistic Deduplication for Cluster-Based Storage Systems". In: *ACM Symposium on Cloud Computing*. San Jose, CA, United States, Oct. 2012, p. 17. DOI: 10.1145/2391229.2391246. URL: https://hal.inria.fr/hal-00728215.
Contribution: SC,WT

[17] **Davide Frey**, **Arnaud Jégou**, and Anne-Marie Kermarrec. "Social Market: Combining Explicit and Implicit Social Networks". In: *International Symposium on Stabilization, Safety, and Security of Distributed Systems*. Grenoble, France: LNCS, Oct. 2011, pp. 193–207. DOI: 10.1007/978-3-642-24550-3_16. URL: https://hal.inria.fr/inria-00624129.
Rating: C
Contribution: SC,CT,WT

[18] Marin Bertier, **Davide Frey**, Rachid Guerraoui, Anne-Marie Kermarrec, and Vincent Leroy. "The Gossple Anonymous Social Network". In: *ACM/IFIP/USENIX 11th International Middleware Conference (MIDDLEWARE)*. Ed. by Indranil Gupta; Cecilia Mascolo. Vol. LNCS-6452. Middleware 2010. Bangalore, India: Springer, Nov. 2010, pp. 191–211. DOI: 10.1007/978-3-642-16955-7_10. URL: https://hal.inria.fr/inria-00515693.
Rating: A
Contribution: CT,EX,WT

[19] **Davide Frey**, Rachid Guerraoui, Anne-Marie Kermarrec, and Maxime Monod. "Boosting Gossip for Live Streaming". In: *P2P 2010*. Delft, Netherlands, Aug. 2010, pp. 1–10. DOI: 10.1109/P2P.2010.5569962. URL: https://hal.inria.fr/inria-00517384.
Rating: C
Contribution: CT,EX,WT

[20] **Davide Frey**, Rachid Guerraoui, Anne-Marie Kermarrec, Maxime Monod, Koldehofe Boris, Mogensen Martin, and Vivien Quéma. "Heterogeneous Gossip". In: *Middleware 2009*. Urbana-Champaign, IL, United States, Dec. 2009, pp. 42–61. DOI: 10.1007/978-3-642-10445-9_3. URL: https://hal.inria.fr/inria-00436125.
Rating: A
Contribution: CT,EX,WT

[21] **Davide Frey**, Rachid Guerraoui, Anne-Marie Kermarrec, Maxime Monod, and Vivien Quéma. "Stretching Gossip with Live Streaming". In: *DSN 2009*. Estoril, Portugal, June 2009, pp. 259–264. DOI: 10.1109/DSN.2009.5270330. URL: https://hal.inria.fr/inria-00436130.
Rating: A
Contribution: CT,EX,WT

[22] **Davide Frey** and Amy L. Murphy. "Failure-Tolerant Overlay Trees for Large-Scale Dynamic Networks". In: *8th International Conference on Peer-to-Peer Computing 2008 (P2P'08)*. Aachen, Germany, Sept. 2008, pp. 351–361. DOI: 10.1109/P2P.2008.30. URL: https://hal.inria.fr/inria-00337054.
Rating: C
Contribution: CT,EX,WT

[23] **Davide Frey** and Gruia-Catalin Roman. "Context-Aware Publish Subscribe in Mobile ad Hoc Networks". In: *Coordination*. Paphos, Cyprus, June 2007, pp. 37–55. DOI: 10.1007/978-3-540-72794-1_3. URL: https://hal.inria.fr/hal-00739641.
Rating: B
Contribution: CT,WT,EX

[24] Gianpaolo Cugola, **Davide Frey**, Amy L. Murphy, and Gian Pietro Picco. "Minimizing the Reconfiguration Overhead in Content-Based Publish-Subscribe". In: *Symposium on Applied Computing*. Nicosia, Cyprus, Mar. 2004. URL: https://hal.inria.fr/hal-00739607.
Rating: B
Contribution: CT,WT,EX

Workshops

[25] Tristan Allard, **Davide Frey**, George Giakkoupis, and Julien Lepiller. "Lightweight Privacy-Preserving Averaging for the Internet of Things". In: *M4IOT 2016 - 3rd Workshop on Middleware for Context-Aware Applications in the IoT*. Trento, Italy: ACM, Dec. 2016, pp. 19 –22. DOI: 10.1145/3008631.3008635. URL: https://hal.inria.fr/hal-01421986.
Contribution: SC,CT,WT

[26] **Davide Frey**, Marc X. Makkes, **Pierre-Louis Roman**, François Taïani, and Spyros Voulgaris. "Bringing secure Bitcoin transactions to your smartphone". In: *Proceedings of the 15th International Workshop on Adaptive and Reflective Middleware (ARM 2016)*. Trento, Italy: ACM, Dec. 2016, 3:1–3:6. DOI: 10.1145/3008167.3008170. URL: https://hal.inria.fr/hal-01384461.
Contribution: SC,CT,WT

[27] **Davide Frey**, Jérôme Royan, Romain Piegay, Anne-Marie Kermarrec, Emmanuelle Anceaume, and Fabrice Le Fessant. "Solipsis: A Decentralized Architecture for Virtual Environments". In: *1st International Workshop on Massively Multiuser Virtual Environments*. Reno, NV, United States, Mar. 2008. URL: https://hal.inria.fr/inria-00337057.
Contribution: CT,WT

[28] **Davide Sormani**, **Gabriele Turconi**, Paolo Costa, **Davide Frey**, Matteo Migliavacca, and Luca Mottola. "Towards lightweight information dissemination in inter-vehicular networks". In: *Proceedings of the 3rd international workshop on Vehicular ad hoc networks*. Los Angeles, United States, Sept. 2006. URL: https://hal.inria.fr/hal-00739632.
Contribution: SC,CT,WT

[29] **Davide Frey** and Paolo Costa. "Publish-subscribe tree maintenance over a DHT". In: *DEBS 2005 Workshop colocated with ICDCS*. Columbus, OHIO, United States, June 2005. URL: https://hal.inria.fr/hal-00739617.
Contribution: CT,EX,WT

Demos

[30] Raziel Carvajal-Gómez, **Davide Frey**, Matthieu Simonin, and Anne-Marie Kermarrec. *WebGC Gossiping on Browsers without a Server [Live Demo/Poster]*. Web Information System Engineering. Nov. 2015. URL: https://hal.inria.fr/hal-01251787.
Contribution: SC,CT,WT

[31] Raziel Carvajal-Gómez, **Davide Frey**, Matthieu Simonin, and Anne-Marie Kermarrec. *WebGC: Browser-Based Gossiping [Live Demo/Poster]*. Middleware 2014. Dec. 2014. DOI: 10.1145/2678508.2678515. URL: https://hal.inria.fr/hal-01080032.
Contribution: SC,CT,WT

[32] Antoine Boutet, **Davide Frey**, Rachid Guerraoui, and Anne-Marie Kermarrec. *WhatsUp: news from, for, through everyone*. Delft, Netherlands, Aug. 2010. DOI: 10.1109/P2P.2010.5569981. URL: https://hal.inria.fr/inria-00515420.
Contribution: SC,CT,EX,WT

Unpublished Reports

[33] **Davide Frey**, Marc X. Makkes, **Pierre-Louis Roman**, François Taïani, and Spyros Voulgaris. *Dietcoin: shortcutting the Bitcoin verification process for your smartphone*. Research Report RR-9162. Univ Rennes, Inria, CNRS, IRISA, France ; Vrije Universiteit Amsterdam, The Netherlands ; Athens University of Economics and Business, Greece, Mar. 2018, pp. 1–17. URL: https://hal.inria.fr/hal-01743995.
Contribution: SC,CT,WT

[34] Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, and Maxime Monod. *Live Streaming with Gossip*. Research Report RR-9039. Inria Rennes Bretagne Atlantique ; RR-9039, Mar. 2017. URL: https://hal.inria.fr/hal-01479885.
Contribution: CT,EX,WT

[35] **Amaury Bouchra-Pilet**, **Davide Frey**, and François Taïani. *Robust Privacy-Preserving Gossip Averaging*. Tech. rep.
Contribution: SC,WT

References

[36] Flavio Chierichetti, George Giakkoupis, Silvio Lattanzi, and Alessandro Panconesi. "Rumor Spreading and Conductance". In: *J. ACM* 65.4 (2018), 17:1–17:21. DOI: 10.1145/3173043. URL: https://doi.org/10.1145/3173043.

[37] Gábor Danner and Márk Jelasity. "Robust Decentralized Mean Estimation with Limited Communication". In: *Euro-Par 2018: Parallel Processing - 24th International Conference on Parallel and Distributed Computing, Turin, Italy, August 27-31, 2018, Proceedings*. 2018, pp. 447–461. DOI: 10.1007/978-3-319-96983-1_32. URL: https://doi.org/10.1007/978-3-319-96983-1_32.

[38] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi. "AT2: Asynchronous Trustworthy Transfers". In: *CoRR* abs/1812.10844 (2018). arXiv: 1812.10844. URL: http://arxiv.org/abs/1812.10844.

[39] Sonia Ben Mokhtar, Antoine Boutet, Pascal Felber, Marcelo Pasin, Rafael Pires, and Valerio Schiavoni. "X-Search: Revisiting Private Web Search using Intel SGX". In: *CoRR* abs/1805.01742 (2018). arXiv: 1805.01742. URL: http://arxiv.org/abs/1805.01742.

[40] Rafael Pires, David Goltzsche, Sonia Ben Mokhtar, Sara Bouchenak, Antoine Boutet, Pascal Felber, Rüdiger Kapitza, Marcelo Pasin, and Valerio Schiavoni. "CYCLOSA: Decentralizing Private Web Search Through SGX-Based Browser Extensions". In: *CoRR* abs/1805.01548 (2018). arXiv: 1805.01548. URL: http://arxiv.org/abs/1805.01548.

[41] J. Sousa, A. Bessani, and M. Vukolic. "A Byzantine Fault-Tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform". In: *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2018, pp. 51–58. DOI: 10.1109/DSN.2018.00018.

[42] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. "Algorand: Scaling Byzantine Agreements for Cryptocurrencies". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. SOSP '17. Shanghai, China: ACM, 2017, pp. 51–68. ISBN: 978-1-4503-5085-3. DOI: 10.1145/3132747.3132757. URL: http://doi.acm.org/10.1145/3132747.3132757.

[43] István Hegedüs and Márk Jelasity. "Differentially Private Linear Models for Gossip Learning through Data Perturbation". In: *OJIOT* 3.1 (2017), pp. 62–74. URL: http://nbn-resolving.de/urn:nbn:de:101:1-2017080613445.

[44] Hugues Mercier, Laurent Hayez, and Miguel Matos. "Brief Announcement: Optimal Address-Oblivious Epidemic Dissemination". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing*. PODC '17. Washington, DC, USA: ACM, 2017, pp. 151–153. ISBN: 978-1-4503-4992-5. DOI: 10.1145/3087801.3087862. URL: http://doi.acm.org/10.1145/3087801.3087862.

[45] Rui Zhu, Bang Liu, Di Niu, et al. "Network Latency Estimation for Personal Devices: A Matrix Completion Approach". In: *IEEE/ACM Trans. Netw.* 25.2 (2017), pp. 724–737.

[46] Jeremie Decouchant, Sonia Ben Mokhtar, Albin Petit, and Vivien Quéma. "PAG: Private and Accountable Gossip". In: *36th IEEE International Conference on Distributed Computing Systems, ICDCS 2016, Nara, Japan, June 27-30, 2016*. 2016, pp. 35–44. DOI: 10.1109/ICDCS.2016.34. URL: https://doi.org/10.1109/ICDCS.2016.34.

[47] George Giakkoupis, Yasamin Nazari, and Philipp Woelfel. "How Asynchrony Affects Rumor Spreading Time". In: *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*. PODC '16. Chicago, Illinois, USA: ACM, 2016, pp. 185–194. ISBN: 978-1-4503-3964-3. DOI: 10.1145/2933057.2933117. URL: http://doi.acm.org/10.1145/2933057.2933117.

[48] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. “Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures”. In: *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*. CCS ’15. Denver, Colorado, USA: ACM, 2015, pp. 1322–1333. ISBN: 978-1-4503-3832-5. DOI: 10.1145/2810103.2813677. URL: http://doi.acm.org/10.1145/2810103.2813677.

[49] F. Maxwell Harper and Joseph A. Konstan. “The MovieLens Datasets: History and Context”. In: *ACM Trans. Interact. Intell. Syst.* 5.4 (Dec. 2015), 19:1–19:19. ISSN: 2160-6455. DOI: 10.1145/2827872. URL: http://doi.acm.org/10.1145/2827872.

[50] Miguel Matos, Hugues Mercier, Pascal Felber, Rui Oliveira, and José Pereira. “EpTO: An Epidemic Total Order Algorithm for Large-Scale Distributed Systems”. In: *Proceedings of the 16th Annual Middleware Conference*. Middleware ’15. Vancouver, BC, Canada: ACM, 2015, pp. 100–111. ISBN: 978-1-4503-3618-5. DOI: 10.1145/2814576.2814804. URL: http://doi.acm.org/10.1145/2814576.2814804.

[51] Roberto Roverso, Riccardo Reale, Sameh El-Ansary, and Seif Haridi. “SmoothCache 2.0: CDN-quality Adaptive HTTP Live Streaming on Peer-to-peer Overlays”. In: *MMSys*. Portland, Oregon, 2015. ISBN: 978-1-4503-3351-1. DOI: 10.1145/2713168.2713182.

[52] Stefano Traverso, Luca Abeni, Robert Birke, Csaba Kiraly, Emilio Leonardi, Renato Lo Cigno, and Marco Mellia. “Neighborhood Filtering Strategies for Overlay Construction in P2P-TV Systems: Design and Experimental Comparison”. In: *IEEE/ACM TON* 23.3 (June 2015). ISSN: 1063-6692.

[53] S. Ataee and B. Garbinato. “EagleMacaw: A Dual-Tree Replication Protocol for Efficient and Reliable P2P Media Streaming”. In: *PDP 2014*. 2014. DOI: 10.1109/PDP.2014.21.

[54] S. Ataee, B. Garbinato, and F. Pedone. “ReStream - A Replication Algorithm for Reliable and Scalable Multimedia Streaming”. In: *PDP 2013*. 2013. DOI: 10.1109/PDP.2013.19.

[55] Daniel Balouek et al. “Adding Virtualization Capabilities to the Grid’5000 Testbed”. In: *Cloud Computing and Services Science*. Vol. 367. Communications in Computer and Information Science. http://www.grid5000.fr. Springer International Publishing, 2013. ISBN: 978-3-319-04518-4. DOI: 10.1007/978-3-319-04519-1_1.

[56] Ranieri Baraglia, Patrizio Dazzi, Matteo Mordacchini, and Laura Ricci. “A Peer-to-peer Recommender System for Self-emerging User Communities Based on Gossip Overlays”. In: *Journal of Computer and System Sciences* 79.2 (2013), pp. 291–308. ISSN: 0022-0000. DOI: 10.1016/j.jcss.2012.05.011.

[57] R. Roverso, J. Dowling, and M. Jelasity. “Through the wormhole: Low cost, fresh peer sampling for the Internet”. In: *P2P 2013*. 2013. DOI: 10.1109/P2P.2013.6688707.

[58] M. Alaggan, S. Gambs, and A-M. Kermarrec. “BLIP: Non-interactive Differentially-Private Similarity Computation on Bloom Filters”. In: *SSS*. 2012.

[59] Robert Birke, Csaba Kiraly, Emilio Leonardi, Marco Mellia, Michela Meo, and Stefano Traverso. “A Delay-based Aggregate Rate Control for P2P Streaming Systems”. In: *Comput. Commun.* 35.18 (Nov. 2012). ISSN: 0140-3664. DOI: 10.1016/j.comcom.2012.07.005.

[60] Pascal Felber, Anne-Marie Kermarrec, Lorenzo Leonini, Etienne Rivière, and Spyros Voulgaris. “Pulp: An adaptive gossip-based dissemination protocol for multi-source message streams”. English. In: *Peer-to-Peer Networking and Applications* 5.1 (2012). ISSN: 1936-6442. DOI: 10.1007/s12083-011-0110-x.

[61] R. Roverso, S. El-Ansary, and S. Haridi. “Peer2View: A peer-to-peer HTTP-live streaming platform”. In: *2012 IEEE 12th International Conference on Peer-to-Peer Computing (P2P)*. 2012. DOI: 10.1109/P2P.2012.6335813.

[62] Roberto Roverso, Sameh El-Ansary, and Seif Haridi. “SmoothCache: HTTP-Live Streaming Goes Peer-to-Peer”. In: *NETWORKING*. 2012.

[63] A. Singh, G. Urdaneta, M. van Steen, and R. Vitenberg. “Robust Overlays for Privacy-Preserving Data Dissemination over a Social Graph”. In: *ICDCS*. 2012.

[64] Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz. “Theory and Practice of Bloom Filters for Distributed Systems”. In: *IEEE Communications Surveys and Tutorials* (2012), pp. 131–155.

[65] M. Wan, A. Jönsson, C. Wang, L. Li, and Y. Yang. “A random indexing approach for web user clustering and web prefetching”. In: *PAKDD*. 2012.

[66] Ranieri Baraglia, Patrizio Dazzi, Matteo Mordacchini, Laura Ricci, and Luca Alessi. “GROUP: A Gossip Based Building Community Protocol”. In: *Proceedings of the 11th International Conference and 4th International Conference on Smart Spaces and Next Generation Wired/Wireless Networking*. NEW2AN’11/ruSMART’11. St. Petersburg, Russia: Springer-Verlag, 2011. ISBN: 978-3-642-22874-2. URL: http://dl.acm.org/citation.cfm?id=2033707.2033766.

[67] R. Birke et al. “Architecture of a network-aware P2P-TV application: the NAPA-WINE approach”. In: *IEEE Communications Magazine* 49.6 (2011). ISSN: 0163-6804. DOI: 10.1109/MCOM.2011.5784001.

[68] G. Giuffrida and C. Zarba. “A recommendation algorithm for personalized online news based on collective intelligence and content”. In: *ICAART*. 2011.

[69] Bernhard Haeupler. “Analyzing network coding gossip made easy”. In: *Proceedings of the forty-third annual ACM symposium on Theory of computing*. ACM, 2011, pp. 293–302.

[70] Sibren Isaacman, Stratis Ioannidis, Augustin Chaintreau, and Margaret Martonosi. “Distributed rating prediction in user generated content streams”. In: *Proceedings of the fifth ACM conference on Recommender systems*. RecSys ’11. Chicago, Illinois, USA: ACM, 2011, pp. 69–76. ISBN: 978-1-4503-0683-6. DOI: http://doi.acm.org/10.1145/2043932.2043948. URL: http://doi.acm.org/10.1145/2043932.2043948.

[71] A. Machanavajjhala, A. Korolova, and A. D. Sarma. “Personalized social recommendations: accurate or private”. In: *VLDB* (2011).

[72] V. Schiavoni, E. Riviere, and P. Felber. “WHISPER: Middleware for Confidential Communication in Large-Scale Networks”. In: *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*. 2011, pp. 456–466. DOI: 10.1109/ICDCS.2011.15.

[73] Niels Zeilemaker, Mihai Capotă, Arno Bakker, and Johan Pouwelse. “Tribler: P2P Media Search and Sharing”. In: *Proceedings of the 19th ACM International Conference on Multimedia*. MM ’11. Scottsdale, Arizona, USA: ACM, 2011, pp. 739–742. ISBN: 978-1-4503-0616-4. DOI: 10.1145/2072298.2072433.

[74] Marin Bertier, François Bonnet, Anne-Marie Kermarrec, Vincent Leroy, Sathya Peri, and Michel Raynal. “D2HT: The Best of Both Worlds, Integrating RPS and DHT”. In: *Eighth European Dependable Computing Conference, EDCC-8 2010, Valencia, Spain, 28-30 April 2010*. 2010, pp. 135–144. DOI: 10.1109/EDCC.2010.25. URL: https://doi.org/10.1109/EDCC.2010.25.

[75] H. Corrigan-Gibbs and B. Ford. “Dissent: accountable anonymous group messaging”. In: *CCS*. 2010.

[76] Sonia Ben Mokhtar, Alessio Pace, and Vivien Quéma. “FireSpam: Spam Resilient Gossiping in the BAR Model”. In: *29th IEEE Symposium on Reliable Distributed Systems (SRDS 2010), New Delhi, Punjab, India, October 31 - November 3, 2010*. 2010, pp. 225–234. DOI: 10.1109/SRDS.2010.33. URL: https://doi.org/10.1109/SRDS.2010.33.

[77] Luca Abeni, Csaba Kiraly, and Renato Lo Cigno. “On the Optimal Scheduling of Streaming Applications in Unstructured Meshes”. In: *NETWORKING*. 2009.

[78] M. Agrawal, M. Karimzadehgan, and C. Zhai. “An online news recommender system for social networks”. In: *SIGIR-SSM*. 2009.

[79] Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer. “Brahms: Byzantine Resilient Random Membership Sampling”. In: *Computer Networks* 53 (2009).

[80] Mary-Luc Champel, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. “FoG: Fighting the Achilles’ Heel of Gossip Protocols with Fountain Codes”. In: *SSS*. 2009.

[81] M. Draief and L. Massoulié. *Epidemics and rumours in complex networks*. Cambridge University Press, 2009.

[82] Rachid Guerraoui, Kevin Huguenin, Anne-Marie Kermarrec, and Maxime Monod. *LiFT: Lightweight Freerider-Tracking Protocol*. English. Research Report RR-6913. INRIA, 2009.

[83] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. “T-Man: Gossip-based fast overlay topology construction”. In: *ComNet* 53.13 (2009). ISSN: 1389-1286. DOI: http://dx.doi.org/10.1016/j.comnet.2009.03.013.

[84] Anne-Marie Kermarrec, Alessio Pace, Vivien Quéma, and Valerio Schiavoni. “NAT-resilient Gossip Peer Sampling”. In: *ICDCS*. 2009.

[85] Nazanin Magharei and Reza Rejaie. “PRIME: Peer-to-Peer Receiver-Driven Mesh-Based Streaming”. In: *TON* 17.4 (2009).

[86] A. Malekpour, F. Pedone, M. Allani, and B. Garbinato. “Streamline: An Architecture for Overlay Multicast”. In: *NCA*. 2009. DOI: 10.1109/NCA.2009.32.

[87] J. J. D. Mol, A. Bakker, J. A. Pouwelse, D. H. J. Epema, and H. J. Sips. “The Design and Deployment of a BitTorrent Live Video Streaming Solution”. In: *2009 11th IEEE International Symposium on Multimedia*. 2009, pp. 342–349. DOI: 10.1109/ISM.2009.16.

[88] Xiaoyuan Su and Taghi M. Khoshgoftaar. “A Survey of Collaborative Filtering Techniques”. In: *Advances in Artificial Intelligence* 2009 (Oct. 27, 2009), e421425. ISSN: 1687-7470. DOI: 10.1155/2009/421425. URL: http://www.hindawi.com/journals/aai/2009/421425/abs/.

[89] N. Bansod, A. Malgi, B. K. Choi, and J. Mayo. “MuON: Epidemic based mutual anonymity in unstructured P2P networks”. In: *Comput. Netw.* (2008).

[90] Thomas Bonald, Laurent Massoulié, Fabien Mathieu, Diego Perino, and Andrew Twigg. “Epidemic Live Streaming: Optimal Performance Trade-offs”. In: *SIGMETRICS*. 2008.

[91] C. Dwork. “Differential privacy: a survey of results”. In: *TAMC*. 2008.

[92] Bo Li, Susu Xie, Yang Qu, Gabriel Y. Keung, Chuang Lin, Jiangchuan Liu, and Xinyan Zhang. “Inside the New Coolstreaming: Principles, Measurements and Performance Implications”. In: *INFOCOM*. 2008.

[93] Harry Li, Allen Clement, Mirco Marchetti, Manos Kapritsos, Luke Robinson, Lorenzo Alvisi, and Mike Dahlin. “FlightPath: Obedience vs. Choice in Cooperative Services”. In: *OSDI*. 2008.

[94] Chao Liang, Yang Guo, and Yong Liu. “Is Random Scheduling Sufficient in P2P Video Streaming?” In: *ICDCS*. 2008.

[95] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2008. URL: http://www.cryptovest.co.uk/resources/Bitcoin%20paper%20Original.pdf (visited on 07/01/2016).

[96] A. Narayanan and V. Shmatikov. “Robust de-anonymization of large sparse datasets.” In: *Proceedings of the 29th IEEE Symposium on Security and Privacy* (2008).

[97] Fabio Picconi and Laurent Massoulié. “Is There a Future for Mesh-Based live Video Streaming?” In: *P2P*. 2008.

[98] Roberto Baldoni, Roberto Beraldi, Vivien Quéma, Leonardo Querzoni, and Sara Tucci Piergiovanni. “TERA: topic-based event routing for peer-to-peer architectures”. In: *Proceedings of the 2007 Inaugural International Conference on Distributed Event-Based Systems (DEBS)*. ACM, 2007, pp. 2–13. ISBN: 978-1-59593-665-3. URL: http://doi.acm.org/10.1145/1266894.1266898.

[99] Olivier Beaumont, Anne-Marie Kermarrec, Loris Marchal, and Etienne Riviere. “VoroNet: A scalable object network based on Voronoi tessellations”. In: *21th International Parallel and Distributed Processing Symposium (IPDPS 2007), Proceedings, 26-30 March 2007, Long Beach, California, USA*. 2007, pp. 1–10. DOI: 10.1109/IPDPS.2007.370210. URL: https://doi.org/10.1109/IPDPS.2007.370210.

[100] Olivier Beaumont, Anne-Marie Kermarrec, and Etienne Riviere. “Peer to Peer Multidimensional Overlays: Approximating Complex Structures”. In: *Principles of Distributed Systems, 11th International Conference, OPODIS 2007, Guadeloupe, French West Indies, December 17-20, 2007. Proceedings*. 2007, pp. 315–328. DOI: 10.1007/978-3-540-77096-1_23. URL: https://doi.org/10.1007/978-3-540-77096-1_23.

[101] J. Bennett and S. Lanning. “The Netflix Prize”. In: *Proceedings of the KDD Cup Workshop 2007*. New York: ACM, Aug. 2007, pp. 3–6. URL: http://www.cs.uic.edu/~liub/KDD-cup-2007/NetflixPrize-description.pdf.

[102] Ken Birman. “The Promise, and Limitations, of Gossip Protocols”. In: *SIGOPS Oper. Syst. Rev.* 41.5 (Oct. 2007), pp. 8–13. ISSN: 0163-5980. DOI: 10.1145/1317379.1317382. URL: http://doi.acm.org/10.1145/1317379.1317382.

[103] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg. “SpiderCast: a scalable interest-aware overlay for topic-based pub/sub communication”. In: *DEBS*. 2007.

[104] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, et al. “Dynamo: Amazon’s Highly Available Key-value Store”. In: *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*. SOSP ’07. Stevenson, Washington, USA: ACM, 2007, pp. 205–220. ISBN: 978-1-59593-591-5.

[105] Niels Drost, Elth Ogston, Rob van Nieuwpoort, and Henri Bal. “ARRG: Real-World Gossiping”. In: *HPDC*. 2007. ISBN: 978-1-59593-673-8. DOI: http://doi.acm.org/10.1145/1272366.1272386.

[106] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. “Gossip-based peer sampling”. In: *ACM Trans. Comput. Syst.* 25 (3 2007). ISSN: 0734-2071. DOI: http://doi.acm.org/10.1145/1275517.1275520. URL: http://doi.acm.org/10.1145/1275517.1275520.

[107] Ugur Kuter and Jennifer Golbeck. “SUNNY: A New Algorithm for Trust Inference in Social Networks Using Probabilistic Confidence Models”. In: *Proceedings of the 22d National Conference on Artificial Intelligence*. Vol. 2. AAAI Press, 2007, 1377–1382. ISBN: 978-1-57735-323-2.

[108] Laurent Massoulié, Andrew Twigg, Christos Gkantsidis, and Pablo Rodriguez. “Randomized Decentralized Broadcasting Algorithms”. In: *INFOCOM*. 2007.

[109] Michael J. Pazzani and Daniel Billsus. “Content-Based Recommendation Systems”. In: *The Adaptive Web: Methods and Strategies of Web Personalization*. Ed. by Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 325–341. ISBN: 978-3-540-72079-9. DOI: 10.1007/978-3-540-72079-9_10. URL: https://doi.org/10.1007/978-3-540-72079-9_10.

[110] Sujay Sanghavi, Bruce Hajek, and Laurent Massoulié. “Gossiping with Multiple Messages”. In: *IEEE Trans. Information Theory* 53.12 (2007), pp. 4640–4654.

[111] Meng Zhang, Qian Zhang, Lifeng Sun, and Shiqiang Yang. “Understanding the Power of Pull-Based Streaming Protocol: Can We Do Better?” In: *JSAC* 25.9 (2007), pp. 1678–1694.

[112] E. Banos, I. Katakis, N. Bassiliades, G. Tsoumakas, and I. P. Vlahavas. “PersoNews: a personalized news reader enhanced by machine learning and semantic filtering”. In: *ODBASE*. 2006.

[113] Supratim Deb, Muriel Médard, and Clifford Choute. “Algebraic gossip: A network coding approach to optimal multiple rumor mongering”. In: *IEEE/ACM Transactions on Networking (TON)* 14.SI (2006), pp. 2486–2507.

[114] Mayur Deshpande, Bo Xing, Iosif Lazardis, Bijit Hore, Nalini Venkatasubramanian, and Sharad Mehrotra. “CREW: A Gossip-based Flash-Dissemination System”. In: *Proc. of ICDCS*. 2006.

[115] C. Dwork, F. McSherry, K. Nissim, and A. Smith. “Calibrating noise to sensitivity in private data analysis”. In: *Theory of Cryptography*. 2006.

[116] Christina Fragouli, Jean-Yves Le Boudec, and Jörg Widmer. “Network Coding: An Instant Primer”. In: *SIGCOMM Comput. Commun. Rev.* 36.1 (Jan. 2006), pp. 63–68. ISSN: 0146-4833.

[117] Simon Funk. https://sifter.org/~simon/journal/20061211.html. 2006.

[118] Pradeep Kyasanur, Romit Roy Choudhury, and Indranil Gupta. “Smart Gossip: An Adaptive Gossip-based Broadcasting Service for Sensor Networks”. In: *MASS*. 2006.

[119] Harry Li, Allen Clement, Edmund Wong, Jeff Napper, Indrajit Roy, Lorenzo Alvisi, and Michael Dahlin. “BAR Gossip”. In: *OSDI*. 2006.

[120] Vidhyashankar Venkataraman, Kaouru Yoshida, and Paul Francis. “Chunkyspread: Heterogeneous Unstructured Tree-Based Peer-to-Peer Multicast”. In: *ICNP*. 2006.

[121] Spyros Voulgaris, Etienne Riviere, Anne-Marie Kermarrec, and Maarten van Steen. “Sub-2-Sub: Self-Organizing Content-Based Publish Subscribe for Dynamic Large Scale Collaborative Networks”. In: *5th International workshop on Peer-To-Peer Systems, IPTPS 2006, Santa Barbara, CA, USA, February 27-28, 2006*. 2006. URL: http://www.iptps.org/papers-2006/Vulgaris-sub06.pdf.

[122] Zhengli Huang, Wenliang Du, and Biao Chen. “Deriving private information from randomized data”. In: *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*. ACM, 2005, 37–48.

[123] Huseyin Polat and Wenliang Du. “SVD-based collaborative filtering with privacy”. In: *Proceedings of the 2005 ACM Symposium on Applied Computing*. ACM, 2005, 791–795.

[124] E. Spertus, M. Sahami, and O. Buyukkokten. “Evaluating similarity measures: a large-scale study in the orkut social network”. In: *KDD*. 2005.

[125] P. Tan, M. Steinbach, and V. Kumar. *Introduction to data mining*. Addison Wesley, 2005.

[126] Spyros Voulgaris, Daniela Gavidial, and Maarten van Steen. “CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays”. In: *Journal of Network and Systems Management* 13.2 (June 2005), pp. 197–217.

[127] Spyros Voulgaris and Maarten van Steen. “Epidemic-Style Management of Semantic Overlays for Content-Based Searching”. en. In: *Euro-Par 2005 Parallel Processing*. 3648. Springer Berlin Heidelberg, 2005, pp. 1143–1152. ISBN: 978-3-540-28700-1, 978-3-540-31925-2.

[128] R. Dingledine, N. Mathewson, and P. Syverson. “Tor: the second-generation onion router”. In: *USENIX Security Symposium*. 2004.

[129] Patrick Eugster, Rachid Guerraoui, Anne-Marie Kermarrec, and Laurent Maussoulie. “From epidemics to distributed computing”. In: *IEEE Computer* 37.5 (2004), pp. 60–67.

[130] Patrick T. Eugster, Rachid Guerraoui, Anne-Marie Kermarrec, and Laurent Massoulié. “Epidemic Information Dissemination in Distributed Systems”. In: *Computer* 37.5 (2004), pp. 60–67.

[131] Peng Han, Bo Xie, Fan Yang, and Ruimin Shen. “A scalable P2P recommender system based on distributed collaborative filtering”. In: *Expert Systems with Applications* 27.2 (2004), pp. 203 –210. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2004.01.003. URL: http://www.sciencedirect.com/science/article/pii/S0957417404000065.

[132] Boris Koldehofe. “Simple gossiping with balls and bins.” In: *Stud. Inform. Univ.* 3.1 (2004), pp. 43–60.

[133] B. N. Miller, J. A. Konstan, and J. Riedl. “PocketLens: toward a personal recommender system”. In: *TOIS* (2004).

[134] M.E.J. Newman. “Fast algorithm for detecting community structure in networks”. In: *Physical Review E* (2004).

[135] A. Singh, M. Castro, P. Druschel, and A. Rowstron. “Defending against eclipse attacks on overlay networks”. In: *SIGOPS*. 2004.

[136] F. Wu, B.A. Huberman, L.A. Adamic, and J.R. Tyler. “Information flow in social groups”. In: *Physica A: Statistical and Theoretical Physics* (2004).

[137] Ranjita Bhagwan, Stefan Savage, and Geoffrey M. Voelker. “Understanding Availability”. In: *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*. 2003.

[138] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony I. T. Rowstron, and Atul Singh. “SplitStream: High-bandwidth Multicast in Cooperative Environments”. In: *SOSP*. 2003.

[139] Philip A. Chou, Yunnan Wu, and Kamal Jain. “Practical Network Coding”. In: *Allerton Conference on Communication, Control, and Computing*. Oct. 2003.

[140] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. “The many faces of publish/subscribe”. In: *ACM Computing Surveys (CSUR)* 35.2 (2003), pp. 114–131. DOI: http://doi.acm.org/10.1145/857076.857078.

[141] Patrick Th. Eugster, Rachid Guerraoui, Sidath B. Handurukande, Anne-Marie Kermarrec, and Petr Kouznetsov. “Lightweight Probabilistic Broadcast”. In: *ACM Transactions on Computer Systems* 21.4 (2003), pp. 341 –374.

[142] Ayalvadi J. Ganesh, Anne-Marie Kermarrec, and Laurent Massoulié. “Peer-to-Peer Membership Management for Gossip-Based Protocols”. In: *IEEE ToC* 52 (2 2003), pp. 139–149.

[143] O. Goldreich. “Cryptography and cryptographic protocols”. In: *Distrib. Comput.* (2003).

[144] Manish Jain and Constantinos Dovrolis. “End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput”. In: *IEEE/ACM Trans. Netw.* 11 (4 2003). ISSN: 1063-6692.

[145] H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar. “On the Privacy Preserving Properties of Random Data Perturbation Techniques”. In: *Third IEEE International Conference on Data Mining*. IEEE Computer Society, 2003, pp. 99–106.

[146] Anne-Marie Kermarrec, Laurent Massoulié, and Ayalvadi Ganesh. “Probabilistic Reliable Dissemination in Large-Scale Systems”. In: *TPDS* 14.3 (2003), pp. 248–258.

[147] Anne-Marie Kermarrec, Laurent Massoulié, and Ayalvadi Ganesh. “Probabilistic Reliable Dissemination in Large-Scale Systems”. In: *TPDS* 14.3 (2003).

[148] Dejan Kostić, Adolfo Rodriguez, Jeannie Albrecht, and Amin Vahdat. “Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh”. In: *SOSP*. 2003.

[149] Greg Linden, Brent Smith, and Jeremy York. “Amazon.com Recommendations: Item-to-Item Collaborative Filtering”. In: *IEEE Internet Computing* 7.1 (2003), pp. 76–80. ISSN: 1089-7801. DOI: 10.1109/MIC.2003.1167344.

[150] H. Polat and W. Du. “Privacy-Preserving Collaborative Filtering Using Randomized Perturbation Techniques”. In: *ICDM*. 2003.

[151] R. S. Prasad, M. Murray, C. Dovrolis, K. Claffy, Ravi Prasad, and Constantinos Dovrolis Georgia. “Bandwidth Estimation: Metrics, Measurement Techniques, and Tools”. In: *IEEE Network* 17 (2003).

[152] J. Canny. “Collaborative Filtering with Privacy”. In: *SP*. 2002.

[153] J. Canny. “Collaborative filtering with privacy via factor analysis”. In: *SIGIR*. 2002.

[154] Andrew Tanenbaum. *Computer Networks*. 4th. Prentice Hall Professional Technical Reference, 2002. ISBN: 0130661023.

[155] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. “Scalability and accuracy in a large-scale network emulator”. In: *OSDI*. 2002.

[156] Ken Goldberg, Theresa Roeder, and Chris Perkins. “Eigentaste: A constant time collaborative filtering algorithm”. In: *Information Retrieval* 4 (2001), pp. 133–151.

[157] P. Kanerva, J. Kristoferson, and A. Holst. “Random Indexing of Text Samples for Latent Semantic Analysis”. In: *CCSS*. 2000.

[158] Richard Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vöcking. “Randomized Rumor Spreading”. In: *FOCS*. 2000.

[159] Kenneth Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky. “Bimodal Multicast”. In: *TOCS* 17.2 (1999), pp. 41–88.

[160] Simon Blake-Wilson and Alfred Menezes. “Authenticated Diffe-Hellman Key Agreement Protocols”. In: *Selected Areas in Cryptography*. 1556. Springer Berlin Heidelberg, 1999, pp. 339–361. ISBN: 978-3-540-65894-8, 978-3-540-48892-7.

[161] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. “An Algorithmic Framework for Performing Collaborative Filtering”. In: *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Berkeley, California, USA: ACM, 1999, pp. 230–237. ISBN: 1-58113-096-1. DOI: 10.1145/312624.312682.

[162] Pierre L’Ecuyer. *Good Parameters And Implementations For Combined Multiple Recursive Random Number Generators*. 1998.

[163] Makoto Matsumoto and Takuji Nishimura. “Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudo-random Number Generator”. In: *ACM Transactions on Modeling and Computer Simulation* 8.1 (1998), pp. 3–30. ISSN: 1049-3301. DOI: 10.1145/272991.272995.

[164] T. Kameda, Y. Ohtsubo, and M. Takezawa. “Centrality in sociocognitive networks and social influence: an illustration in a group decision-making context”. In: *Journal of Personality and Social Psychology* (1997).

[165] Luigi Rizzo. “Effective Erasure Codes for Reliable Computer Communication Protocols”. In: *CCR* 27.2 (1997).

[166] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. “GroupLens: An Open Architecture for Collaborative Filtering of Netnews”. In: *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*. ACM, 1994, 175–186. ISBN: 0-89791-689-1.

[167] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. “Using Collaborative Filtering to Weave an Information Tapestry”. In: *Commun. ACM* 35.12 (Dec. 1992), pp. 61–70. ISSN: 0001-0782. DOI: 10.1145/138859.138867. URL: http://doi.acm.org/10.1145/138859.138867.

[168] Moni Naor. “Bit Commitment Using Pseudo-Randomness”. In: *Journal of Cryptology* 4 (1991), 151–158.

[169] Jussi Karlgren. *An algebra for recommendations : Using reader data as a basis for measuring document proximity*. Tech. rep. 179. QC 20160530. KTH, Computer and Systems Sciences, DSV, 1990.

[170] Alan J. Demers, Daniel H. Greene, Carl Hauser, et al. “Epidemic Algorithms for Replicated Database Maintenance”. In: *Operating Systems Review* 22.1 (1988), pp. 8–32.

[171] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. “Epidemic Algorithms for Replicated Database Maintenance”. In: *PODC*. 1987.

[172] R Cleve. “Limits on the Security of Coin Flips when Half the Processors Are Faulty”. In: *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*. Berkeley, California, USA: ACM, 1986, pp. 364–369. ISBN: 0-89791-193-8. DOI: 10.1145/12130.12168.

[173] Manuel Blum. “Coin Flipping by Telephone a Protocol for Solving Impossible Problems”. In: *ACM Special Interest Group on Algorithms and Computation Theory News* 15.1 (1983), pp. 23–27. ISSN: 0163-5700. DOI: 10.1145/1008908.1008911.

[174] Cornelis Joost van Rijsbergen. *Information Retrieval*. en. 2nd ed. Butterworths, 1979. ISBN: 0408709294, 9780408709293.

[175] Leslie Lamport. “Time, clocks, and the ordering of events in a distributed system”. In: *Communications of the ACM* 21.7 (1978), pp. 558–565.

[176] Burton H. Bloom. “Space/Time Trade-offs in Hash Coding with Allowable Errors”. In: *Communications of the ACM* 13.7 (1970), 422–426. ISSN: 0001-0782. DOI: 10.1145/362686.362692.

[177] Stanley L. Warner. “Randomized response: a survey technique for eliminating evasive answer bias”. In: *Journal of the American Statistical Association* 60.309 (1965), pp. 63–69.

[178] https://mastodon.social/about.

[179] Daniel Julius Bernstein. *NaCl: Networking and Cryptography library*. en. URL: http://nacl.cr.yp.to (visited on 06/26/2014).

[180] Team Rocket 0x 8a5d 2d32 e68b c500 36e4 d086 0446 17fe 4a0a 0296 b274 999b a568 ea92 da46 d533. “Snowflake to Avalanche: A Novel Metastable Consensus Protocol Family for Cryptocurrencies”. https://ipfs.io/ipfs/QmUy4jh5mGNZvLkjies1RWM4YuvJh5o2FYopNPVYwrRVGV.

[181] SNAP. stanford network analysis platform http://snap.stanford.edu.

[182] *Tribler*. http://www.tribler.org.