From online social network analysis to a user-centric private sharing system

by

© *Arastoo Bozorgi*

A thesis submitted to the
School of Graduate Studies
in partial fulfilment of the
requirements for the degree of
Doctor of Philosophy

Department of *Computer Science*
Memorial University of Newfoundland

July 2020

St. John's Newfoundland

Abstract

Online social networks (OSNs) have become a massive repository of data constructed from individuals' inputs: posts, photos, feedbacks, locations, etc. By analyzing such data, meaningful knowledge is generated that can affect individuals' beliefs, desires, happiness and choices — a data circulation started from individuals and ended in individuals! The OSN owners, as the one authority having full control over the stored data, make the data available for research, advertisement and other purposes. However, the individuals are missed in this circle while they generate the data and shape the OSN structure.

In this thesis, we started by introducing approximation algorithms for finding the most influential individuals in a social graph and modeling the spread of information. To do so, we considered the communities of individuals that are shaped in a social graph. The social graph is extracted from the data stored and controlled centrally, which can cause privacy breaches and lead to individuals' concerns. Therefore, we introduced *UPSS: the user-centric private sharing system*, in which the individuals are considered as the real data owners and provides secure and private data sharing on untrusted servers.

The UPSS's public API allows the application developers to implement applications as diverse as OSNs, document redaction systems with integrity properties, censorship-resistant systems, health care auditing systems, distributed version control systems with flexible access controls and a filesystem in userspace. Accessing users' data is possible only with explicit user consent. We implemented the two later cases to show the applicability of UPSS.

Supporting different storage models by UPSS enables us to have a local, remote and global filesystem in userspace with one unique core filesystem implementation and having it mounted with different block stores.

By designing and implementing UPSS, we show that security and privacy can be addressed at the same time in the systems that need selective, secure and

collaborative information sharing without requiring complete trust.

Acknowledgements

I have spent almost five years of my life sitting in my office and working on my thesis until I got to this point writing this page, that is the last words of my thesis. With the support of my supervisors, family and friends, all the days of this long journey were sweet and memorable for me.

I would like to thank my supervisors, Dr. Jonathan Anderson, who helped me to find my way when I was almost lost, Dr. Saeed Samet, who taught me patience and supportiveness, and Dr. Antonina Kolokolova for her wise suggestions. You taught me how to overcome the challenges in both my personal and academic life.

Thanks to my mother, who never gives up with her kind support from thousands of miles away and keeping hope alive in me. Thanks to my father, who was always encouraging me to continue my studies when he was in this world. He devoted himself to his family and kept the smile on her lips even in his hard days. I also thank my brother and sisters for being a mentor for their younger brother for his entire life.

I thank Dr. Todd Wareham that his office door is always open to everyone and he welcomes everyone with his bowl of candy. I never forget his help in proving the $NP - hardness$ of one of my problems while he was busy writing a book.

And finally, I would like to thank my friends for being with me whenever I need them.

Contents

List of Tables

List of Figures

Chapter 1

Introduction and Overview

Starting from 70000 years ago, social networks and living in groups are one of the principal reasons for human survival, claimed Y. N. Harari [Har14]. N. A. Christakis and J. H. Fowler have shown that the relationship between people in social networks not only affects the weight and happiness of them but also someone can have an influence on another's taste, health, wealth and beliefs [CF09, FC08, CF07]. All these studies show the tendency of human beings to live as groups and be affected by other members of the group. The number of people that each person can have a strong tie with in a group is defined as the Dunbar number, which is around 150 [Dun98].

Online social networks (OSNs) are platforms that attempt to simulate this social life in a digital world by connecting people from different languages, beliefs and cultures. They provide people with a new kind of socializing that is not limited to real-life conditions, such as geographical distances and even the Dunbar number. We can have more than 150 friends on those platforms and be sure that the platform reminds us of them from time to time by showing us their life updates, thoughts, or birthdays. However, these are not the only features of OSNs. They also enable us to spread our ideas and thoughts very rapidly without worrying about not being heard. There are examples of this kind, such as an Iranian movement started from a

Facebook page [Ali18], that one single person started a revolutionary movement by just using social media. On the other hand, there is evidence that the spread of fake news on social media can affect the election results [BM19, AG17]. Therefore OSNs can be as a double-edged sword: they can be very beneficial in our lives, or they can cause some concerns, such as privacy concerns that OSN users facing these days.

In this thesis we have demonstrated that we can design a secure and private sharing and collaboration system such as an online social network in which the users are considered as the main data owners with full control over their data, by combining the research from distributed systems, cryptography and filesystems. UPSS, as our solution, protects the users from the adversaries that need to be trusted in the existing systems. More specifically, UPSS stores users' data on untrusted block stores and achieve confidentiality, integrity and availability without sacrificing the performance and users' privacy.

1.1 Definitions

The terms that are used frequently in this thesis are described in this section to give a clear understanding of them.

Online Social Network (OSN) Online social networks serve as a medium for modeling interactions between individuals, groups and organizations. A social network can be modelled as a directed or undirected graph $G = (V, E)$ where V and E are the set of nodes and edges of G. Individuals are modelled as the set of nodes V and the relationships between them are modelled as the set of edges E. The relationships between individuals are established based on the friendships in their real life, being co-tagged in a photo, being co-authored in a book, etc.

Trust We use the definition of trust that is presented in [CH96] that says if "A trusts B" means that "B can act in such a way as to put things into A's set of trust

assumptions without A's explicit consent".

Privacy The term privacy is a multifaceted and complex concept that can be viewed from different perspectives. Based on the research which was done by Gürses [DG10], privacy can be classified into three paradigms: privacy as control, privacy as confidentiality and privacy as practice.

Privacy as control discusses the users' ability to have control their data collection, use and processing. Therefore the trusted organizations should provide a mechanism for the users to address such needs. In privacy as confidentiality, cryptography is used to minimize information disclosure and the organizations are not trusted. However, in privacy as practice, the information flow is as transparent as possible.

In this thesis, we consider the second definition of privacy that is privacy as confidentiality and in some places, we use privacy and confidentiality interchangeably.

Storage A medium for storing individuals' data that can be a hard drive on user's PC, a hard drive on a server, a usb stick, or a cloud storage account such as Google Drive, Box, Dropbox or Amazon S3.

Distributed vs. central server These terms are more about the defined policies about data storage/retrieval on the server and who controls it rather than the physical location of the server. As an example, user data is stored on central servers of a company such as Facebook and there could be data replication for availability and locality, but all the servers are controlled and monitored by the company.

1.2 From social influence to influence maximization

As previously stated, individuals are influenced by their social connections in real life. One example is our tendency to buy a product that is accepted by the majority

of our friends. Marketing companies use the word-of-mouth effect to gain a large number of adoptions in selling their products. Richardson et al. [RD02] defined this effect as a fundamental algorithmic problem in the OSN context: what is the best subset of individuals being selected and convinced to adopt a new product to gain a larger cascade of future adoptions? Kempe et al. [KKT03] defined this problem as an optimization problem called influence maximization and they proved its NP-hardness. The problem is defined as

> INFLUENCE MAXIMIZATION IN OSN
> *Input*: A graph $G = (V, E)$ and a positive integer $k > 0$.
> *Output*: A subset $S \subseteq V$ containing k nodes, such that, as the spread of influence starts from S, the set of nodes that get activated, will be maximized.

Then they modelled the spread of ideas and innovations through OSNs using two basic models: Linear Threshold (LT) and Independent Cascade (IC). They proved that the spread computation function f, that calculates the spread value of a node in a social graph, is submodular for both LT and IC models. Submodularity means that the marginal gain from adding an element to a set S is at least as high as the marginal gain from adding the same element to a superset of S [KKT03].

$$f(S \cup v) - f(S) \geq f(T \cup v) - f(T),$$

for all elements v and all pairs of sets $S \subseteq T$. Spread value of a node v is a probabilistic measure to show how many other nodes can be affected by node v if the spread of information starts from node v.

The other outcome of their work was a greedy hill-climbing algorithm that approximates the optimum solution to within a factor of $(1 - 1/e)$. The main idea of their algorithm was starting from an empty set, repeatedly adding an element

to the set that gives the maximum marginal gain. However, their approach was impractical for large networks.

Having the influence maximization problem defined formally, researchers began to present approximation algorithms for finding the top k influential nodes more efficiently and accurately. The authors of the CELF (Cost-Effective Lazy Forward) algorithm [LKG+07] improved the running time of the greedy algorithm. They calculated the spread value of all nodes in the first iteration of the algorithm, and in the further iterations, the spread value of some — not all — nodes is calculated. However, the time complexity bottleneck of CELF is the first iteration. There are some other approaches that construct a data structure for each node and localize the search spaces of computing the influence spread to smaller subsets of nodes to improve the performance of the greedy algorithm. Constructing a DAG (Directed Acyclic Graph) [CYZ10], extract the vertex cover set [GLL11] and clustering the input graph into communities [KLPL13, BHZR16] are in this category.

The other category of research for influence maximization problem is defining propagation models that mostly are the extensions of LT and IC models. In competitive influence maximization (CIM), two or more competitors try to gain more influence spread in a network by choosing fewer nodes. In [CNWVZ07, SBV+12], the CIM problem is looked from the follower's perspective: there are two competitors trying to find some influential nodes. The second competitor starts the process with knowledge of the seed nodes selected by the first competitor and tries to find some new seed nodes other than the ones selected by the first competitor to achieve more influence spread. In [CCC+11, HSCJ12], the two competitors start spreading the information and one competitor tries to block the effect of the second one. From the host's perspective, the owner of the OSN is responsible for allocating some nodes to each competitor in a fair way, discussed in [LBGL13]. We defined a more realistic scenario for CIM problem [BSKW17] in which the individuals have the ability to think about the incoming influence spread for some time steps and then

be adopted by the spread which is accepted by the majority of their neighbours.

1.3 Private online social network

For solving the influence maximization problem that is reviewed in Section 1.2, we need to have a graph consisting of nodes as the individuals and the edges as the relations between them. Such a graph structure is generated by OSN providers from data and metadata authorized by them. Upon request, the providers can also share more information about individuals to the requesters, which can be advertisement companies, researchers, or governments. This is due to the centralized nature of user data, which is stored on the company's storages and is controlled by one single authority. Users' concerns about their private data increases in Healthcare Social Networks (HSNs) as their personal health data are stored on OSN storage and are vulnerable to any kind of breaches. These concerns go even beyond users'. The Canadian Internet Policy and Public Interest Clinic's (CIPPIC) letter to *Facebook* [Den09] about default privacy settings, collection and use of users' personal information for advertising purposes, disclosure of users' personal information to third-party application developers, and collection and use of non-users' personal information is an example of this kind.

Violating users' privacy in existing OSNs encourages us to find the answer to this question: how can we protect user data in a practical way in the systems with sharing, collaboration and interaction mechanisms so that the data can be accessed by third parties only if the user – as the real data owner – is willing to? We started by investigating the existing approaches that try to address the privacy concerns of OSNs (see Chapter 4). These approaches are categorized into three main groups based on the architecture of OSNs: centralized, decentralized and hybrid. Centralized approaches put their trust in OSN servers and try to preserve users' privacy by sending encrypted data directly or an indirect reference to the data

to OSN servers. Lockr [TGS+08], FlyByNight [LB08], NOYB [GTF08], Facecloak [LXH09], Persona [BBS+09], EASiER [JMB11] and CP2 [RMJ13] are examples of the works in this category.

In decentralized approaches, user data is stored on untrusted servers, users' trusted storages, or friends' storage. Most of these approaches build their decentralized or peer-to-peer (P2P) network on public Distributed Hash Tables (DHTs) and use the DHT nodes for storage, such as PeerSoN [BSVD09], DECENT [JNM+12], Cachet [NJM+12] and PESCA [RJM15]. In some approaches such as Safebook [Str09], Soup [KLF14], Didusonet [GADS+16] and Narendula et al.'s works [NPA10, Nar12], user data is stored on their trusted friends' storages by applying some limitations in the space usage. Among these, just a few of them can provide searching and indexing in the P2P environment [NPA10], which is a critical requirement fo OSNs. In hybrid approaches, user data or parts of it is stored based on users' choices, that can be on personal or untrusted servers, such as Vis-a-vis [SLC+11], Confidant [LSC+11], Raji et al.'s [RMJM11] and Wilson et al.'s [WSW+11] works.

1.4 Private filesystem

None of the proposed approaches discussed in Section 1.3 can guarantee confidentiality, availability and integrity, and some are sacrificed for the others. However, we believe that by providing these properties not in the application level, but in lower levels like a filesystem, we can create a functional system that can be the backbone for higher-level applications as diverse as OSNs, redaction systems and censor-resistant systems. In the filesystem level, we can control the data storage/retrieval and define suitable policies. Moreover, we have the flexibility to decouple the storage from access control to provide confidentiality, availability and integrity, as discussed in Chapters 5 and 6. Having such a filesystem, the existing applications can have all the required properties just by interacting with it without applying

substantial modifications to their code.

The main concerns of traditional filesystems are efficient read and write operations and high availability, which are well-addressed in copy-on-write ZFS [BAH+03] and Coda [SKK+90] filesystems. But privacy remains untouched. Ivy [MMGC02] tries to embed privacy in its design, but sacrifices confidentiality. Ori [MBHM13] could address most of the stated requirements with some extra features such as synchronization, failures handling and data recovery. Another successful modern filesystem is IPFS [Ben14] that synthesizes the key successful ideas behind systems such as DHTs [SMK+01], BitTorrent [Coh03], Git [LM12], and SFS [MK98].

Studying these filesystems inspired us to design and implement *UPSS: the user-centric private sharing system* that can provide secure and selective sharing in collaborative environments (Chapters 5 and 6). Also, UPSS provides a conventional filesystem API using copy-on-write operations around immutable DAGs; this API is accessible directly as an embedded library or proxied via a FUSE interface. Storing everything as fixed-size encrypted blocks on block stores, which are untrusted content-addressable storages, without leaving any footprint about the data structures, guarantees the confidentiality and privacy of blocks' content. In Section 6.5.5, we show that the access patterns can be exposed to an adversary in the current implementation of UPSS; however they can be protected in the next versions of UPSS. The block stores can be individuals' local storages, remote servers, or in scenarios that availability is a strong need, cloud accounts. Using convergent encryption [DAB+02], naming the blocks by the cryptographic hashes of their ciphertext, and embedding the key and the block hash in block pointers, we avoid any central key server. Our extensive evaluation (Section 6.3) shows that UPSS is comparable and in some cases superior to heavily-optimized local, network and global filesystems such as ZFS, NFS and Google's Perkeep. Interacting with UPSS using the public API enables us to build a version control systems with fine-grained access controls (Section 6.3.5).

1.5 Contributions

In this thesis, I started by studying how information can be propagated in OSNs. The outcome of my studies is published in two papers [BHZR16, BSKW17], which are presented in Chapters 2 and 3, respectively. I co-authored these two papers with other people, whose names are listed in the papers, and I was the principal author. In the paper that is included in Chapter 2, I have the following contributions:

- Designing a new framework to incorporate the community structure of online social networks for solving the influence maximization problem
- Developing and evaluating an scalable approximation algorithm called *IN-CIM* (Influential Nodes using Community structure for solving Influence Maximization problem) for finding the top-k influential nodes efficiently with low memory usage

The contributions for the paper that is appeared in Chapter 3 are:

- Introduce a new propagation model called *DCM* (Decidable Competitive Model) for competitive influence maximization problem that gives the decision-making ability to the users
- Develop and evaluate an efficient approximation algorithm called *CI2* to find the influential nodes for competitive influence maximization, based on the DCM model
- Prove the *NP*-Hardness of CI2 by considering the DCM as the propagation model

The content of Chapter 4 explains my investigations about privacy-preserving online social networks and comparing existing approaches from different points of view. This paper is ready to be submitted and all the content is written by me and is edited by Dr. Saeed Samet and Dr. Antonina Kolokolova.

Our cryptographic and distributed filesystem, UPSS: the user-centric private sharing system, is described and evaluated in Chapters 5 and 6. The contents of these chapters are published and submitted respectively, as two papers, which I co-authored as the principal author. The contributions of these two chapters are as follows:

- Design and implement UPSS: the user-centric private sharing system as a mechanism for sharing information securely and selectively without having complete trust in central servers
- Implement a cryptographic hybrid (local, network, global) filesystem called `upss-fuse` on top of UPSS
- Design and implement a novel and confidential version control system called UVC: UPSS Version Control System with flexible and fine-grained access controls
- Design and implement a benchmark framework for filesystem evaluation, that can record a filesystem's behaviour during time

Bibliography

[AG17] Hunt Allcott and Matthew Gentzkow. Social media and fake news in the 2016 election. *Journal of economic perspectives*, 31(2):211–36, 2017.

[Ali18] Masih Alinejad. *The Wind in My Hair: My Fight for Freedom in Modern Iran*. Little, Brown, 2018.

[AR98] Philip E Agre and Marc Rotenberg. *Technology and privacy: The new landscape*. Mit Press, 1998.

[BAH+03] Jeff Bonwick, Matt Ahrens, Val Henson, Mark Maybee, and Mark Shellenbaum. The zettabyte file system. In *Proc. of the 2nd Usenix Conference on File and Storage Technologies*, volume 215, 2003.

[BBS+09] Randy Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. Persona: an online social network with user-defined privacy. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 135–146. ACM, 2009.

[Ben14] Juan Benet. IPFS-content addressed, versioned, P2P file system. *arXiv preprint arXiv:1407.3561*, 2014.

[BHZR16] Arastoo Bozorgi, Hassan Haghighi, Mohammad Sadegh Zahedi, and Mojtaba Rezvani. INCIM: A community-based algorithm for influence maximization problem under the linear threshold model. *Information Processing & Management*, 52(6):1188–1199, 2016.

[BM19] Alexandre Bovet and Hernán A Makse. Influence of fake news in twitter during the 2016 us presidential election. *Nature communications*, 10(1):1–14, 2019.

[BSKW17] Arastoo Bozorgi, Saeed Samet, Johan Kwisthout, and Todd Wareham. Community-based influence maximization in social networks under a competitive linear threshold model. *Knowledge-Based Systems*, 134:149–158, 2017.

[BSVD09] Sonja Buchegger, Doris Schiöberg, Le-Hung Vu, and Anwitaman Datta. Peerson: P2p social networking: early experiences and insights. In *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, pages 46–52. ACM, 2009.

[CCC+11] Wei Chen, Alex Collins, Rachel Cummings, Te Ke, Zhenming Liu, David Rincon, Xiaorui Sun, Yajun Wang, Wei Wei, and Yifei Yuan.

Influence maximization in social networks when negative opinions may emerge and propagate. In *Proceedings of ICDM*, pages 379–390. SIAM, 2011.

[CF07] Nicholas A Christakis and James H Fowler. The spread of obesity in a large social network over 32 years. *New England journal of medicine*, 357(4):370–379, 2007.

[CF09] Nicholas A Christakis and James H Fowler. *Connected: The surprising power of our social networks and how they shape our lives*. Little, Brown Spark, 2009.

[CH96] Bruce Christianson and William S Harbison. Why isn't trust transitive? In *International workshop on security protocols*, pages 171–176. Springer, 1996.

[CNWVZ07] Tim Carnes, Chandrashekhar Nagarajan, Stefan M Wild, and Anke Van Zuylen. Maximizing influence in a competitive social network: a follower's perspective. In *Proceedings of the ninth international conference on Electronic Commerce*, pages 351–360. ACM, 2007.

[Coh03] Bram Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72, 2003.

[CYZ10] Wei Chen, Yifei Yuan, and Li Zhang. Scalable influence maximization in social networks under the linear threshold model. In *Proceedings of ICDM*, pages 88–97. IEEE, 2010.

[DAB+02] John R Douceur, Atul Adya, William J Bolosky, P Simon, and Marvin Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Proceedings 22nd international conference on distributed computing systems*, pages 617–624. IEEE, 2002.

[Den09] Elizabeth Denham. *Report of findings into the complaint filed by the Canadian internet policy and public interest clinic (CIPPIC) against facebook inc. Under the personal information protection and electronic documents act*. Office of the Privacy Commissioner of Canada, 2009.

[DG10] George Danezis and Seda Gürses. A critical review of 10 years of privacy technology. *Proceedings of surveillance cultures: a global surveillance society*, pages 1–16, 2010.

[DG12] Claudia Diaz and Seda Gürses. Understanding the landscape of privacy technologies. *Extended abstract of invited talk in proceedings of the Information Security Summit*, pages 58–63, 2012.

[Dun98] Robin IM Dunbar. The social brain hypothesis. *Evolutionary Anthropology: Issues, News, and Reviews: Issues, News, and Reviews*, 6(5):178–190, 1998.

[FC08] James H Fowler and Nicholas A Christakis. Dynamic spread of happiness in a large social network: longitudinal analysis over 20 years in the framingham heart study. *Bmj*, 337:a2338, 2008.

[GADS+16] Barbara Guidi, Tobias Amft, Andrea De Salve, Kalman Graffi, and Laura Ricci. Didusonet: A p2p architecture for distributed dunbar-based social networks. *Peer-to-Peer Networking and Applications*, 9(6):1177–1194, 2016.

[GLL11] Amit Goyal, Wei Lu, and Laks VS Lakshmanan. Simpath: An efficient algorithm for influence maximization under the linear threshold model. In *Proceedings of ICDM*, pages 211–220. IEEE, 2011.

[GTF08] Saikat Guha, Kevin Tang, and Paul Francis. Noyb: Privacy in online social networks. In *Proceedings of the first workshop on Online social networks*, pages 49–54. ACM, 2008.

[Har14] Yuval Noah Harari. *Sapiens: A brief history of humankind*. Random House, 2014.

[HSCJ12] Xinran He, Guojie Song, Wei Chen, and Qingye Jiang. Influence blocking maximization in social networks under the competitive linear threshold model. In *Proceedings of ICDM*, pages 463–474. SIAM, 2012.

[JMB11] Sonia Jahid, Prateek Mittal, and Nikita Borisov. Easier: Encryption-based access control in social networks with efficient revocation. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 411–415. ACM, 2011.

[JNM+12] Sonia Jahid, Shirin Nilizadeh, Prateek Mittal, Nikita Borisov, and Apu Kapadia. Decent: A decentralized architecture for enforcing privacy in online social networks. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, pages 326–332. IEEE, 2012.

[KKT03] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of SIGKDD*, pages 137–146. ACM, 2003.

[KLF14] David Koll, Jun Li, and Xiaoming Fu. Soup: an online social network by the people, for the people. In *Proceedings of the 15th International Middleware Conference*, pages 193–204. ACM, 2014.

[KLPL13] Chungrim Kim, Sangkeun Lee, Sungchan Park, and Sang-goo Lee. Influence maximization algorithm using markov clustering. In *Database Systems for Advanced Applications*, pages 112–126. Springer, 2013.

[LB08] Matthew M Lucas and Nikita Borisov. Flybynight: mitigating the

privacy risks of social networking. In *Proceedings of the 7th ACM workshop on Privacy in the electronic society*, pages 1–8. ACM, 2008.

[LBGL13] Wei Lu, Francesco Bonchi, Amit Goyal, and Laks VS Lakshmanan. The bang for the buck: fair competitive viral marketing from the host perspective. In *Proceedings of SIGKDD*, pages 928–936. ACM, 2013.

[LKG+07] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *Proceedings of SIGKDD*, pages 420–429. ACM, 2007.

[LM12] Jon Loeliger and Matthew McCullough. *Version Control with Git: Powerful tools and techniques for collaborative software development*. " O'Reilly Media, Inc.", 2012.

[LSC+11] Dongtao Liu, Amre Shakimov, Ramón Cáceres, Alexander Varshavsky, and Landon P Cox. Confidant: protecting osn data without locking it up. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 61–80. Springer, 2011.

[LXH09] Wanying Luo, Qi Xie, and Urs Hengartner. Facecloak: An architecture for user privacy on social networking sites. In *Computational Science and Engineering, 2009. CSE'09. International Conference on*, volume 3, pages 26–33. IEEE, 2009.

[MBHM13] Ali José Mashtizadeh, Andrea Bittau, Yifeng Frank Huang, and David Mazieres. Replication, history, and grafting in the Ori file system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 151–166. ACM, 2013.

[MK98] David Mazieres and M Frans Kaashoek. Escaping the evils of centralized control with self-certifying pathnames. In *Proceedings of the 8th ACM SIGOPS European workshop on Support for composing distributed applications*, pages 118–125. ACM, 1998.

[MMGC02] Athicha Muthitacharoen, Robert Morris, Thomer M Gil, and Benjie Chen. Ivy: A read/write peer-to-peer file system. *ACM SIGOPS Operating Systems Review*, 36(SI):31–44, 2002.

[Nar12] Rammohan Narendula. The case of decentralized online social networks. Technical report, Technical report, EPFL, 2012.

[NJM+12] Shirin Nilizadeh, Sonia Jahid, Prateek Mittal, Nikita Borisov, and Apu Kapadia. Cachet: a decentralized architecture for privacy preserving social networking with caching. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 337–348. ACM, 2012.

[NPA10] Rammohan Narendula, Thanasis G Papaioannou, and Karl Aberer. Privacy-aware and highly-available osn profiles. In *Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on*, pages 211–216. IEEE, 2010.

[RD02] Matthew Richardson and Pedro Domingos. Mining knowledge-sharing sites for viral marketing. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 61–70, 2002.

[RJM15] Fatemeh Raji, Mohammad Davarpanah Jazi, and Ali Miri. Pesca: a peer-to-peer social network architecture with privacy-enabled social communication and data availability. *IET Information Security*, 9(1):73–80, 2015.

[RMJ13] Fatemeh Raji, Ali Miri, and Mohammad Davarpanah Jazi. Cp2: Cryptographic privacy protection framework for online social networks. *Computers & Electrical Engineering*, 39(7):2282–2298, 2013.

[RMJM11] Fatemeh Raji, Ali Miri, Mohammad Davarpanah Jazi, and Behzad Malek. Online social network with flexible and dynamic privacy policies. In *Computer Science and Software Engineering (CSSE), 2011 CSI International Symposium on*, pages 135–142. IEEE, 2011.

[SBV+12] Shahrzad Shirazipourazad, Brian Bogard, Harsh Vachhani, Arunabha Sen, and Paul Horn. Influence propagation in adversarial setting: how to defeat competition with least amount of investment. In *Proceedings of CIKM*, pages 585–594. ACM, 2012.

[SKK+90] Mahadev Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on computers*, 39(4):447–459, 1990.

[SLC+11] Amre Shakimov, Harold Lim, Ramón Cáceres, Landon P Cox, Kevin Li, Dongtao Liu, and Alexander Varshavsky. Vis-a-vis: Privacy-preserving online social networking via virtual individual servers. In *2011 Third International Conference on Communication Systems and Networks (COMSNETS 2011)*, pages 1–10. IEEE, 2011.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[Str09] Thorsten Strufe. Safebook: A privacy-preserving online social net-

work leveraging on real-life trust. *IEEE Communications Magazine*, page 95, 2009.

[TGS+08] Amin Tootoonchian, Kiran Kumar Gollu, Stefan Saroiu, Yashar Ganjali, and Alec Wolman. Lockr: social access control for web 2.0. In *Proceedings of the first workshop on Online social networks*, pages 43–48. ACM, 2008.

[WB90] Samuel D Warren and Louis D Brandeis. The right to privacy. *Harvard law review*, pages 193–220, 1890.

[Wes68] Alan F Westin. Privacy and freedom. *Washington and Lee Law Review*, 25(1):166, 1968.

[WSW+11] Christo Wilson, Troy Steinbauer, Gang Wang, Alessandra Sala, Haitao Zheng, and Ben Y Zhao. Privacy, availability and economics in the polaris mobile social network. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, pages 42–47. ACM, 2011.

Chapter 2

INCIM: A community-based algorithm for influence maximization problem under the linear threshold model

(This chapter is based on a paper published in Information Processing & Management, 2016 [BHZR16])

A social network is modeled as a graph $G = (V, E)$ where V and E are the set of nodes and edges of G. In a real world social network, people are modelled as the set of nodes V and the relationship between them, e.g., friendship or being co-tagged in a photo is modelled as the set of edges E of the graph. Information can propagate via links between people which leads to word-of-mouth advertising and its famous application, viral marketing. In viral marketing, the owner of a product, gives free or discounted samples of a product to a group of people to gain a large number of adoptions through the word-of-mouth effect. The influence maximization problem is motivated by the idea of viral marketing. Kempe et al. [KKT03] defined the influence maximization problem in a graph $G = (V, E)$ as finding a subset of $S \subseteq V$

containing of k nodes, such that, as the spread of influence starts from S, the set of nodes that get activated, will be maximized.

Two propagation models are defined in [KKT03] by Kempe *et. al.*, namely, the Linear Threshold (LT) and Independent Cascade (IC) models. Also, different algorithms have been proposed based on the IC model[CWW10], [CSH$^+$14], [GZZ$^+$13], [KLPL13] and LT model[CYZ10], [GLL11b], [GLL11a]. Our proposed algorithm in this paper is based on the Linear Threshold model.

The greedy algorithm [KKT03] needs to calculate the influence of every node in the graph which is very time consuming for large graphs. In recent years, several attempts have been made to solve the influence maximization problem more efficiently. In CELF [LKG$^+$07], first, the spread of each node is calculated and in the next iterations, based on the sub-modularity of spread function, only the spread of some nodes needs to be updated that causes CELF algorithm to run faster than the greedy algorithm. However, the time complexity bottleneck of CELF [LKG$^+$07] is the first iteration.

In some approaches, the authors try to compute the spread of nodes in smaller subgraphs to have a faster first iteration than CELF algorithm [LKG$^+$07] and also, improve the overall running time of their algorithms [GLL11b, YKK13, KLPL13]. In IPA [YKK13], the influence paths starting from each node v to other nodes are found and each influence path is considered as an influence evaluation unit and the node with maximum influence propagation probability over influence paths is a candidate for a seed node. However, the results of our experiments in Section 2.4 show that the quality of seeds in IPA is not satisfying.

In SIMPATH [GLL11b], to avoid calculating the spread of every node in the graph, the vertex cover of the graph is computed and the spread of each node in the vertex cover is calculated. In this paper, we improve the running time of the SIMPATH algorithm [GLL11b] by using the community structure of the input graph.

Despite useful and meaningful characteristics of communities in social networks, very little attention has been made to incorporate the role of communities in influence maximization problem. In real social networks, people live in a cluster with whom they have strong relations. Information circulates at a high velocity within these clusters and each person tends to know what the other people know. Therefore, the spread of information on new ideas and opportunities must come through the weak ties that connect people in separate clusters. The weak ties so often ignored by social scientists are in fact a critical element of social structure. Weak ties are essential to the flow of information that integrates otherwise disconnected social clusters into a broader society [Gra73]. Furthermore, some communities in social networks play a significant role, as they are central and other communities monitor them to get the updates. Moreover, real networks contain a huge number of nodes, and computing the influence of each node is very expensive. Calculating the spread of each node locally inside its community can be done very quickly which improves the running time.

Kim et al. [KLPL13] approached the influence maximization problem from a community based perspective and their main reason is to limit the search space to some nodes inside communities and decrease the running time. In [KLPL13], first the graph is clustered into a set of communities, and then the most influential node in each community is considered as the seed candidate for the influence maximization problem. Finally, only k nodes from candidate nodes are chosen as final seed nodes. In this model, the role of communities is ignored. For example, if some communities are very small and have no influence on other communities, the quality of seeds in that communities is expected to be lower.

Contribution. Our major contribution in this research is summarized as follows:

- We design a new framework to incorporate community structure in the influence maximization problem.
- We devise an efficient algorithm for finding the top-k influential nodes in a

social network based on communities in the graph.

Methodology. In this paper, we incorporate the role of communities of the social network in a meaningful manner and propose an algorithm under Linear Threshold model. In our algorithm, first we find the local spread of each node inside its community. Then we construct the graph of communities, where each community is a vertex and there is an edge between two vertices if there is at least one edge between the corresponding communities in the actual graph. After that, we compute the global influence that is the spread of each community in the graph of communities and finally calculate the final spread of each node as a combination of its local spread and its global spread. So, a particular node that belongs to a community with higher spread, has more chance to be chosen as a seed node candidate.

We conduct the empirical studies on large real graphs. Extensive performance studies demonstrate that the proposed algorithm significantly outperforms the state-of-the-art algorithms in term of the quality of outputted seeds while still has an acceptable running time and memory usage.

Organization. The rest of this paper is organized as follows. In Section 2.1, we summarize the related works, and in Section 2.3, we devise a new algorithm for finding the top influential nodes in the graph called INCIM. Section 2.4 introduces the datasets which are used to examine our algorithm and presenting the experimental results. The last section is devoted to conclusions and some directions for future work.

2.1 Related Work

In the previous section, we mentioned some works closer to ours and in this section, these works and other related works are reviewed in a more detailed way. The greedy algorithm presented by Kempe et al. [KKT03] can guarantee a good quality of seeds, but it is very time consuming for large graphs. Therefore, efficiently and

accurately finding the influential nodes in social networks, under the LT and IC models, has recently drawn a great deal of attention.

CELF algorithm presented in [LKG+07] owes to the fact that the influence spread function is a sub-modular function, which means the marginal gain from adding an element to a set S is at least as high as the marginal gain from adding the same element to a superset of S [KKT03]. Therefore, CELF reduces the number of calls to the spread estimation function (because only the spread of some nodes needs to be updated in each iteration) and improves the running time of the greedy algorithm. However, the time complexity bottleneck of CELF is the first iteration, where the spread of every node is calculated and thus the CELF's first iteration is the same as the greedy's first iteration.

In the heuristic approach in [CYZ10], for each node v, a DAG (Directed Acyclic Graph) is constructed and the spread of nodes is computed locally within the resulting DAGs and the seed nodes are selected based on the greedy algorithm. This algorithm has a reasonable running time, but it takes a lot of memory to store a DAG for each node in the graph. The main idea of the algorithm presented in [CWW10] is making local trees with edges starting or ending at each node called MIA; then the probability of activating a node by other nodes in each tree is computed locally. Finally a node with maximum probability is chosen as seed node.

In SIMPATH [GLL11b], to reduce the number of calls to the influence estimation function, first the vertex cover of G is computed, and only the spreads of nodes within the vertex cover are calculated. Having the spread values, the seed nodes are found in k iterations. In IMRank [CSH+14], an initial rank is assigned to every node in the graph which is computed from a known heuristic; then, based on these initial ranks, the spreads of nodes are estimated, while in the greedy algorithm the spreads are computed exactly in each iteration. In IPA [YKK13], for each node v in the graph, the algorithm finds the influence paths starting from v to other nodes and computes an influence propagation probability for each node. The nodes

with maximum influence propagation probability are selected as seed nodes. IPA algorithm runs very fast and also is parallelizable to run faster in multicore systems, but it achieves seed nodes with low quality over different datasets comparing to other algorithms.

In [GZZ+13], the influence maximization problem is viewed from a new perspective. More specifically, k nodes are chosen as seed sets that have maximum influence over a set of target nodes to obtain a personalized set of influential nodes. The personalized influence is specific to its own applications where the seeds are selected from a general social network for a specific type of product.

Very closely related to our work is [KLPL13], where communities of the graph are constructed by Markov Clustering algorithm. Then, in each community, a node with maximum spread is chosen as the seed node candidate. Finally, k nodes with bigger influence spread are chosen from candidate nodes. The main problem with this algorithm is that, it does not consider the role of each community as a unit to spread the influence and the size of each community.

2.2 Preliminaries

In this part, we introduce fundamental concepts of influence calculation which are needed for better understanding of the paper.

Community detection. Communities are subsets of nodes in the graph, with more edges between them and fewer edges between nodes in different communities [LP49]. Community detection is formulated as a clustering problem. That is, given the full graph $G = (V, E)$, partition the vertex set into k subsets $S_1, S_2, \cdots, S_k$, such that $\bigcap_{i=1}^{k} S_i = \emptyset$ and $\bigcup_{i=1}^{k} S_i = V$. A quality metric $Q(S_1, \cdots, S_k)$ is defined over the partitions and a community detection algorithm will try to find a partitioning that maximize or minimize Q depending on its nature. This is for non-overlapping community detection and one can simply remove the constraint $\bigcap_{i=1}^{k} S_i = \emptyset$ to get

the overlapping version [HCL14].

Linear Threshold model. In this model which is defined in [KKT03], activation probability of a node depends on a uniform function of its neighbors that are activated before. More specifically, a weight function f_v maps the neighbors of v to $[0, 1]$, then assigns a threshold value $\theta_v \in [0, 1]$ to each node v uniformly at random. Node v would be activated in time t if $f_v(S) > \theta_v$, where S is the subset of neighbors v that have been activated in time $t - 1$. Based on [KKT03], the value of f_v is initialized as Equation (2.1):

$$f_v(S) = \sum_{u \in S} b_{v,u} \tag{2.1}$$

where $b_{v,u}$ is the weight of edge (v, u) and sum of all edges between v and its neighbors should be less than 1 as Equation (2.2).

$$\sum_{u \ neighbors \ of \ v} b_{v,u} \leq 1 \tag{2.2}$$

In some cases, the weight of the edges are not set in the input graph and they should be computed. To do so, the number of edges from node i to node j is defined as w_{ij} (in this chapter, we have one edge between each two nodes, therefore $w_{ij} = 1$). Also, the number of input edges to node j is defined as d_j. Finally the weight of edge (i, j) in LT model is as follows:

$$\frac{w_{ij}}{d_j}. \tag{2.3}$$

2.3 INCIM algorithm

In this section, we devise a new algorithm, called INCIM (Influential Nodes using Community structure for solving Influence Maximization problem) for the influence maximization problem from a community-based perspective. First, we present a general overview of the algorithm and then we explain each part in more details.

2.3.1 Algorithm overview

Given a set of communities $C = \{C_1, C_2, \cdots, C_l\}$ in a graph $G = (V, E)$, such that $C_1 \cup C_2 \cup \cdots \cup C_l = V$, the intuition of INCIM can be explained using the characteristics of community structure in G. Our algorithm proceeds in two phases. In the first phase (preprocessing), we find the communities of the main graph by a partition algorithm. In the second phase, we first find the influence of each community among other communities based on the links between their nodes and then find the degree to which each node spreads the influence inside its own community. We determine the final spread of each node in the graph as a combination of its community influence and its influence inside its community.

Our innovations (which will be explained in Section 2.3.2 in more detail) to decrease the running time and increase the efficiency of our algorithm are as follows:

- Computing the spread of nodes locally in the communities which causes a decrease in overall running time of the algorithm. Also, by determining those communities whose nodes spread should be updated, we decrease the number of calls to the SIMPATH algorithm which again results in decreasing the running time.

- By using the idea of local and global spreads of nodes and their combination, we track the role of communities and also, the influence of a node in its own community.

- Using a combination of the SIMPATH [GLL11b] and CELF [LKG+07] algorithms; more precisely, using the SIMPATH algorithm to compute the spread of nodes and storing and updating these spread values by the idea of the CELF algorithm cause our algorithm to find the seed nodes which have the most influence spread in the graph in a reasonable time and less memory usage.

2.3.2 Algorithm description

INCIM contains the following steps:

1. **Preprocessing:** In this step, we partition the input graph into its communities. The partition algorithm we have used is SLPA [XSL11].

2. **Computing spread of nodes:** In this step, we create the graph of communities whose nodes are the communities of the graph and then we calculate the spread values. We use the manner of the SIMPATH algorithm [GLL11b] to compute the spread of nodes which is done in 3 steps as follows:

   (a) Computing global spread which is the spread value of each node in the graph of communities, i.e., the spread values of the communities.

   (b) Computing local spread which is the spread value of the nodes inside their communities.

   (c) Computing final spread which is a combination of local and global spreads per each node.

3. **Making CELF lists:** We store the resulting spread values of the nodes by the idea of the CELF [LKG$^+$07] algorithm. For each community, we have a list which the number of its elements is equal to the number of nodes in that community.

4. **Determining communities to be updated:** We determine which communities should be updated based on the nodes in the seed set.

5. **Selecting seed nodes:** In this step, INCIM iterates k times to finds the k most influential nodes.

Now, we can describe INCIM algorithm by explaining the details of each step.

2.3.2.1 Preprocessing step

To partition the input graph into communities, we use the SLPA algorithm as the partition algorithm which is proposed in [XSL11]. The SLPA is categorized as a dynamic and agent-based algorithm and is an extension of the Label Propagation Algorithm (LPA). Both SLPA and LPA algorithms tries to maximize the modularity measure Q. Modularity measure Q is defined as the difference between the observed density of edges within communities and the expected density of edges within the same communities but with random connections [NG04]. In SLPA, each node can be a listener or a speaker. The roles are switched depending on whether a node serves as an information provider or information consumer. A node can hold as many labels as needed to decide which label to accept. The more a node observes a label, the more likely it will spread that label to the other nodes. The input of the SLPA algorithm is the input graph containing the id of the nodes and the edge weights in the LT model. The weight of the edges are considered when a listener wants to accept a label from its neighbors. For example, a listener x, has three neighbors $y1$, $y2$ and $y3$, with edge weights 0.1, 0.2 and 0.3 respectively. When node x consider the labels from its neighbors, it will weight each label as $0.1/(0.1 + 0.2 + 0.3)$ for node $y1$, $0.2/(0.1 + 0.2 + 0.3)$ for node $y2$ and $0.3/(0.1 + 0.2 + 0.3)$ for node $y3$. Then, the listener accepts one label from the collection of labels received from neighbors following certain listening rule, such as selecting the most popular label from what it observed in the current step.

The main reasons to choose this algorithm are its less time consuming than the other ones in finding communities and its high quality in finding communities of the graph; the experiments done in [XSL11] confirm such specifications of SLPA.

2.3.2.2 Computing spread of nodes

In our framework, we use the approach of the SIMPATH algorithm [GLL11b] to compute the spread of a node in the input graph. As mentioned in [GLL11b], the

spread of a node v is computed by summing the weights of all simple paths starting from v. The weight of a simple path is calculated as the products of the edge weights on the path:

$$W[P] = \prod_{(v_i,v_j)\in P} b_{v_i,v_j} \tag{2.4}$$

In Equation (2.4), P is a path and b_{v_i,v_j} is the weight on edge (v_i, v_j).

Also the spread of a seed set S, that includes the influential nodes, is the sum of that of nodes $u \in S$ in subgraphs induced by $V - S + u$ (considering just one node u a time in S and removing the other nodes and their connections in S other than u). So, by considering simple paths starting from nodes of set S, we can compute the spread of set S in different subgraphs. To find simple paths in the graph, SIMPATH uses BACKTRACK algorithm that is presented in [Kro67, Joh75]. As the problem of enumerating all simple paths is #P-hard [Val79], in the SIMPATH algorithm, only the paths are considered which their weights increase rapidly as the length of the path increases. Thus, the influence can be captured by exploring the paths within a small neighborhood, where the size of the neighborhood can be controlled by a threshold value. The BACKTRACK algorithm gets a node u, a threshold value η and a subgraph as input, find all the simple paths from node u by considering the threshold value and output the spread of the node, which is the sum of the simple path weights. In out algorithm, we use the SIMPATH default value for η, which is 10^{-3}.

The SIMPATH algorithm is shown in Algorithm 2.1. In Algorithm 2.1, S is the seed set, η is the threshold value to control the size of the neighborhood, U is the set of nodes of the input graph and $\delta(S)$ is the spread value of S.

Algorithm 2.1 SIMPATH

Input: S, η, U, V

Output: $\delta(S)$

1: $\delta(S) \leftarrow 0$

2: **for** each $u \in S$ **do**

3: $\quad \delta(S) \leftarrow \delta(S) + BACKTRACK(u, \eta, V - S + u, U)$

4: return $\delta(S)$

Algorithm 2.1 shows how the SIMPATH algorithm computes the spread value of a seed set. In an iterative manner in line 3 of the algorithm, all the nodes of S are given to the BACKTRACK algorithm and their spreads are computed. The final value returned by Algorithm 2.1 is the sum of the spread values of all nodes in set S.

As we mentioned in section 2, CELF algorithm [LKG+07] uses the sub-modularity of the influence spread function. In our algorithm, we take the advantages of CELF algorithm to reduce the number of calls to the SIMPATH algorithm. To achieve this goal, in our approach the spread values of nodes computed by SIMPATH are stored in a list sorted in decreasing order, as shown in Algorithm 2.3. Then, in each iteration of CELF algorithm, only the marginal gain of the top node of the list is re-computed and if needed, the list is resorted. If a node remains at the top, it is picked as the next seed [LKG+07]. As we compute the spread of nodes in the communities of the input graph, we have lists separated per each community. By using the CELF idea to store the spread values and update the lists, we improve the running time of our algorithm.

Next, we formalize the notions of *local spread* and *global spread* of each node and describe the way we determine them in more details.

Global spread Once we identify the communities, we can determine which communities are the central ones since they are monitored by individuals and other

communities that look for ideas, news and innovations. For instance, consider a big community of researchers working on social networks; it can play a central role in spreading ideas and topics for other researchers as they follow this community to monitor the recent trends. Therefore, it makes sense to consider the position of each community inside these research groups and find the degree to which each community can influence other communities. In this way, we can determine central communities and find the influential nodes in these communities.

To determine central communities, we build the graph $G_c = (V_c, E_c)$ of communities and compute the spread of each of its nodes. Each community C_i is a node of G_c and if there is a directed edge from node $u \in C_i$ to node $v \in C_j$ in graph G, then there is a directed edge from the corresponding node C_i to C_j in G_c. We define W_{ij} as the maximum weight of the edges from community C_i to C_j. We consider only the maximum edge weight between the nodes of the graph of communities as the maximum influence spread that we can achieve can flow on the edge with maximum weight. We also count the number of input edges to community C_i shown by d_{C_i} as the degree of this community. Similar to LT model, the weight of edge (C_i, C_j) in G_c is considered as $\frac{W_{ij}}{d_{C_j}}$. At the beginning of the algorithm, a random threshold value is assigned to each node C_i in G_c. As we mentioned before, the model of information propagation which is used for the main graph is LT model, so we should also have the same information spread model for the graph of communities. That is why we use the LT edge weight computation manner for graph of communities.

The spread of nodes in graph $G_c = (V_c, E_c)$ is computed by the SIMPATH algorithm which takes $G_c = (V_c, E_c)$ graph as its input. We call this spread as $\bar{\delta}$.

Local spread Suppose that there is a social network in which communities are constructed based on research interests in the academia. An influential node in a community has more influence on the nodes of its community than it does on the nodes of other communities with different research interests; so, in INCIM

algorithm, each node has a local spread which is its influence inside its community. By computing the spread of nodes locally in the communities, we have an improvement in running time and also, we choose the most influential nodes in their own communities which results in choosing the most influential nodes as the algorithm's final output when combining the local spread with the global spread as will be explained below. The results of our experiments in Section 2.4 confirm this improvements.

In the first iteration of the algorithm, the spread of each node v in a community is computed as $\delta_l(v)$ and stored in list δ_i based on CELF optimization. For each community i, we have a list δ_i whose size is equal to the number of nodes in that community. In other words, the input nodes of the SIMPATH algorithm are the ones inside the communities separately for each community, and SIMPATH finds simple paths in the communities locally. As Equation (2.5), the total number of elements of all δ_i is equal to the number of all nodes in graph G, so the total memory needed by INCIM is the same as CELF algorithm.

$$\sum_{i=1}^{n} |\delta_i| = |V| \tag{2.5}$$

In Equation (2.5), n is the number of communities, $|\delta_i|$ is the size of δ_i and $|V|$ is the number of nodes in graph G.

Since the communities in a network are not necessarily well separated, there are some edges between them. Therefore, individuals in the same community can have different influence on the neighbor communities. To tackle this issue, we consider the notion of border nodes in the graph. A *border node* in a community has at least one edge that connects it to another community in the network and influence can enter or leave a community through border nodes. So, it is important to compute the spread of border nodes over the graph. We call SIMPATH with graph G as its input to compute the border node's spread and store the spread values in a vector called α with an element per each border node. These values will be combined with

local and global spreads to track border node's influence above nodes from their linked communities. In Figure 2.1, node v is a border node as it has a link to another node from another community.

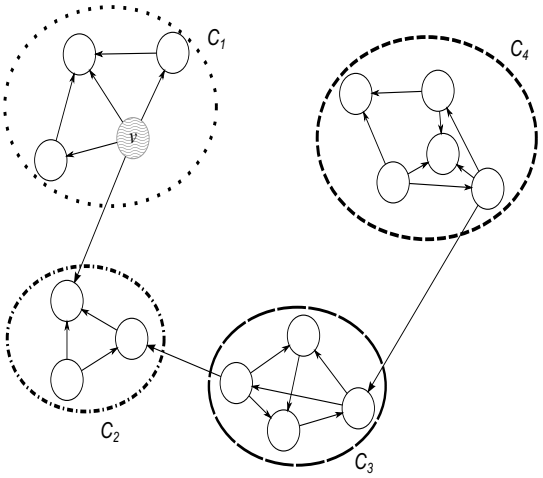


Figure 2.1: A sample network with its communities

Final spread, combination of local spread and global spread Final spread of a node is a combination of its local spread and spread of its community. By final spread, we determine which nodes are the most influential nodes over the graph.

Definition 1. *Given a social network $G = (V, E)$ and a set of communities $\{C_1, C_2, \ldots, C_l\}$, where $\delta_l(v)$ is the local spread of node v in community C_i and $\bar{\delta}(v)$ is the global spread of community C_i which v belongs to, along with $\alpha(v)$ which is the spread values of border nodes, the final Spread of node v is defined as follows:*

$$\delta(v) = \delta_l(v) + \alpha(v).\bar{\delta}(v) \tag{2.6}$$

In Equation (2.6), $\alpha(v)$ is equal to 1 if v is a non-border node; if v is a border node, $\alpha(v)$ is equal to the influence spread from v to the nodes inside its linked communities. As Goyal et al. showed in [GLL11b], the majority of the influence to a node flows in from a small neighborhood and can be captured by enumerating

paths within that neighborhood. Also, Wang *et. al.* discussed in their community-based greedy approach [WCSX10] that the difference between the node's influence degree in its community and its influence degree in the whole network is small. In INCIM, we compute the spread of nodes locally in their own communities, too. But as we discussed earlier, the communities can also influence each other by the links which exist between them and we can not ignore the influence which flows between the nodes belong to different communities. Such influence is considered by combining the global spread with the local spread of each node. The global spread of a community is computed by summing up the edge weights of all simple paths starting from the community. Also, the local spread of a node is computed in the same way. So to combine the local spread and the weighted global spread of a node, we should sum up them together, as we did in Equation (2.6). In Section 2.4.3, we will show that by using Equation (2.6), we can achieve a good approximation for spread computation near to the approximation achieved by simple greedy algorithm [KKT03] which is $(1 - 1/e - \varepsilon)$, or of 63% of the optimized solution.

2.3.2.3 Seed nodes selection

After finishing the first iteration of the algorithm, the node with maximum spread in community i takes the first place of δ_i, and the node with maximum spread between all first elements of all δ_i is chosen as the first seed node.

After choosing the first seed node in the first iteration, the algorithm will be executed $k - 1$ times to find the $k - 1$ remaining seed nodes. In each iteration, the SIMPATH algorithm is called and the δ_i lists would be updated based on the new marginal gains achieved by new nodes. Since the seed nodes are chosen from different communities, the spread of nodes in graph of communities $G_c = (V_c, E_c)$ should be also updated. So, in the beginning of each iteration, SIMPATH is called to compute the spread of nodes of graph $G_c = (V_c, E_c)$.

Suppose that Figure 2.1 is a part of a network whose node v is chosen as a seed

node at time t. As node v belongs to community C_1, the spread of other nodes in community C_1 should be updated for the next iteration. Since there is a simple path from node v to a node in community C_2, so the spread of nodes in community C_2 should be updated, too. But for communities C_3 and C_4, such updates are not required because there is no simple path from v to C_3 and C_4.

The INCIM algorithm calls SIMPATH to compute the spread of nodes in only those communities determined by the *getCommunitiesToUpdate* subroutine; in this way, the number of calls to the SIMPATH algorithm will be decreased remarkably in each iteration. This is one of the contributions for decreasing the running time of the algorithm for the graphs with connected communities structure, rather than separated communities that are not reachable from each other. For the later case, our algorithm should check all the disconnected communities in each iteration, which increases the running time. The *getCommunitiesToUpdate* subroutine is shown in Algorithm 2.2. In Algorithm 2.2, S is the seed set and $G_c(V_c, E_c)$ is the graph of communities.

Algorithm 2.2 *getCommunitiesToUpdate*

Input: $S, G_c(V_c, E_c), G(V, E)$

Output: $UpdLst$

1: $UpdLst \leftarrow \emptyset$
2: $Neighbours \leftarrow \emptyset$
3: **for** each $v \in S$ **do**
4: $C_i =$ the community containing v
5: $UpdLst = UpdLst \cup C_i$
6: $Neighbours =$ every u that there is a simple path from v to u
7: **for** each $u \in Neighbours$ **do**
8: $C_j =$ the community containing u
9: $UpdLst = UpdLst \cup C_j$
10: return $UpdLst$

The $UpdLst$ is the list of communities which should be updated and returned by Algorithm 2.2. In the loop of line 3, the nodes which there is a simple path starting from nodes of set S to them are considered. Then the communities of such nodes and the communities of the nodes of set S are considered as the output of the algorithm.

In the framework proposed in this paper, we consider the role of each community by its global spread. Also, we track the influence spread achieved by each node in its community as local spread. By combining global and local spreads, we choose nodes as seed nodes that have most influence spread in their community and this influence can spread as much as possible to other communities. This is why we claim that our algorithm chooses the most influential nodes with higher quality comparing to other state-of-the-art algorithms. Also, by computing the spread of nodes locally in the communities and using the idea of CELF algorithm [LKG+07], we have improvements in running time and memory usage. The experimental results of Section 2.4 confirm our claims.

INCIM algorithm is shown in Algorithm 2.3. In Algorithm 2.3, $G(V, E)$ is the input graph and S is the set of size k containing seed nodes returned by INCIM algorithm.

In lines 3 and 4 of Algorithm 2.3, the graph of communities is constructed based on the communities discovered by the SLPA algorithm. Then in lines 5–7, the spread values of the nodes in the graph of communities are computed. The influence spread of border nodes is computed in lines 9–11. In line 11, where $\bar{\delta}(v)$ is the global spread of v's community, we multiply the spread value of each border node with the global spread value of its community. In the beginning of each iteration starting from line 12, we call Algorithm 2.2 to determine which communities should be updated in line 14 and store them in list CP. In the first iteration, after calling Algorithm 2.2, CP contains all the communities of the graph because there is no seed node in the beginning of the algorithm. In lines 15–29, for all nodes of the communities which

Algorithm 2.3 $INCIM$

Input: $G(V, E), k, \eta$

Output: S

$S \leftarrow \emptyset$
$CommunitiesSet \leftarrow \emptyset$
call SLPA algorithm to find the communities of G and store them in $CommunitiesSet$
make graph of communities $G_c(V_c, E_c)$ such that each community is a graph node
for each $u \in G_c$ **do**
    call *Algorithm* 2.1 to compute $\bar{\delta}(u)$
    add $\bar{\delta}(u)$ to $\bar{\delta}$ which is ordered decreasingly
find border nodes and store them in $BorderNodesSet$
for each $v \in BorderNodesSet$ **do**
    call *Algorithm* 2.1 to compute $\delta(v)$
    $\delta(v) = \delta(v) * \bar{\delta}(v)$ that v belongs to
while $|S| < k$ **do**
    $CP \leftarrow \emptyset$
    call *Algorithm* 2.2 to determine the communities which should be updated and store them in CP
    $maxSpread = 0$
    $maxNode = \emptyset$
    for each $c \in CP$ **do**
        $\forall x \in \delta_c$ call $SIMPATH(S, \eta, V - x, V)$ to compute $\delta^{V-x}(S)$
    for each $x \in \delta_c$ **do**
        call $SIMPATH(S, \eta, V - S, V)$ to compute $\delta^{V-S}(x)$
        if $x \notin BorderNodesSet$ **then**
            $\alpha(x) = 1$
        $\delta(x) = \delta(x) + \alpha(x) * \bar{\delta}$ that x belongs to
        update δ_c based on the new spread value of x
        u = top node of δ_c
        if $\delta(u) > maxSpread$ **then**
            $maxSpread \leftarrow \delta(u)$
            $maxNode \leftarrow u$
    $S = S \cup maxNode$
return S

should be updated, we compute their spread values by the SIMPATH algorithm, and the node with the maximum spread is chosen as the seed node. In line 18, $\delta^{V-x}(S)$ is the spread value of seed set S in the graph containing nodes in set V except node x and its edges.

2.4 Evaluation

To compare our algorithm with other approaches, we have done our experiments on four real datasets and compared the algorithms based on running time, the quality of seed nodes and memory usage. The code is written in C++ and all experiments are run on a Linux (Ubuntu 10.04) machine with 3.2GHz Intel Xeon CPU and 128GB memory.

2.4.1 Datasets

We have used four real-world datasets to run our experiments on; their specifications are shown in Table 2.1 and available on the SNAP library of Stanford University website[1].Using SLPA algorithm, we find the communities of the datasets to be used by INCIM algorithm. The specifications of the found communities of the datasets are shown in Table 2.2. In both Tables 2.1 and 2.2, the # sign indicates the number of elements.

2.4.2 Algorithms to compare

INCIM: The algorithm presented in this paper.
LDAG: The approach proposed in [CYZ10]. We set the parameter $\theta = 1/320$ as recommended by authors.
SIMPATH: The algorithm in [GLL11b] running with parameters $\eta = 10^{-3}$ and $l = 4$ as recommended by authors.

[1]http://snap.stanford.edu/

Table 2.1: Specifications of real standard datasets

	NetHEPT	**Slashdot**	**Amazon**	**DBLP**
#Nodes	15K	82K	262K	914k
#Edges	62K	948K	1.2M	6.6M
Average out-degree	4.12	9.8	9.4	7.2
Maximum out-degree	64	1527	425	950
#Connected components	1781	1	1	41.5K
Largest component size	6794	82K	262K	789K

Table 2.2: Specification of communities of the datasets

	NetHEPT	**Slashdot**	**Amazon**	**DBLP**
#Communities	2901	12K	32674	73489
#Nodes in the biggest community	407	6050	2852	771
#Edges in the biggest community	2938	16294	7169	3193

IPA: The proposed algorithm in [YKK13] running with $threshold = 0.005$ as recommended by authors.

PageRank: The algorithm proposed in [BP98]. In this paper, nodes with maximum ranking are chosen as seed nodes. The algorithm stops when the score vectors from two consecutive iterations differ by at most 10^{-6} as per $L1 - norm$.

HighDegree: This algorithm [KKT03] chooses the nodes with maximum out-degree as seed nodes.

Our reasons to compare INCIM with the aforementioned algorithms are as follows: PageRank and HighDegree are two well-known and basic algorithms which are compared with most of the other works. SIMPATH is an algorithm with good results in running time, memory usage and quality of seed nodes. Also, LDAG has good results in quality of seed nodes and reasonable response time. At last, IPA uses the idea of communities to find the most influential nodes.

2.4.3 Experimental results

Comparison of memory usage The memory usage of HighDegree and the PageRank algorithms is almost zero because they do not need to store any structures when they are running, but LDAG has the most memory usage among compared algorithms because this algorithm makes a DAG for each node in the graph. INCIM uses higher amount of memory than SIMPATH but lower than LDAG. The results are shown in Figure 2.2 for 50 seed sets.

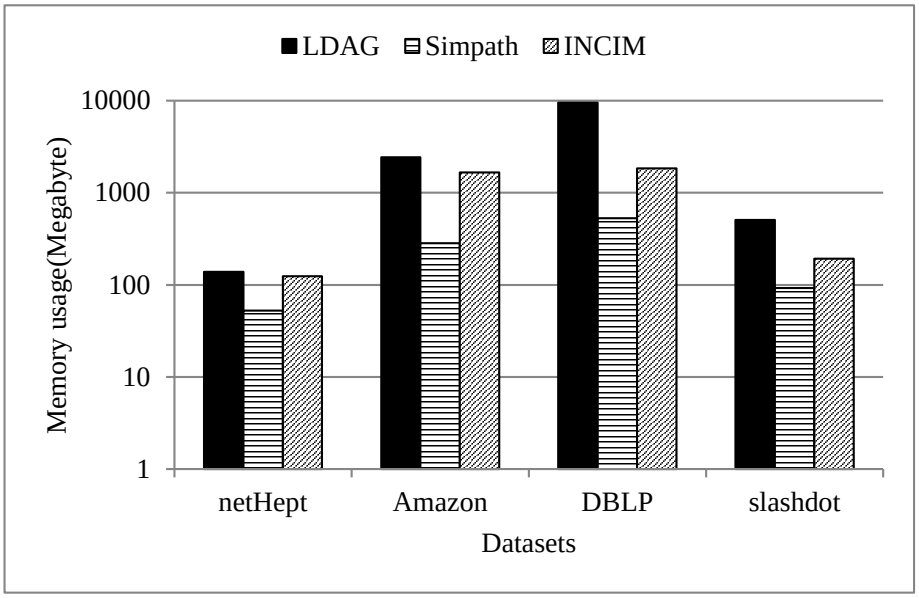
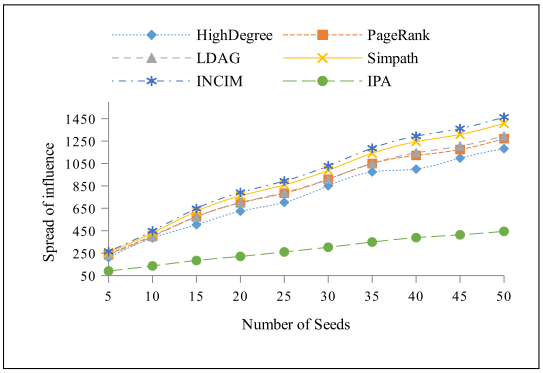


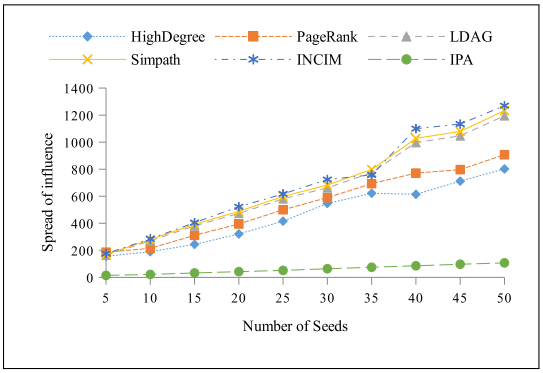
Figure 2.2: Comparison based on memory usage in different algorithms

Comparison of seed set quality An algorithm with higher quality is the algorithm that has higher influence spread. Based on the experimental results shown in Figure 2.3, the INCIM algorithm has the highest quality seed set among other algorithms, except in the Slashdot dataset where the IPA algorithm achieves the best results. While INCIM uses the SIMPATH algorithm to compute the spread of nodes, its spread of seed set is a little better than SIMPATH in most of the cases

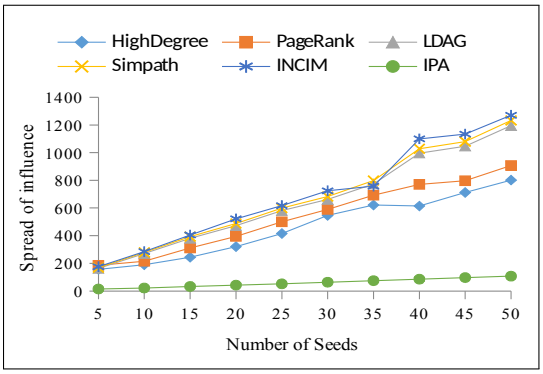
because INCIM uses a combination of local and global spreads and thus tracks both the effect of each node in its community and the effect of each community.



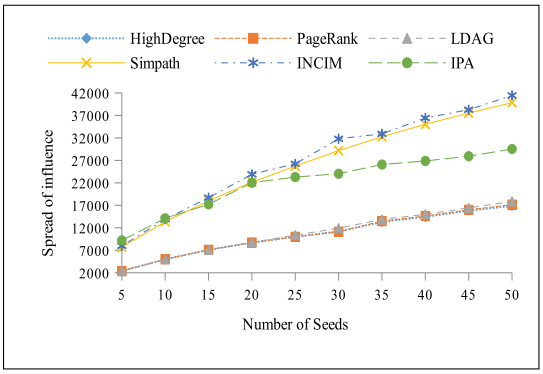
(a) netHept



(b) slashDot



(c) Amazon



(d) DBLP

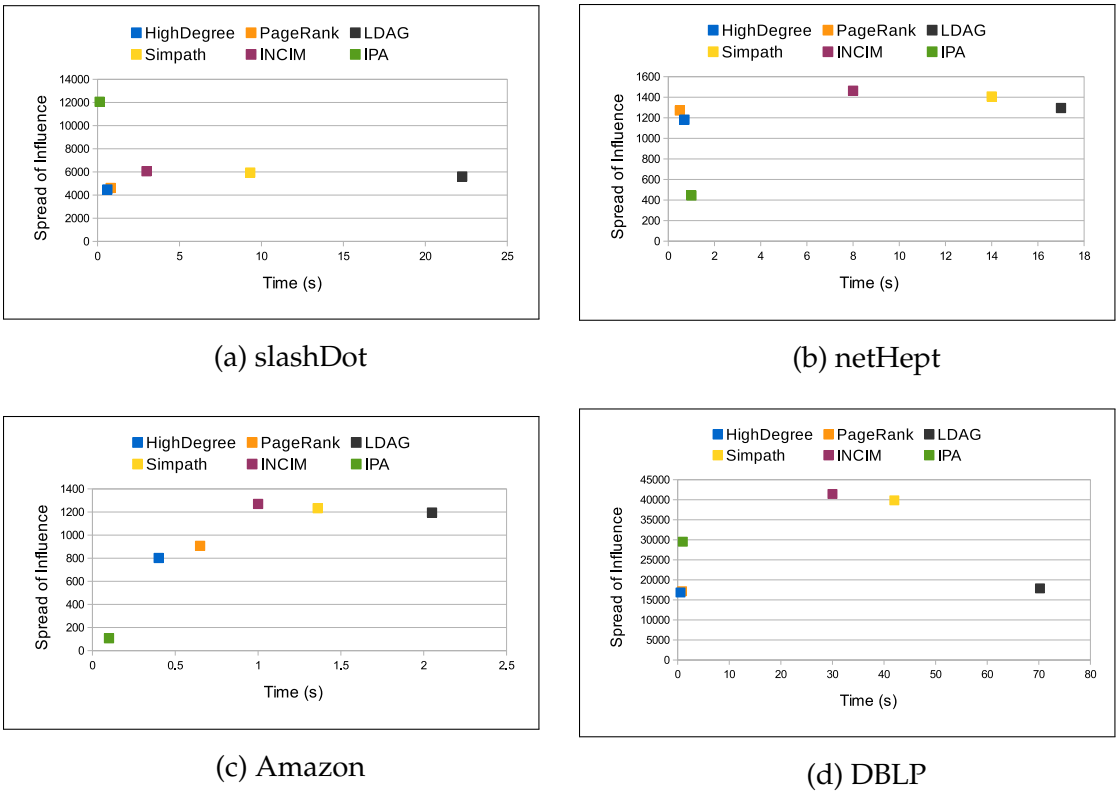
Figure 2.3: Influence spread achieved by various algorithms

The IPA algorithm has lower quality of seed sets than other algorithms in all datasets, except the slashdot dataset. In netHept and Amazon, IPA has the lowest quality of seed set and in DBLP, its quality is higher than the PageRank and HighDegree algorithms but lower than INCIM, SIMPATH and LDAG. As we can see, IPA has an uncertain behavior because for some datasets, the quality of its seed nodes is higher than other algorithms while for most others, it results in lower quality; hence, the IPA algorithm is not so reliable.

Comparison of running times Figure 2.5 shows the results of comparisons based on running times. In the NetHept dataset, the running time of PageRank and HighDegree are considerably low, so the plots for these algorithms in NetHept dataset are omitted. IPA has better running time than INCIM, SIMPATH and LDAG and runs in a time close to that of PageRank and HighDegree. The INCIM algorithm has the best running time among other algorithms, except IPA, PageRank and HighDegree. These three algorithms have better running times, however they fail in finding the seed nodes with high quality, which is the main target of influence maximization problem. They trade off the quality of seed nodes for running time. We plotted the running time versus the influence spread achieved by different algorithms for seed set of size 50, which is the seed size used in our other comparisons and also used by other approaches [GLL11b, CYZ10, YKK13], in Figure 2.4.

As we see in Figures 2.5(a) and 2.5(d), INCIM performs slower than SIMPATH at the beginning of its running time. For instance, in Figure 2.5(a), the time needed to find seed nodes until the number of seed nodes reaches 15, is higher than the time needed in the SIMPATH algorithm. The reason is that, the INCIM algorithm finds the communities of the graphs in the preprocessing step. Also, in the first iteration, INCIM computes the spread of the nodes of all communities, but in other iterations, only the communities which are considered by Algorithm 2.2, should be updated. Table 2.3 shows the running time improvement of our proposed algorithm for choosing 50 seed nodes comparing with SIMPATH, which we used for calculating the spread values. As we can see, our approach has an improvement in running time between 27% - 68% based on different datasets.

The effect of communities As we mentioned in this paper, in the INCIM algorithm, seed nodes are chosen using communities of the input graph. In each iteration of the algorithm, computing the spread of each node is done locally in



(a) slashDot

(b) netHept

(c) Amazon

(d) DBLP

Figure 2.4: Spread of influence vs. running time for seed sets of size 50

its community. Also, when node v is chosen from community C_i, the spread of other nodes of community C_i along with the spread of nodes in communities with a simple path from C_i, should be updated. Thus, the spread of only a limited number of nodes would be updated in each iteration, instead of updating all of the nodes in the graph. By this contribution, the algorithm runs faster in other iterations. Figure 2.6 shows the number of communities that are updated in each iteration.

INCIM in its first iteration, computes the spread of all nodes in all communities, but in other iterations, the number of communities to be updated is decreased, and the number of spread computation calls is decreased, too. As the number of seed nodes is increased, there are more simple paths from seed nodes to other nodes in other communities. So, after some iterations, the number of communities which should be updated, is increased in comparison to the second iteration where there

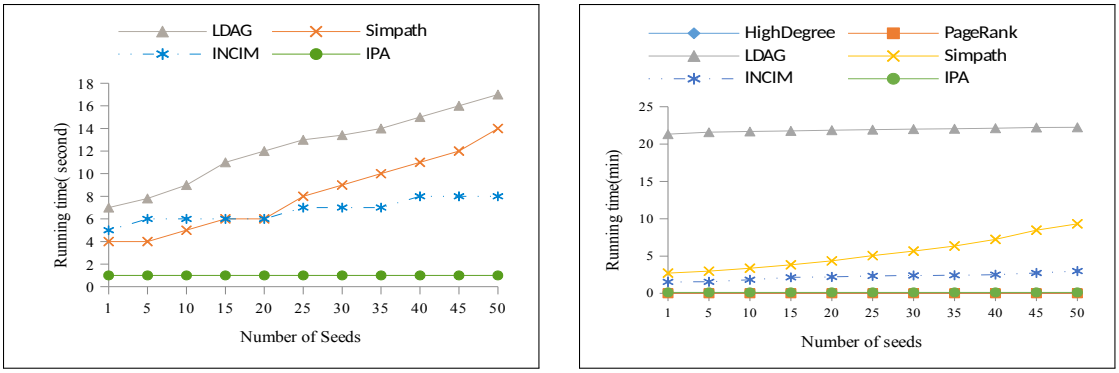
Table 2.3: Speed up in different datasets

Algorithms compared	**Datasets**			
	DBLP	Amazon	Slashdot	NetHept
Simpath	29%	27%	68%	43%

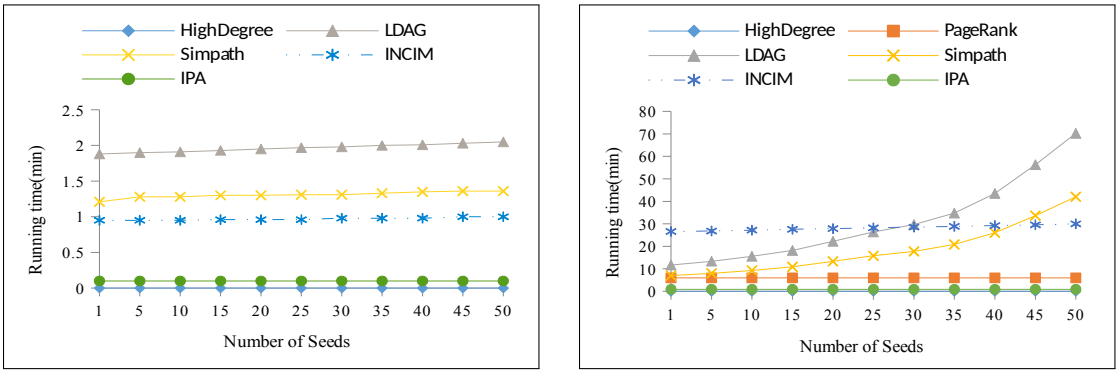
is only one seed node; but the number of communities in later iterations is anyway less than that of the first iteration.

The efficiency of final spread computation To study how efficiently we compute the spread of nodes in our approach, we run INCIM and simple greedy algorithm [KKT03] which uses Monte Carlo simulations (MC) on two moderate and two larger datasets (Amazon and DBLP respectively) and compare the spread achieved by them. We choose 5 different randomly selected set of nodes as seed sets of size 10, 20, 30, 40 and 50 and run INCIM and MC simulations 10000 times to compute the spread of the seed sets. The results are shown in Figure 2.7.

As we can see in Figure 2.7, the values which are computed by INCIM are very close to the values computed by Monte Carlo simulations for different sets of nodes. For Amazon dataset, the differences between the values computed by INCIM and MC are 0.68%, 0.5%, 0.8%, 0.62% and 0.85% for sets of size 10, 20, 30, 40 and 50 respectively. Also, For DBLP dataset, the differences between the values are 0.8%, 0.69%, 0.73%, 0.32% and 0.47% for sets of size 10, 20, 30, 40 and 50 respectively. Kempe. et al. showed that their approximation algorithm [KKT03], that we refer as the Monte Carlo simulations, can achieve at most $(1 - 1/e - \varepsilon)$ of the optimal solution and the other algorithms that can achieve near results are considered as practical algorithms. By the experiments in this section, we showed that the INCIM algorithm can calculate the spread values, which is a combination of local and global spreads, with an approximation very close to the MC results.



(a) netHept (b) slashDot



(c) Amazon (d) DBLP

Figure 2.5: Comparison of running time of different algorithms

2.5 Conclusion and future work

We can categorize the recent works in influence maximization problem in three groups. Some papers such as [LCL12, SBV+12, CCC+11, HSCJ12] study the influence maximization problem by considering competitors. In such papers, a competitive model is described which is in most cases an extension of linear threshold or independent cascade model. Some papers such as [CLZ12] study the influence maximization problem with temporal constraints. The main goal of such works is that influence propagation will be done in a limited time. Other works try to propose an algorithm to solve the influence maximization problem in a reasonable time with lower memory usage and higher quality of seed sets. INCIM, our

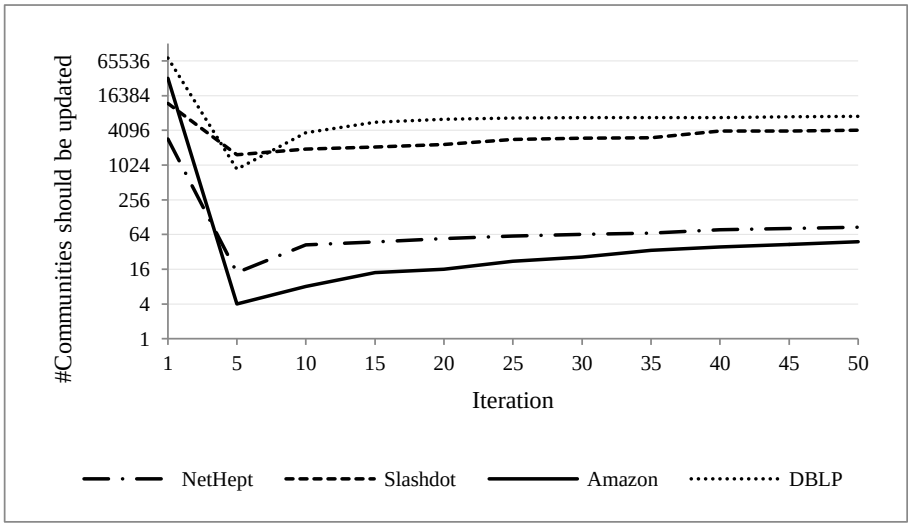
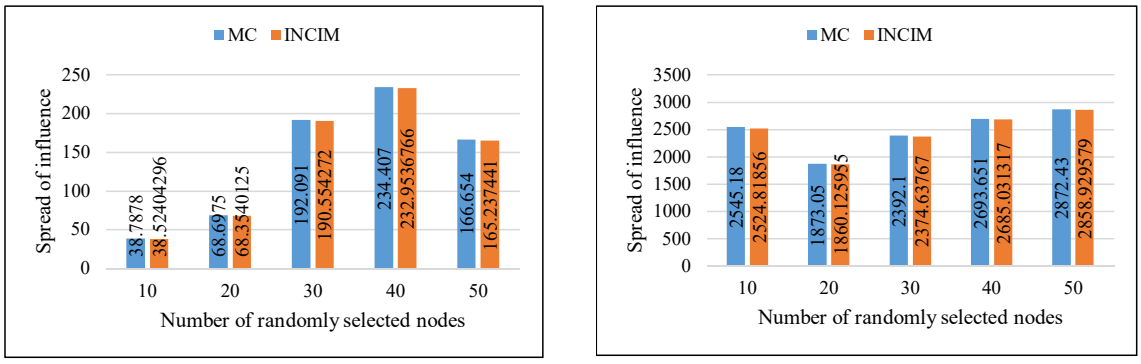


Figure 2.6: Number of communities that should be updated in each iteration



(a): Amazon

(b): DBLP

Figure 2.7: The spread achieved by different randomly selected seed sets

proposed algorithm, finds the influential nodes in communities of the graph. In this algorithm, spread computation of nodes is done locally in communities of the graph which causes a reasonable decrease in running time. Also, in each iteration of the algorithm, the marginal gain of only a limited number of nodes is computed, based on the communities they belong to, which causes the number of calls to the subroutine computing the spread of nodes, to be decreased, and thus the running time is also decreased.

For future work, we are interested in studying the competitive influence maxi-

mization problem in which there is more than one influence spread from different competitors. Considering the graph content and choosing the influential nodes based on the topic given as input, is the other path for our future work.

Bibliography

[BHZR16] Arastoo Bozorgi, Hassan Haghighi, Mohammad Sadegh Zahedi, and Mojtaba Rezvani. INCIM: A community-based algorithm for influence maximization problem under the linear threshold model. *Information Processing & Management*, 52(6):1188–1199, 2016.

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1):107–117, 1998.

[CCC+11] Wei Chen, Alex Collins, Rachel Cummings, Te Ke, Zhenming Liu, David Rincon, Xiaorui Sun, Yajun Wang, Wei Wei, and Yifei Yuan. Influence maximization in social networks when negative opinions may emerge and propagate. In *SDM*, pages 379–390. SIAM, 2011.

[CLZ12] Wei Chen, Wei Lu, and Ning Zhang. Time-critical influence maximization in social networks with time-delayed diffusion process. In *AAAI*, 2012.

[CSH+14] Suqi Cheng, Hua-Wei Shen, Junming Huang, Wei Chen, and Xue-Qi Cheng. Imrank: Influence maximization via finding self-consistent ranking. In *Proceedings of SIGIR*, pages 475–484, 2014.

[CWW10] Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of SIGKDD*, pages 1029–1038. ACM, 2010.

[CYZ10] Wei Chen, Yifei Yuan, and Li Zhang. Scalable influence maximization in social networks under the linear threshold model. In *Proceedings of ICDM*, pages 88–97. IEEE, 2010.

[GLL11a] Amit Goyal, Wei Lu, and Laks VS Lakshmanan. Celf++: optimizing the greedy algorithm for influence maximization in social networks. In *Proceedings of the 20th international conference companion on World wide web*, pages 47–48. ACM, 2011.

[GLL11b] Amit Goyal, Wei Lu, and Laks VS Lakshmanan. Simpath: An efficient algorithm for influence maximization under the linear threshold model. In *Proceedings of ICDM*, pages 211–220. IEEE, 2011.

[Gra73] Mark S Granovetter. The strength of weak ties. *American Journal of Sociology*, pages 1360–1380, 1973.

[GZZ+13] Jing Guo, Peng Zhang, Chuan Zhou, Yanan Cao, and Li Guo. Personalized influence maximization on social networks. In *Proceedings of CIKM*, pages 199–208. ACM, 2013.

[HCL14] Pili Hu, Sherman SM Chow, and Wing Cheong Lau. Secure friend discovery via privacy-preserving and decentralized community detection. *arXiv preprint arXiv:1405.4951*, 2014.

[HSCJ12] Xinran He, Guojie Song, Wei Chen, and Qingye Jiang. Influence blocking maximization in social networks under the competitive linear threshold model. In *SDM*, pages 463–474. SIAM, 2012.

[Joh75] Donald B Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1):77–84, 1975.

[KKT03] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread

of influence through a social network. In *Proceedings of SIGKDD*, pages 137–146. ACM, 2003.

[KLPL13] Chungrim Kim, Sangkeun Lee, Sungchan Park, and Sang-goo Lee. Influence maximization algorithm using markov clustering. In *Database Systems for Advanced Applications*, pages 112–126. Springer, 2013.

[Kro67] D Kroft. All paths through a maze. *Proceedings of the IEEE*, 55(1):88–90, 1967.

[LCL12] Long-Foong Liow, Shih-Fen Cheng, and Hoong Chuin Lau. Niche-seeking in influence maximization with adversary. In *Proceedings of ICEC*, pages 107–112. ACM, 2012.

[LKG+07] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *Proceedings of SIGKDD*, pages 420–429. ACM, 2007.

[LP49] R Duncan Luce and Albert D Perry. A method of matrix analysis of group structure. *Psychometrika*, 14(2):95–116, 1949.

[NG04] Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[SBV+12] Shahrzad Shirazipourazad, Brian Bogard, Harsh Vachhani, Arunabha Sen, and Paul Horn. Influence propagation in adversarial setting: how to defeat competition with least amount of investment. In *Proceedings of CIKM*, pages 585–594. ACM, 2012.

[Val79] Leslie G Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[WCSX10] Yu Wang, Gao Cong, Guojie Song, and Kunqing Xie. Community-based greedy algorithm for mining top-k influential nodes in mobile social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1039–1048. ACM, 2010.

[XSL11] Jierui Xie, Boleslaw K Szymanski, and Xiaoming Liu. Slpa: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process. In *Proceedings of ICDMW*, pages 344–349. IEEE, 2011.

[YKK13] Hwanjo Yu, Seung-Keol Kim, and Jinha Kim. Scalable and parallelizable processing of influence maximization for large-scale social networks. In *Proceedings of ICDE*, pages 266–277. IEEE Computer Society, 2013.

Chapter 3

Community-based influence maximization in social networks under a competitive linear threshold model

(This chapter is based on a paper published in Knowledge-Based Systems, 2017 [BSKW17])

The effect of online social networks (OSNs) in our daily life is undeniable as they have introduced new ways of communication and serve as a medium for propagating news, ideas, thoughts and any type of information. Such information can propagate via links between people, which leads to word-of-mouth advertising and its famous application, viral marketing. In viral marketing, the owner of a product gives free or discounted samples of a product to a group of people to gain a large number of adoptions through the word-of-mouth effect. The influence maximization problem, which is motivated by the idea of viral marketing, was introduced by Kempe et al. [KKT03] as finding a subset $S \subseteq V$ containing of k nodes in a graph $G = (V, E)$, such that the spread of influence from S will be maximized. There exists a huge amount of work on solving the influence maximization problem [KKT03, LKG$^+$07, YKK13, GLL11, CYZ10a]. Most of

these works assume that there is only one party trying to find influential users in the social network. However, in the real world, multiple parties typically compete simultaneously with similar products. This is called competitive influence maximization (CIM). Recently, several works have tried to solve the CIM problem [HSCJ12, CCC$^+$11, LBGL13, BKS07, BAEA11, LYW$^+$16, KAAB17, WLY$^+$16] by proposing new propagation models which are extensions of Linear Threshold and Independent Cascade models [KKT03] or the Distance-based and Wave-propagation models [SBV$^+$12].

In this paper, we examine the CIM problem from the follower's perspective and propose a new propagation model called DCM (Decidable Competitive Model) which is an extension of the Linear Threshold model. In DCM, each node has the ability to think about the incoming influence spreads from its neighbors for d timesteps and then decide to be activated by the neighbour with the majority of adoption.

In real social networks, people interact with each other based on common interests and strong ties between themselves. Such strong ties between individuals create community structures in social networks, which in turn allow information to circulate within these networks at a high velocity. We propose an algorithm called Competitive Influence Improvement (CI2) which finds the minimum number of influential nodes within their respective communities. Closely related to our work are [BHZR16, KLPL13, WCSX10, HPZNN17, ZLJ16] which also exploit community structure within social networks to find influential nodes.

Contribution. Our major contributions in this research are summarized as follows:

- We propose the DCM propagation model, the primary intent of this work, which gives decision-making power to nodes based on incoming influence in a competitive version of the LT propagation model.
- We prove the NP-hardness of competitive influence improvement under the DCM model.

- We propose the CI2 algorithm to find the minimum number of the most influential nodes for a competitor C_2. This algorithm uses knowledge of the nodes selected by a competitor C_1 so that C_2 can achieve more influence spread by spending less budget. Computing the spread of seed nodes is done locally inside communities of the input graph, which results in a substantial decrease in running time.

- We conduct experiments using three real and three synthetic datasets to show that CI2 can find influential nodes in an acceptable running time. Synthetic datasets are generated with the same number of nodes and edges but different community structures in order to track the effect of community structure of networks on our approach. Also, we consider the effect of the algorithm which finds the seed nodes for the first competitor on the seed nodes which will be selected by the second competitor by conducting different experiments which use well-known algorithms [BHZR16, KKY13, CYZ10b] to extract the first competitor's seed set.

Organization. In Section 3.1, we review some background knowledge to enable a better understanding of the upcoming concepts. In Section 3.2, we describe our Linear-Threshold-based propagation model, prove that competitive influence improvement under this model is NP-hard, and propose our CI2 algorithm. Section 3.3 describes the experiments performed with real and synthetic data to evaluate the proposed approach. Finally, in Section 3.4, we give our conclusions and directions for future work.

3.1 Background and Related Work

In [KKT03], Kempe et al. introduced two propagation models to address the influence maximization problem, the Linear Threshold (LT) and Independent Cascade (IC) models. In both models, a threshold value $\theta \in [0, 1]$ is assigned to each node

and each node can be active or inactive. Also, each edge from node u to node v has an influence weight $p_{u,v} \in (0,1]$. At first, all nodes are inactive except the nodes in set S which have been activated before as seed nodes and the propagation process is started from them. In LT, an inactive node v can be activated at time t if $f_v(S) > \theta_v$, where S stands for v's neighbors which are activated at time $t-1$. As Kempe et al. mentioned in [KKT03], the value of f_v is initialized as

$$f_v(S) = \sum_{u \in S} p_{v,u}$$

where $p_{v,u}$ is the weight of edge (v,u). In the LT model, the sum of all edge weights between v and its neighbors should be less than or equal to 1 [KKT03].

In IC, the activation process is the same as that in LT except that in IC, an activated node u has only one chance to activate its inactive neighbor v with probability $p_{u,v}$.

Community structure. In a graph, the communities are subsets of nodes with more connections between them and fewer ones to the nodes in different communities [LP49]. In community detection, a graph $G = (V, E)$ is given and the target is to partition the graph nodes into k subsets $S_1, S_2, \cdots, S_k$, such that $\bigcap_{i=1}^{k} S_i = \emptyset$ and $\bigcup_{i=1}^{k} S_i = V$. This condition is for none-overlapping community detection and for the overlapping version, we can remove the $\bigcap_{i=1}^{k} S_i = \emptyset$ constraint [HCL14].

Competitive influence maximization. Recently, several works have tried to solve the competitive influence maximization by introducing new propagation models to simulate the competitive manner of the competitors which are mostly an extension of the LT or IC models. Some efforts such as the ones introduced in [CNWVZ07, SBV$^+$12] look to this problem from the follower's perspective, i.e. they assume that there are two competitors trying to find some influential nodes and the second competitor starts his process with knowledge of the seed nodes selected by the first competitor and tries to find some new seed nodes other than the ones selected by the first competitor to achieve more influence spread. In some other works such as

the ones presented in [HSCJ12, CCC+11] one competitor tries to block the effect of the other competitor. In *K*-LT [LBGL13] and WPCLT [BFO10] models, the authors solve the competitive influence maximization problem from the host's perspective, i.e. the owner of the social network is responsible for fairly allocating some specific number of seed nodes to the competitors. In the next section, we explain *K*-LT and WPCLT models in more details as they are more related to our work.

3.2 Propagation model and algorithm

In this section, we introduce the DCM propagation model (which is an extension of the LT model), compare DCM with the Weighted-proportional (WPCLT) [BFO10] and *K*-LT [LBGL13] models, and prove the NP-hardness of competitive influence improvement under DCM. Finally, we introduce the CI2 algorithm to find the influential nodes in a social network under our competitive propagation model.

3.2.1 DCM propagation model

In the DCM propagation model, each node can be in one of the following states: *inactive*, *thinking*, $active^+$ or $active^-$. Suppose there are two competitors who try to advertise for their products over a social network. We denote the first competitor with the $+$ sign and the second competitor with the $-$ sign and each node v, picks a threshold value θ_v uniformly at random from [0,1]. Let S_1 be the seed set selected by the first competitor and S_2 be the seed set selected by the second one. At first all nodes except those in the seed set are *inactive*. The activation process of node v is as follows: at time $t > 1$ if the total incoming influence weight from the in-neighbors of v which are active ($N^{in}_{active}(v)$) reaches the threshold value of v, its state changes to *thinking*, which means the state of node v would be changed if

$$\sum_{u \in N^{in}_{active}(v)} p_{u,v} \geq \theta_v \tag{3.1}$$

Node v remains in *thinking* state after this state change for d timesteps and after that, it decides to become $active^+$ or $active^-$ based on the maximum total incoming influence weight from its in-neighbors. Let A^+_{t+d} be the set of in-neighbor nodes of v with state $active^+$, A^-_{t+d} be the set of in-neighbor nodes of v with state $active^-$, and A_{t+d} be the set of all in-neighbor nodes of v that are active at time $t+d$. The state of node v changes from *thinking* to $active^+$ or $active^-$ as follows:

$$v_{state} = \begin{cases} active^+, & \text{if } \sum_{u \in A^+_{t+d}} p_{u,v} > \sum_{u \in A^-_{t+d}} p_{u,v} \\ active^-, & \text{otherwise} \end{cases} \tag{3.2}$$

In the WPCLT model, which was proposed by Allan Borodin et al. [BFO10], the state of a node v changes to $active^+$ with probability $\sum_{u \in A^+_{t-1}} p_{u,v} / \sum_{u \in A_{t-1}} p_{u,v}$. This means that a node v would be activated as $active^+$ ($active^-$) at time t with probability equal to the ratio between the total weight from the in-neighbors with state $active^+$ ($active^-$) and that from all active in-neighbors. Wei Lu et al [LBGL13] noted that in the WPCLT model, when a node is about to activate, the neighbors which have been activated in all previous timesteps are considered; this, however, does not assure recency, which is when the customer's choice among competing products relies more on recent than old information [PM11, HS09]. Hence, Wei Lu et al proposed the *K*-LT model [LBGL13] in which the activation probability of node v at time t relies on its in-neighbors which have just been activated at time $t-1$ rather than all past exposures such that the state of a node v changes to $active^+$ with probability $\sum_{u \in A^+_{t-1} \setminus A^+_{t-2}} p_{u,v} / \sum_{u \in A_{t-1}} p_{u,v}$.

In both the WPCLT and *K*-LT models, a node cannot decide whether it would be activated or not, while in the real world when people decide to whether to purchase or consume a product, they are influenced by the decisions made by others. Even when individuals seem to be making decisions separately, they are likely to be mindful of the preferences of others [WH12].

As an example, imagine that the structure in Figure 3.1 is part of a social graph

in which, nodes w_1 and w_3 have been activated before as $active^+$ and $active^-$ respectively and nodes w_2 and w_4 haven't been activated yet and $\theta_v = 0.25$ (we show that nodes w_2 and w_4 are connected to other nodes of the graph by the dotted lines around them). At time t_1, in both WPCLT and K-LT models node v is influenced by the total incoming influence which is larger than its threshold value θ_v. Thus node v would be activated as $active^+$, as $p_{w_1,v} > p_{w_3,v}$. Now imagine that nodes w_2 and w_4 are activated as $active^-$ by other nodes of the graph at time t_2; this means that in both the WPCLT and K-LT models, node v is in $active^+$ state at time t_2, while the majority of its neighbors have been activated as $active^-$. But in DCM, the state of node v changes from *inactive* to *thinking* at time t_1 and its state remains stable for d timesteps so that it can consider different influence spreads, after which it decides to be activated by the influence spread which is accepted by the majority of its neighbors. This causes the state of node v to change from *thinking* to $active^-$ after d timesteps in the DCM model. Therefore, it is reasonable to give the ability to the nodes to decide about the incoming influence spread.

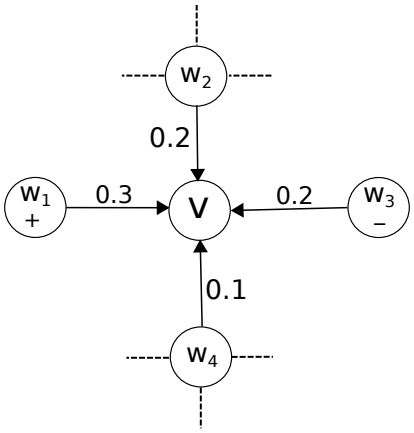


Figure 3.1: An example graph substructure.

3.2.2 NP-hardness of competitive influence improvement under DCM

Let a vertex-labeled and arc-weighted community-structure graph be $G = (V, A, l, p)$ where V is the vertex-set, A is the arc-set, $l : V \to \{inactive, thinking, active^+, active^-\}$ is the vertex-label function, and $p : A \to (0, 1]$ is the arc-weight function. Consider the vertex-labels in G after influence propagation under the DCM model is complete, and let N_- and N_+ be the number of vertices in G with labels $active^-$ and $active^+$, respectively. We want to select a set of vertices S_+ of size at most k relative to a given set S_- such that S_+ ultimately has more influence than S_- — that is, we want N_+ to be larger than N_-. Let $D(S_-, S_+) = \max(0, N_+ - N_-)$ and $\mathbb{E}_D(S_-, S_+)$ be the expected value of $D(S_-, S_+)$ (with this expectation arising from the various θ_v being chosen uniformly at random from the interval $[0, 1]$[1]).

The problem of influence improvement under DCM can now be stated formally as follows:

> DCM COMPETITIVE INFLUENCE IMPROVEMENT (DCI-Imp)
> *Input*: A community structure graph $G = (V, A, l, p)$, a seed-set S_-, and positive integers $d, k > 0$ and $c \geq 0$.
> *Output*: A seed-set S_+ of size at most k such that $\mathbb{E}_D(S_-, S_+) > c$, if such a S_+ exists, and special symbol $\perp$ otherwise.

This problem looks easy, as we are only requiring some fixed amount of improvement (which is the smallest possible, *i.e.*, *any* improvement, when $c = 0$). However, looks can be deceiving.

Theorem 1. *If DCI-Imp is polynomial-time solvable when $d = 1$ and $c = 0$ then $P = NP$.*

[1]Uniform choice of θ_v is consistent with previous linear-threshold-based models of influence propagation such as that in [KKT03]. However, the results in this section apply relative to θ_v choice under *any* distribution over $[0, 1]$ (including, but not limited to a uniform distribution) as long as that choice is ergodic, *i.e.*, there must be a finite non-zero probability for every $\theta_v \in [0, 1]$ being picked, including 0 and 1 as border cases.

Proof. We first show that DCI-Imp$_D$, the decision version of DCI-Imp (which asks whether or not the requested S_+ exists), is NP-hard by a polynomial-time reduction from the following NP-hard problem:

> DOMINATING SET [GJ79, Problem GT2]
> *Input*: A graph $G = (V, E)$ and an integer k.
> *Question*: Is there a dominating set in G of size at most k, *i.e.*, is there a subset $V' \subseteq V$, $|V'| \leq k$, such that for each $v \in V$, either $v \in V'$ or $\exists (v, v') \in E$ such that $v' \in V'$?

For any graph $G = (V, E)$, let $N(u)$ be the set of all vertices in G that are in-neighbors of vertex u in G (including u itself). Given an instance $(G = (V, E), k)$ of DOMINATING SET, construct the following instance $(G' = (V', A, l, p), S_-, d, k', c)$ of DCI-Imp$_D$:

- $V' = V_1 \cup V_2 \cup V_3$ where $V_1 = \{v_1^1, v_2^1, \ldots, v_{|V|}^1\}$, $V_2 = \{v_1^2, v_2^2, \ldots, v_{|V|}^2\}$, and $V_3 = \{v_1^3, v_2^3, \ldots, v_{|V|+(k-1)}^3\}$.
- $A = A_1 \cup A_2$ where $A_1 = \{(u, v) \mid u \in V_1, v \in V_2, \text{ and } v \in N(u)\}$ and A_2 ensures that each each vertex in V_2 has incoming arcs from exactly two distinct vertices in V_3.
- The initial labeling l of V is such that all vertices in V_3 have label *active*$^-$ and all other vertices have label *inactive*.
- p is such that the weight of each arc in A_1 is $1/2|V|$ and the weight of each arc in A_2 is $1/4|V|$.
- $S_- = V_3$.
- $d = 1$, $k' = k$, and $c = 0$.

This construction can be done in polynomial time in the size of the given instance of DOMINATING SET.

By the construction of G' above, the only vertices that can change label from *inactive* to *thinking* (and thereafter to either *active*$^+$ or *active*$^-$) under DCM are the vertices in V_2. A vertex v in V_2 will only be able to change label from *inactive* to *thinking* if $\theta_v > 1/2|V|$. Such a *thinking* vertex v will then have final label *active*$^-$

unless there is at least one vertex u in V_1 with label $active^+$ that has an arc to v (as the weight of such an arc would outweigh the weights of the two incoming arcs from V_3 and hence force v to have label $active^+$). As $|V_3| = |V| + (k-1)$, $D(S_-, S_+)$ can only have value 0 or 1 for a given S_+, with the value of 1 occurring if and only if $\theta_v > 1/2|V|$ for each vertex v in V_2 and the k vertices in S_+ force all vertices in V_2 to have label $+$ under DCM. However, by the construction of G', the vertices in such a S_+ correspond to a dominating set of size k in G. Given that θ_v is drawn uniformly from $[0, 1]$, there is a S_+ such that for some values of θ_v, $D(S_-, S_+) = 1$ and hence $\sigma(D(S_-, S_+)) > c = 0$ if and only if there is a dominating set of size k in the given instance of DOMINATING SET.

The above establishes that DCI-Imp$_D$ is NP-hard. To complete the proof, note that any polynomial-time algorithm for DCI-Imp can be used to solve DCI-Imp$_D$ in polynomial time, which, by the definition of NP-hardness, would imply that $P = NP$. □

This result shows that if the conjecture $P \neq NP$ is true (which is widely believed within Computer Science [For09, GJ79]), the simplest type of competitive influence improvement cannot be computed correctly for all inputs in polynomial time.

One might still hope that this problem is practically solvable in polynomial time. Two senses in which this might be possible are:

1. DCI-Imp is solvable in effectively polynomial time under certain restrictions. For example, there might be an algorithm for DCI-Imp that is exponential-time in general relative to the number k of nodes in S_+ but runs in polynomial time when k is a small constant. Such an algorithm would have runtime $f(k)n^x$ where f is an arbitrary function, n is the input size, and x is a constant. This notion of effective polynomial-solvability is the fixed-parameter tractability underlying Downey and Fellow's theory of parameterized computational complexity [DF99].

2. DCI-Imp is solvable in polynomial time by a probabilistic algorithm with high probability, *e.g.*, $> 2/3$. This notion of solvability is essentially what

many types of stochastic heuristics (in particular, those based on evolutionary computation) promise.

However, it turns out that these types of solvability are also unavailable to us, as proved in Theorems 2 and 3.

Theorem 2. *If DCI-Imp is fixed-parameter tractable relative to parameter k when $d = 1$ and $c = 0$ then $FPT = W[2]$.*

Proof. In the reduction given in the proof of Theorem 1, the size k of the requested dominating set in the given instance of DOMINATING SET is equal to the size k' of S_+ in the constructed instance of DCI-Imp. Hence, this reduction is also a parameterized reduction relative to parameter k' in the constructed instance of DCI-IMP. This result then follows from the $W[2]$-hardness of DOMINATING SET relative to parameter k and the inclusion of FPT in $W[2]$ [DF99]. □

Theorem 3. *If $P = BPP$ and DCI-Imp is polynomial-time solvable by a probabilistic algorithm which operates correctly with probability $\geq 2/3$ then $P = NP$.*

Proof. BPP is considered the most inclusive class of problems that can be efficiently solved using probabilistic methods (in particular, methods whose probability of correctness is $\geq 2/3$ and can be efficiently boosted to be arbitrarily close to probability one) [Wig07, Section 5.2]. If DCI-Imp has a probabilistic polynomial-time algorithm which operates correctly with probability $\geq 2/3$, DCI-Imp$_D$ is in BPP. However, as $BPP = P$ and DCI-IMP$_D$ is NP-hard by Theorem 1 above, the definition of NP-hardness then implies that $P = NP$. □

These results show that if, in addition to $P \neq NP$, the conjectures $FPT \neq W[2]$ and $P = BPP$ are also true (both of which are widely believed within Computer Science (see [DF99, DF13] and [Wig07, Section 5.2], respectively)), in general, the simplest type of competitive influence improvement cannot be practically computed in either of the senses above (the former relative to small-sized S_+).

To summarize, the results in this section effectively rule out several popular types of efficient algorithms for competitive influence improvement under the DCM

model. As such, they also justify the search for and use of heuristic algorithms such as the greedy community-based algorithm described in the remainder of this paper.

3.2.3 Community-based algorithm

Motivated by the useful characteristics of communities in social networks which we mentioned previously in Section 3.1, we decided to base our CI2 algorithm for competitive influence improvement on influential nodes in the community structure of input graph G. An overview of CI2 is shown in Figure 3.2. At first, the communities of the input graph are extracted; these communities are denoted by labels C_1, C_2 and C_3 in the figure. Then, in each community, the most influential node is selected as a seed candidate. Finally, the node which has the maximum influence spread among candidate nodes is selected as a seed node. The selected seed node for the second competitor is denoted with a $+$ sign in Figure 3.2. The CI2 algorithm is explained in more detail in the following section.

Community detection. Many approaches have been proposed to solve the community detection problem in online social networks. MLAMA-Net [MM16] is an evolutionary algorithm, which solves the community detection problem in a network of chromosomes using evolutionary operators and local searches. In MLAMA-Net, each node includes a chromosome and a learning automaton. Each chromosome explores a community for its corresponding node using evolutionary operators and improves the community by a local search. The learning automaton is responsible for saving the histories of local searches of each node. Very related to MLAMA-Net, Khomami et al. proposed DLACD [KRM16], which extracts the community structure of complex networks based on distributed learning automata.

To find the communities of the input graph, we use the listener-speaker approach introduced in [XSL11] in conjunction with the information diffusion model. First, each node v is considered as a unique community with community label equal to its ID. Then one node is selected as the consumer of the information and receives

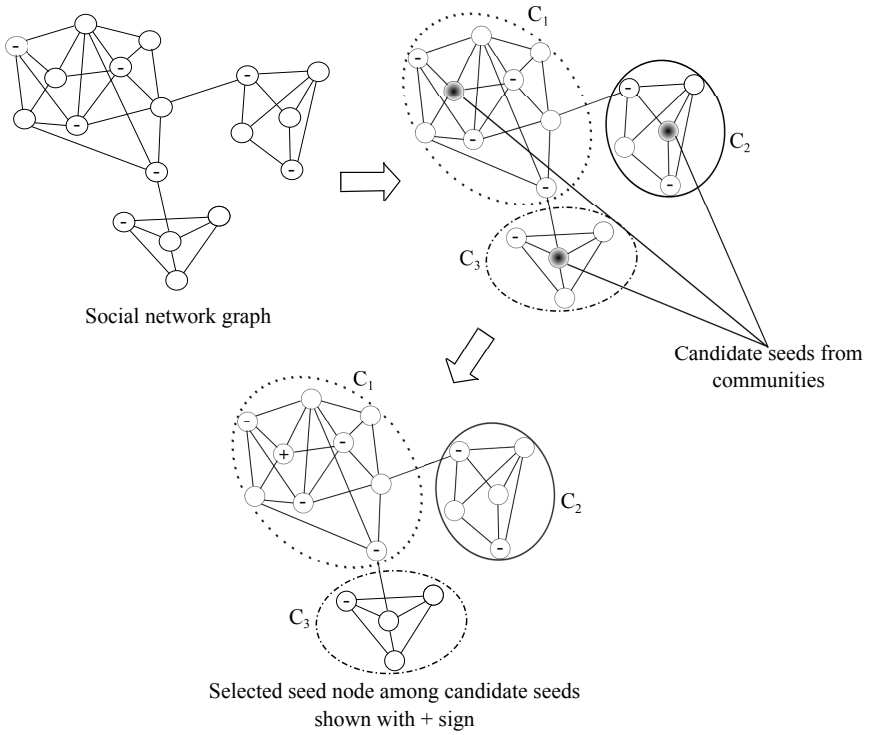


Figure 3.2: An overview of the CI2 community-based algorithm.

the community labels from its neighbors. To decide which label will be accepted by the selected node, the weights of the incoming edges are considered. For example, suppose a selected node v has four in-neighbors u_1, u_2, u_3 and u_4, with edge weights 0.1, 0.05, 0.3 and 0.25 respectively. When node v considers the labels from its neighbors, it will weight each label as $0.1/(0.1 + 0.05 + 0.3 + 0.25)$ for node u_1, $0.05/(0.1 + 0.05 + 0.3 + 0.25)$ for node u_2, $0.3/(0.1 + 0.05 + 0.3 + 0.25)$ for node u_3 and $0.25/(0.1 + 0.05 + 0.3 + 0.25)$ for node u_4. Then, the selected node accepts one label from the collection of the received labels from its neighbors based on a specified listening rule, such as the popularity of the observed labels in the ongoing step. This process is then repeated and at each step, one new node is selected as the consumer of the information. The main reason for using the approach of [XSL11] to find graph communities is its ability (verified by experiments done in [XSL11]) to efficiently find high-quality communities.

Seed selection. After constructing the community structure from graph G, we need to find the minimum number of nodes for the second competitor which achieve higher influence spread than the influence spread achieved by the nodes selected by the first competitor. The spread value of node v is the number of nodes which can be accessed and activated by node v and the spread of nodes in set S is the sum of spreads of each node in the set.

In each community C_i, we locally run an algorithm which uses DCM as its propagation model to find the most influential node in C_i and store the node ID and its spread value in candidate seed set S'. In this step we can use any approximation algorithms, even the simple greedy algorithm [KKT03], to show the applicability of our propagation model in solving the competitive influence maximization problem. Note that the node which is selected as a candidate node in this step should be different from the nodes which have been selected for the first competitor. The size of S' is equal to the number of communities and in each step, this set is updated to hold the new candidate seeds of each community. Among the candidate seeds, the one which has the maximum marginal gain is selected and added to S_2. S_1 and S_2 are seed sets of the first and second competitors respectively such that

$$S_2 = S_2 \cup arg\ max_{v \in S'}(\delta(S_2 \cup \{v\})) \tag{3.3}$$

In Equation (3.3), $\delta(S_2 \cup \{v\})$ is the marginal gain of adding node v to seed set S_2.

Stop criterion. The above seed selection process is continued until the influence spread achieved by nodes in S_2 reaches the influence spread achieved by nodes in S_1. The steps of our community-based algorithm are shown in Algorithm 3.1.

Algorithm 3.1 The CI2 community-based algorithm.

Input: $G = (V, E), S_1$

Output: S_2

1: $S_2 \leftarrow \{\}$
2: Construct the communities of the input graph G and store them in *CommunitiesSet*
3: **while** $\delta(S_2) \leq \delta(S_1)$ **do**
4: $S' \leftarrow \{\}$
5: **for** each $C_i \in CommunitiesSet$ **do**
6: call simple greedy algorithm to find most influential node $s_i \notin S_1$ in community C_i
7: $S' = S' \cup \{s_i\}$
8: $S_2 = S_2 \cup arg\ max_{v \in S'}(\delta(S_2 \cup \{v\}))$
9: return S_2

3.3 Evaluations

To evaluate the efficiency of our community-based algorithm in finding high-quality seed sets in an acceptable running time, we have done our experiments on three real-world datasets. These experiments show that there is a trade-off between running time and the quality of seed nodes selected by a competitor by changing parameter d, the number of timesteps a node can think about the incoming influence spread. To track the effect of community structure of networks on our approach, we also have used three synthetic datasets with the same numbers of nodes and edges but different community structures. Our code is implemented in C++ and all experiments were run on a Linux (CentOS 7.0) machine with a 3.6GHz Intel Core i7 CPU and 16GB of memory.

Table 3.1: Specifications of real standard datasets.

	NetHEPT	**Slashdot**	**Amazon**
#Nodes	15K	82K	262K
#Edges	62K	948K	1.2M
Average out-degree	4.12	9.8	9.4
Maximum out-degree	64	1527	425
#Connected components	1781	1	1
Largest component size	6794	82K	262K

Table 3.2: Specification of communities in the real standard datasets.

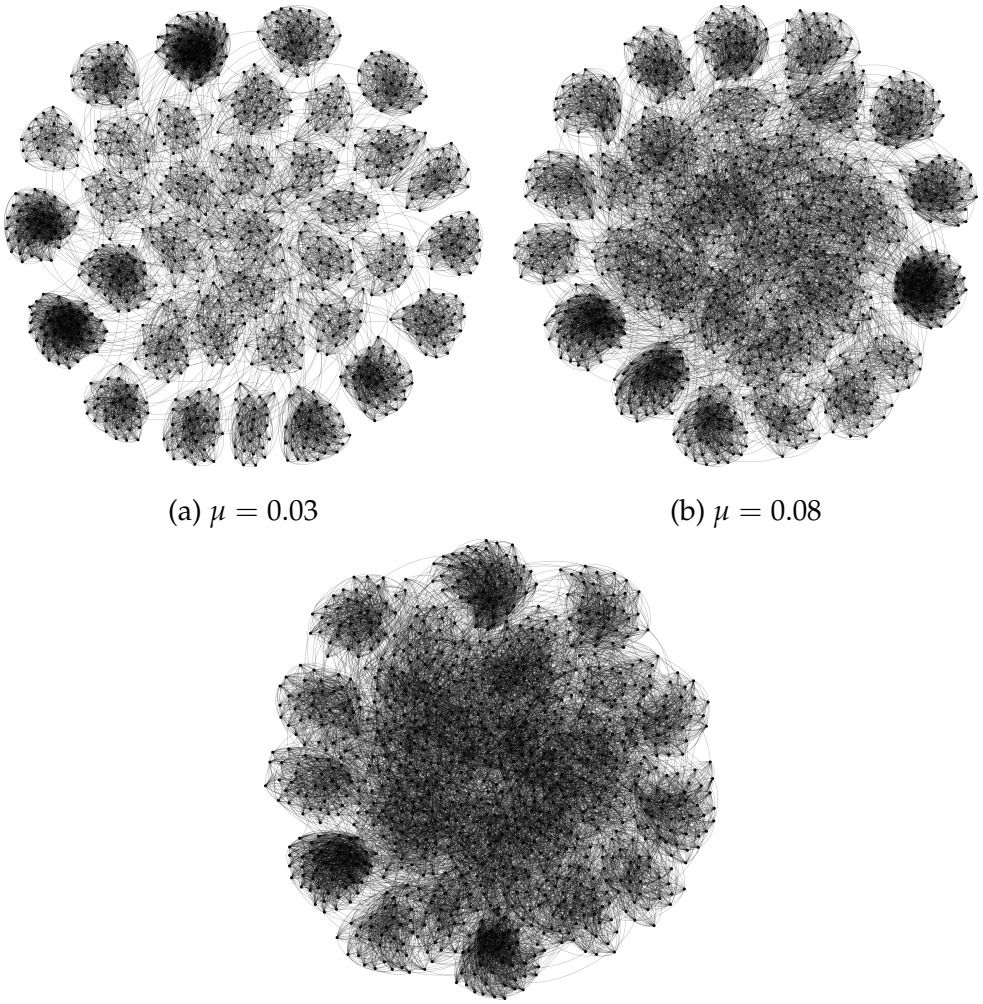
	NetHEPT	**Slashdot**	**Amazon**
#Communities	2901	12K	32674
#Nodes in the biggest community	407	6050	2852
#Edges in the biggest community	2938	16294	7169
Average out-degree in the biggest community	6.2	11.67	2.52
Maximum out-degree in the biggest community	48	1407	5

3.3.1 Experiments setup

Dataset. The real-world datasets that we have used in our experiments are available from the SNAP library on the Stanford University website[1] and their specifications are shown in Table 3.1. Using the community detection algorithm described in Section 3.2.3, we found the communities in these datasets; the specifications of these communities are shown in Table 3.2. In both Tables 3.1 and 3.2, the # sign indicates the number of elements.

To generate our synthetic datasets, we used LFR benchmark [LFR08], which specifies the heterogeneity of the networks by the distributions of node degrees

[1] http://snap.stanford.edu/



(a) $\mu = 0.03$ (b) $\mu = 0.08$

(c) $\mu = 0.15$

Figure 3.3: Three networks generated by LFR benchmark with 1000 nodes and different mixing parameter values.

and community sizes. The node degrees and community sizes are taken from power law distributions with exponents γ and β respectively. By assigning three different values (0.03, 0.08 and 0.15) to the mixing parameter μ, setting $N = 1000$ (the number of nodes), $\gamma = 2$ and $\beta = 1$, and varying the in-degree of nodes between 0 to 50 with average 15 and the community size between 20 and 50, we generated three different datasets, which are visualized in Figure 3.3. Note that the mixing parameter determines the fraction of one node's links to other nodes inside its community and nodes outside its community. More specifically, each node shares a fraction of $1 - \mu$ of its links with the nodes inside its community and a fraction of μ with nodes belonging to other communities.

Algorithms. We ran the implementation of our CI2 algorithm with the above described datasets as well as the following algorithms:

- The greedy approximation algorithm [KKT03], which uses Monte Carlo (MC) simulations to compute the spread of a node within a factor of $(1 - 1/e - \epsilon)$ for any $\epsilon > 0$. In this algorithm, MC simulations were performed 10,000 times to compute the spread of the seed sets.

- The INCIM algorithm [BHZR16], which computes the spread value of each node such that it is very close to that computed by Monte Carlo simulations. This algorithm finds the influence of each node as a combination of its local and global influences to track the effect of each node in its community and also, the effect of each community in the input graph.

- The IPA algorithm [KKY13], which selects as seed node the node which has maximum influence propagation probability in each iteration. Based on the recommendations in [KKY13], we set parameter $threshold = 0.005$.

- The LDAG algorithm [CYZ10b], which computes the spread of each node in its belonging DAG (Directed Acyclic Graph) locally and achieves good results

in quality of seed nodes. Based on the recommendations in [CYZ10b], we set parameter $\theta = 1/320$.

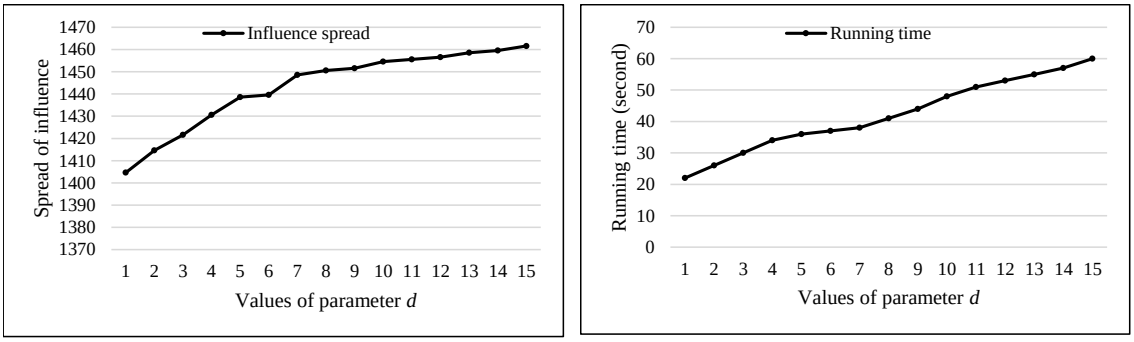
- The HighDegree algorithm [KKT03], which selects as seed node the node which has maximum out-degree.

The results of these algorithm runs are discussed in the next section. Two characteristics of these algorithms are worth noting. First, both the INCIM and IPA algorithms use the idea of communities to find influential nodes and like LDAG have reasonable running times and find good quality nodes. Second, though the HighDegree and greedy approximation algorithms are now almost 15 years old and may thus appear to be obsolete, they are still very commonly used in comparisons involving recently-proposed approaches to competitive and non-competitive influence maximization [OCC16, PHN$^+$16, KAAB17, WLY$^+$16, LYW$^+$16].

3.3.2 Experimental Results

Setting parameter d As we mentioned in previous sections, in the DCM propagation model, each node can think about incoming influence spread for d timesteps and then decides to be activated based on the majority of its neighbor's adoptions. When $d = 1$, information propagates the same as in the LT propagation model, where nodes can think for only one timestep about incoming influence spread, and we have the best running time. As the value of parameter d is increased, nodes have more time to think about incoming influence, which allows the selection of seed nodes with higher quality; however, this also results in increased running time. In Figure 3.4, we can see the changes in influence spread and running time associated with values of parameter d from 1 to 15. We did this experiment on the NetHEPT dataset with a seed set of size 50.

As we can see in Figure 3.4(a), as the value of parameter d changes from 1 to 7, there is a remarkable increase in influence spread. However, changing the value of



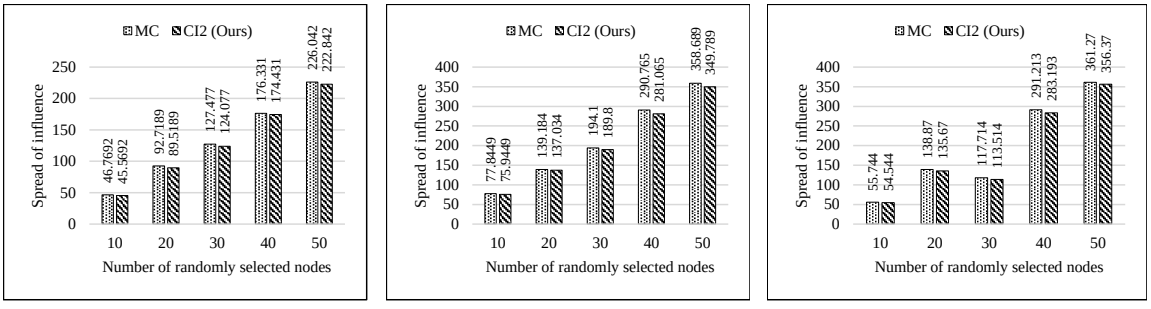
(a): Spread of influence

(b): Running Time

Figure 3.4: The effect of parameter d on influence spread and running time.

parameter d from 8 to 15 provides a marginal increase in influence for a great cost of increased running time (see Figure 3.4(b)). We performed the same experiment on the Amazon and Slashdot datasets and got essentially the same results. Hence, we set the value of parameter d to 7, the point of diminishing returns, in the remainder of our experiments. The value of d may be different for other datasets.

Efficiency of proposed algorithm. To study how efficiently we compute the spread of nodes by extracting seed nodes from communities, we randomly selected 5 different set of nodes as seed sets of size 10, 20, 30, 40 and 50 from each of the Amazon, NetHept and Slashdot datasets and ran CI2 and Monte Carlo simulations to compute the spread of these seed sets. The results are shown in Figure 3.5.



(a): Amazon

(b): NetHEPT

(c): Slashdot

Figure 3.5: The influence spreads achieved by different randomly selected seed sets.

Table 3.3: Differences between the influence spread values computed by MC and CI2 (in percentage).

Datasets	**Seed set size**				
Used	**10**	**20**	**30**	**40**	**50**
Amazon	2.07%	2.63%	2.51%	1.81%	2.13%
NetHEPT	2.44%	1.54%	2.21%	3.33%	2.48%
Slashdot	2.15%	2.30%	3.56%	2.75%	1.35%

The differences between the values computed by MC and CI2 are shown in Table 3.3. The results in Figure 3.5 and Table 3.3 show that the values which are computed by CI2 are very close to the values computed by Monte Carlo simulations which computes the spread of a node with a good approximation guarantee.

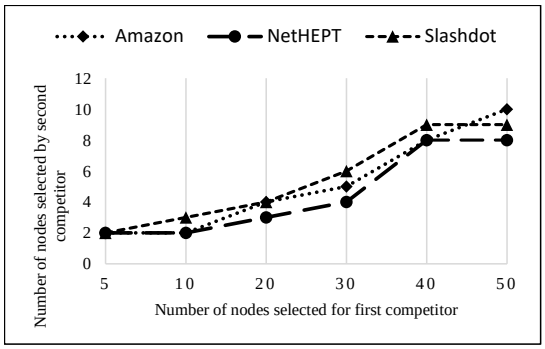
In these runs, calculating the spread of nodes locally inside the communities they belong to seems to cause a huge decrease in running time. To verify this, we used our CI2 algorithm to find a seed set of size 50 from the NetHEPT dataset by (1) considering existing communities and calculating the spread values locally inside communities and (2) calculating the spread of each node in the whole graph without considering their own community. In the former case, CI2 finds the seed nodes in approximately 22 seconds, while it finds such seed nodes in approximately 70 minutes in the later case. This clearly shows the effect of localizing the spread calculations in running time, which is the result of considering community structure in the CI2 algorithm.

Seed selection To simulate the competitive condition from the follower's perspective, we chose some seed nodes randomly and activated them for the first competitor as negative and ran CI2 to select the minimum number of nodes with higher influence spread for the second competitor. The nodes selected for the second competitor should be different from the ones selected for the first competitor.

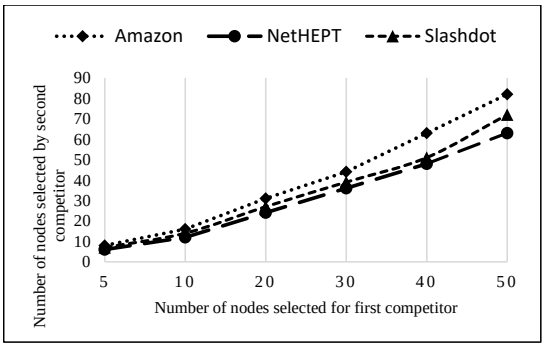
We also did the same process by running the greedy approximation algorithm and the INCIM [BHZR16], IPA [KKY13], LDAG [CYZ10b] and HighDegree [KKT03] algorithms with different values for k as their budgets. The generated seed sets are of size 5, 10, 20, 30, 40 and 50. The minimum number of nodes selected by CI2 to defeat the first competitor in each case is shown in Figure 3.6.

As we can see in Figure 3.6, the minimum number of nodes which is required to be selected by the second competitor to achieve higher influence spread depends deeply on how the seed nodes are selected by the first competitor. In Figure 3.6(a), in which the seed nodes of the first competitor are selected randomly, fewer nodes in each set are required to defeat the first competitor. However, when we extract the actual seed nodes by running the algorithms mentioned above, in each set of nodes, more nodes must be selected by the second competitor. For example, in Slashdot dataset in Figure 3.6(b), 72 seed nodes must be selected to achieve higher influence spread than the spread achieved by an actual seed set of size 50, while only 13 nodes need to be selected to achieve higher influence spread when the nodes in the set of size 50 are selected randomly. Also, the algorithm which is used to extract the actual seed nodes for the first competitor affects the number of seed nodes that need to be selected by the second competitor, as different algorithms achieve different levels of quality in their seed node extraction. In Figure 3.6(c-f), the number of seed nodes that need to be selected to defeat the first competitor are 69, 67, 64 and 61 if the seed sets of size 50 are extracted by the INCIM, IPA, LDAG and HighDegree algorithms respectively from the Slashdot dataset. These figures tell us that as seed nodes are selected with higher quality for the first competitor, more seed nodes other the selected ones must be selected by the second competitor.

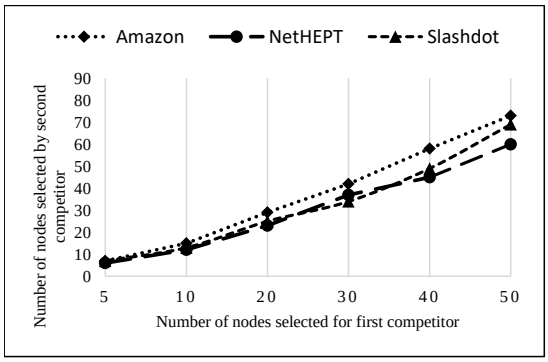
The effect of community structure on seed selection To study how the structure of communities can affect the quality of seed nodes, we ran CI2 on three synthetic LFR networks [LFR08] which vary their community structures by assigning differ-



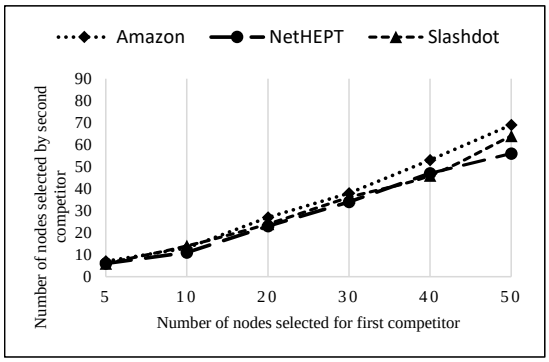
(a): Randomly seed nodes selected for first competitor



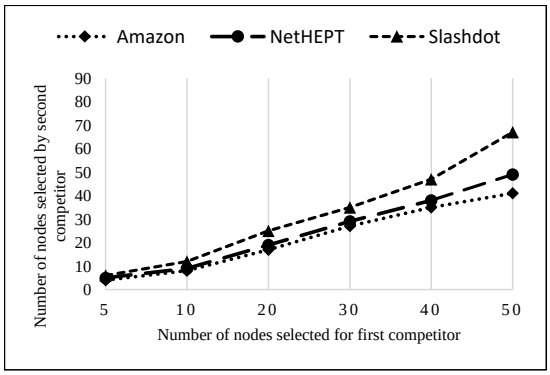
(b): Actual seed nodes selected for first competitor using MC



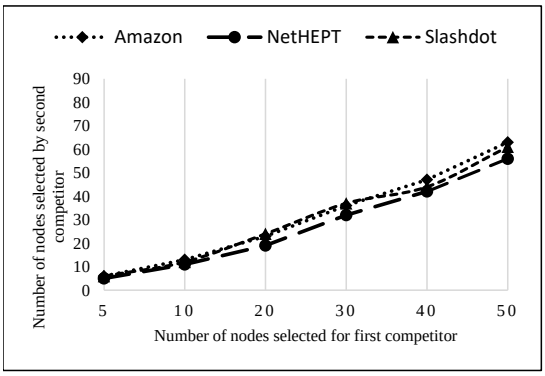
(c): Actual seed nodes selected for first competitor using INCIM



(d): Actual seed nodes selected for first competitor using LDAG



(e): Actual seed nodes selected for first competitor using IPA



(f): Actual seed nodes selected for first competitor using HighDegree

Figure 3.6: Minimum numbers of nodes which need to be selected by the second competitor to achieve larger influence spread than the spread achieved by the first competitor.

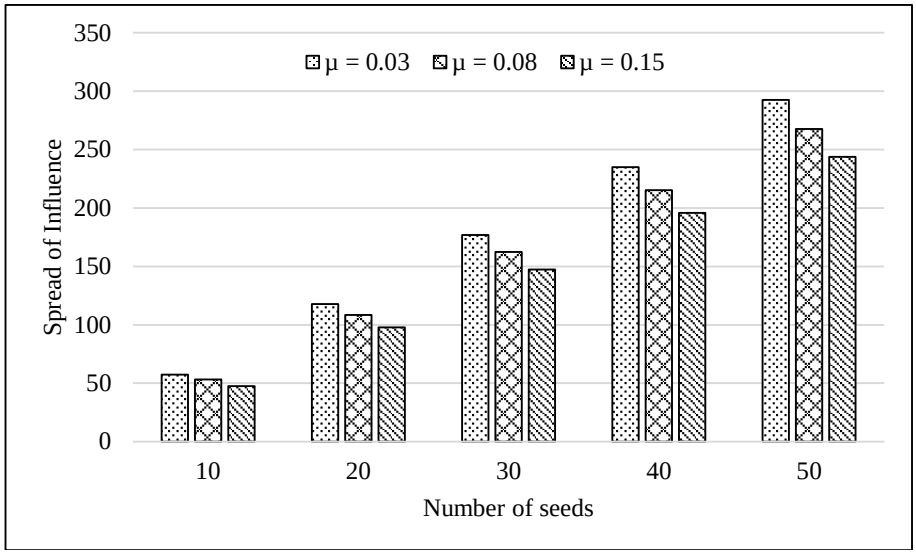


Figure 3.7: Influence spread achieved on LFR networks with different mixing parameter values.

ent values to the mixing parameter in LFR benchmark. If the value of the mixing parameter is smaller, the communities are loosely connected to each other and there are few links between nodes in different communities. The results of our runs on LFR networks in Figure 3.7 show that CI2 is better at finding seed nodes in networks whose community structures are more prominent. This is demonstrated by the observation that in the first network with the smallest mixing parameter, the influence spread achieved by the extracted seed set is higher than the two other networks with larger mixing parameter values. One way to help CI2 act better in networks with less prominent community structure is to consider the effect of border nodes [BHZR16] on influence spread computations. Border nodes have at least one link to nodes in other communities, which allows the spread of influence from a border node's own community to others and vice versa. As the main point of this paper is to propose the DCM propagation model and our aim of using community structure in CI2 algorithm is to improve the running time of finding seed nodes, we will address the issue of border nodes in future work.

3.4 Conclusion

In this paper we studied competitive influence maximization from the follower's perspective and introduced the Decidable Competitive Model (DCM), an extended version of the LT model, for influence propagation in a competitive fashion. To find the influential nodes in a social network graph, we proposed an efficient algorithm which extracts the communities of the input graph and finds the most influential node in each community as a seed candidate. Then the final seed nodes are selected from the set including seed candidates. The size of the final seed set should be as small as possible, i.e. we assign the seed nodes to the second competitor so as to achieve higher influence spread comparing with the spread achievement of the first competitor's seed set by spending less budget. The ability of nodes to think about incoming influence in the DCM propagation model simulates a realistic situation in which a node's tendency is toward the spread of influence which has been adopted by the majority of their neighbors after d timesteps. Adding parameter d to simulate the thinking ability of nodes results in finding influential nodes with higher quality; moreover, by calculating the spread values of each node locally inside its community, we achieved an acceptable running time. The results of our experiments on different real and synthetic datasets prove the effectiveness of our propagation model.

There are several promising directions for future research. First, faster algorithms than the greedy algorithm used here could be used inside communities to find influential nodes. Second, the effect of border nodes on the quality of seed nodes should be investigated in networks with different community structures. Finally, the effect of temporal evolution of networks on influence maximization should be analyzed.

Bibliography

[BAEA11] Ceren Budak, Divyakant Agrawal, and Amr El Abbadi. Limiting the spread of misinformation in social networks. In *Proceedings of the 20th international conference on World Wide Web*, pages 665–674. ACM, 2011.

[BFO10] Allan Borodin, Yuval Filmus, and Joel Oren. Threshold models for competitive influence in social networks. In *Internet and network economics*, pages 539–550. Springer, 2010.

[BHZR16] Arastoo Bozorgi, Hassan Haghighi, Mohammad Sadegh Zahedi, and Mojtaba Rezvani. INCIM: A community-based algorithm for influence maximization problem under the linear threshold model. *Information Processing & Management*, 52(6):1188–1199, 2016.

[BKS07] Shishir Bharathi, David Kempe, and Mahyar Salek. Competitive influence maximization in social networks. In *Internet and Network Economics*, pages 306–311. Springer, 2007.

[BSKW17] Arastoo Bozorgi, Saeed Samet, Johan Kwisthout, and Todd Wareham. Community-based influence maximization in social networks under a competitive linear threshold model. *Knowledge-Based Systems*, 134:149–158, 2017.

[CCC+11] Wei Chen, Alex Collins, Rachel Cummings, Te Ke, Zhenming Liu, David Rincon, Xiaorui Sun, Yajun Wang, Wei Wei, and Yifei Yuan. Influence maximization in social networks when negative opinions may emerge and propagate. In *Proceedings of ICDM*, pages 379–390. SIAM, 2011.

[CNWVZ07] Tim Carnes, Chandrashekhar Nagarajan, Stefan M Wild, and Anke Van Zuylen. Maximizing influence in a competitive social network: a

follower's perspective. In *Proceedings of the ninth international conference on Electronic Commerce*, pages 351–360. ACM, 2007.

[CYZ10a] Wei Chen, Yifei Yuan, and Li Zhang. Scalable influence maximization in social networks under the linear threshold model. In *Proceedings of ICDM*, pages 88–97. IEEE, 2010.

[CYZ10b] Wei Chen, Yifei Yuan, and Li Zhang. Scalable influence maximization in social networks under the linear threshold model. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 88–97. IEEE, 2010.

[DF99] R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer, Berlin, 1999.

[DF13] R.G. Downey and M.R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, Berlin, 2013.

[For09] L. Fortnow. The Status of the P Versus NP Problem. *Communications of the ACM*, 52(9):78–86, 2009.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, San Francisco, CA, 1979.

[GLL11] Amit Goyal, Wei Lu, and Laks VS Lakshmanan. Simpath: An efficient algorithm for influence maximization under the linear threshold model. In *Proceedings of ICDM*, pages 211–220. IEEE, 2011.

[HCL14] Pili Hu, Sherman SM Chow, and Wing Cheong Lau. Secure friend discovery via privacy-preserving and decentralized community detection. *arXiv preprint arXiv:1405.4951*, 2014.

[HPZNN17] Maryam Hosseini-Pozveh, Kamran Zamanifar, and Ahmad Reza Naghsh-Nilchi. A community-based approach to identify the most influential nodes in social networks. *Journal of Information Science*, 43(2):204–220, 2017.

[HS09] Tad Hogg and Gabor Szabo. Diversity of user activity and content quality in online communities. In *Third International AAAI Conference on Weblogs and Social Media*, 2009.

[HSCJ12] Xinran He, Guojie Song, Wei Chen, and Qingye Jiang. Influence blocking maximization in social networks under the competitive linear threshold model. In *Proceedings of ICDM*, pages 463–474. SIAM, 2012.

[KAAB17] Mehrdad Agha Mohammad Ali Kermani, Seyed Farshad Fatemi Ardestani, Alireza Aliahmadi, and Farnaz Barzinpour. A novel game theoretic approach for modeling competitive information diffusion in social networks with heterogeneous nodes. *Physica A: Statistical Mechanics and its Applications*, 466:570–582, 2017.

[KKT03] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of SIGKDD*, pages 137–146. ACM, 2003.

[KKY13] Jinha Kim, Seung-Keol Kim, and Hwanjo Yu. Scalable and parallelizable processing of influence maximization for large-scale social networks? In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 266–277. IEEE, 2013.

[KLPL13] Chungrim Kim, Sangkeun Lee, Sungchan Park, and Sang-goo Lee. Influence maximization algorithm using markov clustering. In *Database Systems for Advanced Applications*, pages 112–126. Springer, 2013.

[KRM16] Mohammad Mehdi Daliri Khomami, Alireza Rezvanian, and Mohammad Reza Meybodi. Distributed learning automata-based algorithm for community detection in complex networks. *International Journal of Modern Physics B*, 30(8):1650042, 2016.

[LBGL13] Wei Lu, Francesco Bonchi, Amit Goyal, and Laks VS Lakshmanan. The bang for the buck: fair competitive viral marketing from the host perspective. In *Proceedings of SIGKDD*, pages 928–936. ACM, 2013.

[LFR08] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78(4):046110, 2008.

[LKG+07] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *Proceedings of SIGKDD*, pages 420–429. ACM, 2007.

[LP49] R Duncan Luce and Albert D Perry. A method of matrix analysis of group structure. *Psychometrika*, 14(2):95–116, 1949.

[LYW+16] Weiyi Liu, Kun Yue, Hong Wu, Jin Li, Donghua Liu, and Duanping Tang. Containment of competitive influence spread in social networks. *Knowledge-Based Systems*, 109:266–275, 2016.

[MM16] Mehdi Rezapoor Mirsaleh and Mohammad Reza Meybodi. A Michigan memetic algorithm for solving the community detection problem in complex network. *Neurocomputing*, 214:535–545, 2016.

[OCC16] Han-Ching Ou, Chung-Kuang Chou, and Ming-Syan Chen. Influence maximization for complementary goods: Why parties fail to cooperate? In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1713–1722. ACM, 2016.

[PHN+16] Canh V Pham, Dung K Ha, Dung Q Ngo, Quang C Vu, and Huan X Hoang. A new viral marketing strategy with the competition in the large-scale online social networks. In *2016 IEEE International Conference on Computing & Communication Technologies, Research, Innovation, and Vision for the Future (RIVF)*, pages 1–6. IEEE, 2016.

[PM11] Gang Peng and Jifeng Mu. Technology adoption in online social networks. *Journal of Product Innovation Management*, 28(s1):133–145, 2011.

[SBV+12] Shahrzad Shirazipourazad, Brian Bogard, Harsh Vachhani, Arunabha Sen, and Paul Horn. Influence propagation in adversarial setting: how to defeat competition with least amount of investment. In *Proceedings of CIKM*, pages 585–594. ACM, 2012.

[WCSX10] Yu Wang, Gao Cong, Guojie Song, and Kunqing Xie. Community-based greedy algorithm for mining top-k influential nodes in mobile social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1039–1048. ACM, 2010.

[WH12] Wendy Wood and Timothy Hayes. Social influence on consumer decisions: Motives, modes, and consequences. *Journal of Consumer Psychology*, 22(3):324–328, 2012.

[Wig07] A. Wigderson. P, NP and mathematics — A computational complexity perspective. In *Proceedings of ICM 2006: Volume I*, pages 665–712, Zurich, 2007. EMS Publishing House.

[WLY+16] Hong Wu, Weiyi Liu, Kun Yue, Jin Li, and Weipeng Huang. Selecting seeds for competitive influence spread maximization in social

networks. In *International Conference of Young Computer Scientists, Engineers and Educators*, pages 600–611. Springer, 2016.

[XSL11] Jierui Xie, Boleslaw K Szymanski, and Xiaoming Liu. Slpa: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process. In *Proceedings of ICDM*, pages 344–349. IEEE, 2011.

[YKK13] Hwanjo Yu, Seung-Keol Kim, and Jinha Kim. Scalable and parallelizable processing of influence maximization for large-scale social networks. In *Proceedings of ICDE*, pages 266–277. IEEE Computer Society, 2013.

[ZLJ16] Yuxin Zhao, Shenghong Li, and Feng Jin. Identification of influential nodes in social networks with community structure based on label propagation. *Neurocomputing*, 210:34–44, 2016.

Chapter 4

Privacy-preserving Online Social Networks: Challenges and Solutions

(This chapter is ready to be submitted as a paper to a suitable journal or conference)

Online social networks (OSN) such as *Facebook*, *Google+* and *Twitter* have attracted billions of users which are using their free services. Such OSNs allow their users to communicate with each other in different ways such as sending messages, publishing their ideas which can be seen by huge number of people, sharing photos, like other user's activities, etc. Besides these general-purpose social networks, there are some other social networks which are made for professional purposes such as *Linkedin* which is a business-oriented social network or *Sermo* [1] and *Doximity* [2] which are social networks for healthcare providers with over half a million users.

The centralized nature of user data which is stored on the company's storage servers and is controlled by one single entity causes a range of privacy concerns. The administration cost of general-purpose social networks like *Facebook*, forces the providers to monetize user data far beyond the user's sharing interests. Users' concerns about their private data increases in Healthcare Social Networks (HSNs) as their personal health data are stored on OSN servers and are vulnerable to any

[1] http://www.sermo.com/

[2] https://www.doximity.com/

kind of breaches.

The worry about OSNs privacy concerns goes beyond the users'. In July 2009, the Canadian Internet Policy and Public Interest Clinic (CIPPIC) complained about *Facebook* approaches to default privacy settings, collection and use of users' personal information for advertising purposes, disclosure of users' personal information to third-party application developers, and collection and use of non-users' personal information [oCD09]. Also, in April 2010, the Privacy Commissioner of Canada and the heads of the data protection authorities in France, Germany, Israel, Italy, Ireland, Netherlands, New Zealand, Spain and the United Kingdom sent a letter to the chief executive officer of Google Inc. to express their concerns about privacy issues related to Google Buzz [Car10].

Based on the importance of OSNs in people's daily life and the sensitivity of their data, a mechanism which can block or at least minimize the users' privacy violation while preserve the OSN advantages is strongly needed. In the last few decades, a huge amount of work has been devoted to research on privacy and to address the privacy problems over OSNs. The term privacy is a multifaceted and complex concept which can be viewed from different perspectives. Based on the research which was done by Gürses [DG10], privacy can be classified into three paradigms: privacy as control, privacy as confidentiality and privacy as practice.

Privacy as control states that there should be a mechanism to enable individuals to control and oversee the collection, processing, and use of their data. The organizations that collect and process user data are supposed to act honestly. This paradigm relates to the definition of privacy by Westin: "the right of the individual to decide what information about himself should be communicated to others and under what circumstances"[Wes68]. In privacy as confidentiality, trusting organizations is avoided and disclosing any information by individuals is prevented or minimized by some mechanisms such as cryptography. This paradigm is related to the definition of privacy in [WB90]: "the right to be let alone ". In both privacy

as control and privacy as confidentiality, the main focus is on security and both concepts try to allow individuals to prevent information disclosure or organizations to enhance the security of data they hold and prevent its abuse for illegal purposes. However, privacy has some other social dimensions beyond the decisions made in isolation [DG12]. Technologies in the privacy as practice paradigm try to make the information flow more transparent instead of concealing and controlling it. This paradigm relates to Agre's definition of privacy: "the freedom from unreasonable constraints on the construction of ones own identity "[AR98].

The architecture of OSNs can be viewed from different points of view based on the way user data is stored and supervised. In most known OSNs such as *Facebook*, *Google+* and *Twitter* which use *centralized* client-server architectures, the users' data and their interactions are stored on OSN servers and are supervised by a single entity, i.e., the OSN provider. In such centralized architectures, users' privacy is always facing potential privacy violations by the provider. To protect user data from a "big-brother" scenario with OSN providers, *decentralized* or *peer-to-peer* (P2P) architectures for social networks have begun to emerge. In such OSNs, user data can be stored on data owners' computers, friends' computers, random peers over the social network or any trusted third-parties' external storage [BB13]. Diaspora [RG] with over 400000 users is the most successful decentralized OSN. A mixture of centralized and decentralized architectures is called *hybrid* architecture which stores user data on both providers' dedicated servers and users' trusted servers.

To the best of our knowledge, just a few studies have been done to address the privacy concerns about Healthcare Social Networks (HSNs) [Li13, Cha16, LBL+12, Li15]. However, the users of such online communities share sensitive information about their health conditions which may negatively affect their job opportunities, reputation, relationships and insurance choices if the data would be revealed to unauthorized entities.

In this chapter, we explore different privacy solutions proposed for OSNs and

explain each solution and their pros and cons in applying them to OSNs. In Section 4.1 we discuss concepts that the reader should be familiar with to better understand the paper. Section 4.2 is the section in which we will introduce known privacy solutions for OSNs in three categories and we discuss the applicability of each solution for different types of OSNs in Section 4.3. Then in Section 4.4, we explain in detail why we recommend the decentralized architectures for OSNs and discuss the design limitations and how to overcome them to have a practical P2P architecture. We discuss what can be a suitable approach for healthcare OSNs in Section 4.5. We conclude our chapter in Section 4.6.

4.1 Preliminaries

In this section, we introduce fundamental concepts of security and privacy and review some basic encryption approaches which are needed for better understanding of the chapter.

4.1.1 Online Social Networks (OSNs)

Social networks serve as a medium for modeling interactions between individuals, groups and organizations. A social network can be modeled as a directed or undirected graph $G = (V, E)$ where V and E are the set of nodes and edges of G. Individuals are modeled as the set of nodes V and the relationships between them is modeled as the set of edges E. The relationships between individuals are established based on the friendships in their real world life, being co-tagged in a photo, co-authoring a book, etc. The edges in graph G can be weighted or unweighted based on the problem which needs to be solved.

4.1.2 Peer-to-Peer (P2P) overlays

P2P overlays can provide solutions for pervasive environments like wireless and mobile networks which need a flexible and extensible underlying network to support different levels of diversity and personalization concerning both the users and the applications [Mal15]. In such environments, P2P overlays allow for multiple virtual network topologies to be built on top of the actual physical networks.

The P2P overlays are classified into four groups: structured P2P, unstructured P2P, multi-layer and bio-inspired P2P [Mal15]. In structured overlay, the nodes are connected to each other in a specific way like Distributed Hash Tables (DHTs) which allow the uniform distributions of the resources among the nodes. However, these tight structures between nodes make these overlays inefficient for dynamic networks [Mal15] like social networks in which the churn rates are high, because in each churn, the network should be updated to fix the structure.

Unstructured overlays are the second type of overlays in which the node relationships and lookup operations are more flexible in comparison with structured ones. There are some pros and cons in using such overlays over the underlying network. The flexibility of the connections guarantees resilience and robustness in dynamic networks and reduces the maintenance costs such as the overhead of the message exchanges. But the lookup operations are done by flooding [Mal15] which make them less inefficient for lookups in comparison with structured overlays. These types of operations are useful for single- and multi-attribute range queries [Mal15].

In some networks in which different types of applications need to work together, two or more overlays can be employed to minimize the management overheads and also, benefit from efficient resource discovery of structured overlays and flexible membership of unstructured ones together [Mal15]. These are called multi-layer overlays.

The bio-inspired overlays are inspired from the biology field and are character-

ized by their adaptive and reactive behaviour in distributed operations, that are resilience to failure of components and are self-organized. These features make them suitable to be used in heterogeneous environments with distributed operations [Mal15]. The resource discovery in these overlays is done using swarm intelligence techniques which make their implementation computationally difficult as they are based on self-organization and the plethora of independent agents interact with each other using indirect means [Mal15].

4.1.3 Group Key Management

One of the important functional building blocks for secure multicast architectures like video conferencing, software updates and broadcasting stock quotes is group key management protocol. In group encryptions, there exists a sender which sends the data to a group of receivers in a secure multicast session handled by two main entities called Group Controller and Key Server [CS05]. The sender sends a secret symmetric key SK to all group members and asks the key server to generate another secret key K_i per each user i and all the keys are stored in the key server. The secret symmetric key itself is communicated to the group member on a secured public channel between the server and the group members, that can be established by DiffieHellman key exchange protocol [Res99]. To multicast a message, the sender encrypts data with SK using a symmetric encryption algorithm so that the receivers can decrypt the data with key SK. When a group member leaves the group, re-keying is required as follows: the key server generates a new SK' and encrypts it with K_i of each user except the leaving member and broadcasts the new key to the group members. So, the next encryptions will be done by the new generated key. Another scenario in which re-keying is required is when a new member wants to join an existing group. In this scenario, a key server generates a secret key K_j for the new member j and generates a new key SK', then encrypts SK' with previous SK to the existing members and encrypts it with K_j for the new member and broadcasts

the new keys. By this way of key broadcasting, new members cannot decrypt the previous messages sent in the group [CS05].

4.2 Privacy Solutions

To address the privacy concerns over OSNs, several approaches have been proposed in recent years. Some of these approaches try to present a framework which can be integrated with centralized architecture of current OSNs and preserve the privacy of users by some mechanisms like data encryption [TGS+08, LB08, GTF08, LXH09, BBS+09, SZF10, JMB11, BKW11, RMJ13, RMJ14]. These approaches trust the OSN provider and user data is stored in OSN servers. However, there are some other approaches which believe that the privacy concerns of OSNs are related to their centralized nature. Such approaches presented a decentralized or peer-to-peer architecture for OSNs and try to introduce new features which cannot be supported by current OSNs to motivate users to use the new architecture [BSVD09, Str09, JNM+12, Nar12, BB13, RJM15, NJM+12, GADS+16]. In such frameworks, user data is stored on peers or user-trusted storage servers. Besides these two categories, some other approaches have been proposed which are a hybrid of centralized and decentralized architectures which use a hybrid of OSN providers' storage servers and users' trusted storages which can be users' machines, users' friends' machines or cloud storage servers which are managed by users themselves [RMJM11, SLC+11, WSW+11, LSC+11]. In the reminder of this section, we will introduce the known approaches, which have been done to the date of this paper with the aim of preserving users' privacy in OSNs, and categorize them based on the way they store user data into three groups, centralized, decentralized and hybrid. Finally, we conclude this section with discussing the privacy solutions presented for health-care OSNs.

4.2.1 Centralized OSNs

In Figure 4.1, an overview of centralized approaches is shown. In the centralized architectures, user data is stored on the servers which are controlled by the provider centrally and different approaches try to protect user data by encrypting it before it would be stored.

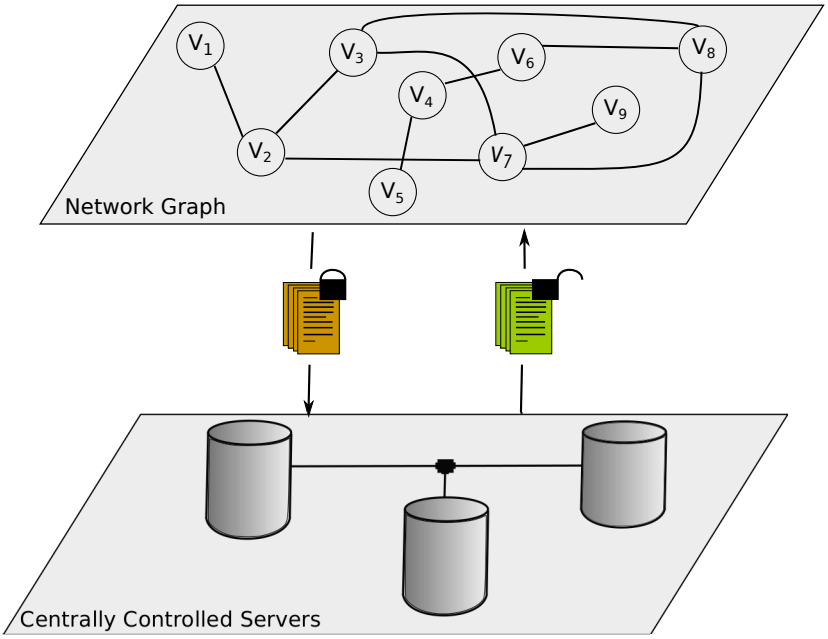


Figure 4.1: An example of centralized approaches: encrypting user data before storing on centrally controlled storage servers, but the relations between users can be seen by the provider.

Lockr [TGS+08] is a browser plug-in in which users can define social access control lists (ACL) to restrict their friends access to specific data and the defined ACLs are reusable in different websites. A user who wants to grant access of her published content to one of her friends should define a small piece of meta-data called "attestation" including an issuer, a recipient, a social relationship between two parties, an expiration date, a relationship key and a digital signature. The attestation is signed by the sender's private key and encrypted by the receiver's

public key and is stored by the receiver after verification and decryption. This approach can hide private data but the relationship between users is plain and the data is stored on OSN servers in plain-text form. Lockr needs to reissue the attestations with new relationship keys when a revocation is done and supports only one-to-one communication, which means one attestation is issued for one receiver, not a group of receivers.

flybyNight [LB08] uses a JavaScript implementation of AES and RSA and their own implemented El Gamal to transmit encrypted messages to Facebook using a Facebook API. The El Gamal algorithm is used for encrypting one-to-one communications while for one-to-many communication (group encryption), the authors uses proxy cryptography. To to so, the user generates a group key for each group member which would be the public key and a proxy key. Then encrypted data is transfered along with the public key to a proxy server. In the proxy server, the encrypted data would be re-encrypted again with the proxy key, n times for each group member. flyByNight does not support efficient revocation as a new group key and new proxy keys are required per each group member after revoking just one member from the existing group.

The idea behind NOYB (short for "None Of Your Business") [GTF08], which was introduced at the same time as Lockr [TGS+08] and flyByNight [LB08], is that the profiles of users are partitioned into smaller clusters called *atoms*. The atoms of one user are substituted with atoms of another user in the same cluster pseudo-randomly and then the encrypted index of each atom is stored in a dictionary. The dictionaries are stored on a different server and key exchange between users is done out-of-band (OOB). NOYB does not introduce any mechanism to define permissions to different friends or groups of friends. Also, the completeness of the dictionaries is related to the number of users which use NOYB. Therefore, the number of NOYB users affects its effectiveness.

Persona [BBS+09] is another framework for centralized OSNs in which users'

private information is hidden using ciphertext policy attribute-based encryption (CPABE). Using ABE allows friend-of-friend interactions without requiring enumerations of friend and attribute lists. A friend may limit who may read a response to a wall post to a more restricted group. The dynamic grouping feature of Persona is the result of using an attribute-based mechanism for encryption. But Persona does not support an efficient revocation mechanism. Persona OSN is a Mozilla plugin which can also work with other social networks such as Facebook and can integrate with Facebook applications.

Submitting encrypted data to OSN servers may attract the attention of the providers and cause some profile tracking, which may worry the users. Therefore, the authors of FaceCloak [LXH09] introduced a mechanism in which the users' private data is encrypted and stored on a third-party server which is trusted by the user herself while some other fake data related to encrypted data are stored in OSN servers in plaintext form. The fake data is generated using some dictionaries for replacing sample texts such as names, but for more complex texts such as a poem, some related fake text is replaced from Wikipedia. FaceCloak consists of three phases: setup, encryption and decryption. In the setup phase, three keys are generated: a master key and personal index key that are distributed to the user's friends by some out-of-band mechanism and an access key that is stored locally in the user's machine. In the encryption phase, text that starts with a unique separator such as @@ is encrypted and stored in a third party server, and fake data is sent to the social network server. Private data is stored as an index-value pair on a third-party server, where the value is the data encrypted by the master key and the index is a unique number generated from a cryptographic hash of a personal index key and the type of private data. In FaceCloak, modifying the real content of posts published by other users on one's profile is not possible. Also, there is no mechanism to define one-to-many communication.

One of the weaknesses of the previous approaches is that none of them support

an efficient revocation mechanism which would be suitable for the frequently changing social group memberships. The need of such a principal feature for encryption-based systems which are used to preserve the privacy in social networks encourages Sun et al. [SZF10] to introduce a privacy preserving approach which supports dynamic revocation and search over encrypted data without decrypting it. The main difference of this approach is in using broadcast encryption for efficient revocation coupled with role-based searchable encryption. When a member is revoked from a group, her public key is also revoked. So, in the decryption phase of the broadcast scheme, she cannot obtain the renewed secret key to decrypt the ciphertexts anymore.

EASiER [JMB11] is another approach which supports efficient revocation by introducing an attribute-based encryption approach, in which the decryption process is done by the participation of a minimally trusted proxy server that handles revoked users and attributes. In EASiER, the centralized OSN provider acts as the proxy server which cannot decrypt the messages directly since it doesn't have the attribute keys. When an unprovoked user wants to decrypt a ciphertext, she sends a part of the ciphertext to the proxy server and receives some information, which can be combined with the secret key of the user so the ciphertext can be decrypted. In each revocation, just the proxy key would be renewed and the users re-key their proxy keys. The superiority of EASiER's attribute-based approach, compared to the role-based approach of [SZF10], lies in its ability for multiple encryptions per each role, which may be needed on same scenarios.

Scramble [BKW11] is a Firefox extension which allows users to define access control lists (ACL) of authorized users for each piece of data, based on their preferences, and guarantees confidentiality of users' data towards the social network site (SNS) providers by storing encrypted data in a TinyLink server and its corresponding link in the OSN servers. PGP's web-of-trust (Pretty Good Privacy)[Zim95], which is a data encryption and decryption concept that provides cryptographic privacy and

authentication for data communication, is the key distribution mechanism used in Scramble. There is no efficient revocation mechanism in Scramble and users are required to distribute a new public key upon key revocation or key update.

CP2 (short for "Cryptographic privacy protection") [RMJ13] introduces a public key broadcast encryption scheme which protects not only users' private data from the OSN provider and unauthorized users, but also the relationships between users in the OSN. To protect the relationships, the user's friends are determined by a unique index which is a pseudonym other than the friend's real name. Then the mapping between users' real names and their pseudonym is stored securely on OSNs servers. Also, the groups' names are specified by a pseudonym so that no one will be aware of the user's relationship with her friends. Using flexible access controls of CP2, users can define new access policies using the combinations of friends and relations with some boolean operations such as AND, OR, NOT. CP2 supports one-to-many communication and also provides an efficient mechanism for group revocation.

Raji et al. [RMJ14] introduced another public key broadcast encryption scheme which enables users to protect their shared data while keeping their connections/relationships with other users anonymous. Also, users can employ the privacy setting of other users in the same group. This is done by using a proxy server. When a user wants to share data for the first time in a group in which another user has already specified a privacy setting, the proxy server runs an algorithm to prepare some parameters for the user and sends the result and the header information related to the group to the user. This way, the user can use the defined policies of the group to which she belongs to share her own data. The approach of this paper [RMJ14] supports one-to-many communication and employs an efficient revocation mechanism.

Table 4.1 gives a comparison of the illustrated solutions for centralized OSNs based on different privacy concerns. There are different types of key distributions

which we categorize into two groups: in-band and out-of-band. In in-band distribution, the encryption keys are generated and stored on OSN servers and are distributed to the users. In out-of-band distribution, the key generation is done on a third-party server and the keys are distributed by an out-of-band mechanism like email, phone call, letter or via a third-party server. Also, based on the number of senders of encrypted data in a group, communications can be one-to-one, one-to-many and many-to-many [RMJ14]. In one-to-one communication, a sender encrypts her data which can be decrypted by one specific receiver, while in one-to-many communication, just one user can send encrypted data to the group and the other members of the group can only be the receivers. But in many-to-many communication, each user in a group can be both sender and receiver. As we mentioned, most of the privacy-preserving approaches for centralized OSNs use some encryption mechanisms to protect user data. By encrypting data, some functionalities of OSNs such as searchability are not accessible, especially in the cases where private data is stored on a third-party server. Among the summarized approaches in the current section, just Sun et al. [SZF10] presented a solution to search over encrypted data without decrypting it in OSNs.

Methods	Protect from OSN Provider	Protect Relationships	Efficient Revocation	Search without Decryption	Key Distribution	Communication Support
Lockr [TGS+08]	–	–	–	–	In-bound	One-to-one
flyByNight [LB08]	–	–	–	–	Out-of-bound (proxy server)	One-to-many
NOYB [GTF08]	✓	–	–	–	Out-of-bound	One-to-many
FaceCloak [LXH09]	✓	–	–	–	Out-of-bound	Many-to-many
Persona [BBS+09]	✓	–	–	–	Out-of-bound	Many-to-many
Sun et al. [SZF10]	✓	–	✓	✓	In-bound	One-to-many
EASiER [JMB11]	✓	–	✓	–	Out-of-bound (proxy server)	One-to-many
Scramble [BKW11]	✓	–	–	–	Out-of-bound (PGP)	One-to-many
CP2 [RMJ13]	✓	✓	✓	–	In-bound	One-to-many
Raji et al. [RMJ14]	✓	✓	✓	–	Out-of-bound (proxy server)	Many-to-many

Table 4.1: Comparing the privacy approaches proposed for Centralized OSNs

4.2.2 Decentralized OSNs (Peer-to-Peer)

The centralized nature of OSNs in which all user data is accessible by a single entity, i.e. the OSN provider, encourages researchers to change their way of thinking about preserving user data, which leads to a shift from client-server to a peer-to-peer architecture coupled with encryption so that the users can protect their own data and can have direct data exchange with other users without constant Internet connectivity. The idea of decentralized social networks comes from the P2P file sharing systems like Napster[1] and Soulseek[2] which were mostly used for music. There are also other file sharing systems like Groove[3], which is the Microsoft project for music sharing that supports some communication mechanisms such as discussion forums and search features. This way, Groove fulfills the requirements of OSNs. Figure 4.2 shows a big picture of decentralized approaches.

In this section, we discuss the decentralized approaches that try to address the OSNs' privacy concerns with the main focus on data availability, users' privacy and searching in distributed environments. The approaches that are selected are the state-of-the-art ones that have attracted significant attention in this scope.

PeerSoN (short for "P2P Social Networking") [BSVD09] tried to overcome two limitations of OSNs which are privacy issues and the requirement of Internet connectivity while keeping OSN features like searchability. The main properties of PeerSoN are encryption, decentralization, and direct data exchange. Encryption provides users' privacy and decentralization based on a P2P infrastructure. This limits the full accessibility of an OSN provider to user data and makes it easier to integrate direct data exchange between users' devices into the system without the need for Internet connectivity. PeerSoN benefits from a two-tier architecture to achieve its goals: look-up service and peers. For look-up service, a Distributed

[1]http://ca.napster.com/
[2]http://www.slsknet.org/
[3]https://music.microsoft.com/

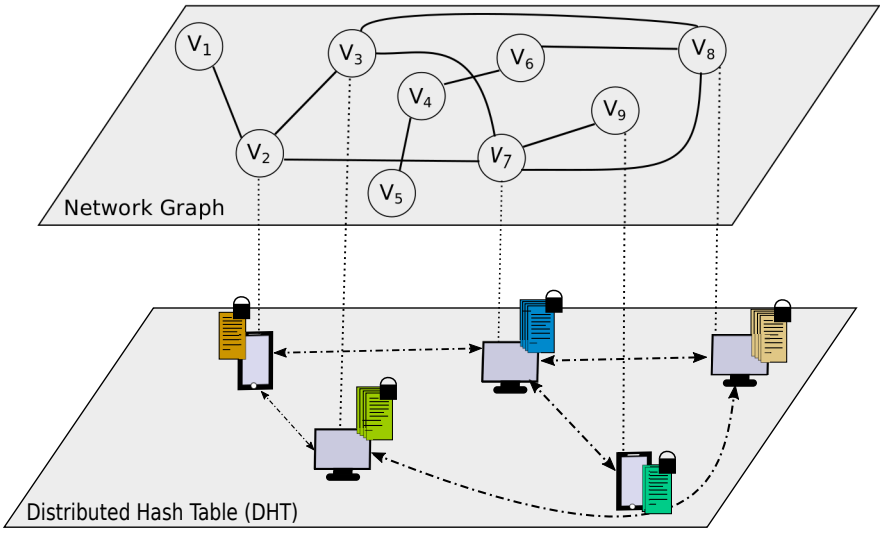


Figure 4.2: An example of decentralized approaches: user data is stored in users' storage servers

Hash Table (DHT), which is implemented using OpenDHT[1], is used in which the meta-data about users and their data are stored. Users' offline messages are stored on DHT for further accessibility.

To protect users from potential privacy violations by the provider, Safebook [Str09] introduced a three-tier decentralized architecture relying on cooperation among some social network users called peers. Safebook nodes form three types of overlays: matryoshkas, P2P lookup service, and trusted identification service (TIS). Matryoshkas form concentric rings of nodes surrounding each central node. Matryoshkas act as trusted data stores, and as interaction points with other central nodes. Direct contacts of each node shape the innermost shell of a matryoshka called mirrors, which stores the core's data in an encrypted form. Entrypoints are the nodes in the outermost shell of a matryoshka, which acts as a gateway for all data requests addressed to the core. In Safebook, each node is known by an unambiguous identifiers or pseudonym which the mapping between nodes and their pseudonyms are stored on TIS. In Safebook, each user has to trust other

[1]www.opendht.org

nodes as her private data are stored encrypted on other node's storages and all data transfers are done through those nodes. This may be one of the weaknesses of Safebook which causes some worries for its users.

LifeSocial.KOM [GGS+11] is another P2P architecture for OSNs with a number of plugins that provide critical features of OSNs such as search, chat and group generation. The core network layer of LifeSocial.KOM is a structured DHT-based P2P overlay. Each user can define a privacy policy for their contents and include the public key of the users authorized to access the contents.

LotusNet [AR12] is a P2P architecture for OSNs with a flexible and fine-grained access control system. The framework is shaped by three layers: identity management, services and application logic. Identity management resides in a DHT structure for handling interactions. Content is grouped by content type. Upon sharing, the identity of the owner and the receiver, an expiration time and a regular expression that specify the granted content types, is grouped into a grant certificate. In this way, grants are paired with social contacts, rather than with shared resources and limits the number of grant certificates.

Like LotusNet, the Prometheus access policies are a combination of some elements, such as type of relationship, weight that specify the trust level of the relationship and the users locations [KFA+10]. The user is able to define both white lists and black lists to include or exclude specific users. The P2P architecture of Prometheus is based on a DHT overlay. In their overlay, each user has a group of trusted peers that act as user's replicas and are responsible to provide the data availability in the when the user is offline.

Porkut [NPA10] is another decentralized OSN which focuses mainly on availability by replicating user data on trusted friends' storages. This approach considers the geographical locations and online time patterns of users to select the best replica nodes by different greedy algorithms, each of which concentrates on different cost minimization objectives such as access cost, number of replicas and storage cost.

Also, a privacy preserving indexing mechanism is introduced which facilitates content discovery among friends. The indexes are stored on a DHT-based table in the form of $(key, value)$ pairs. Search terms are the keys of the index table and the user profile identifiers are mapped to the value fields. In order to protect content and owner privacy, the pairs are K-anonymized and will be published into index table. By K-anonymizing the pairs, each pair is identical with at least $K-1$ other pairs [Swe02].

DECENT [JNM+12] is a modular and object-oriented architecture which uses a DHT to store user data and supports flexible attribute policies and fast revocation. There are three access policies for each object in DECENT: Read policy (R-Policy), Write policy (W-Policy) and Append policy (A-Policy). R-Policy is an attribute-based policy which describes the read access policy of each object. W-Policy is an identity-based policy which is assigned to the object owner and describes who can modify or delete the content of an object. A-Policy is an attribute-based policy which describes who can add a comment/annotation to an object. DECENT uses a mechanism for fast revocation which is the same with the one used in EASiER [JMB11] with attribute delegation support. So, the users can define a friend-of-a-friend attribute and ask all her contacts to delegate it to all of their contacts. DECENT can protect users' relationships from third-parties that may have no relationship with the object owner and are therefore untrusted, such as storage nodes.

As we stated earlier, one of the final goals of P2P architectures is to introduce a mechanism to make social networks available without the need for Internet access. To achieve this goal, users' profiles should be accessible even when they are offline. Rammohan Narendula [Nar12] introduced a mechanism to model user online times in OSNs from their activity times. As in a decentralized architecture, user data is stored in replicas which are mainly users' friends, the online time of each user depends on the online time of their replicas. Therefore, there should be an overlap

between user's online time and, at least, one of the nodes which is selected as a replica node. The solutions which are suggested in [Nar12] to choose the best replicas are: selecting the minimum number of replicas which have a time overlap with the other replicas, selecting the top-k most active friends and finally, selecting random friends which should be connected in time.

The authors of Pesca [RJM15] believe that the techniques which are used to consider the availability of content in decentralized OSNs without Internet access should consider not only the user's status in terms of being online/offline in the network but also the access control assigned to the published data. Therefore, they introduced PESCA which enables the privacy of user data while considering its availability by employing a broadcast encryption mechanism. In this framework, the availability of each user is defined as a user online table (UOT) consisting of users' online patterns. Users' online patterns are extracted from the times each user communicate with her social friends by tracking her resource usages such as storage or processing power usage. In PESCA, the best candidates to be selected as replicas are the ones which are online at the time of sharing a content and also are online when data audiences may access the content. A dynamic algorithm selects the minimum number of online friends as replicas based on the union of the up-times of data owner and her friends.

The main focus of eXO [LNTM11] is on searching and content discovery in distributed OSNs. It uses a DHT overlay resource discovery for retrieving stored content and they also enable the users to add tags to the content. However, tags are not included in the global indexer and are stored on the owners' computer. These tags are used for query expansion: upon receiving a response form the DHT, the user can contact the corresponding owner in the query response to retrieve the related tags and issue a new query based on those tags.

Cachet [NJM+12], which is the improvement of DECENT [JNM+12], is an architecture which protects confidentiality by an attribute-based encryption, integrity

by digital signature and availability of data in distributed OSNs. User data is stored encrypted on untrusted nodes which shape a DHT. The authors argued that showing a newsfeed to the user with hundreds of friends in a decentralized OSN protected by an encryption mechanism, requires fetching the wall updates from all their friends' profiles and decrypting them, which takes minutes to be completed. This process would be non-practical if the number of updates increases. Therefore, they introduced a gossip-based social caching algorithm, which increases the performance of newsfeed displaying. More specifically, when a user posts content on her wall, online contacts who satisfy the ABE policy defined by the user provide cached, decrypted content to other contacts who also satisfy the policy. So when an offline contact gets online and wants to view the latest newsfeeds, an algorithm locates other online contacts which have a cached version of new updates and query them to be retrieved. The DHT is used for retrieving updates which may not be cached, which ensures higher level of data availability.

Self-Organized Universe of People (SOUP) [KLF14] is another framework to guarantee data availability with minimal replication overhead and without assuming any permanent online storage. This framework is able to handle both high churn of regular participants and attacks from malicious users. The replica nodes are selected in two modes: bootstrapping mode and regular mode. When a node joins the network newly, it is entered in the bootstrapping mode in which the node begins to learn from the experience of its neighbors about their mirrors. After that, the node goes into the regular mode and in this mode, the set of mirrors may be changed periodically based on the exchanged experiences between nodes. SOUP also employs a protective dropping algorithm to enable the nodes to decide they accept extra mirror requests which may need to drop the data of some of the nodes.

The authors of My3 [NPA12] also put their focus on providing the data availability in distributed OSNs and also try to solve the data inconsistency problem. Based on their results, the authors believe that with having 4-5 replicas for each user, they

can achieve availability higher that 90%. In their replica placement algorithm, the minimum number of replicas are selected among the user's trusted friends and the content is stored on the replicas as plaintext. Therefore there is no confidentiality involved in the My3 approach, in contrast to the previous approaches in this chapter such as Pesca [RJM15]. To provide data consistency, when a replica comes online, it notifies all other online replicas and updates its content with updates from the others.

A recent work discussing the availability problem in decentralized OSNs is DiDuSoNet, proposed by Guidi et al. [GADS+16]. In this work, the authors proposed a Dunbar-based, specific kind of sub network with a limited number of relations per each user. The dynamic framework in which the replicas are selected from a user's trusted friends in a way that the replica set is changed dynamically due to users churn. When a user u joins the network, it selects a user among its online friends, say v, as a replica and replicates its profile on that node. When v becomes offline, u is responsible to select another replica when online, otherwise v should select another replica and replicate u's profile on the new replica and then, v can leave the network. It means that the set of replicas for each user is selected dynamically and the users' profile is replicated each time on one's storage. The best candidate to be a replica node in each join/leave is selected based on a combination of common friends between the data owner and the replica, online duration periods of nodes and the number of contact frequencies between nodes. It is stated in the paper that by replicating a user's profile on only two online replica, maximum availability is guaranteed in this framework. The authors also introduced FRoDO [AGGR15], which is a protocol that can be applied to any P2P systems including structured and unstructured P2P networks.

In Table 4.2, we can see a comparison of proposed approaches for decentralized OSNs. This table clearly shows that just a few of these approaches can present a mechanism which preserves the search feature of OSNs. Using search features in

OSNs, users can find their friends from real life in the OSN or make new connections based on common interests [BD09].

Methods	Storage	Interaction Method	Availability	Search
PeerSoN [BSVD09]	Nodes in DHT	DHT	–	–
Diaspora [RG]	User trusted storage (pods)	DHT	–	–
Safebook [Str09]	Trusted friends	DHT	–	–
LifeSocial.KOM [GGS+11]	DHT	DHT	–	✓
LotusNet [AR12]	DHT	DHT	–	✓
Prometheus [KFA+10]	Trusted peers	DHT	✓	–
Porkut [NPA10]	DHT	Trusted friends	✓	✓
DECENT [JNM+12]	Random nodes in DHT	DHT	–	–
Narendula et al. [NPA10]	Trusted friends	Direct or using a third-party server	✓	✓
Narendula et al. [Nar12]	Trusted friends	Direct or using a third-party server	✓	–
PESCA [RJM15]	DHT resides in users' storage space	DHT	✓	–
eXO [LNTM11]	DHT	DHT	–	Partially supported
Cachet [NJM+12]	Untrusted nodes in DHT	DHT	✓	–
SOUP [KLF14]	Trusted friends	DHT	✓	–
My3 [NPA12]	Trusted friends	DHT	✓	–
DiDuSoNet [GADS+16]	Trusted friends	DHT	✓	–

Table 4.2: Comparing the privacy approaches proposed for decentralized OSNs

4.2.3 Hybrid OSNs

In most hybrid approaches, users can decide about where to store their private data. In Figure 4.3, an overview of hybrid approaches is shown in which user's private data is stored on personal servers, while public data is stored on the provider's servers which are controlled centrally.

Raji et al. [RMJM11] proposed a privacy protection mechanism which employs an identity-based broadcast encryption in which the relation keys are not stored anywhere and they are obtained by a broadcast encryption algorithm in each request. In this approach, the setup algorithm of the BE scheme is responsible for generating the public/secret key pair for each user and the users can choose a storage server for storing their private data. The selected storage server can be a third-party server or it can be the storage provided by the OSN provider.

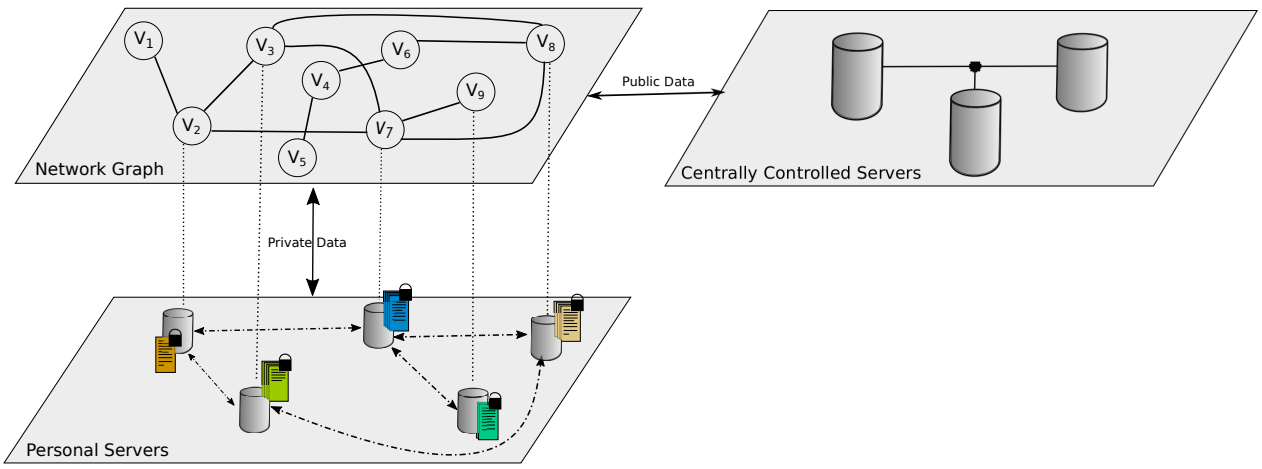


Figure 4.3: An example of hybrid approaches: private data is stored on personal storages

Vis-a-Vis [SLC+11] is a decentralized framework for OSNs which is based on the privacy-preserving notion of a Virtual Individual Server (VIS). A VIS is a virtual machine which is running in a paid cloud-computing utility such as Amazon Elastic Compute Cloud (EC2) or Rackspace Cloud Servers. The main focus of Vis-a-Vis is to protect users' shared locations from unauthorized entities by storing them encrypted on VIS. Also, Vis-a-Vis allows users to apply range queries to data items by using a hierarchical structure rather than a distributed hash table (DHT).

Polaris [WSW+11] is another distributed OSN which improves users' privacy and preserves economic incentives for the OSN providers. In Polaris, users can select different types of storages based on different application domains. As an example, users can store their public photos on the photo sharing websites and store their private data on their own mobile devices. In the end, there is a trade-off between users' sensitivity about their data and their economic preferences.

In Confident [LSC+11], two different types of servers are used for their decentralized architecture: desktop and enterprise storage servers and cloud-based name servers. The storage servers host user data and the authentication information are selected from the users' trusted friends. Confident relies on the social trust between

friends. Therefore, data is stored plain on the storage servers. A list of available replicas and a logical clock per each user are stored on the name servers, but name servers cannot access the plain data as they are not trusted to access them.

4.2.4 Healthcare online social networks

To the best of our knowledge, a limited number of studies have been done with the main focus on preserving users' privacy in Healthcare online Social Networks (HSNs). Most of the works try to present some recommendations to protect the privacy of health data and make the users aware about the potential privacy breaches over HSNs and teach them how to protect their data using different settings provided by OSNs.

In [WWJ10], Williams et al. discuss the existing challenges of information management in HSNs and explain how software developers can design a built-in privacy protection OSN. The authors suggest that all the organizations which deal with user data must adopt certain privacy principles such as: proactiveness of privacy not its reactiveness, which means to fix privacy problems before they happen, privacy by default, privacy by design, end-to-end lifecycle protection and respect for users' privacy.

The authors of [Li13] believe that policy makers and stakeholders are responsible for keeping the online health data private and they should consider the suggestions below in designing a secure OSN:

- Privacy awareness: minimize the amount of data which is shared by users to accomplish the intended purpose.
- Privacy by education: prepare the users with a user-friendly way of privacy settings.
- Privacy by design: building data protection and privacy by design into the platform.

- Privacy by regulation: using users' data with their consent and prohibiting inappropriate uses of health data.

Charbonneau et al. [Cha16] report a content analysis of privacy policies and disclosure practices for 25 online ovarian cancer communities and they present a coding sheet instrument to collect data from online ovarian cancer communities. The data were collected based on four primary areas:

- notification to participants about how personal information is collected, used, or shared
- choices for participants to opt out from sharing their data with third parties
- details about security measures used to protect personal health information
- ability for participants to access, modify, or delete the personal information

The results of the studies in this paper show that 96% of sites collect personal information from their users and share them with third parties, 56% of them use cookie technology to track users' behaviors, 36% of them offered opt-out choices for sharing data with third parties and just 28% of them allow users to delete their accounts whenever they want.

Jingquan Li [Li15] also analyzes the privacy and security characteristics of HSNs and believe that an effective protection for both the HSNs and their users is accessible based on a shared responsibility between the OSN provider and the users. Loiselle et al. [LA17] apply decentralization to protect users' sensitive data and they suggest to create HSNs on existing blockchain architecture.

HealthShare [LBL$^+$12] is one of the approaches which presents a practical solution to protect HSNs user data by two attribute-oriented authentication and transmission schemes. The attribute-oriented authentication scheme enables users to generate a tree structured attribute proof for themselves to anonymize their

sensitive attributes. The attribute-oriented transmission scheme enables the users to encrypt their health information into a ciphertext bonded with a customized access policy with two modes: direct and indirect. For direct mode, users create the access policies by themselves; for indirect mode, a delegated user may help to create an appropriate access policy for the received ciphertext without having an access to the content.

The most recent work on designing a healthcare-focused application is a Mindfulness Virtual Community (MVC) that Morr et al. [EMMA+20] have presented. They have developed a user-centered platform for York University students that enables them to interact with other students and psychologists in a P2P environment. As with previous approaches, the authors of MVC employ decentralization for solving the privacy problems in healthcare environments.

4.3 Discussion

Privacy Education. As we stated before, one of the main concerns of OSN users is about where their private data is stored and for which purposes their personal information is used. However, in general-purpose OSNs such as *Facebook* and *Instagram*, users trust OSN providers to keep their data private and control the spreading of their data using pre-defined settings of the platform. But the fact about how users' data is used is far beyond the users' thoughts. Based on an article which was published by ProPublica[1] in late 2016 [AMPJ16], *Facebook* is buying data about users' online and offline life from commercial data brokers to enhance its advertisements. One of the brokers which *Facebook* signed a deal with in 2012 is *Datalogix* which filed a complaint with Federal Trade Commission alleging *Facebook*'s violation of privacy. This article indicates that *Facebook* is willing to tell its users many things which it knows about them, but not all the things which it

[1]https://www.propublica.org/

provides to advertisers.

These facts show that in a general purpose OSN like Facebook, an educating mechanism is needed to teach new users how to protect their own data using default settings of the platform and make the users aware of the way OSN provider may use their data for other purposes.

Types of relationships. In each type of OSN we have different types of users. In general-purpose OSNs such as *Facebook* users are connected to each other based on their common language, the geographical region in which users are living and the friendship relations which they have with each other in real life. In professional OSNs such as *LinkedIn*, relationships are based on common professional skills and job preferences of the users. *Sermo* and *Doximity* connect experts and patients to each other based on their medical concern or their users' expertise. More specifically, we investigate the type of relationships in some HSNs and we compare them with the type of relationships in general-purpose OSNs. A notable difference between HSNs and general-purpose ones is in the type of friend relationships between users. In general-purpose OSNs, the relations between users are mostly based on the friendship relations which they have with each other in real life. But in HSNs, the relations are mainly based on common health problems and expertises.

In some HSNs like PatientsLikeMe, the users are people which have some health problems and want to find other people with the same problem in order to benefit from each others experiences about how to deal with their problem. If we call these type of users as patients, the relationship between users is of type *patient-patient*. In some other HSNs like Doximity[1] and Sermo, users are health-care professionals which want to stay connected to each other and broaden their expertise by sharing their thoughts and experiences. If we call these users as experts, the relationship between users are of type *expert-expert*. Besides these two types of HSNs, there are some other OSNs like Inspire which provides a platform to connect patients to other

[1]https://www.doximity.com/

patients and experts to access online help from other users. The type of relations between users are *patient-patient* and *patient-expert* in such platforms.

Privacy Solutions. As the type of users are different in different OSNs, their privacy concerns are different, too. Moreover, new types of requirements may emerge to fulfill the users' needs. For example, a more powerful group and friend recommendation is needed in HSNs in comparison with general-purpose OSNs. The reason is that in HSNs the relations are established based on some common health problems between the users who do not know each other in real life, while in general-purpose OSNs, the relations are mostly established based on the relations the users have in their real life. Therefore, it is not applicable to design a unique privacy framework which can be applied to all types of OSNs.

Another important factor in designing a privacy-enabled framework is its applicability in current OSNs. As we discussed in previous sections, remarkable efforts have been done to protect user data by encrypting it before they would be stored on providers' storages. However, in most OSNs such as *Facebook*, the provider does not allow the users to send encrypted data to their storages and if any application leaves some traces of encryption or other means which prevents the provider from learning user data for different purposes including advertisements, they may be removed by the provider [LXH09].

4.4 Recommended Privacy Solution

If we want to present a user-centric architecture with the main purpose of preserving users' privacy and overcome to some limitations such as need for Internet connectivity, P2P architecture would be a good choice; while in HSNs where users' generated contents are used for researching and analyzing purposes by other companies, a hybrid approach would be a better choice for protecting user data with the possibility to provide users' generated content for researchers if the user apply

permissions on releasing her private data anonymously.

However, there are still some limitations to design an acceptable architecture for decentralized OSNs to convince users to shift from traditional client-server to P2P infrastructure. Some of the limitations that we can mention are:

- Privacy approach: what mechanism should be used to preserve users' data privacy while it should be dynamic and flexible?
- Key management and distribution: what mechanism should be used to store encryption keys confidentially and how to distribute them among users?
- Interaction method: how to handle communications between users?
- Storage: how and where user data should be stored?
- Topology: how the users should be connected to each other?
- Availability: how to guarantee the availability of published content even when the users are offline?
- Searching: how to search for your real life friends in OSN or find new friends based on common interests?
- Openness: how to design an open platform to support other third-party applications?
- Robustness: how to handle disruptive behavior of users while there is no single entity to define the rights?

In Figure 4.4, an overview of the recommended P2P architecture is shown. Our architecture consist of a two-tier architecture: Data Exchange Tier (DET) and Lookup Tier (LT). User data is stored encrypted on users' devices which are located on DET and the lookup operations are done on LT. The colors which are used for the nodes in LT are related to the color of the users' stored documents in DET. In this section,

we will introduce the steps needed to overcome most of the above challenges to design a P2P architecture and discuss the possible solutions in each step.

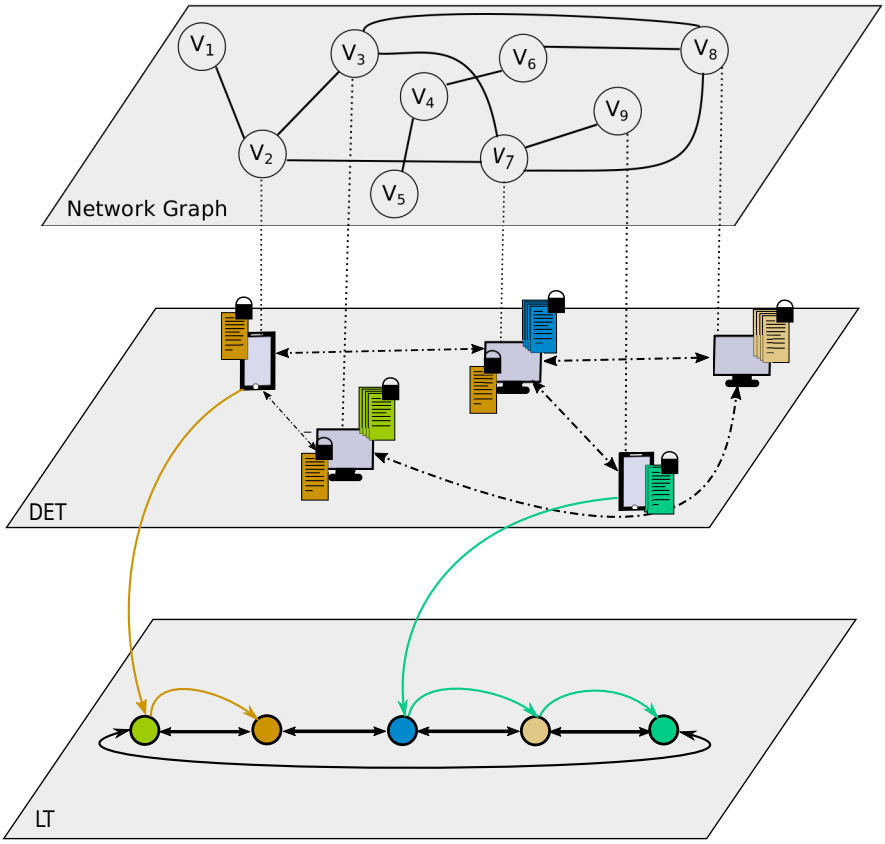


Figure 4.4: Recommended privacy solution: user data is stored on Data Exchange Tier (DET) on users' devices and the lookup operations are done on Lookup Tier (LT). Data replication is used to increase the data availability.

4.4.1 Encryption mechanism

Using a P2P architecture for OSNs, we eliminate the privacy breaches caused by OSN providers which act as a big-brother for all the users and we give back data control to the users. However storing user data on storages other than central storages cannot guarantee the data privacy for users as there may be other unau-

thorized data accesses based on where data is stored. So, an important issue which should be considered in designing a P2P architecture is choosing an appropriate encryption mechanism as well as a good approach for key management and distribution. Moreover, data availability and searching are the other important concerns in P2P architectures. In the remainder of this section, we propose our solution for these challenges.

To provide the data confidentiality and access control for OSN users, we have to employ a suitable encryption technique to support group encryption. Such type of encryption techniques are called secure group communication, which is the first requirement for secure data communication in OSNs. Also, the technique which is used should have following features so that it could be applied to the OSN platforms:

1. Dynamic: OSN platforms are dynamic environments in which the relationships between users are changing frequently during time. So, the selected technique should be able to handle such dynamic changes efficiently.

2. Efficient: the cost of encryption/decryption should be independent of the number of recipients.

3. Low storage: the overhead storage which is needed to store encryption headers should be minimum so that the cryptosystem would be scalable.

4. Stateless: revoking some users from a group should not cause the remaining users to update their private keys.

5. Fully collusion resistant: all the users other than group users cannot collude to decrypt a broadcast message.

6. One-to-Many communication: the initializer of a communication group would be the broadcaster in her group and can encrypt her data and broadcast it to other group members.

7. Forward/backward secrecy: a newly joined member should not be able to decrypt former encrypted data (forward free) and a revoked member from a group should not be able to decrypt later encrypted data.

One of the encryption mechanisms which can be applied to decentralized OSNs is the adaptive public-key broadcast encryption which was introduced by Gentry et al. [GW09]. This approach is secure against any number of colluders and the ciphertext generated by this approach is of constant size for any number of receivers. The randomized algorithms which made the broadcast encryption are *Setup*, *keyGen*, *Encrypt* and *Decrypt*. The details of each algorithm is as follows:

$\boldsymbol{Setup(\lambda, n) \rightarrow \langle PubK, SecK \rangle}$. This algorithm runs $GroupGen(\lambda, n)$ to generate two groups $\mathbb{G}$ and $\mathbb{G}_T$, which are of prime order p with bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. Then picks $\alpha \xleftarrow{R} \mathbb{Z}_p$ and $g, h_1, \cdots, h_n \xleftarrow{R} \mathbb{Z}^{n+1}$.

$$PubK = (g, e(g,g)^{\alpha}, h_1, \cdots, h_n)$$
$$SecK = g^{\alpha}$$

The *Setup* algorithm takes n, which determines the number of receivers in the group, and λ as security parameter, which determines the maximum size of broadcast group, and it outputs a public/secret key pair $\langle PubK, SecK \rangle$, which is kept by the broadcaster.

$\boldsymbol{KeyGen(i, SecK) \rightarrow PrvK_i}$. Picks $r_i \xleftarrow{R} \mathbb{Z}_p$. Then

$$PrvK_i = \langle d_{i,0}, \cdots, d_{i,n} \rangle$$
$$PrvK_{i,0} \leftarrow g^{-r_i}, PrvK_{i,i} \leftarrow g^{\alpha} h_i^{r_i}, \forall_{j \neq i} PrvK_{i,j} \leftarrow h_j^{r_i}$$

The *KeyGen* algorithm takes index $i \in \{i, \cdots .n\}$, which is member's identity, and secret key *SecK*. It outputs a private key $prvK_i$ for member i of broadcasting group.

$\boldsymbol{Encrypt(S, PubK) \rightarrow (Header, K)}$. Picks a random $t \xleftarrow{R} \mathbb{Z}_p$. Then

$$K = e(g, g)^{\alpha.t}$$
$$Header = \left(g^t, (\prod_{j \in S} h_j)^t \right)$$

The *Encrypt* algorithm takes an $S \subseteq \{1, \cdots, n\}$ as the subset of users and *PubK* as a public key and outputs a pair $(Header, K)$. *Header* contains the data needed to help users in S find the message encryption key and K is the message encryption key, which is a symmetric key.

$\boldsymbol{Decrypt(S, i, PrvK_i, Header, PubK) \rightarrow K}$. If we imagine $Header = (C_0, C_1)$, then

$$K = e(PrvK_{i,i}. \prod_{j \in S, j \neq i} PrvK_{i,j}, C_0).e(PrvK_{i,0}, C_1)$$

The *Decrypt* algorithm takes $S \subseteq \{1, \cdots, n\}$ as the subset of users, user id $i \subseteq \{1, \cdots, n\}$, private key $PrvK_i$ of user i, a header *Header* and public key *PubK* as inputs and the output of the algorithm is the decryption key K if i belongs to S. The proofs of correctness of the above algorithms are explained in [GW09].

4.4.2 Topology

The way the nodes communicate with each other establishes an overlay on top of the underlying physical network and is different from the connectivity between nodes in the physical network which is called network infrastructure [Mal15]. For P2P architectures, we can apply a two-tier architecture: Data Exchange Tier (DET) and Lookup Tier (LT). In DET, the peers are connected to each other for direct data exchange and in LT, peers are connected to a structured overlay such as DHTs for lookup services.

Data Exchange Tier (DET). The connections between peers in DET are based on their friendship connections, i.e., there exist a connection between two nodes v_1

and v_2 in DET in our P2P overlay if a friend request was exchanged between these two nodes through a secure channel and the friendship was established. Here we only consider undirected edges for our graph. This rules out the both parties are involved in establishing the connection. For other types of connections, such as follower-followee in Twitter, we need to have directed edges, which we do not consider in this chapter. Following this strategy causes our social network graph to be partitioned into communities of friends and the size of the communities depends on the number of peers collaborating to each other within them. The existence of such communities affects directly the data availability in our P2P overlay [BD09] if the replica placement strategy selects the replicas among users' friends. Imagine that one of the generated communities, say C_i, would be small in size because of the number of its participants. The data availability in C_i would be low as data is stored on a small number of replicas, which have been selected from C_i's peers. Therefore, in the replica placement strategy, we should consider the community size and the availability rate of the data.

Lookup Tier (LT). Most of the structured overlays provide a key-value interface to work with them. Therefore, in our solution we need a key-value pair for each piece of meta-data related to a user, which can be stored in LT. The key is the corresponding index of a value and is used for searching the value. The keys could be a hash function of users' *UIDs* which we discussed in previous section.

The structured overlay we choose for our LT affects the overall performance of the overlay. In [KR07] the authors did some experiments to investigate the percolation effect in two known structured overlays, Chord [SMK+01] and Symphony [MBR+03] and they found out that 4% of peers are not reachable in a same component with the network size of 10^6. Also, in [WCZJ04], it is shown that direct communication between 36% of peers is impossible because of the blocks caused by firewalls and Network Address Translators (NATs). These experiments show the defects of the structured ring-based overlays in some special cases. Moreover, the

maintenance cost of rings is high in the environments with high churn, because in each join/leave of the peers in the overlay, their successors and predecessors should be updated. Therefore, in dynamic environment of OSNs, we need an overlay for our LT which can handle lookup and write operations correctly and its maintenance cost would be low.

In [GGD+08], a ring-less structured overlay called Fuzzynet is presented, which has all the functionalities of ring-based overlays, while achieving better performance in its lookup and publish operations with no explicit maintenance. As it is stated in the paper, Fuzzynet is ideal for high churn environments in which joining/leaving of peers are done frequently. This approach can be applied to the Lookup tier of the overlay.

4.4.3 Data availability

A crucial feature of OSNs is their need to a mechanism to make data available for all the users whenever it is accessed even if the data owner would be offline. In centralized OSNs, the providers are responsible for guaranteeing the data availability for their users by employing different data storages and distributing the overload between them. But in decentralized OSNs, the users themselves are responsible for storing their own data on their storages and ensure the data availability. However, one user which contributes to the OSN by generating a content is not online 24/7 to ensure the data availability for other users who want to access to generated content. Therefore, profile replication is a good solution to keep the data available when the data owner is offline, meaning that the data can be replicated on other users' storages, called replicas. Thus, we need a strategy to choose the best replicas for each user to make the data available whenever it is accessed.

The set which is selected as replica nodes for each node u should satisfy the following features:

1. At least one of the replica nodes should be online with high probability at the

time of sharing data items by u.

2. There should be time overlaps between u's friends which are authorized to access the data items and the replica nodes so that in each time, one or more replica nodes are online to serve data accesses by authorized users.

3. The size of replica set should be minimum as the nodes in our P2P architecture are ordinary users which use devices with limited amount of storage (even the device can be a cellphone) and we can not replicate users' data as much as possible just to increase the data availability.

4. The selected replica nodes should maximize the data availability. Our strategy may find different replica sets with the same size but different availability in the case of the overall online time periods of the replicas. In such cases, the set should be selected which achieves maximum availability.

5. The replica set should be resilient to dynamic nature of OSNs and the selection strategy should keep the set updated upon each join/leave without performance decrease.

We can model our decentralized OSN as an undirected graph $G = (U, E, T)$, where U is the set of users that have joined the OSN and are represented by the vertices in the graph, E is the set of friendship relations between users and are represented by graph edges and T is the set of online time patterns of each user with predefined granularity (e.g., seconds, minutes, hours), which can be stored in a table. As the online times are not certain and the status of one user may be changed, in each slot of the table, we can store the probability of the user being online or offline in that specific time period.
The replica placement problem can be stated formally as follows:

PROBLEM **1.** (REPLICA PLACEMENT). *Given an undirected graph $G = (U, E, T)$ and a node u, select a replica set $R_u \subseteq N_u$ such that*

1. $\forall v \in N_u, \exists r_i \in R_u$ *that is online with a probability bigger than a threshold value for each time slot*

2. $\forall t\ \exists r_i \in R_u$*, which the data stored on* r_i *should be consistent with the data stored on* u

3. $|R_u|$ *would be minimized*

In Problem 1, N_u is the set of u's neighbors. The list of nodes in set R_u should be stored in u's storage. Table 4.3 shows an example of the online time patterns for three nodes v_1, v_2 and v_3. Let us assume that v_2 and v_3 are some of the neighbours of v_1 and we want to find a replica set for node v_1 that satisfies the conditions stated in Problem 1. As the three nodes are online in the first time slot with a low probability, the replicas that are chosen does not need to guarantee a high availability for that time slot. This means that nodes v_2 and v_3 are less likely to request v_1's data in the first time slot. We can calculate the probability values of two nodes v_1 and v_2 being online in a specific time slot as:

$$Pr(v_1 \text{ or } v_2 \text{ being online}) = 1 - Pr(v_1 \text{ not online}).Pr(v_2 \text{ not online}) \tag{4.1}$$

We calculate the online probabilities for all v_1's neighbours using Equation (4.1) to find the online threshold values per each time slot. Then the replicas should be selected in a way so that the online probability of the replicas be bigger than or equal to the threshold value for each time slot.

In Section 4.2.2, we introduced and analyzed some approaches such as the ones proposed by Narendula et al. [Nar12, NPA12] and Raji et al. [RJM15] that try to solve Problem 1. In the later case, the authors consider extra parameters, such as the list of authorized users for each data item.

Discussion 1. As we stated before, online time availabilities of the users are initialized by themselves when joining the OSN, but it is very probable that the

Table 4.3: The online time patterns for three nodes

v_1 :

Time slot	1	2	3	...	23
Online probability	0.5	0.2	0.75	...	0.4

v_2 :

Time slot	1	2	3	...	23
Online probability	0.05	0.3	0.8	0.2	0.75

v_3 :

Time slot	1	2	3	...	23
Online probability	0.01	0.3	0.65	0.35	0.5

online time patterns would be changed due to the possible changes in users' habits in using the platform or changes in the users' geographical locations. Thus, we should consider such changes in online time patterns if we want to guarantee the data availability provided by the replicas. To keep the online time patterns updated, we can track the log records of the system, which hold some data of when a specific user goes online or offline in the system. Using such records of data, the online time patterns of users can be extracted automatically and if the changes in the online time patterns of user u exceeds a threshold value, the replica selection algorithm should be run again to update the set R_u.

Also, joining new users or leaving current users to/from the OSN platform is another scenario which causes some changes to the replica set R_u. If one of the users, which leave the OSN, would be a replica node, the data availability would be affected based on the storage space and time availability provided by the leaved node. Also, if a new user is added to the set which has access to the data item, new replicas might be needed if online time of this new user has a portion not overlapping with any existing replicas or u. These are the other cases in which the replica placement algorithm needs to be run again for the user which has been affected by the other nodes' leave or join.

Discussion 2. The replica placement algorithm finds the replica nodes for user u to guarantee the availability of u's data for all her friends, but as we stated in Section 4.4.1, a user can encrypt her data in order to be accessible by just a specific group of users who are determined in the $Setup$ phase of the broadcast encryption mechanism. This means that by defining such access levels, u's data is not needed to be available for all her friends, but for users in encryption group. In such cases, the replica placement algorithm should find the replica nodes for u to guarantee the availability for a smaller group of nodes called $Authorized$ nodes ($Auth_u$). Also, for each data item which is shared by u, we need to have a different set of replicas. Following this approach, we store the data items of one user on different sets of replicas, which prevents the overloading of a particular replica set if all data items would be stored on one replica set. On the other side, we have a decrease in the overall performance of our platform as the replica placement algorithm needs to be run per each data item. Therefore, there should be a trade-off between the performance and the load balancing.

Discussion 3. We replicate user data to increase the availability, but this replication brings with itself some downsides such as managing data consistency on all replica nodes, which is costly in terms of performance. Data consistency means that the data on all replicas should be the same and any modification on data should be carried out on all the copies to ensure consistency [TVS07].

As we stated before, the list of replicas for each node u is stored on its storage. Therefore, node u can act as a master node for other replica nodes in the set R_u and nodes $r_i \in R_u$ are backup nodes. The reason that node u can be a master node is that u is the data owner and can make any modifications to its own data. The master node is responsible to hold the list of replicas, list of changes to its own storage including updating a data item or creating new data item and should be

able to track which replica nodes have applied the changes to their local storage.

Discussion 4. Another important factor which needs to be considered in selecting the best candidates as replicas is the relation between node u being replicated and the candidate nodes. In some papers such as [BSVD09, JNM+12, NJM+12] the candidate nodes are selected from the random nodes or untrusted nodes and evaluating their effectiveness as a replica node is done by the replica placement algorithm. However, in most of the recent studies ([Nar12, KLF14, GADS+16]), the replica nodes are selected from the trusted friends of u. De Salve et al. [DSDGR16, DSGR17] discussed that there is a strong relationship between each user and its direct friends by considering online times patterns. They showed in their experiments that the stronger ties exist between a user and its friends, the more probable those users are similar in their online times. Also, they concluded that users have more probability to be online when at least 10 of their Dunbar friends [Dun98] are online. Dunbar friends are defined as the direct friends of a user with strong tie strength, where the tie in OSNs is measured as the contact frequency or the number of direct interactions and the number of social interactions such as posts, comments and tags.

4.4.4 Searching

An important feature, which all OSNs should support, is a mechanism to enable users to search for their friends from social life or find new friends based on common interests. In centralized architectures, enabling this feature is possible without too much effort as the list of users and the metadata related to each user is stored centrally and executing queries on stored data to extract required results is possible. But in P2P architectures where data is stored decentralized, executing such queries efficiently is hard to achieve. Therefore, we need a mechanism to relate each data item to its owner and determine each user by its interests based on what he/she had published to enable efficient searching. In meanwhile, users' privacy should be

preserved and the relations between them should not reveal any information.

One of the ways for annotating content in OSNs is data tagging. In [GSS08], the authors introduced a tagging approach for decentralized environments, which is based on vector space model [SWY75], to characterize users, tags and resources. This approach can be applied to our decentralized OSN as it can provide the requirements we need for our architecture. In this framework, each feature (user, tag or resource) is represented as a feature vector and the weight of elements is calculated as a combination of *item-to-item frequency* and *inverse item frequency*. Then the similarity between feature vectors is calculated based on the common notion of IR-style cosine measure [NTW06]. The users which have bigger similarity values can be considered as users with common interests based on what they have tagged on their published data.

To preserve users' privacy in our search framework, we can use unique identifiers for users, tags and resources. This identifiers can be obtained from a hashed value of a unique attribute of the features and should be stored encrypted in our overlay. Another way to protect the identifiers and to hide the relations between each identifier and its main source is to *K*-anonymize each identifier [Swe02].

4.5 Requirements of a privacy-enabled approach for healthcare online social networks

In Section 4.4 we explained how to construct the required foundations of a P2P architecture, which preserves users' data by distributing and storing it on users' storages. It also enables users to apply a broadcast encryption approach to their private data whenever needed. The privacy requirements of Healthcare online Social Networks (HSNs) are almost the same with the privacy requirements of general-purpose OSNs. In both environments, users have some concerns about how their data is accessed and for which purposes their data can be used. However,

there are some differences between these two types of OSNs. In this section, we investigate the HSN requirements.

4.5.1 Privacy Policies

The users of HSNs have more privacy concerns in comparison with general-purpose OSNs as the data which is stored on HSN storages are related to health conditions of users and are needed to be kept as private as possible. In order to know how existing HSNs respect to their users' concerns, we investigate the privacy policies of four known HSNs, which have a reasonable number of active users:

- PatientsLikeMe[1]: a sharing web site with the aim of providing a platform for patients to interact with each other and benefit from each others' experiences.
- Sermo[2]: a platform to enable doctors to virtually meet each other and exchange their knowledge.
- Inspire[3]: an online community for patients, family members, caregivers and health professionals, which is provided by ClinicaHealth, Inc.
- QuantiaMD[4]: an online physicians' community provided by Aptus Health Holdings, Inc.

Advertisement and Third party companies. All the above platforms need user registration to let them write comments, communicate with other users, post messages, upload photos, etc. In the process of registration, the users are required to input personal information about themselves which is stored and kept in companies' servers. In the privacy policy of all above companies, it is stated that the personal

[1]https://www.patientslikeme.com/
[2]http://www.sermo.com/
[3]https://www.inspire.com/
[4]https://secure.quantiamd.com/

information of the users would be provided for third parties with whom the company has business relationship and the company's sponsors. Also, in some special cases, users' information would be accessible to governments or other entities in connection with a legal process. Interestingly, the companies assert that the users themselves are responsible for what they share in the platforms and there is no guarantee to keep their data private.

In the privacy policy of PatientsLikeMe, it is stated that "Members are encouraged to share health information but should consider that the more information that is entered, the more likely it is that a Member could be located or identified" [Tea20b].

Cookies and session cookies are the other technologies which are used in above platforms in order to not only improve their platform's functionality, but also to track the users' behavior in seeing advertisements provided by third parties and as its result, to improve the advertisement policies. Besides these technologies, Inspire uses the "tracking feature of Google AdWords to measure the effectiveness of AdWords advertisements" [Tea20a].

Account deactivation. In all the introduced platforms, there is no option for account deactivation in the user profile and if a user wants to deactivate her account, it can be done by sending an email to the provider with a keyword like "DEACTIVATE" in its subject line. However, deactivating an account does not mean user data also is deleted from the servers. In the privacy policy of PatientsLikeMe, it is stated that "If a Member chooses to deactivate their account, PatientsLikeMe will not display or sell the Members Personal Data as of the date of deactivation. However, the Members Personal Data, including Shared and Restricted Data, will remain in the system unless you contact our community team to request that your data be deleted." [Tea20b].

Opt-out options. As a user is registered to one of the introduced platforms, her information is available for third parties and companies' sponsors, as they stated in their privacy policy, and there is no opt-out option for users to disallow the company in selling their information to third parties. In privacy policy of PatientsLikeMe, it is stated that "When a Member chooses to share Personal Information via a free text field (e.g. forum, treatment evaluations, annotations, journals, feeds and adverse event reports) and photos or images, the information shall be treated as Shared Data" [Tea20b]. The only option which is provided in Sermo is that "We may then provide our partners with engagement effectiveness metrics that include the sharing of some minimal personal data for performance assessment purposes. To the extent required by law, we will collect your explicit consent prior to providing such identifiable information" [Tea20c].

Merging. In the privacy policy of all above companies it is stated that users' data including personal information collected from them would be completely transfered to another company which buys the company as a result of sale, acquisition, merges or bankruptcy. Upon merging, "PatientsLikeMe may transfer the Shared Data, Restricted Data, and Platform Use Data to any successor to its business as a result of any merger, acquisition, asset sale, bankruptcy proceeding, or similar transaction or event, with such successor bound by the terms of this Privacy Policy with respect to its use and disclosure of such information" [Tea20b], "ClinicaHealth will seek to obligate the acquiring company to use any personal information transferred by this Site in a manner consistent with this Privacy Statement, but cannot guarantee that it will be able to impose that requirement, or that the acquiring company will comply" [Tea20a].

All the above items clearly demonstrate that users do not have any control over their personal data as the data would be stored in the companies' storages and there is even no settings or options for the users to define access levels for their data.

Moreover, user data is accessible by third-party companies for advertisement and research purposes upon the user's registration. While in the P2P platform that we introduced in the previous sections, user data is stored on the storages under their own control and the users can also define fine-grained access levels for each part of their data to determine who can access to what data. Furthermore, users can delete their shared data whenever they want and remove accesses just by defining new keys for the data items. Therefore, it is reasonable to apply our P2P architecture to HSN platforms to preserve users' privacy.

4.5.2 Friend and group recommendation

As we stated in Section 4.3, the main difference between HSNs and general-purpose OSNs is in the type of friend relationships between users. Based on the types of relations in HSNs, which are *expert-expert*, *patient-patient* and *patient-expert*, a powerful searching mechanism is strongly needed in decentralized HSNs to enable the users to find others which they don't know before but have common health problems or common expertises. The approach which was discussed in Section 4.4.4 can be applied to decentralized HSNs to support the search feature. When a new user wants to join the platform, she should define her role in the HSN as patient or expert. Then the user should selects some attributes for herself from the tag set T. The attributes are stored as tags and annotations for the user. For example, if a user, which has some heart problems and wants to join the platform to benefit from the experiences of other user with same problem and be tracked by an expert, she should introduce herself as a patient and assign herself some attributes about heart problems from the existing tag set T. The profile of the user is interpreted as the resource in our search framework which is tagged with appropriate attributes. The tags help users to find other patients and experts related to their health problems. Moreover, by combining the results of extracted tags related to specific users with extracted resources related to specific tags, we can find communities of users which

annotated themselves with tags related to the same health problems. Thus, new users can join to the communities of existing patients and experts with the same concerns.

4.6 Conclusion

In this chapter, we reviewed the existing solutions which have been presented to overcome the privacy problems which Online Social Network (OSN) users are facing with. A significant number of these solutions try to preserve user data by applying encryption mechanisms to existing client-server architectures. This type of solution is suitable for general-purpose OSNs which their services are freely available for their users. However, user data is controlled centrally by a unique authority. Therefore, decentralized approaches try to store user data on the servers controlled by the users and provide additional features such as enabling P2P and direct communications between users. Hybrid approaches try to combine these two approaches by encrypting just sensitive data and storing it on trusted servers.

The solution which should be applied to the OSN platform depends on the type of relationship between the users and the features which the OSN should provide for their users. By investigating the privacy policy of well-known general-purpose OSNs as well as Healthcare Social Networks (HSNs), we observe that the users are responsible for what they share on online platforms and the providers make user data available for third-party companies for advertisement and research purposes upon user's registration. Therefore, we recommend decentralized solutions to enable the users to have full control over their own data. However, defining a practical decentralized approach for OSNs is possible by overcoming the existing challenges which are explained in this paper. Also, in order to be able to apply a decentralized architecture to HSNs, a powerful search, and friend and group recommendation is needed as the friendship relations are established based on

common health problems or common expertises, and not based on the friendships in social life.

In Chapters 5 and 6, we will present a solution for storing users' data securely on untrusted content-addressable storage servers and provide partial sharing/subsetting in the filesystem level. That is our first step for designing and implementing a distributed private online social network.

Bibliography

[AGGR15] Tobias Amft, Barbara Guidi, Kalman Graffi, and Laura Ricci. Frodo: Friendly routing over dunbar-based overlays. In *2015 IEEE 40th Conference on Local Computer Networks (LCN)*, pages 356–364. IEEE, 2015.

[AMPJ16] Julia Angwin, Surya Mattu, and Terry Parris Jr. Facebook doesnot tell users everything it really knows about them. `https://www.propublica.org/article/facebook-doesnt-tell-users-everything-it-really-knows-about-them`, Dec 2016.

[AR98] Philip E Agre and Marc Rotenberg. *Technology and privacy: The new landscape*. Mit Press, 1998.

[AR12] Luca Maria Aiello and Giancarlo Ruffo. Lotusnet: Tunable privacy for distributed online social network services. *Computer Communications*, 35(1):75–88, 2012.

[BB13] Oleksandr Bodriagov and Sonja Buchegger. Encryption for peer-to-peer social networks. In *Security and Privacy in Social Networks*, pages 47–65. Springer, 2013.

[BBS+09] Randy Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. Persona: an online social network with user-defined privacy. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 135–146, 2009.

[BD09] Sonja Buchegger and Anwitaman Datta. A case for P2P infrastructure for social networks-opportunities & challenges. In *Wireless On-Demand Network Systems and Services, 2009. WONS 2009. Sixth International Conference on*, pages 161–168. IEEE, 2009.

[BKW11] Filipe Beato, Markulf Kohlweiss, and Karel Wouters. Scramble! your social network data. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 211–225. Springer, 2011.

[BSVD09] Sonja Buchegger, Doris Schiöberg, Le-Hung Vu, and Anwitaman Datta. Peerson: P2P social networking: early experiences and insights. In *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, pages 46–52. ACM, 2009.

[Car10] Nicholas Carlson. Letter about privacy concerns of google buzz. `https://www.priv.gc.ca/media/nr-c/2010/let_100420_e.asp`, April 2010.

[Cha16] Deborah H Charbonneau. Privacy practices of health social networking sites: Implications for privacy and data security in online cancer communities. *Computers, informatics, nursing: CIN*, 2016.

[CS05] Yacine Challal and Hamida Seba. Group key management protocols: A novel taxonomy. *International journal of information technology*, 2(1):105–118, 2005.

[DG10] George Danezis and Seda Gürses. A critical review of 10 years of privacy technology. *Proceedings of surveillance cultures: a global surveillance society*, pages 1–16, 2010.

[DG12] Claudia Diaz and Seda Gürses. Understanding the landscape of privacy technologies. *Extended abstract of invited talk in proceedings of the Information Security Summit*, pages 58–63, 2012.

[DSDGR16] Andrea De Salve, Marco Dondio, Barbara Guidi, and Laura Ricci. The impact of users availability on on-line ego networks: a facebook analysis. *Computer Communications*, 73:211–218, 2016.

[DSGR17] Andrea De Salve, Barbara Guidi, and Laura Ricci. Evaluation of structural and temporal properties of ego networks for data availability in dosns. *Mobile Networks and Applications*, pages 1–12, 2017.

[Dun98] RI Dunbar. The social brain hypothesis. *brain*, 9(10):178–190, 1998.

[EMMA+20] Christo El Morr, Catherine Maule, Iqra Ashfaq, Paul Ritvo, and Farah Ahmad. Design of a mindfulness virtual community: A focus-group analysis. *Health informatics journal*, 26(3):1560–1576, 2020.

[GADS+16] Barbara Guidi, Tobias Amft, Andrea De Salve, Kalman Graffi, and Laura Ricci. Didusonet: A P2P architecture for distributed dunbar-based social networks. *Peer-to-Peer Networking and Applications*, 9(6):1177–1194, 2016.

[GGD+08] Sarunas Girdzijauskas, Wojciech Galuba, Vasilios Darlagiannis, Anwitaman Datta, and Karl Aberer. Fuzzynet: Zero-maintenance ringless overlay. Technical report, 2008.

[GGS+11] Kalman Graffi, Christian Gross, Dominik Stingl, Daniel Hartung, Aleksandra Kovacevic, and Ralf Steinmetz. Lifesocial.KOM: A secure

and P2P-based solution for online social networks. In *2011 IEEE Consumer Communications and Networking Conference (CCNC)*, pages 554–558. IEEE, 2011.

[GSS08] Olaf Görlitz, Sergej Sizov, and Steffen Staab. Pints: peer-to-peer infrastructure for tagging systems. In *IPTPS*, page 19. Citeseer, 2008.

[GTF08] Saikat Guha, Kevin Tang, and Paul Francis. Noyb: Privacy in online social networks. In *Proceedings of the first workshop on Online social networks*, pages 49–54. ACM, 2008.

[GW09] Craig Gentry and Brent Waters. Adaptive security in broadcast encryption systems (with short ciphertexts). In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 171–188. Springer, 2009.

[JMB11] Sonia Jahid, Prateek Mittal, and Nikita Borisov. Easier: Encryption-based access control in social networks with efficient revocation. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 411–415. ACM, 2011.

[JNM+12] Sonia Jahid, Shirin Nilizadeh, Prateek Mittal, Nikita Borisov, and Apu Kapadia. Decent: A decentralized architecture for enforcing privacy in online social networks. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, pages 326–332. IEEE, 2012.

[KFA+10] Nicolas Kourtellis, Joshua Finnis, Paul Anderson, Jeremy Blackburn, Cristian Borcea, and Adriana Iamnitchi. Prometheus: User-controlled P2P social data management for socially-aware applications. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 212–231. Springer, 2010.

[KLF14] David Koll, Jun Li, and Xiaoming Fu. Soup: an online social network by the people, for the people. In *Proceedings of the 15th International Middleware Conference*, pages 193–204. ACM, 2014.

[KR07] Joseph S Kong and Vwani P Roychowdhury. Price of structured routing and its mitigation in P2P systems under churn. In *Peer-to-Peer Computing, 2007. P2P 2007. Seventh IEEE International Conference on*, pages 97–104. IEEE, 2007.

[LA17] Carmen G Loiselle and Saima Ahmed. Is connected health contributing to a healthier population? *Journal of Medical Internet Research*, 19(11):e386, 2017.

[LB08] Matthew M Lucas and Nikita Borisov. Flybynight: mitigating the privacy risks of social networking. In *Proceedings of the 7th ACM workshop on Privacy in the electronic society*, pages 1–8. ACM, 2008.

[LBL+12] Xiaohui Liang, Mrinmoy Barua, Rongxing Lu, Xiaodong Lin, and Xuemin Sherman Shen. Healthshare: Achieving secure and privacy-preserving health information sharing through health social networks. *Computer Communications*, 35(15):1910–1920, 2012.

[Li13] Jingquan Li. Privacy policies for health social networking sites. *Journal of the American Medical Informatics Association*, 20(4):704–707, 2013.

[Li15] Jingquan Li. A privacy preservation model for health-related social networking sites. *Journal of medical Internet research*, 17(7), 2015.

[LNTM11] Andreas Loupasakis, Nikos Ntarmos, Peter Triantafillou, and Darko Makreshanski. eXO: Decentralized autonomous scalable social networking. In *CIDR*, pages 85–95, 2011.

[LSC+11] Dongtao Liu, Amre Shakimov, Ramón Cáceres, Alexander Varshavsky, and Landon P Cox. Confidant: protecting osn data without locking it up. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 61–80. Springer, 2011.

[LXH09] Wanying Luo, Qi Xie, and Urs Hengartner. Facecloak: An architecture for user privacy on social networking sites. In *Computational Science and Engineering, 2009. CSE'09. International Conference on*, volume 3, pages 26–33. IEEE, 2009.

[Mal15] Apostolos Malatras. State-of-the-art survey on P2P overlay networks in pervasive computing environments. *Journal of Network and Computer Applications*, 55:1–23, 2015.

[MBR+03] Gurmeet Singh Manku, Mayank Bawa, Prabhakar Raghavan, et al. Symphony: Distributed hashing in a small world. In *USENIX Symposium on Internet Technologies and Systems*, page 10, 2003.

[Nar12] Rammohan Narendula. The case of decentralized online social networks. Technical report, EPFL, 2012.

[NJM+12] Shirin Nilizadeh, Sonia Jahid, Prateek Mittal, Nikita Borisov, and Apu Kapadia. Cachet: a decentralized architecture for privacy preserving social networking with caching. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 337–348. ACM, 2012.

[NPA10] Rammohan Narendula, Thanasis G Papaioannou, and Karl Aberer. Privacy-aware and highly-available osn profiles. In *Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on*, pages 211–216. IEEE, 2010.

[NPA12] Rammohan Narendula, Thanasis G Papaioannou, and Karl Aberer. A decentralized online social network with efficient user-driven replication. In *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing*, pages 166–175. IEEE, 2012.

[NTW06] Nikos Ntarmos, Peter Triantafillou, and Gerhard Weikum. Counting at large: Efficient cardinality estimation in internet-scale data networks. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 40–40. IEEE, 2006.

[oCD09] Privacy Commissioner of Canada and Elizabeth Denham. *Report of findings into the complaint filed by the Canadian internet policy and public interest clinic (CIPPIC) against facebook inc. Under the personal information protection and electronic documents act*. Office of the Privacy Commissioner of Canada, 2009.

[Res99] Eric Rescorla. Rfc2631: Diffie-hellman key agreement method, 1999.

[RG] I. Zhitomirskiy R.S.D. GRippi, M. Salzberg. Diaspora. `https://joindiaspora.com/`.

[RJM15] Fatemeh Raji, Mohammad Davarpanah Jazi, and Ali Miri. Pesca: a peer-to-peer social network architecture with privacy-enabled social communication and data availability. *IET Information Security*, 9(1):73–80, 2015.

[RMJ13] Fatemeh Raji, Ali Miri, and Mohammad Davarpanah Jazi. Cp2: Cryptographic privacy protection framework for online social networks. *Computers & Electrical Engineering*, 39(7):2282–2298, 2013.

[RMJ14] Fatemeh Raji, Ali Miri, and Mohammad Davarpanah Jazi. A centralized privacy-preserving framework for online social networks. *The ISC International Journal of Information Security*, 6(1):35–52, 2014.

[RMJM11] Fatemeh Raji, Ali Miri, Mohammad Davarpanah Jazi, and Behzad Malek. Online social network with flexible and dynamic privacy policies. In *Computer Science and Software Engineering (CSSE), 2011 CSI International Symposium on*, pages 135–142. IEEE, 2011.

[SLC+11] Amre Shakimov, Harold Lim, Ramón Cáceres, Landon P Cox, Kevin Li, Dongtao Liu, and Alexander Varshavsky. Vis-a-vis: Privacy-preserving online social networking via virtual individual servers. In *2011 Third International Conference on Communication Systems and Networks (COMSNETS 2011)*, pages 1–10. IEEE, 2011.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[Str09] Thorsten Strufe. Safebook: A privacy-preserving online social network leveraging on real-life trust. *IEEE Communications Magazine*, page 95, 2009.

[Swe02] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[SWY75] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[SZF10] Jinyuan Sun, Xiaoyan Zhu, and Yuguang Fang. A privacy-preserving scheme for online social networks with efficient revocation. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[Tea20a] Inspire Team. Inspire privacy policy. `https://www.inspire.com/about/privacy/`, October 2020.

[Tea20b] PatientsLikeMe Team. Patientslikeme privacy policy. `https://www.patientslikeme.com/users/sign_up`, October 2020.

[Tea20c] Sermo Team. Sermo privacy policy. `http://www.sermo.com/what-is-sermo/privacy-policy`, August 2020.

[TGS+08] Amin Tootoonchian, Kiran Kumar Gollu, Stefan Saroiu, Yashar Ganjali, and Alec Wolman. Lockr: social access control for web 2.0. In *Proceedings of the first workshop on Online social networks*, pages 43–48. ACM, 2008.

[TVS07] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[WB90] Samuel D Warren and Louis D Brandeis. The right to privacy. *Harvard law review*, pages 193–220, 1890.

[WCZJ04] Wenjie Wang, Hyunseok Chang, Amgad Zeitoun, and Sugih Jamin. Characterizing guarded hosts in peer-to-peer file sharing systems. In *Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE*, volume 3, pages 1539–1543. IEEE, 2004.

[Wes68] Alan F Westin. Privacy and freedom. *Washington and Lee Law Review*, 25(1):166, 1968.

[WSW+11] Christo Wilson, Troy Steinbauer, Gang Wang, Alessandra Sala, Haitao Zheng, and Ben Y Zhao. Privacy, availability and economics in the

polaris mobile social network. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, pages 42–47. ACM, 2011.

[WWJ10] James B Williams and Jens H Weber-Jahnke. Social networks for health care: addressing regulatory gaps with privacy-by-design. In *Privacy Security and Trust (PST), 2010 Eighth Annual International Conference on*, pages 134–143. IEEE, 2010.

[Zim95] Philip R Zimmermann. *The official PGP user's guide*. MIT press, 1995.

Chapter 5

Challenges in designing a distributed cryptographic file system

(This chapter is based on a paper published in Twenty-seventh International Workshop on Security Protocols, Trinity College, Cambridge, UK, April 2019 [BJA19])

Private information about individuals is, today, stored in centralized repositories that must be trusted fully to perform access control faithfully. Alternative approaches have been proposed that distribute data in peer-to-peer networks, but they lack the availability required by real-world systems that process personal information. Online social networks, censorship resistance systems, document redaction systems and health care information systems have apparently-disparate requirements for confidentiality, integrity and availability. However, we believe that these problems are tractable if re-cast as filesystems problems, with a research goal of developing filesystems that incorporate both centralized and distributed components without sacrificing user privacy.

The four use cases we have identified have differing, even contradictory, requirements (Section 5.1). For example, the requirement for centralized auditing of access to health care data is in direct opposition to the needs of a censorship-resistant online social network. However, we believe that there is a set of filesystems and cryptographic techniques that can be employed together, in various combinations,

to meet each use case's requirements individually.

Techniques from modern copy-on-write filesystems can be combined with cryptographic capabilities, convergent encryption and distributed systems concepts in order to implement filesystem primitives with untrusted storage at global scale (Section 5.2). Untrusted block stores may be centralized for high performance or distributed for availability in the face of a censor, with local storage acting as either a cache or a seed as appropriate. Subsets of both files and directory trees can be selectively shared among users and applications, with mutability controlled by application-specific policy enforced only on user- or organization-controlled systems. Separating the control plane of policy enforcement from the data plane of bulk storage yields a hybrid filesystem that can be applied to centralized or distributed use cases.

We have begun to build a prototype of such a hybrid filesystem: *UPSS: the user-centric private sharing system*. This filesystem can be accessed as a traditional Unix filesystem or — perhaps more compellingly — incorporated directly into applications as a library. The UPSS API allows applications to interact with the system without knowledge of underlying structures such as the storage medium, and to provide more sophisticated sharing protocols than can be supported by the traditional POSIX filesystem API. Combining the research in both filesystems and security protocols, the UPSS project strives to enable systems with rich collaboration *and* strong user control in contexts as disparate as health care and censorship-resistant social networks.

5.1 Motivation: use cases

The requirements for our privacy-preserving distributed filesystem stem from four use cases that are not well-served by the state of the art:

- online social networking with untrusted service providers,

- censorship-resistant social networking with network partitioning,
- corporate file sharing with redaction integrity and
- health care data sharing with privacy and audit requirements.

A summary of the four use cases' requirements can be found in Table 5.1.

Table 5.1: Requirements derived from OSN (S), censorship-resistant network (C), redaction integrity (R) and health care (H) use cases

Requirement	S	C	R	H
High availability (connected network)	✓		✓	✓
Availability in network partition		✓		
Untrusted storage	✓	✓		✓
Partial/subset sharing	✓	✓	✓	✓
Scalable to large user base	✓		✓	✓
Sharing reciprocity ("merge requests")	✓	✓	✓	✓
Peer-to-peer storage and caching		✓		
Access auditing				✓

5.1.1 Online social network

General-purpose online social networks require high availability, frictionless content sharing and high overall standards for ease of use. Users — and the applications they employ — interact with shared content; any social application platform must be able to provide this access while still allowing for user control over that sharing. In addition to the above, however, it is desirable for any design incorporating centralized elements to *not trust the provider*. That is, although a central store of data may be required for availability and performance reasons, that provider need not have access to the plaintext of users' data.

As we describe in Section 5.2.1, it is possible to share immutable directory trees among users of a centralized data store while maintaining strong confidentiality, integrity and availability properties. This can be accomplished with well-understood cryptographic and filesystem techniques. The challenge of building a practical OSN from these raw materials is in the handling of shared *mutable* content. It must be possible to describe mutation in terms of operations that can be checked for consistency when performed by multiple authorized users. For example, updating a tree of shared content can be expressed as a "merge request" that replaces one immutable tree with another one, as long as the new tree references the original as its predecessor.

Thus, we make the following observations about the requirements for a privacy-preserving distributed filesystem being used as the basis for an OSN:

1. The requirement for high availability rules out techniques that base their security and functionality guarantees on the use of peer-to-peer networks — some centralized storage may be required to achieve better performance.

2. Centralized providers must only have access to encrypted data, ideally without information about metadata such as file sizes.

3. Users should be able to easily delegate access to shared content, both to applications and to other users.

4. The requirement for massively scalable performance rules out techniques that impose heavy computational or network burdens on servers, such as secure multi-party computation or private information retrieval.

5. It should be possible to manage mutable directory trees with operations that can be checked for consistency, e.g., "merge requests".

5.1.2 Censorship-resistant social networking

Reliable and widely adopted information systems need to be resilient against any censors by authorities such as governments. Decentralization of server modules could be an effective approach for this issue, as any central point is a kind of weakness for the system. Inevitably, most systems contain centralized management and decision making modules, for example, user authorization on the cloud storage should be done centrally. Peer-to-peer connections are a solution to reduce the risk of being blocked. Along with this idea, caching techniques are another effective ways to improve system's high availability. Thus, to provide a reliable and safe censorship-resistant information system, we can accommodate the following fundamentals about the system's underlying filesystem to support the stated requirements:

1. The requirement for decentralization leads to a different sharing protocol on the filesystem that lets users to have peer-to-peer data sharing. Necessarily, we need a backup storage for those cases in which one or more principals are offline. The sharing approach should handle backups as well. The connection protocol should work for different users behind NATs (Network Address Translators). Similar inspiring protocols have been introduced before, such as Session Traversal Utilities for NAT (STUN) [RMMW08], for finding users' public IP addresses and ports. Figure 5.1 shows this situation, in which the connection between the users and the cloud storage account is blocked and users are able to interact with each other in a peer-to-peer environment.

2. Along with peer-to-peer connections, availability in a network partition can be supported by different caching techniques which support various periods of time for cached data. The filesystem can handle data caching based on users' share requests. In this way, having permanent cached data is also possible. Expired cached data could be removed in different ways such as having a

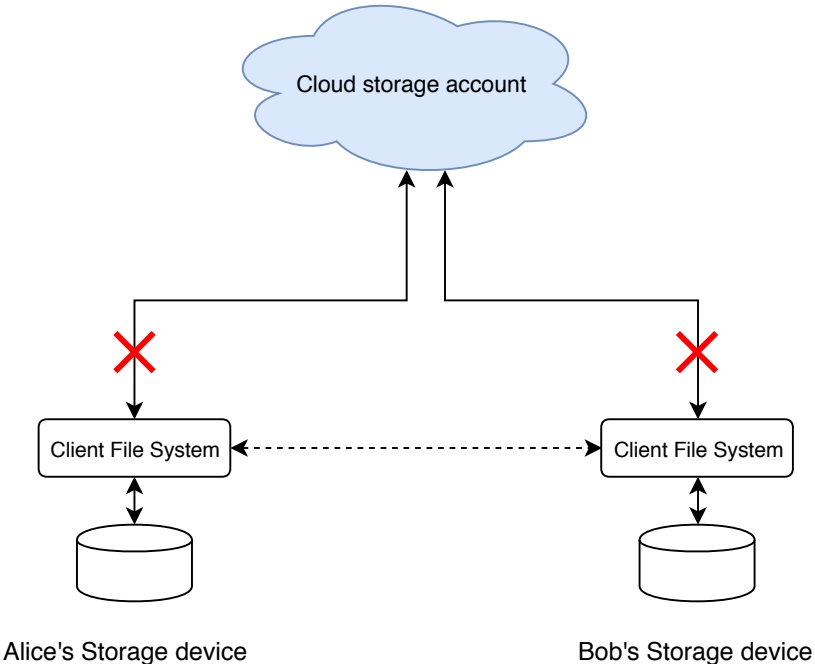


Figure 5.1: Applied censorship to the central point of the system

garbage collector that runs periodically.

5.1.3 Redaction with integrity

A common requirement for large organizations — both commercial and governmental — is the ability to release documents and information selectively in response to information requests and legal discovery. Current approaches to such information sharing involve the redaction of documents and a one-way release of information. Linking redacted documents to their original versions is a manual process that provides little technical assurance of integrity. A chain of custody for such information may be asserted by the releasing organization but may not, using current techniques, be verified by the receiving party.

It is desirable to be able to release portions of documents in a manner that provides strong integrity verification and linking to the original document. Sharing part of a file or a directory hierarchy should be efficient and should allow linking to

commitments for unredacted versions without revealing any redacted information. In addition, it may be desirable to provide a mechanism for changes to released content to be shared back to the original, unredacted document, enabling new communication patterns and — potentially — better strategies for internal information compartmentalization. We can thus observe that:

1. Directory hierarchies and even files should be sharable *in part*, as subsets of their unredacted originals.

2. Shared subsets should be linkable to original, unredacted documents with strong integrity.

3. Changes to redacted documents should be re-sharable back to redacting parties in a way that permits two-way collaboration over partial views.

5.1.4 Health care data sharing

Health care information systems have several requirements in common with OSNs such as high availability and data sharing within the network. One additional requirement, however, is the need for auditing of accesses to patient data. This acts as a disincentive for health care workers to access the private information of patients they are not caring for. However, in the context of a privacy-preserving system, the requirement for audit records should not cause arbitrary patient data to be exposed to the network security team.

One barrier to innovation in the health care context is the dichotomy between "trusted" and "untrusted" systems and the enormous effort required to certify a system as "trusted". A filesystem that afforded the ability to securely share strict subsets of patient data could enable new innovations. Applications running on such a platform could be executed with lower stakes, as the impact of an application accidentally leaking data without context, e.g., image data with no patient identifiers attached, would be less than if the application implicitly had access to complete

patient records. This is anologous on the partial data sharing requirement described in Section 5.1.3, but in a very different environment.

We thus make the following observations about the challenges of building health information systems atop a privacy-preserving distributed filesystem:

1. Confidentiality must be maintained from both system administrators and health care workers who do not require patients' information in the course of providing care.

2. Access to patient data should be auditable without revealing patient information to auditors.

3. It should be possible to provide new applications with access to data subsets without implying access to complete patient records.

5.2 UPSS: the user-centric private sharing system

To design a system which can provide a fundamental basis for the majority of the requirements discussed in Section 5.1, we use some key ideas of existing systems, mainly discussed in Section 5.3, to design our system as a new filesystem associated with efficient sharing mechanism. By introducing UPSS, we try to meet confidentiality, high availability, data integrity, and an efficient sharing mechanism, all integrated into a system to serve a wide range of applications.

To support confidentiality, we use cryptographic techniques to provide an end-to-end sharing system, which stores user data in a secure way on the storage, with user-controlled privacy. To provide a high level of availability, users' storages or cloud storage accounts can be used as temporary or even permanent caches to maintain other users' data online. Consequently, this data replication leads to data inconsistency problem which is reduced to a version control problem by storing user data in immutable objects in our system. UPSS suggests a content-addressing

mechanism for data blocks on storage through cryptographic hashes obtained from blocks' content and their physical locations on storage.

Our system is constructed out of four main layers, which are depicted in Figure 5.2. In this section, we discuss UPSS, a User-centric Private Sharing System in more detail and explain how UPSS can meet the discussed requirements.

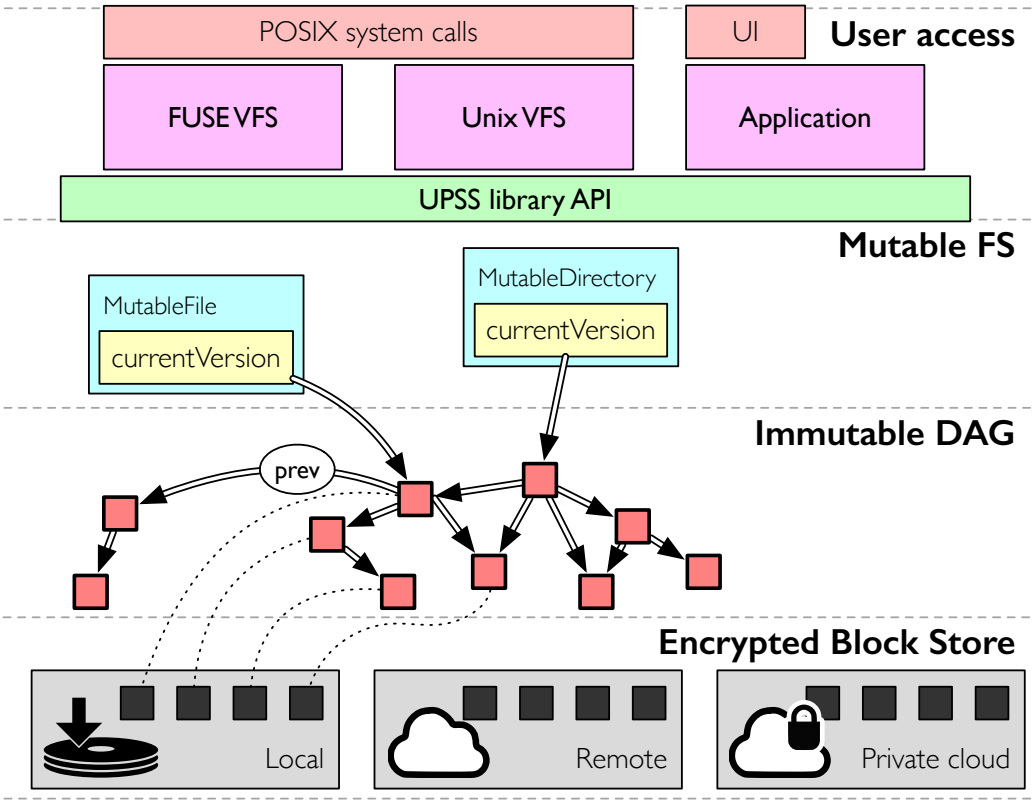


Figure 5.2: Layers in the UPSS prototype. Encrypted block store stores ciphertext blocks, which are generated from plaintext blocks in immutable Merkle DAGs (thick arrows represent block pointers). A mutable filesystem is exposed as a library API, which can be consumed directly by applications or adapted to a conventional filesystem API.

5.2.1 Immutable DAGs

Data is stored in UPSS as a set of fixed-size immutable encrypted blocks. These blocks, which are encrypted and named according to a hash of their ciphertext, are linked together in a Merkle DAG (Directed Acyclic Graph). Merkle DAGs are used to describe files, directories and versions of both. Immutable blocks are encrypted using symmetric keys derived from cryptographic hashes of their content, a technique known as *convergent encryption* [DAB+02], and named using the cryptographic hash of their ciphertext, a technique known as *content-addressable storage*. The name of a block and the key that can be used to access it are referred to as a *block pointer* $BP_B = (n_B, k_B)$, given by:

$$k_B = h(B)$$
$$n_B = h\left(E_{k_B}\{B\}\right)$$

where B is the plaintext block, E is a deterministic symmetric encryption, n_B is the name of a block and k_B is the key used to decrypt it. A block pointer can thus be seen as a cryptographic *capability* [DVH66] to read a block, though not necessarily to modify it (see Section 5.2.3). The block pointer to the root node of a file or directory implies the ability to access arbitrary quantities of content, up to an entire filesystem.

Convergent encryption does not provide semantic security as it is vulnerable to content-guessing attacks [BDPR98]. However, the de-duplication effect of the convergent encryption makes it suitable for storage efficiency. For making a trade-off between the security and the storage efficiency, Stanek et al. [SSAK14] introduced a convergent threshold cryptosystem in which popular files — identical files that are uploaded to the storage server by many users — are encrypted using convergent encryption, while unpopular files are protected using another symmetric encryption

around the inner convergent encryption. However, for enabling a the storage server to identify the popular and unpopular files, two central server components need to be fully trusted by the users, which is the opposite of our target. Currently, we are not using the convergent threshold encryption. However, we are embedding random padding inside the blocks with content size lower than the UPSS's block size. We can choose the deterministic and non-deterministic random padding, where in the former case the de-duplication is enabled.

By defining blocks to be immutable, we reduce the data inconsistency problem to a version control problem. A file modification causes a new version of the file to be created and this modification affects the whole path up to the parent blocks until the root block. We borrowed this feature from the Copy-on-Write (CoW) file systems. The modifications are done with the block size granularity, means that even if one bit in a block is modified, a new block is generated and the CoW updates are applied. Version controls are met by keeping a pointer to the previous version of the modified files in their corresponding root blocks.

The symmetric key of each block is stored inside its preceding block. In the same way, the symmetric key of the root block in a sub-tree is stored in the blocks of the parent directory. In this way, we avoid any central server to keep the chain of our decryption keys. Moreover, we avoid duplication of same blocks generated by different users as the blocks would eventually have the same ciphertext.

5.2.2 Mutable Filesystem API

All the details about the immutable DAGs and the underlying storage model are hidden from the top level applications. UPSS provides an object view of the underlying encrypted blocks for the applications and enables them to interact with the system through the provided API. Each file and directory in mutable filesystem layer is interpreted as an object called `FSObject`. The `FSObjects` are in-memory objects constituting the mutable DAGs for our system. In this layer, the following

metadata attributes are defined and used for each file or directory: access time, creation time and the modification time.

Applications interact with UPSS using `FSObject` references. In case of any modification on the content of a file or directory, the inner state of the corresponding `FSObject` is changed and the modifications is applied to the underlying layers to reflect the file/directory's modification. However, the object reference remains the same. This process is done in background and is transparent from the top level applications. The results can be returned by callback functions provided by the API's methods. This approach can also make a non-blocking modification process from the applications' viewpoint running on the upper level.

The `FSObjects` enables us to have structured files, which can support some functionalities not supported by the classical Unix filesystems that interpret the files as unstructured byte arrays. One of the functionalities supported by UPSS is guaranteeing the data consistency of the shared files between users. To do so, the system should define a consistency model, suitable for distributed systems. One applicable solution is to define the file structures as a Conflict-free Replicated Data Type (CRDT) [SPBZ11b, SPBZ11a, KB17] to guarantee that shared files on different replicas converge, by defining a Strong Eventual Consistency (SEC) model, which leverages mathematical properties such as monotonicity in a semi-lattice and/or commutativity, which ensure the absence of conflict. An add-only DAG is an example of a CRDT data type [SPBZ11a] suitable for a distributed filesystem with file sharing and redacting integrity.

5.2.3 Share Control Module

Providing a mechanism to share data, either on a multiuser system or over the network in distributed systems, is a crucial feature for various information systems. According to the requirements described in Section 5.1, a sharing module on top of our filesystem is responsible to support different scenarios for both centralized and

peer-to-peer data sharing. To have a very flexible, and also extensible design for the future, we considered the sharing module as another application above the UPSS API library, shown in Figure 5.3, along with other applications such as Unix VFS or FUSE VFS, etc. Thus, the subsequent integrated system potentially supports a wide range of required use cases for information systems.

As a primary prototype, this module is divided into two principal sub-modules that handle share requests and data transmissions separately. The share control module is an application, mainly responsible to manage share requests, in terms of the sharing protocol, version control, users' privileges and access controls. This module receives or sends share requests. Once the request is processed and the user and privileges are authorized, the share control module makes related modification on shared data through the UPSS API.

On the other hand, data transmission is accomplished using the sharing block store which communicates with the local block store or the local caches of the system and other remote block stores. Figure 5.3 represents this module. These features distinguish UPSS from the other systems described in Section 5.3.

As another important issue that rises about data sharing, this module handles users' access controls. When we talk about data sharing between users and their privacy, accessibility appears as the other important issue. In UPSS, the concepts of permissions and privileges are raised in different layers and vary based on the scope in which the user is defined. For example, for local users in a multiuser system, a file can be shared with traditional POSIX permissions such as "read", "write" and "execute". However, for non-local users in the network involved with share and merge requests, we need to define additional policies and definitions to provide a user-centered privacy scheme through access controls. For example, any user can share data with others, but nobody obtains or modifies data without the right privileges. Data modification would be done after user authorization, which can be achieved using cryptographic techniques such as public keys and certificates.

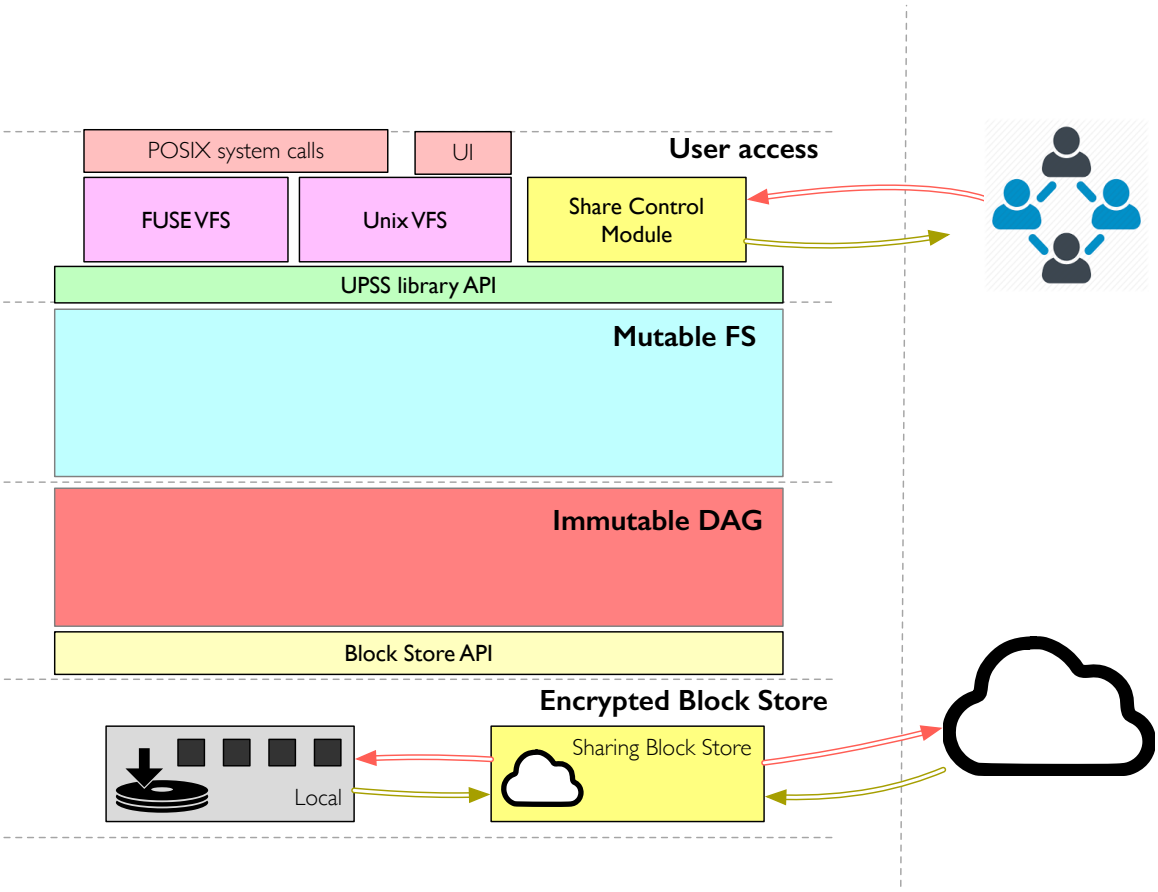


Figure 5.3: Sharing module includes two sub-modules, share control module and sharing remote block store, to manage share procedure and data transmission control

For remote users, modifying data could be followed with merge or pull requests. The sharing protocol considers users' access rights before the main procedure of the request is accomplished. The rest of the procedure would be done in the mutable layer, as it is implicitly shown in Section 5.2.2. The corresponding mutable block would be added to the mutable Merkle DAG associated with the block pointer to its ciphertext block. Here, the application of block pointers is identical to capabilities [DVH66], as unforgeable references to ciphertext blocks of our system. Therefore, we can see different security approaches in UPSS to guarantee user privacy and data confidentiality.

5.2.4 VFS layer(s)

Classical filesystem abstractions — files, directories and directory entries — represent a subset of the abstractions that can be represented in UPSS, but they are important abstractions. These can be exposed to applications and users using existing virtual filesystem (VFS) layers provided in userspace (FUSE) or kernels (Unix VFS). Higher-level applications may prefer to interact directly with the UPSS API, but existing applications can work with UPSS without modification via an existing VFS.

Requests from VFS layers can be addressed with inode numbers which are the low-level files or directories identifiers. Mapping the low-level names to UPSS entities enables the VFS and FUSE APIs to interact with UPSS. Besides the mappings from low-level names to object references, VFS layers also provide metadata that is only meaningful for the local system and its users, such as permissions for local users.

5.3 Related Work

As a primary target of our study, we investigated existing privacy-preserving approaches mainly in online social networks. First, we started with studying different systems and modules regardless of their type, scope and the system level in which they are integrated and employed, to find their key ideas, cons, and pros. The only common feature between all of them is their effort to place additional user privacy, especially against OSN server providers. As we expanded our target to have a secure and privacy-preserving filesystem, we continued with an investigation on several filesystems, focusing on their provided security features. We summarize these studies starting from OSN tools to filesystems.

5.3.1 Online Social Network Systems and Tool

Privacy on social networks can be discussed in different ways. A well-known challenge is protecting users' privacy against each other. However protecting users' privacy against OSN servers and providers is also an important issue. Therefore, attempts to improve privacy in centralized OSN systems appeared. Because of the centralized architecture of most adopted OSNs, such as Facebook and Google Drive, primary proposed approaches tended to be centralized in design. Lockr [TGS+08], FlyByNight [LB08], NOYB [GTF08], Scramble [BKW11] and CP2 [RMJ13], all can be counted as important and effective tools in this category. In addition to centralization, we can categorize privacy countermeasures and approaches in other ways. Using cryptographic techniques or providing user-controlled privacy are other effective features.

Lockr [TGS+08] and Scramble [BKW11] are browser plug-ins that restrict access through user-defined access control lists. Although this feature makes a flexible privacy scheme, it still suffers from storing plaintext data on OSN servers, including user data and users relationships. FlybyNight [LB08] and CP2 (short for "Cryptographic privacy protection") [RMJ13] protect user data by some encryption mechanism with this difference that CP2 can also protect the relationships between users by determining the users with some unique pseudonyms. In NOYB [GTF08], users' profiles are partitioned into smaller clusters called *atoms*, and the atoms of one user are substituted with atoms of another user in the same cluster pseudo-randomly, and then the encrypted index of each atom is stored in a dictionary. The authors of FaceCloak [LXH09], introduced a mechanism in which the users' private data is stored encrypted in a user-trusted third-party server, while some other fake data related to encrypted data are stored in OSN servers in plain-text format. One important point about using data encryption in communication with OSNs is that these systems and their providers should allow encrypted messages and data to be stored or transmitted. Moreover, data encryption should not affect OSNs' main

functionalities such as search use cases.

Another beneficial feature toward real privacy for users is user-controlled or user-centered privacy. For example, Lockr, FaceCloak, and Scramble, allow users to define and control their preferred privacy mainly through access control lists.

As it is stated above, the centralized nature of OSNs in which all users' data is accessible by a single entity, i.e., OSN provider, encourages the researchers to change their mind about how to preserve users' data, which leads to a shift from client-server to a decentralized architecture coupled with encryption so that the users can protect their data. Diaspora [BHG+12] is an example of such architecture for OSNs.

In decentralized P2P systems, connectivity and high availability are other issues that bring several other challenges to be discussed, along with other existing problems such as privacy. PeerSoN (short for "P2P Social Networking") [BSVD09] , Safebook [Str09] and Porkut [NPA10] are instances of decentralized OSNs that have tried to come up with solving some parts of these challenges. PeerSoN [BSVD09] tried to overcome connectivity and privacy limitations. The main properties of PeerSoN are encryption, decentralization, and direct data exchange. Safebook [Str09] is another system that tried to protect users, from potential privacy violations, against providers. It relies on the concept "peers", which points to the cooperation between users on the social network. Although Safebook has presented a decentralized structure, it still relies on central servers that keep pseudonyms of interacting nodes in the network. Also, there are criticisms of its privacy scheme. Porkut [NPA10], as another example in this category, focuses mainly on data availability by replicating users' data on trusted friends' storages. Also, a privacy preserving indexing mechanism is introduced which facilitates content discovery among friends. The indexes are stored on a Distributed Hash Table (DHT) [SMK+01] in the form of $(key, value)$ pairs. As another different example, we can consider Cachet [NJM+12] which is an improvement of DECENT [JNM+12]. Cachet proposed a decentralized architecture

to be used in OSNs, which protects confidentiality by an attribute-based encryption. It also provides integrity and availability by digital signature and gossip-based social caching algorithm, respectively.

However, all the discussed decentralized approaches rely on distributed data structures such as DHTs to enable the users to interact with each other and discover other users and resources. The nodes which construct the DHT network should be trusted to keep critical information about the users and the network topology.

To summarize, in both centralized and decentralized architecture of discussed OSNs, we can still see unaddressed privacy, availability, and connectivity problems. Despite all efforts done to overcome these restrictive issues, proposed approaches seem superficial rather than beneficial, for our requirements. This matter leads us to think about lower layers of the system, where we are involved with the filesystem and its communication with applications in higher layers. If we can provide confidentiality, data integrity, and availability at the same level as the filesystem, higher level applications can benefit from these properties even while interacting with the filesystem using standard POSIX APIs.

5.3.2 File systems

In most traditional filesystems, data integrity and availability is preferred over confidentiality and privacy. For several years, the concept of privacy was something beyond filesystems functionalities, and data writing and retrieval throughput was the most important feature. As an example, (ZFS) [BAH+03] is placed in the group of efficient widely adopted filesystems through Copy-on-Write (CoW) techniques. Thus, distributed filesystems were introduced with the main target of providing high availability along with the former feature. Examples of such filesystems are Coda [SKK+90], Ivy [MMGC02] and Ori [MBHM13].

Coda [SKK+90] is one of the inspiring distributed filesystems with the idea of shared data repositories. It retrieves data and resolves conflicts using the concept

called accessible volume storage group (AVSG) as data replicas. Similar to Coda, Ivy [MMGC02], as a multi-user peer-to-peer filesystem, has focused on data availability with a different approach which relies on private snapshots from filesystem for each participant. Ivy stores logs from the state of the filesystem in a distributed hash table, called DHash [DKK+01]. A log constituted of the dedicated private snapshot for each user, contains all user's modifications to filesystem data and meta-data. Thus, we can find signs of user privacy in Ivy's approach, although confidentiality has not been stated as its main feature. We can find these features collected in Ori [MBHM13], plus its own data sharing mechanism, grafting, across user multiple devices. Synchronization, failures handling and data recovery, are expanded and emphasized in Ori more than two previous stated filesystems. Over time, filesystems and other sharing stores expanded their functionalities, such as content-addressing, based on new requirements in the community. As a modern filesystem, IPFS [Ben14] synthesizes the key successful ideas behind systems such as DHTs [SMK+01], BitTorrent [Coh03], Git [LM12], and SFS [MK98]. Moreover, IPFS deals with encrypted mutable objects to improve confidentiality.

Like UPSS, Tahoe [WOW08] is a cryptographic filesystem that stores the content encrypted in Merkle DAGs and provides access control by cryptographic capabilities. In Tahoe's design, both mutability and immutability is supported, which the later case may cause data inconsistency in collaborative environments. The files are encrypted with one symmetric key and they are erasure coded using Reed-Solomon codes [Riz97] into N shares to be written to N servers. However, Tahoe is designed for file sharing and archival storage; using Tahoe with POSIX-like read-write workloads can cause "its performance to crawl to a halt" [tah20]. Having the files encrypted with one key cannot provide the partial sharing/subsetting.

Having been inspired by discussed filesystems, we strongly believe that UPSS, as a cryptographic content-addressable filesystem can serve typical filesystem requirements associated with many of the modern requirements that are explained

in Section 5.1. UPSS stores data in content-addressed fixed-length blocks and controls block accessibility through its sharing module that supports both peer-to-peer and client-server connections. The UPSS's APIs, discussed in the Section 5.2, provide a higher level of abstraction about the underlying storage model and enable a variety of applications to use the system without any assumption about the physical storage.

5.4 Conclusion

Distrustful information sharing is a common problem that is not well-addressed by the existing state of the art. In online social networks, sharing *any* information with friends requires sharing *all* information with a potentially-untrustworthy provider. Countermeasures to the all-seeing provider are brittle, ineffectual or else perform too poorly for general consumption. In the general social networking case, users lose control of how widely their data is shared; the stakes are even higher in censorship resistance scenarios. In environments with strong confidentiality properties, conversely, a lack of secure sharing techniques stifles collaboration, transparency and innovation. This is seen when organizations apply redaction with no linkability to original data or when health-care authorities silo patient information off from potentially-innovative applications.

One linkage among all four of these use cases is the need to share information *selectively* and *securely* among parties without requiring complete trust. Any technique for enabling such sharing must not be strictly one-way: it must provide the possibility of reciprocation and collaboration. We have argued that all of these goals can be met by recasting the above problems in terms of a privacy-preserving filesystem. By decoupling storage from access control, a distributed user-centred filesystem can provide confidentiality, integrity and availability properties to support systems in all four of these use cases.

Using encrypted fixed-length blocks in a content-addressable store, information can be stored with untrusted providers, cached locally and/or distributed opportunistically via contact or peer-to-peer networks. Using convergent encryption and Merkle DAGs, file and directory structures can be stored as immutable DAGs in a manner that both preserves privacy and enables global deduplication. Higher-level mutable filesystem objects can be maintained using higher-level sharing protocols with application-specifiable authorization schemes. Finally, this filesystem can be exposed to users via direct embedding within applications, via local Web frontends or as a traditional filesystem within FUSE or a Unix VFS layer.

We are exploring these ideas in our prototype filesystem, *UPSS: the user-centric private sharing system*. We believe that the availability of such a filesystem will enable the development of applications and platforms that provide both strong user privacy and rich collaborative sharing. Designing such systems may yet demonstrate that sharing and security can go hand in hand.

Bibliography

[BAH+03] Jeff Bonwick, Matt Ahrens, Val Henson, Mark Maybee, and Mark Shellenbaum. The zettabyte file system. In *Proc. of the 2nd Usenix Conference on File and Storage Technologies*, volume 215, 2003.

[BDPR98] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In *Annual International Cryptology Conference*, pages 26–45. Springer, 1998.

[Ben14] Juan Benet. IPFS-content addressed, versioned, P2P file system. *arXiv preprint arXiv:1407.3561*, 2014.

[BHG+12] Ames Bielenberg, Lara Helm, Anthony Gentilucci, Dan Stefanescu,

and Honggang Zhang. The growth of diaspora-a decentralized online social network in the wild. In *2012 Proceedings IEEE INFOCOM Workshops*, pages 13–18. IEEE, 2012.

[BJA19] Arastoo Bozorgi, Mahya Soleimani Jadidi, and Jonathan Anderson. Challenges in designing a distributed cryptographic file system. In *Cambridge International Workshop on Security Protocols*, pages 177–192. Springer, 2019.

[BKW11] Filipe Beato, Markulf Kohlweiss, and Karel Wouters. Scramble! your social network data. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 211–225. Springer, 2011.

[BSVD09] Sonja Buchegger, Doris Schiöberg, Le-Hung Vu, and Anwitaman Datta. PeerSoN: P2P social networking: early experiences and insights. In *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, pages 46–52. ACM, 2009.

[Coh03] Bram Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72, 2003.

[DAB+02] John R Douceur, Atul Adya, William J Bolosky, P Simon, and Marvin Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Proceedings 22nd international conference on distributed computing systems*, pages 617–624. IEEE, 2002.

[DKK+01] Frank Dabek, M Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 202–215. ACM, 2001.

[DVH66] J. B. Dennis and E. C. Van Horn. Programming semantics for multiprogrammed computations. *Communications of the ACM*, 9(3):143–155, 1966.

[GTF08] Saikat Guha, Kevin Tang, and Paul Francis. NOYB: Privacy in online social networks. In *Proceedings of the first workshop on Online social networks*, pages 49–54. ACM, 2008.

[JNM+12] Sonia Jahid, Shirin Nilizadeh, Prateek Mittal, Nikita Borisov, and Apu Kapadia. Decent: A decentralized architecture for enforcing privacy in online social networks. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, pages 326–332. IEEE, 2012.

[KB17] Martin Kleppmann and Alastair R Beresford. A conflict-free replicated JSON datatype. *IEEE Transactions on Parallel and Distributed Systems*, 28(10):2733–2746, 2017.

[LB08] Matthew M Lucas and Nikita Borisov. FlyByNight: mitigating the privacy risks of social networking. In *Proceedings of the 7th ACM workshop on Privacy in the electronic society*, pages 1–8. ACM, 2008.

[LM12] Jon Loeliger and Matthew McCullough. *Version Control with Git: Powerful tools and techniques for collaborative software development*. " O'Reilly Media, Inc.", 2012.

[LXH09] Wanying Luo, Qi Xie, and Urs Hengartner. FaceCloak: An architecture for user privacy on social networking sites. In *Computational Science and Engineering, 2009. CSE'09. International Conference on*, volume 3, pages 26–33. IEEE, 2009.

[MBHM13] Ali José Mashtizadeh, Andrea Bittau, Yifeng Frank Huang, and David Mazieres. Replication, history, and grafting in the Ori file system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 151–166. ACM, 2013.

[MK98] David Mazieres and M Frans Kaashoek. Escaping the evils of centralized control with self-certifying pathnames. In *Proceedings of the 8th ACM SIGOPS European workshop on Support for composing distributed applications*, pages 118–125. ACM, 1998.

[MMGC02] Athicha Muthitacharoen, Robert Morris, Thomer M Gil, and Benjie Chen. Ivy: A read/write peer-to-peer file system. *ACM SIGOPS Operating Systems Review*, 36(SI):31–44, 2002.

[NJM+12] Shirin Nilizadeh, Sonia Jahid, Prateek Mittal, Nikita Borisov, and Apu Kapadia. Cachet: a decentralized architecture for privacy preserving social networking with caching. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 337–348. ACM, 2012.

[NPA10] Rammohan Narendula, Thanasis G Papaioannou, and Karl Aberer. Privacy-aware and highly-available OSN profiles. In *Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on*, pages 211–216. IEEE, 2010.

[Riz97] Luigi Rizzo. On the feasibility of software fec. *Univ. di Pisa, Italy*, pages 1–16, 1997.

[RMJ13] Fatemeh Raji, Ali Miri, and Mohammad Davarpanah Jazi. CP2: Cryptographic privacy protection framework for online social networks. *Computers & Electrical Engineering*, 39(7):2282–2298, 2013.

[RMMW08] Jonathan Rosenberg, Rohan Mahy, Philip Matthews, and Dan Wing. Session traversal utilities for NAT (STUN). Technical report, 2008.

[SKK+90] Mahadev Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Steere. Coda: A highly avail-

able file system for a distributed workstation environment. *IEEE Transactions on computers*, 39(4):447–459, 1990.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[SPBZ11a] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. *A comprehensive study of convergent and commutative replicated data types*. PhD thesis, Inria–Centre Paris-Rocquencourt; INRIA, 2011.

[SPBZ11b] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *Symposium on Self-Stabilizing Systems*, pages 386–400. Springer, 2011.

[SSAK14] Jan Stanek, Alessandro Sorniotti, Elli Androulaki, and Lukas Kencl. A secure data deduplication scheme for cloud storage. In *International conference on financial cryptography and data security*, pages 99–118. Springer, 2014.

[Str09] Thorsten Strufe. Safebook: A privacy-preserving online social network leveraging on real-life trust. *IEEE Communications Magazine*, page 95, 2009.

[tah20] Tahoe frequently asked questions. `https://tahoe-lafs.org/trac/tahoe-lafs/wiki/FAQ`, 2020.

[TGS+08] Amin Tootoonchian, Kiran Kumar Gollu, Stefan Saroiu, Yashar Ganjali, and Alec Wolman. Lockr: social access control for web 2.0. In *Proceedings of the first workshop on Online social networks*, pages 43–48. ACM, 2008.

[WOW08] Zooko Wilcox-O'Hearn and Brian Warner. Tahoe: the least-authority filesystem. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*, pages 21–26, 2008.

Chapter 6

UPSS: the user-centric private sharing system

(This chapter is submitted as a paper to ACM Transaction on Storage (ACM TOS), June 2020)

Across a broad spectrum of use cases, there is an acute need for private storage *and* sharing, with strong security *and* performance properties. Existing systems provide security *or* performance, with strong protection *or* ease of user-directed sharing, but a failure to adequately address "both-and" requirements forces users to settle for systems that do not fully meet their needs. This is true in environments as diverse as social networking, electronic health records and surveillance data management.

Online social networking applications present users with a difficult choice between centralized systems with high levels of performance but weak privacy properties and distributed systems that attempt to provide stronger privacy properties but lack reliability and practical levels of performance. The desirable scenario for users is to provide them the ability to easily share arbitrary quantities of content with others while maintaining a *private-by-default* posture. In particular, it is desirable for a social networking system to ensure that the operators of network and storage infrastructure have as little visibility as possible into the social interactions of users.

Contemporary approaches to health records include thick perimeters and high

levels of trust in selected organizations and individuals. Such approaches introduce needless risk even in an age of small systems with known players, and they do not scale up to enable innovative health care ecosystems that improve outcomes without placing patient information at risk. It is desirable to be able to share minimized information — or indirect references to information — in such a way that access can be audited without revealing patient details even to auditors.

In surveillance and law enforcement, there is often a need to redact identifying or other operational details from surveillance imagery. While performing this redaction, however, there is also a need to maintain a demonstrably strong chain of custody from an original source image or video through to a redacted image presented in a legal proceeding. Confidentiality is important in this scenario, but so is the integrity of multiple versions of content and the linkages among them. In Section 6.5, we will discuss how UPSS can address this requirement.

In all of these cases, what is needed is a mechanism for *least-privileged* storage of information that facilitates *simple sharing* of arbitrary quantities of content at users' discretion. Such a system should provide strong confidentiality and integrity properties, such that it can rely on commodity cloud services from untrusted providers. We have built such a system in UPSS: the user-centric private sharing system, a cryptographic filesystem designed to be "global first", with no assumptions made about the trustworthiness of storage infrastructure (Section 6.1) or even on common agreed-on definitions of users or user identities. Relying on key concepts from capability systems [DVH66], distributed systems, log-structured filesystems (Section 6.1.1) and revision control, we have developed a new approach to filesystems (Section 6.2) that offers novel features while being usable in ways that are compatible with existing applications.

We demonstrate the utility of this new approach to privacy-oriented filesystems, conceptually described in [BJA19], via four case studies (Section 6.3): a comparison with conventional Unix filesystems, locally (Section 6.3.2), remotely (Section 6.3.3)

and globally (Section 6.3.4), and as the basis for a new model of private revision control (Section 6.3.5). Through all of these case studies, we demonstrate that UPSS provides a solid underpinning for new approaches to user-centric private data storage and sharing, enabling users to benefit from both strong security *and* high performance, with both private, least-privileged storage *and* simple user-directed sharing.

6.1 Background

In this section, we define some concepts that are needed for a better understanding of the remaining paper.

6.1.1 Preliminaries

In 1992, Rosenblum et al. designed a disk storage management technique called Log-structured filesystem (LFS) [RO92] to improve the write performance of the existing filesystems of the time, such as Fast File System (FFS) [MJLF84]. Log-structured filesystems buffer large writes into memory and persist them to the disk along with their metadata sequentially in big chunks called *segments*. In this way, the disk rotation latency related to random accesses is avoided as all writes are done sequentially. Therefore, the blocks related to a file are accessed sequentially.

Efforts on optimizing the procedure of writing on filesystems continued on further popular filesystems such as ZFS [BAH+03], which employs the idea of copy-on-write (COW), used in our filesystem, `upss-fuse`, as well. The main idea of COW is to have immutable data, reducing the risks associated with concurrent accesses to mutable state and enabling important techniques such as cryptographic checksumming and greater parallelism. More specifically, when a data block is needed to be copied from one address to another, a pointer to the source is created for the target instead of actual copying, and the block is marked as read-only. In

this way, all the read operations from the target are served by referring to the source address. Upon writes in source or target, the block is copied to a new address, and the pointers are updated. In this way, write procedures are postponed until they actually are needed, which leads to performance improvement in the COW filesystems.

Merkle DAG (Directed Acyclic Graph), which is a general version of a Merkle tree, is a suitable data structure for managing files in filesystems such as ZFS [BAH+03] and Btrfs [RBM13], or in version control systems such as Git. The nodes of the Merkle DAG contain the cryptographic hashes of its children's content, and they also can contain some data. The root node can be used to compare different Merkle DAGs. If the hash values of the root nodes of two Merkle DAGs would be equal, it means that the two Merkle DAGs are identical. These features have made Merkle DAGs useful to content-related procedures like data integrity checks on filesystems.

6.2 UPSS: the user-centric private sharing system

UPSS is a private sharing system, which can provide confidentiality, integrity, and availability properties. UPSS enables its users to share information *selectively* and *securely* without requiring complete trust. The system is built from the best practices of successful approaches such as version control systems, content-addressable storage, and convergent encryption.

UPSS is designed in a layered architecture, shown in Figure 6.1. The block store layer (Section 6.2.1) provides storage of immutable encrypted blocks. On top of that, we have the immutable layer, described in Section 6.2.2, in which the relation between the data items in various granularities is defined. Mutability is provided by the in-memory mutable objects in the mutable filesystem layer (Section 6.2.3). The public UPSS library API is the interaction point between applications and UPSS.

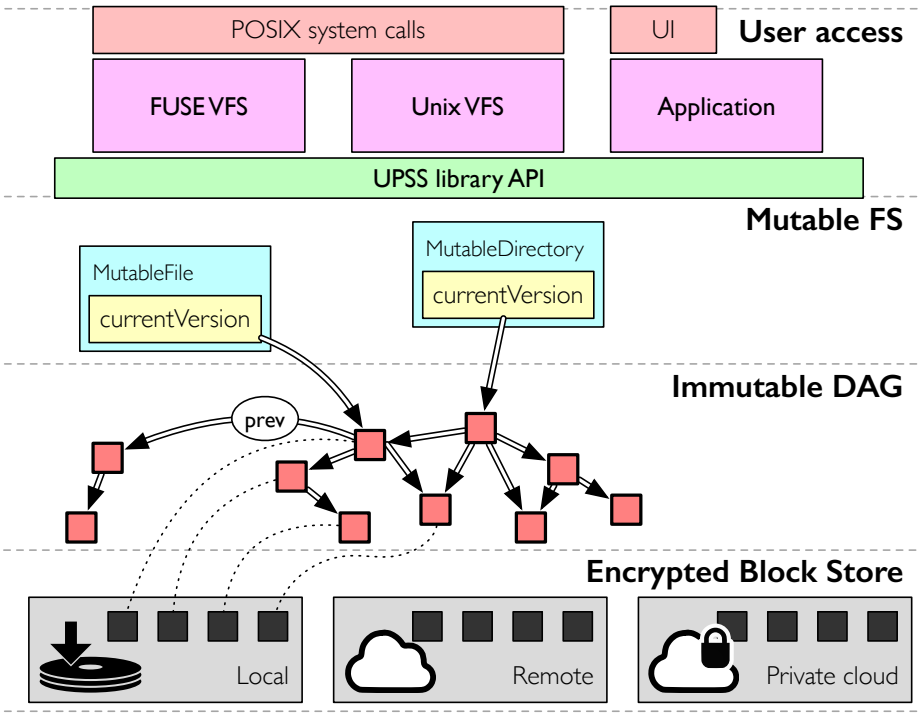


Figure 6.1: The layered structure of UPSS.

6.2.1 BlockStore layer

As we have targeted UPSS to be used as a distributed system and to support data sharing, its design includes a layer called block store, which is the API for storing and retrieving data blocks, on and from the data storage. This layer has a minimal interface, consisting of two main functions to *read* and *write*, and two other functions for reporting the used blocks and outputting the block hashes. In the current implementation, there is no garbage collector and this is left for future work.

A block storage is a *sea of encrypted blocks*. This API supports *local*, *remote*, *in-memory* and *cloud* data storage. We can have a local block store, which is writing data on a local data storage, on a local system, or we can have a shared data storage located on a remote system or on a cloud storage provider.

The *write* method accepts an encrypted block and generates the cryptographic

hash of its content and a lookup operation is done on a map including the cryptographic hashes of the previously stored encrypted blocks. The content is persisted to the storage if its cryptographic hash does not exist in the map. This enables safe deduplication across many users even at a global scale. The generated cryptographic hash is returned by the *write* method. The *read* method accepts the cryptographic hash of a block and returns its corresponding encrypted block to the above layer. The performance of *read* and *write* methods affects any other systems which are using UPSS. We have implemented a caching block stores coupled with journaling to write the blocks in faster block stores when possible. The caching block store consists of two *near* and *far* stores, each of which is an implementation of the block store. Upon finishing the writes in the near block store, the returned block pointer is journaled to a local file and the write to the far block store, which is a more expensive write, is done in background. In this way, we are providing non-blocking writes in the caching block store and by journaling, we ensure that we are not missing the expensive writes.

We have implemented a local block store for our system, which uses a local file as the data storage. UPSS' local block store can be a user's storage medium, or a temporary cache for other remote block stores, or permanent storage, such as what is used in peer-to-peer systems in which everything is stored on the local storages. It stores fixed-size encrypted blocks without any plaintext metadata. Besides the local block store, we also have implemented amazon block store, that is connected to Amazon S3 [AWS20] service, a remote block store, described in Section 6.3.3.1, which is used as the storage integrated with our version control system (Section 6.3.5), a memory block store for keeping everything in memory, and a caching block store.

Listing 6.1 shows a caching block store that includes memory, local and remote block stores.

```
let memory_store = MemoryBlockStore::new()?;
let local_store = LocalBlockStore::new(file, block_size)?;
let remote_blockstore = RemoteBlockStore::new(server_addr,
 block_size)?;
let l1_cache = CacheingBlockStore::new(memory_store, local_store)
 ?;

let store = CacheingBlockStore::new(l1_cache, remote_blockstore)?;

```

Listing 6.1: An example of creating a caching block store.

6.2.2 Immutable DAGs

In UPSS, each data item is stored as a set of fixed-size immutable encrypted blocks, which are linked together in a DAG (Directed Acyclic Graph). For example, an immutable file version can be represented as a tree of immutable blocks, a file's history as a DAG of versions and blocks, and a directory as a mapping from names to files. Immutable blocks are encrypted using symmetric keys derived from cryptographic hashes of their plaintext, a technique known as *convergent encryption* [DAB+02, LCL+13, ASA17], and named using the cryptographic hash of their ciphertext, a technique known as content-addressing. The name of a block and the key that can be used to access it are referred to as a *BlockPointer* $BP_B = (n_B, k_B)$, given by:

$$\begin{aligned} k_B &= h(B) \\ n_B &= h\left(E_{k_B}\{B\}\right) \end{aligned} \tag{6.1}$$

In Equation (6.1), n_B is the name of a block and k_B is the key used to decrypt it. A block pointer can thus be seen as a cryptographic *capability* [DVH66] to read a block, though not necessarily to modify it. The block pointer to the root node of a file or directory implies the ability to access arbitrary quantities of content, up to an

entire directory tree in a filesystem. We integrate the convergent encryption with optionally random padding to protect low-entropy block contents. UPSS supports both deterministic and non-deterministic padding; De-duplication is only enabled in the former case. UPSS defaults to SHA3 [Dwo15] algorithm for generating the digests for the encrypted blocks, but the algorithm choice is not hard-coded in the UPSS design and other hash functions can be used. We encrypt the blocks using AES, which is a symmetric key algorithm. In the current design, we can encrypt data blocks with 16, 24, or 32-byte keys using AES128, AES192, or AES256 algorithms, respectively. The hashing and encryption algorithms are embedded in the block pointers to allow future cryptographic algorithm updates.

When fully convergent encryption is used (the UPSS default), de-duplication is possible across multiple users, as the same content always hashes to the same key, producing the same ciphertext. We reduce the data inconsistency problem to a version control problem by defining blocks to be immutable. A content modification causes a new version of the content to be created, and this modification affects the parent block. This approach is similar to the update approach of copy-on-write (COW) filesystems, which apply the updates all the way up until the root block. UPSS keeps the old versions of content by storing a pointer to the previous version of the modified content in their corresponding root blocks. Therefore, if the new version of a block is not persisted to a block store yet, the older versions of the block are accessible using the previous pointers until the updated version is ready to be used.

6.2.3 Mutable Filesystem API

UPSS provides an object-oriented view of the underlying encrypted blocks and enables the applications to interact with the system using the provided public API. The traditional filesystem concepts such as files, directories, and directory trees are implemented as in-memory objects that provide the mutability. Both `File` and

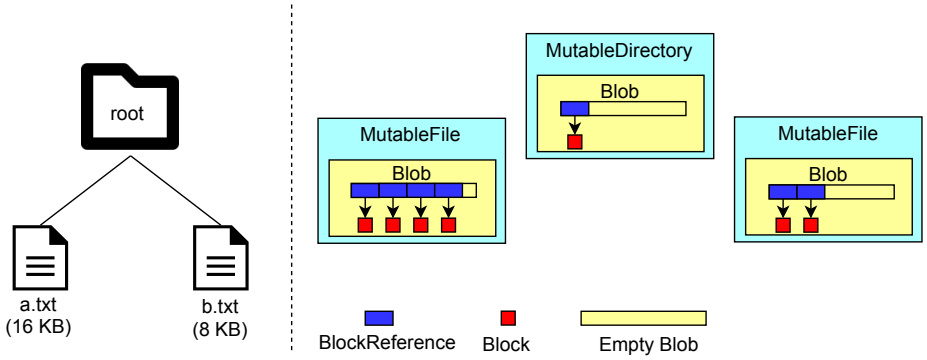


Figure 6.2: The corresponding in-memory objects related to each file and directory. Each in-memory object contains a Blob that keeps a list of copy-on-write references to immutable in-memory Blocks.

`Directory` contents are managed by a binary `Blob` structure, which maps data into immutable blocks via copy-on-write references. Figure 6.2 shows how we relate the in-memory objects to files and directories.

In UPSS, all the modifications are handled in the mutable layer by updating the in-memory objects, unless UPSS asks for persisting the objects explicitly. For persisting a `File` or `Directory`, the list of `BlockReferences`, each of which points to a `Block`, are persisted and their block pointers are added to a `MetaVersion` structure. The final step is persisting the `MetaVersion` and adding or updating its root block pointer in the object's parent. Having the `MetaVersion` structure enables us to implement partial sharing and redaction with intergrity check. However, this feature is not fully implemented and we postponed it to the furure work (see Section 6.5.2).

The process of persisting a `MetaVersion` is different from persisting data blocks. The `MetaVersion`, which holds a list of block pointers to the encrypted data blocks, is chunked into fixed-size encrypted blocks if its size is more than the UPSS's block size. The encrypted blocks related to the `MetaVersion` are linked together in a linked list in which each block's block pointer is embedded inside its preceding block. A history of object versions is kept by the `Prev` pointers.

UPSS is implemented as a library that can be linked directly into applications; a simple example of such integration is shown in Listing 6.2. Currently, applications integrating the UPSS library access it via Rust calling conventions, as the library is written in Rust [Tea20a]. However, in the future, we will support other programming languages such as C, Python and JavaScript/WASM via foreign function interfaces as described in Section 6.5.1.

```
let store = LocalBlockStore::new(file, block_size)?;
let fs = upss::UPSS::new(Box::new(store));

// [...]

let f = fs.new_file()?;
f.write(&some_bytes)?;

```

Listing 6.2: An example of integrating the UPSS library directly into an application, accessing FS objects via API calls.

6.3 Case studies

In this section, we demonstrate the practicality of UPSS via four case studies. Each case study demonstrates UPSS's qualitative ease of use as well as its quantitative performance, with comparisons to other systems drawn where appropriate. These four case studies are:

- UPSS as a local filesystem (Section 6.3.2),
- UPSS as a network filesystem (Section 6.3.3),
- UPSS as a global filesystem (Section 6.3.4) and
- UVC: UPSS as a version control system (Section 6.3.5).

6.3.1 Benchmark description

We evaluated the cost of creating files and directories and reading and writing from/into on-disk local and remote block stores. For evaluating file and directory creation, we generated a user-defined number of files and directories, added them to an ephemeral root directory and persisted the results into file-backed block stores.

For evaluating read and write operations, we generated 1000 files filled with random data of size 4 KiB, the natural block size of our underlying storage, select a file randomly and processed the sequential read and write operations on it.

We also implemented a macrobenchmark that simulates a web server behaviour. We selected a file randomly from a file set and performed 10 consecutive read and write operations with different I/O sizes: 4 KiB, 256 KiB, 512 KiB and 1 MiB. The results of our macrobenchmark is discussed in Section 6.3.2.4.

The Filebench [fil16] framework gave us an idea about how to implement our benchmark functions. We did not use the Filebench framework for our evaluations as it did not provide the level of detail about filesystem's behaviour during time that is reported in Sections 6.3.2.2, 6.3.3.2 and 6.3.4.1.

We ran each microbenchmark function 1,000 times for the direct usage of UPSS API and 5 times for the `upss-fuse` micro and macro benchmarks; the results are reported as the arithmetic mean of the runs along with their standard deviations. We ran the benchmarks on a 4-core, 8-thread 3.6 GHz Intel Core-i7-4790 processor with 24 GiB of RAM and 1 TB of ATA 7200 RPM magnetic disk, running Ubuntu Linux 4.15.0-72-generic (the machines' configurations for the network filesystem evaluations are different, explained in Section 6.3.3.2). The results of these benchmarks for direct usage of UPSS API — as well as a comparison with benchmark results from Sections 6.3.2.2 and 6.3.3.2 — can be seen in Figure 6.3. This figure clearly shows the performance degradation caused by FUSE [VAM+19] in comparison with the direct API usage.

One of the most expensive operations in UPSS is persisting data into block stores,

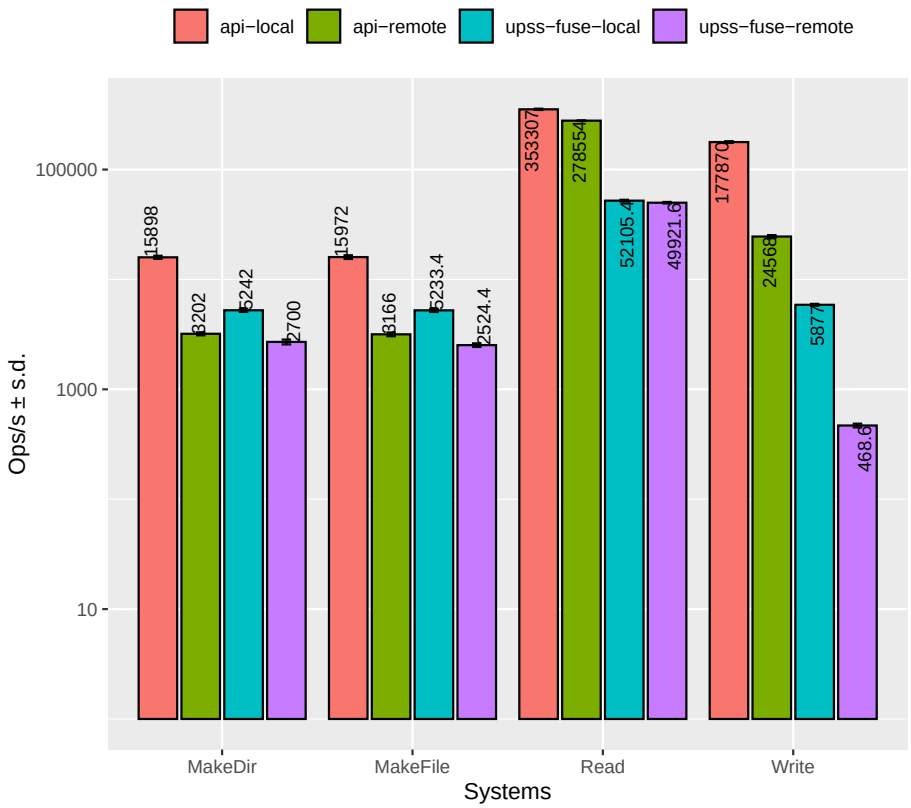


Figure 6.3: A comparison of the performance of UPSS when accessed directly via the UPSS API and via `upss-fuse` connected to a local or remote block store. The numbers reported as the average number of operations done in 60 seconds for 5 runs. Confidence intervals are represented with error bars.

as discussed in Section 6.2.3. Since in-memory data structures may be written many times in a short interval, "dirty" blocks are only written to the block store when explicitly requested (or, in the case of `upss-fuse`, every 5 s), which leads to a new version of the block; only at this point are any cryptographic hashes computed, blocks encrypted, etc. The cost of this process is illustrated in Figure 6.4; it is superlinear due to the larger amounts of metadata required to describe larger amounts of data. This relationship is also seen in UPSS' total storage requirements, as shown in Equation (6.2), where the total space s_t required to store s B of data is

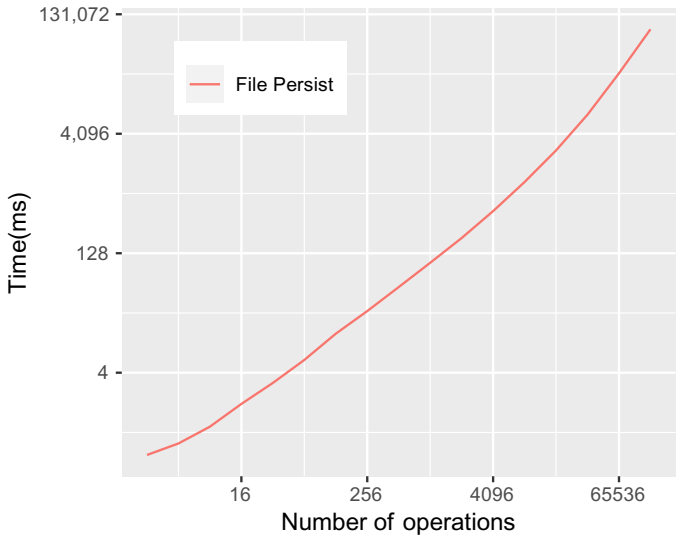


Figure 6.4: The time required to persist files to a block store scales superlinearly with the number of files, as larger amounts of data require larger amounts of metadata to describe them (directory persist times are almost identical to those of files).

very slightly superlinear.

$$s_t = (1.09 + 0.001613s)\, s \tag{6.2}$$

6.3.2 UPSS as a local filesystem

Direct usage of the UPSS API requires program modification — and, today, the use of a specific programming language. In order to expose the benefits of UPSS to a wider range of software, we have implemented a *filesystem in userspace (FUSE)* [fus19] wrapper that exposes UPSS objects to other applications via a hook into the Unix VFS layer. As shown in Figure 6.5, `upss-fuse` maps FUSE inode numbers to in-memory UPSS objects to service VFS requests. This allows conventional applications to access an UPSS directory mounted as a Unix directory with POSIX semantics, though there is one unsupportable feature: hard links. Hard links are defined within the context of a single filesystem, but UPSS is designed to allow any direc-

tory to be shared as a root directory of a filesystem. Owing to this design choice, it is impossible to provide typical hard link semantics and, e.g., update all parents of a modified file so that they can perform their own copy-on-write updates (see Section 6.2.3). Therefore, we do not provide support for hard links — a common design choice in network file systems such as NFS.

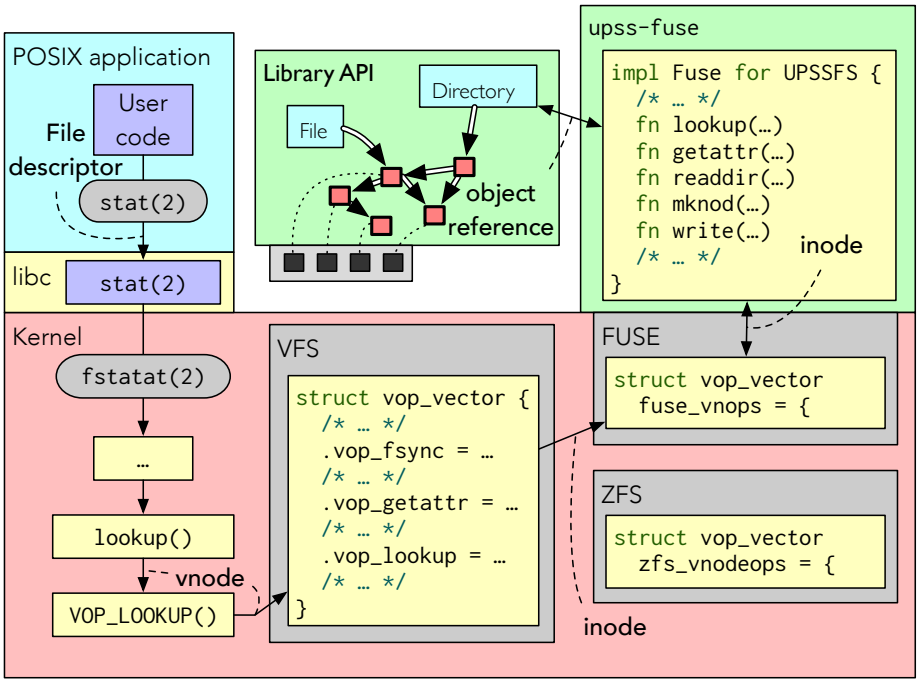


Figure 6.5: `upss-fuse` exposes a UPSS directory to POSIX applications via an in-kernel FUSE device.

The `upss-fuse` wrapper exposes an ephemeral plaintext view of an UPSS's directory underneath a Unix mount point, allowing conventional file and directory access, while keeping all data and metadata encrypted at rest in a local or remote block store (see Section 6.3.5). Unlike existing cryptographic filesystems such as NCryptFS [WMZ03] and EncFS [LFS16a], no plaintext directory structure is left behind in the mount point after the filesystem has been unmounted. The only meta-

data that is kept in the clear by the `upss-fuse` compatibility layer is the block pointer of the root directory, which is automatically updated as the filesystem contents are modified and the root directory is persisted to the block store. This block pointer is currently kept in a plaintext file; we plan to protect it with symmetric or asymmetric cryptography in the future (see Section 6.5).

6.3.2.1 Snapshots and consistency

As a copy-on-write filesystem, UPSS provides cheap snapshots. As a user-empowering *sharing* system, these snapshots can be quickly shared with other users for read-only access: a user need only share the block pointer to a file or directory with another user, and that user will be able to retrieve the content from a block store and decrypt it. To facilitate such sharing, `upss-fuse` exposes both cryptographic hashes (which provide integrity guarantees over Merkle DAGs for blockchain-like applications) and full block pointers (which allow content sharing) to users via POSIX *extended attributes*, an of which is shown in Listing 6.3.

```
% xattr -p user.hash mnt/a-file-in-upss
sha3-512:hdd3P80hjERoF1PO9ezuOEQQwG/Goey2Up5je...

```

Listing 6.3: `upss-fuse` exposes UPSS cryptographic details to users via POSIX extended attributes. This allows users to verify the integrity of a directory tree or to share a directory's contents via its block pointer.

UPSS creates snapshots whenever requested by asking for a directory's cryptographic name (which depends on its entries' names, depending on their contents, etc. — see Section 6.2.2). In order to provide data consistency, `upss-fuse` requests that UPSS persist a "dirty" — i.e., modified — root directory every five seconds, or after a tunable number of dirty objects require persisting. As described in Section 6.2.3, persisting a `Directory` object causes its versioned children to be recursively persisted (if dirty), after which the cryptographic block pointer for the new root directory

version can be stored in the `upss-fuse` metadata file. As in other copy-on-write filesystems, the cost of persisting an entire filesystem depends on the amount of "dirty" content in the filesystem. The trade-off between the demand for frequent data synchronization and the requirement for more frequent — though smaller — persistence operations is illustrated in Figure 6.6.

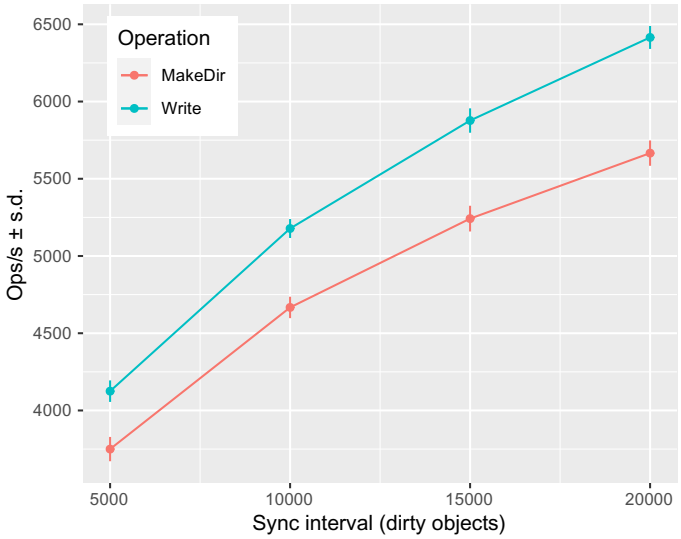


Figure 6.6: Trade-off of sync frequency vs performance. The average number of operations (in five runs, each 60 seconds) that can be done per second for different sync intervals are shown in y-axis (the content size in the write operations is 4 KiB). Sync interval x means x objects are kept in memory until the next sync.

6.3.2.2 Performance comparisons

To illustrate the performance of UPSS when used as a conventional local filesystem, we compared `upss-fuse` with the FUSE-based cryptographic filesystems CryFS [MRAMQ17] and EncFS [LFS16b, LFS16a], as well as the mature, heavily-optimized ZFS [BAH+03]. The latter has been included because, although it is not a cryptographic filesystem designed for fine-grained confidentiality, it is a log-structured filesystem with some

common features such as copy-on-write updates and cryptographic hashes used to name blocks. In contrast to `upss-fuse`, ZFS has been extensively optimized over the past two decades to become a high-performance, widely-deployed filesystem.

We mounted each of these four filesystems on different paths in the Linux host referenced in Section 6.3.1 and ran four microbenchmarks to test their speed in creating empty directories (**MakeDir**), creating empty files (**MakeFile**), reading randomly select files sequentially including 4 KiB of data (**ReadFile**) and writing random data to files (**WriteFile**). Each of these four benchmarks was run for 60 seconds and the operations per second were calculated as the average of 5 runs; the results are shown in Figure 6.7. UPSS outperforms EncFS and CryFS for all operations, with performance especially exceeding these existing systems in the critical **Read** and **Write** benchmarks. As might be expected, ZFS significantly outperforms UPSS in three out of four benchmarks, with **Read** performance 6.176× and **Write** performance 20.03× faster than `upss-fuse`, but `upss-fuse` does outperform ZFS in one benchmark: **MakeDir**. In `upss-fuse`, creating files and directories have the same cost, as they are both backed by empty collections of blocks, but ZFS is optimized for the creation of files as the cost of directory creation speed. We also note that `upss-fuse` performs $1.47 - 8.2\times$ more operations per second in various benchmarks than CryFS and EncFS while also providing stronger security properties (see Section 6.4).

Figure 6.8 shows a more detailed examination of the behavior of the four comparison filesystems. In these plots, a fixed number of benchmark operations were performed; the x-axis represents the time needed to complete all 100k operations. These plots show the bursty nature of real filesystems, and in the case of CryFS, they reveal performance that scales poorly as the number of requested operations increases.

Much of the bursty nature of these plots derives from how each filesystem synchronizes data to disk. For example, by default, ZFS synchronizes data every 5 s

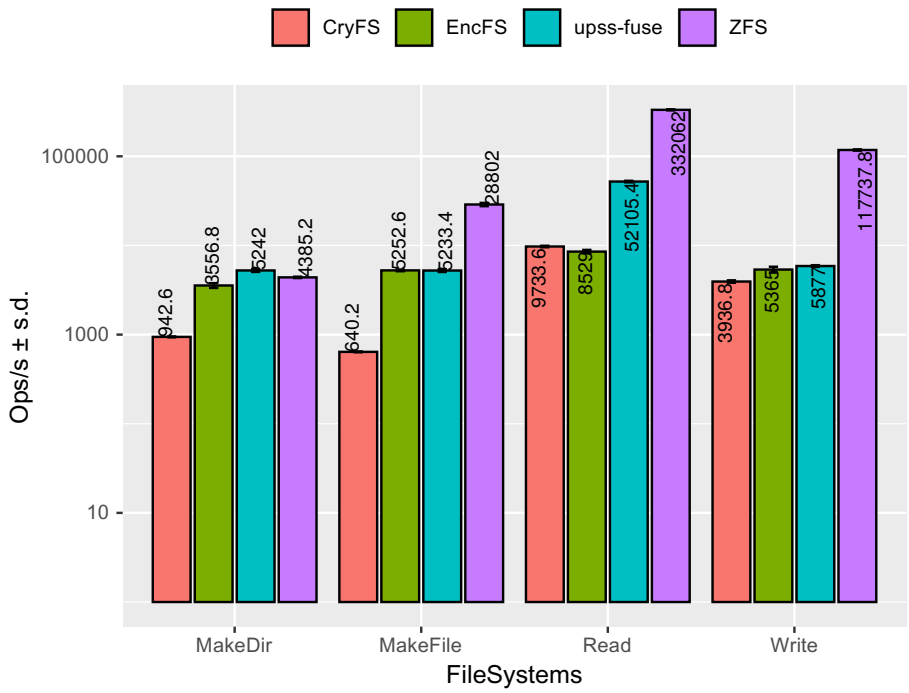


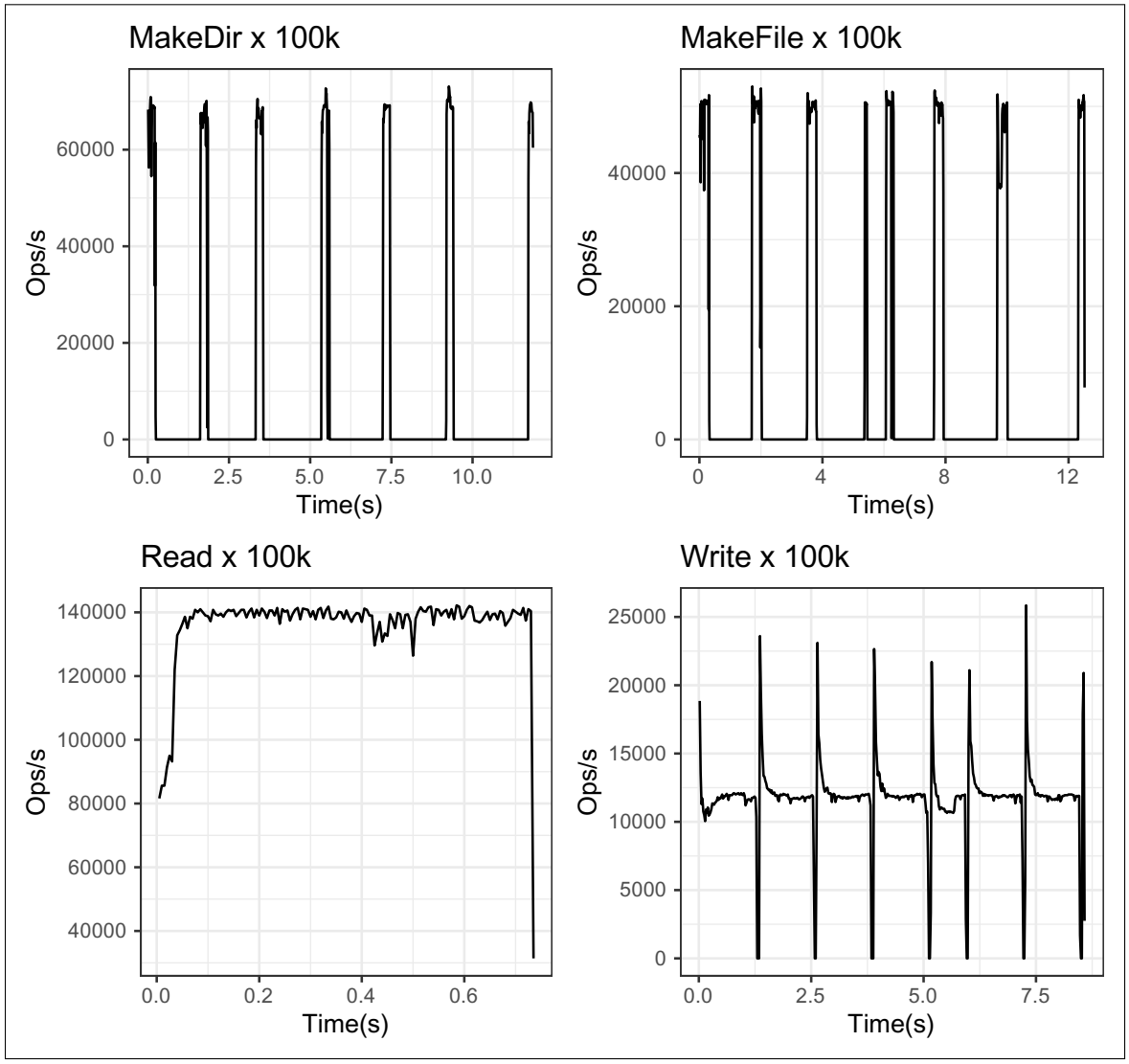
Figure 6.7: Operations that can be performed per second by CryFS, EncFS, `upss-fuse` and ZFS for our four microbenchmarks. The numbers reported as the average number of operations done in 60 seconds for 5 runs along with their standard deviations as error bars.

or when 64 MiB of data has accumulated to sync, whichever comes first. Similarly, to provide a fair comparison, `upss-fuse` is configured to synchronize after 5 s or 15,000 writes (close to 64 MiB of data when using 4 KiB blocks). These periodic synchronizations cause performance to drop, even on dedicated computers with quiescent networks and limited process trees.

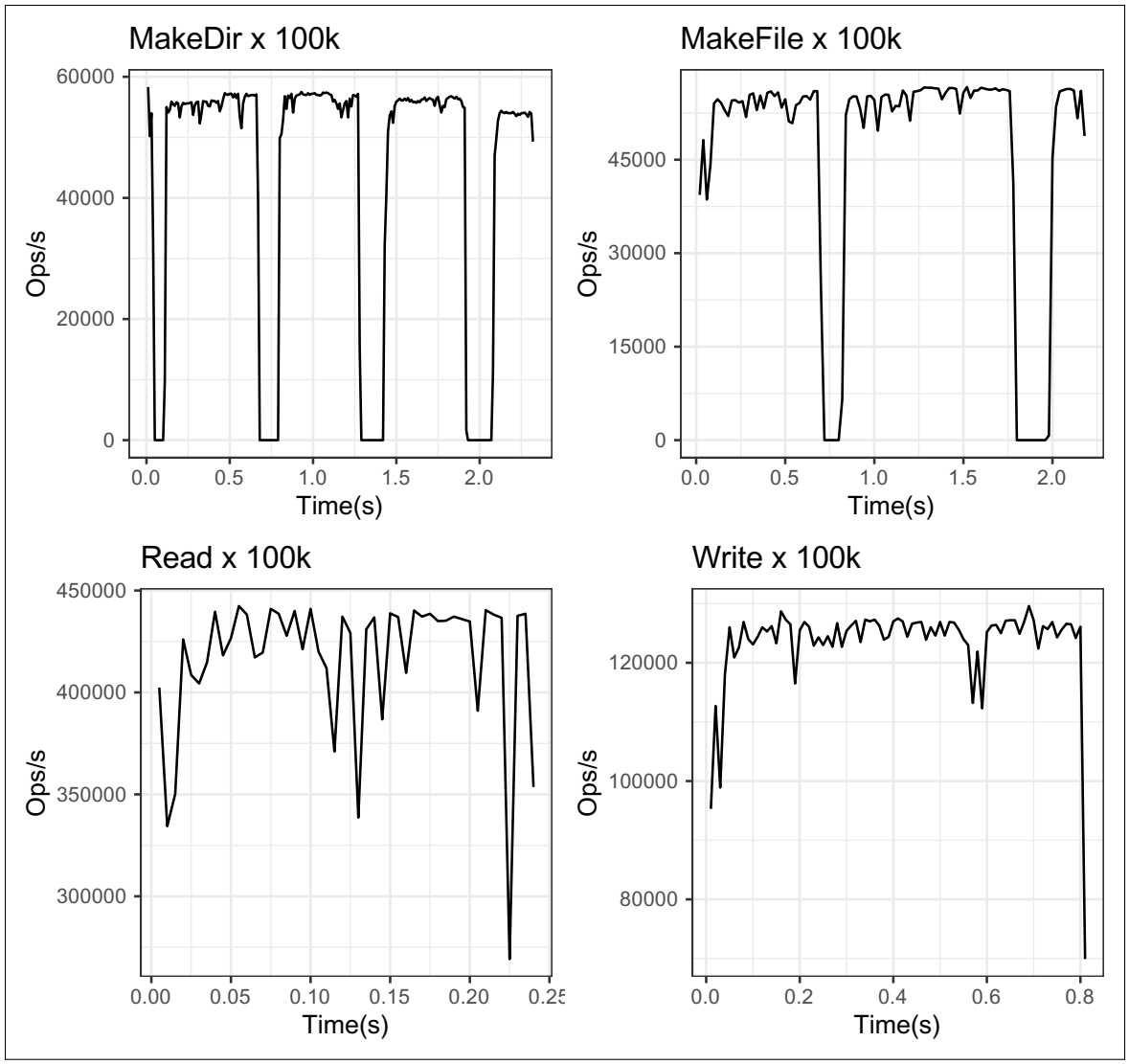
6.3.2.3 Deduplication

As stated in Section 6.2.2, UPSS's convergent encryption provides natural de-duplication for blocks containing the same content, even if they are saved and encrypted by mutually-distrustful users. To evaluate the effect of de-duplication on

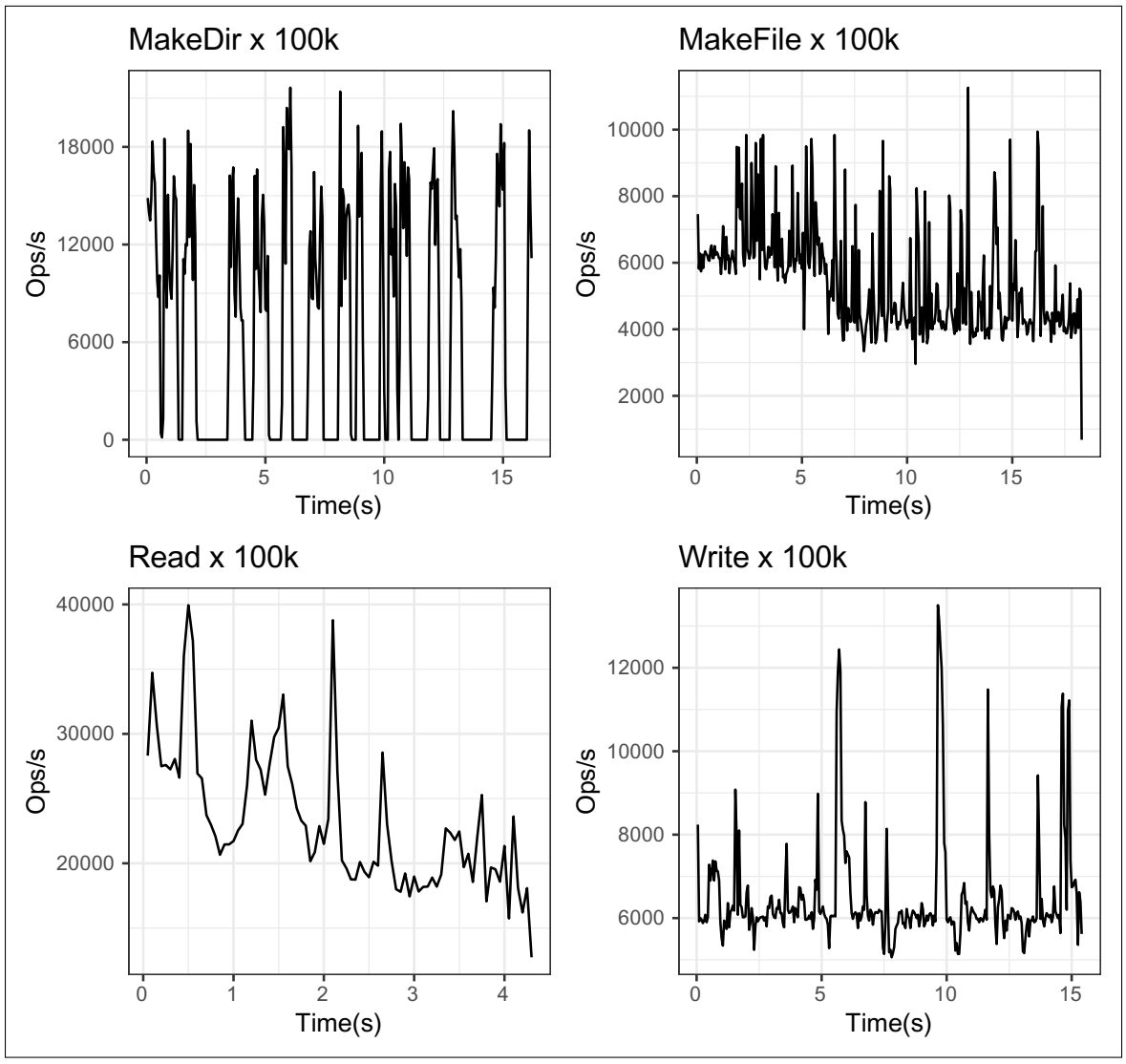
Figure 6.8: Operations that can be performed per second by CryFS, EncFS, `upss-fuse` and ZFS for our four microbenchmarks. The behaviour of the filesystems are reported for 100k operations during time.



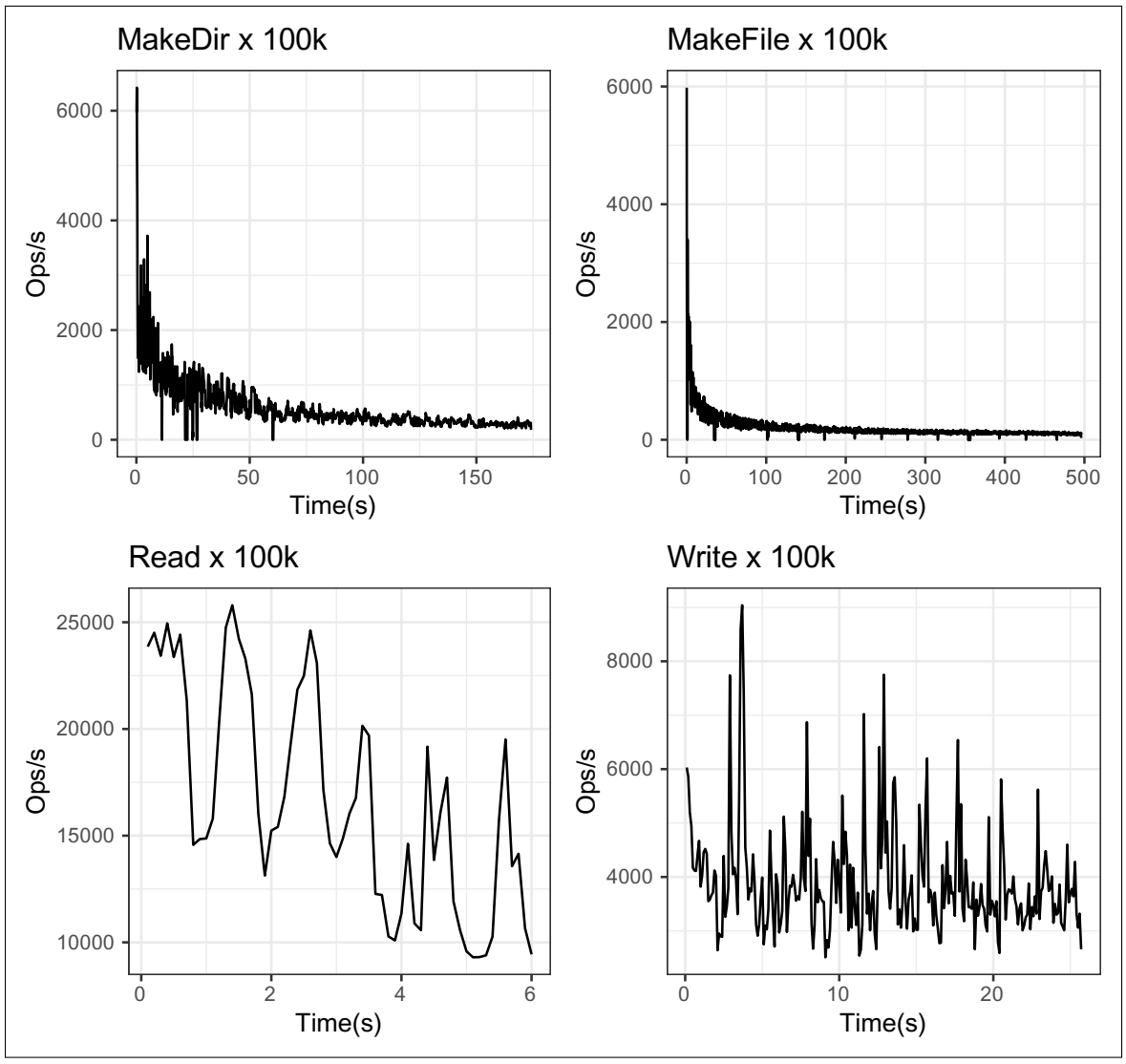
(a) `upss-fuse`


MakeDir x 100k
Ops/s
60000
40000
20000
0
0.0
0.5
1.0
1.5
2.0
Time(s)
MakeFile x 100k
Ops/s
45000
30000
15000
0
0.0
0.5
1.0
1.5
2.0
Time(s)
Read x 100k
Ops/s
450000
400000
350000
300000
0.00
0.05
0.10
0.15
0.20
Time(s)
Write x 100k
Ops/s
120000
100000
80000
0.0
0.2
0.4
0.6
0.8
Time(s)

(b) ZFS


MakeDir x 100k
Ops/s
18000
12000
6000
0
0
5
10
15
Time(s)
MakeFile x 100k
Ops/s
10000
8000
6000
4000
2000
0
5
10
15
Time(s)
Read x 100k
Ops/s
40000
30000
20000
0
1
2
3
4
Time(s)
Write x 100k
Ops/s
12000
10000
8000
6000
0
5
10
15
Time(s)

(c) EncFS


MakeDir x 100k
MakeFile x 100k
Read x 100k
Write x 100k
Ops/s
Time(s)
0
2000
4000
6000
50
100
150
200
300
400
500
10000
15000
20000
25000
8000
10
20

(d) CryFS

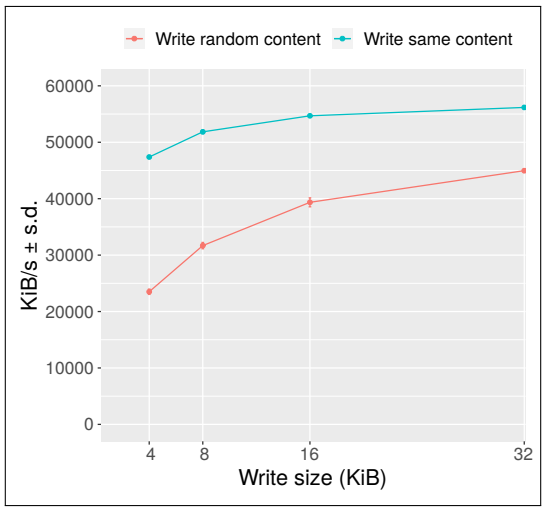
overall performance, we compared writing the same content and different contents into files by measuring the amount of data that can be written per second. As shown in Figure 6.9, `upss-fuse` and ZFS benefit from de-duplication. This effect becomes more pronounced for ZFS as write sizes increase.

6.3.2.4 Macro-benchmark

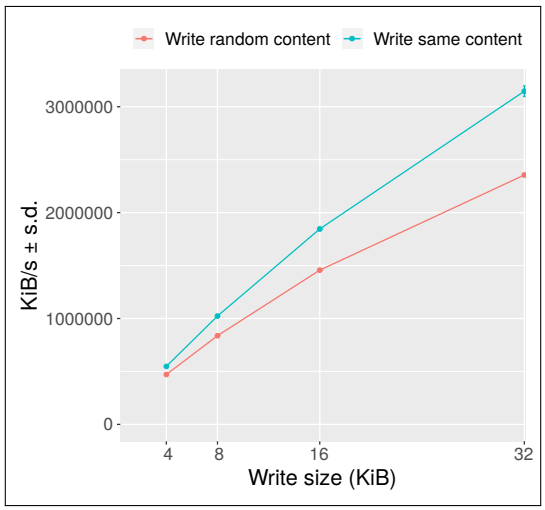
We ran our macrobenchmark function discussed in Section 6.3.1 on `upss-fuse`, CryFS, EncFs and ZFS, to evaluate `upss-fuse` in a a web server simulation in which consecutive read and write operations with different I/O sizes are performed on different files. The results are reported in Figure 6.10. As in previous benchmarks, ZFS outperforms the other filesystems for different I/O sizes. `upss-fuse` achieved better results than CryFS and EncFS for the 4 KiB case. However, as the I/O size increases, CryFS outperforms `upss-fuse`. The reason is that the bigger files we have, the more number of fixed-sized blocks are generated by `upss-fuse`, each of which needs to be encrypted with a different key and then persisted. But in CryFS all the fixed-size blocks related to a file are encrypted with one symmetric key. Therefore, the key generation is done once per each file in CryFS and this cause a better performance for larger files, and in the same time, makes CryFS inapplicable for partial file sharing and redaction scenarios that are supported by UPSS. Also, several studies have shown that the files in a filesystem are small with the mean size of only a few kilobytes [RO92, BHK+91, LZCZ86].

6.3.3 UPSS as a network filesystem

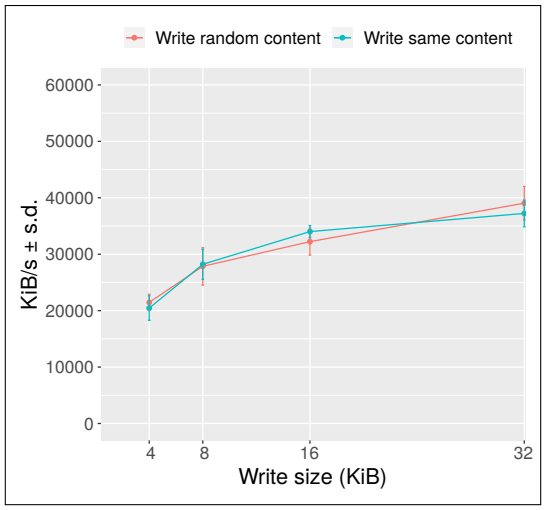
Although UPSS can be used as a local filesystem, it is primarily designed as a system for sharing data across networks with untrusted storage providers. Taking advantage of UPSS's unique properties requires an evaluation that is not directly comparable to other systems. Thus, we have compared the performance of `upss-fuse` when connected to a *remote block store* (Section 6.3.3.1) to that of SSHFS [Tea20b] and



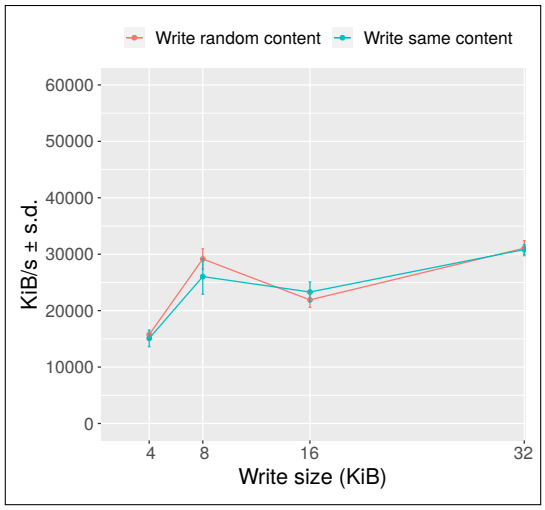
(a) `upss-fuse`



(b) ZFS (larger y-axis scale)



(c) EncFS



(d) CryFS

Figure 6.9: Performance of writing the same content compared with writing random contents into files. The numbers are the average of KiB of content written per second for five runs. Note the different y-axis for ZFS.

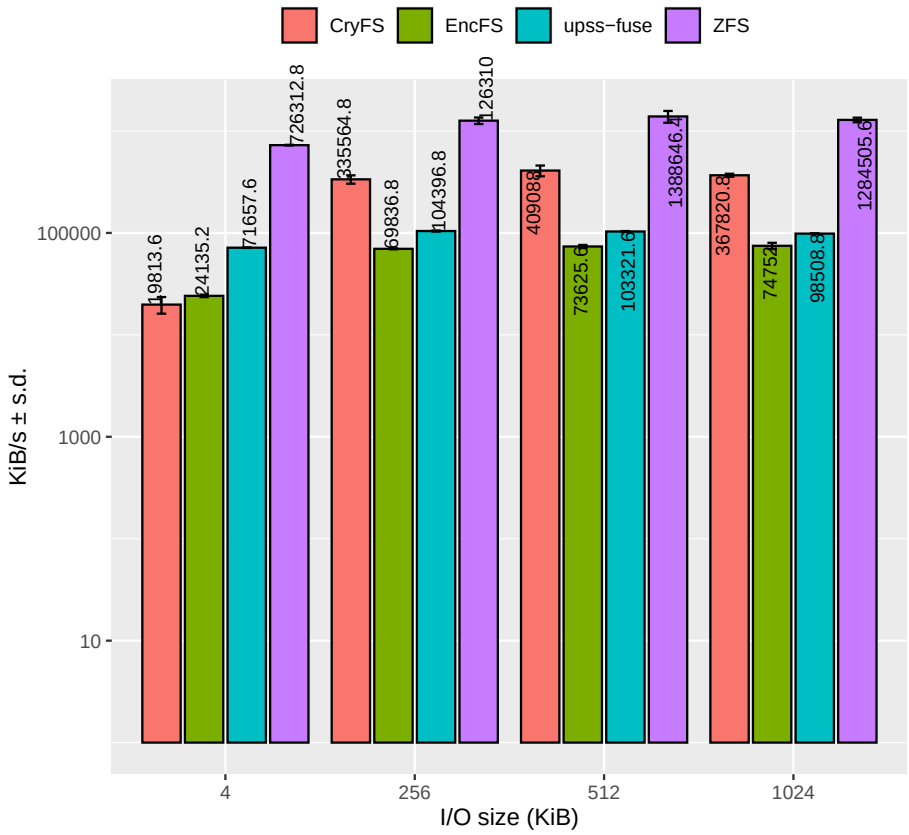


Figure 6.10: Performance of CryFS, EncFS, `upss-fuse` and ZFS for the web server macrobenchmark. The numbers are the average of KiB of I/O per second for five runs, each 60 seconds.

the venerable NFS [SCR+03] (Section 6.3.3.2).

6.3.3.1 Remote block store

In UPSS, a block store caches encrypted fixed-size blocks on behalf of users. The confidentiality and integrity of these blocks' content is assured by cryptographic operations performed by clients before blocks are sent to the block store, so the underlying storage medium may be untrusted. The block store sees the blocks as immutable ciphertext blobs, named by the cryptographic hash of their contents. This approach allows us to build a block store in which a centralized server exploits

high-quality network links and disks to receive and transmit large numbers of encrypted blocks — the data plane — regardless of what block pointers are shared between users — the control plane. Clients send a data block to be stored or request to get previously stored data, identified by block pointers. This design is amenable to multi-layer caching, with a system accessing a remote block store having the option of caching immutable blocks in a local block store, and even caching those results in memory.

6.3.3.2 Performance comparison

As in Section 6.3.2.2, we evaluated the performance of UPSS by mounting an `upss-fuse` filesystem in a Unix mount point and comparing it to other filesystems using four microbenchmarks. In this section, however, we connected our `upss-fuse` filesystem to a remote block store and compared our performance results against two other remote filesystems: the FUSE-based SSHFS [Tea20b] and the venerable NFS [SCR+03]. Similar to Section 6.3.2.2, one comparison filesystem is primarily designed for security and the other has higher performance after a long history of performance optimization.

The remote block store server was run on a 4-core, 2.2 GHz Xeon E5-2407 processor with 16 GiB of RAM and 1 TB of magnetic disk, running FreeBSD 12.1-RELEASE. The client machine, that runs `upss-fuse`, is a 4-core, 3.5 GHz Xeon E3-1240 v5 processor with 32 GiB of RAM and 1 TB of magnetic disk, running Ubuntu Linux 16.04. Both the client and the server machines were located on the same LAN, connected to each other via a Gigabit switch dedicated to test machines (and therefore with little traffic). For our comparisons, we ran the benchmarks discussed in Section 6.3.2.2 for 60 seconds and the operation per seconds are reported in Figure 6.11. Figure 6.12 shows the behaviour of the benchmarked filesystems with executing 100k **MakeDir**, **MakeFile**, **Read** and **Write** operations. In the network environment UPSS outperforms SSHFS and NFS for **MakeDir**, **MakeFile** and **Read**

operations and for **Write**, it achieves comparable results. For the **Read** benchmark, `upss-fuse` has a slow start as the encrypted blocks are read from the remote block store and are loaded into the memory. The **Read** benchmark generates 1000 files, each of which of size 4 KiB, filled with random content. After the files being loaded into memory, the other read operations are served from the in-memory objects. This causes `upss-fuse` to be about 5× faster than NFS in the **Read** benchmark. For a fair comparison, we also included the results of **Read** benchmark by clearing the caching block store and the in-memory objects, shown as Read-CacheMiss in Figure 6.11. More specifically, we generated the files on the remote block store, clear all the related in-memory objects and remove the files from the caching block store. Then we started reading the files from the remote block store and writing them to the caching block store. The other two filesystems achieve better read results in comparison with the Read-CacheMiss benchmark.

6.3.4 UPSS as a global filesystem

In addition to local and network filesystem, `upss-fuse` can also be connected to untrusted cloud storage providers. To do so, we have implemented an UPSS block store backed in the Amazon S3 service [AWS20] and compared its performance with S3FS [GNr20] and Perkeep [LN18] (Section 6.3.4.1).

6.3.4.1 Performance comparison

We mounted `upss-fuse` backed with the Amazon block store (with and without local caching), S3FS and Perkeep in a Unix mount point and compared them using our four microbenchmarks. S3FS allows Linux and macOS to mount an Amazon S3 bucket via FUSE without any security properties. Perkeep, formerly called Camlistore, is a FUSE-based cryptographic filesystem that can be backed by memory, local or cloud storage. We configured Perkeep to use an Amazon S3 account for our evaluation.

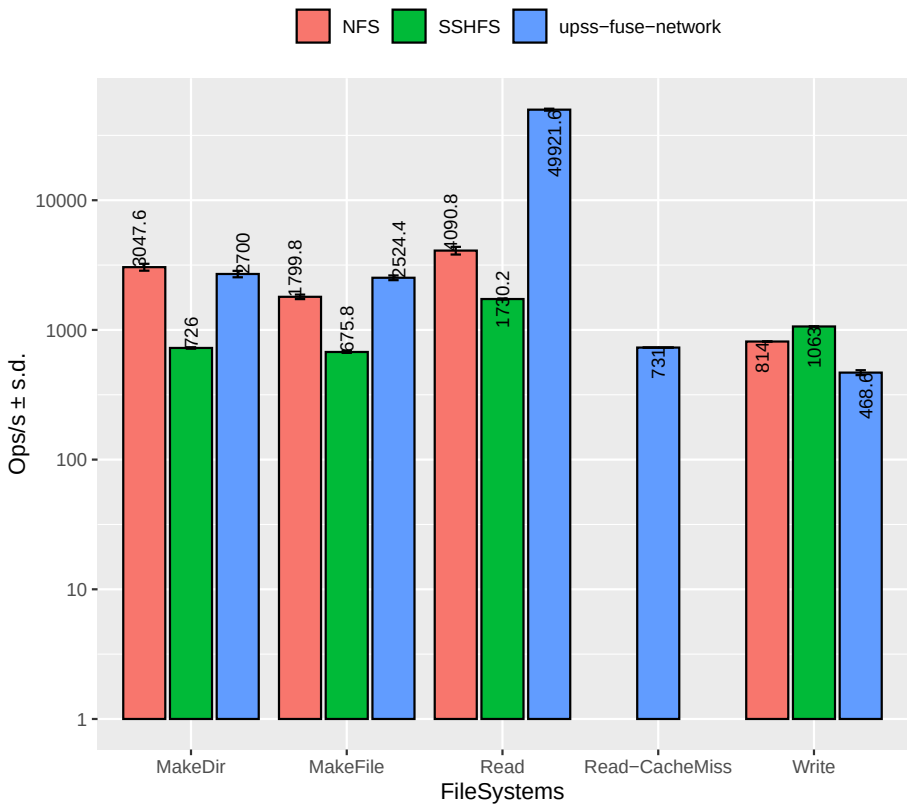
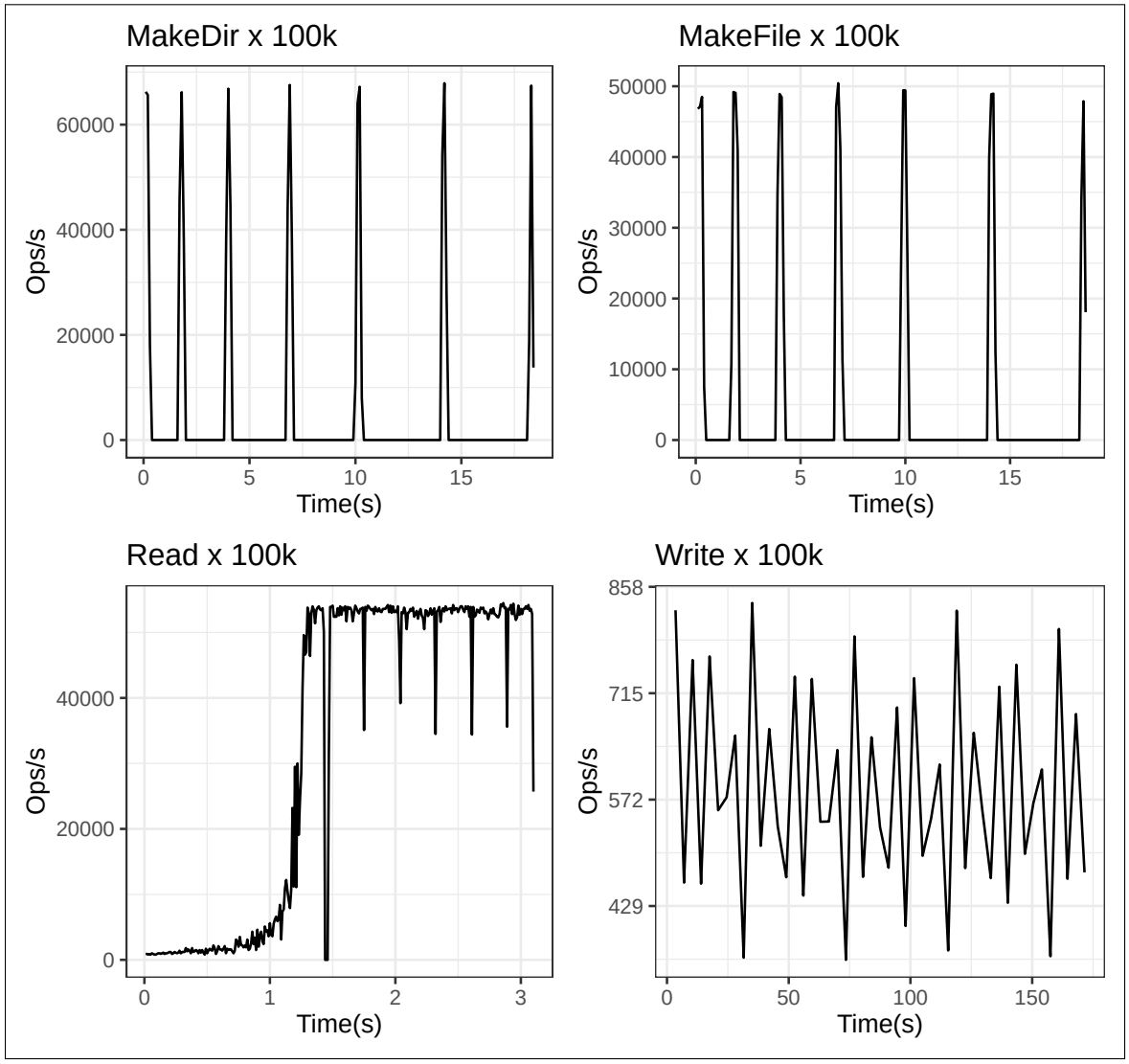


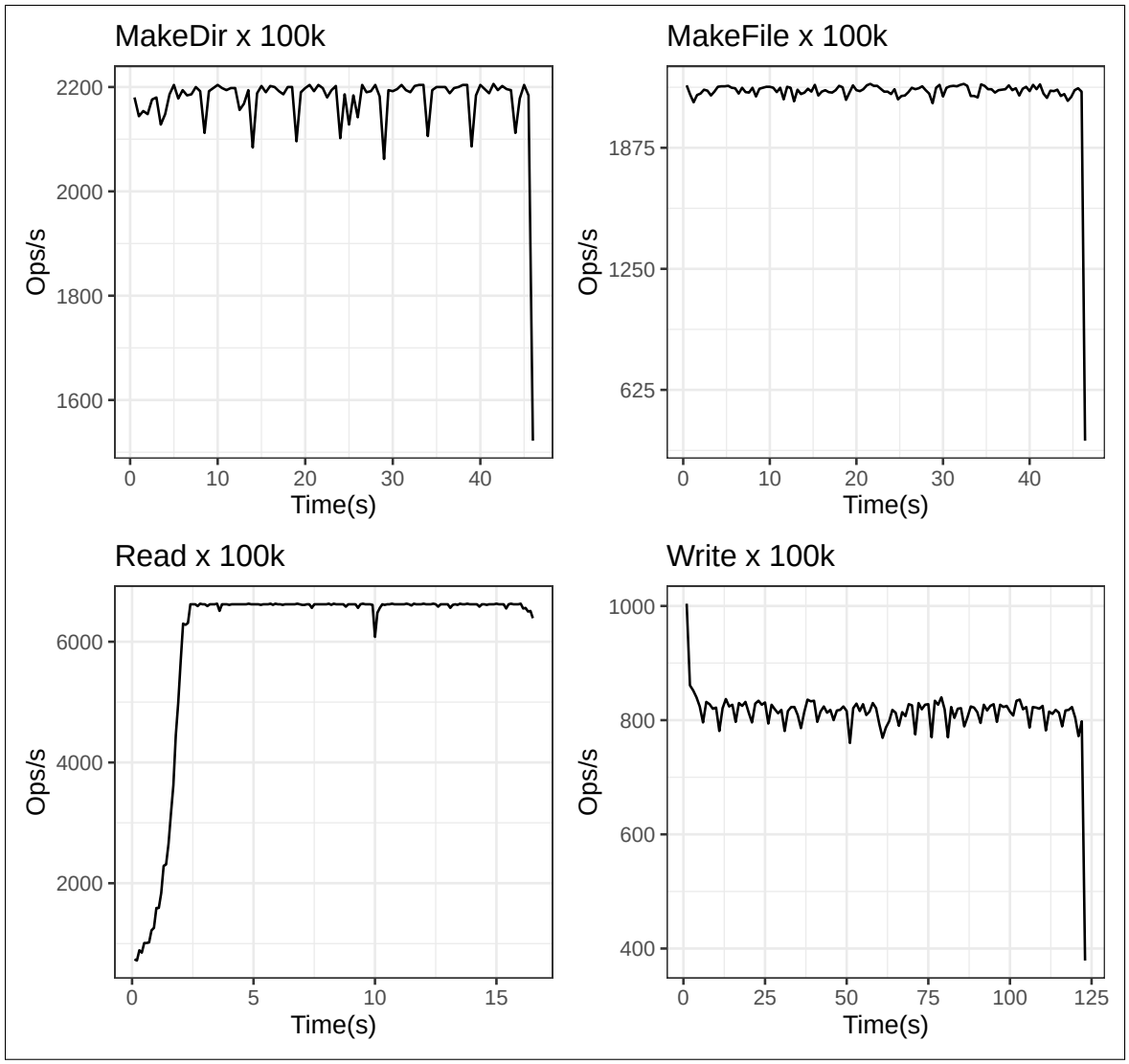
Figure 6.11: Network performance comparisons. The numbers reported as the average number of operations done in 60 seconds for 5 runs. The Read-CacheMiss bar for `upss-fuse` shows the read Ops/s for the files that do not exist in the caching block store and they are read from the remote block store and also are written to the caching block store.

We ran the benchmarks discussed in Section 6.3.2.2 with 5k **MakeDir**, **MakeFile**, **Read** and **Write** operations and the behaviours of `upss-fuse`-network, Perkeep and S3FS during time are reported in Figure 6.13. In all of these cases, Amazon S3's response time is the bottleneck. To have a fair comparison, we ran the benchmarks for `upss-fuse` with and without caching. With caching enabled, we write the encrypted blocks in a caching block store and journal the block names to an on-disk file, then we write to Amazon S3 bucket by processing the journal using a background thread. This makes a large difference in the number of operations that can

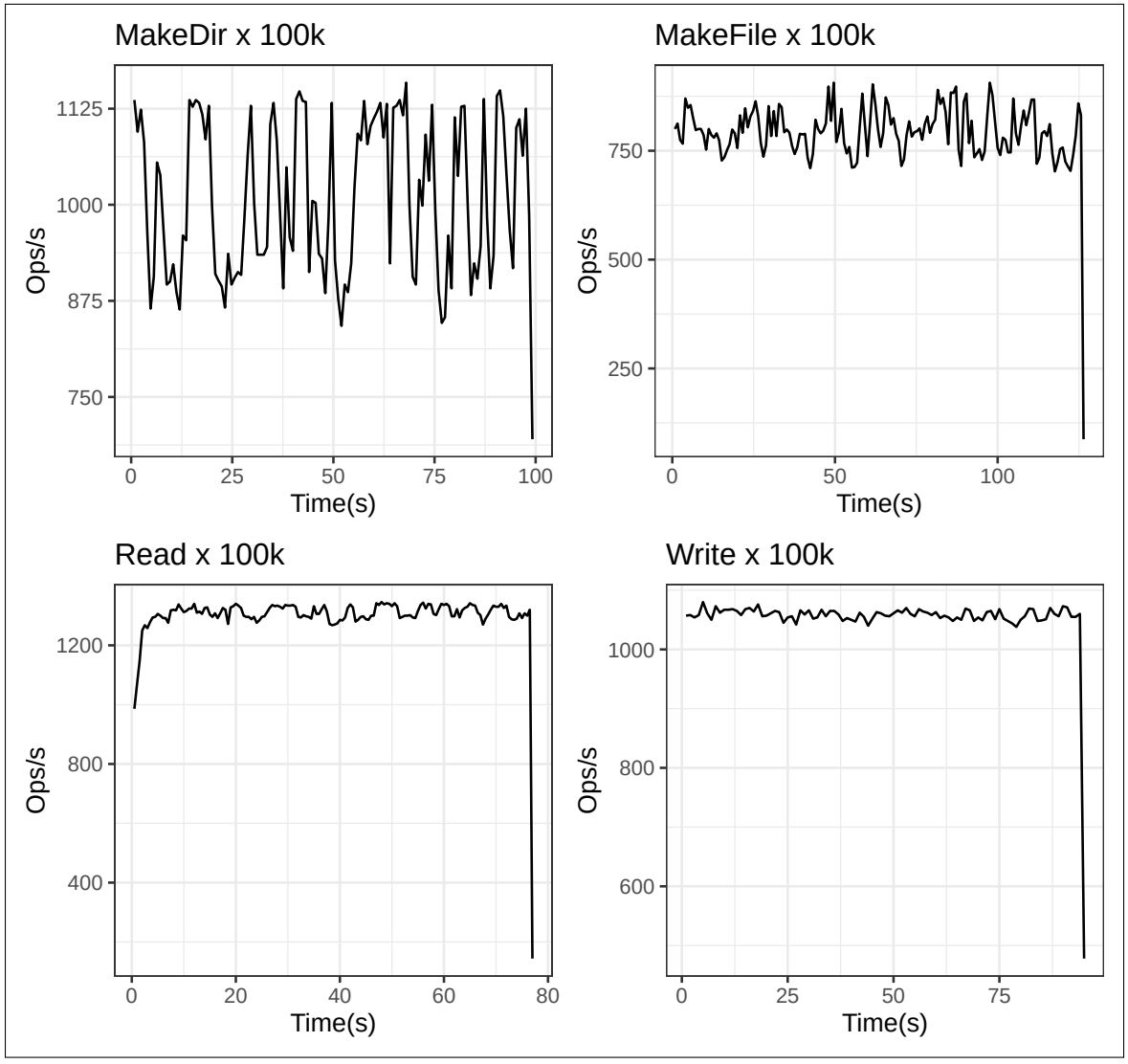
Figure 6.12: Operations that can be performed per second by `upss-fuse`-network, NFS and SSHFS for four microbenchmarks. The behaviour of the filesystems are reported for 100k operations during time.



(a) upss-fuse-network


MakeDir x 100k
Ops/s
2200
2000
1800
1600
0
10
20
30
40
Time(s)
MakeFile x 100k
Ops/s
1875
1250
625
0
10
20
30
40
Time(s)
Read x 100k
Ops/s
6000
4000
2000
0
5
10
15
Time(s)
Write x 100k
Ops/s
1000
800
600
400
0
25
50
75
100
125
Time(s)

(b) NFS


MakeDir x 100k
Ops/s
1125
1000
875
750
0
25
50
75
100
Time(s)
MakeFile x 100k
Ops/s
750
500
250
0
50
100
Time(s)
Read x 100k
Ops/s
1200
800
400
0
20
40
60
80
Time(s)
Write x 100k
Ops/s
1000
800
600
0
25
50
75
Time(s)

(c) SSHFS

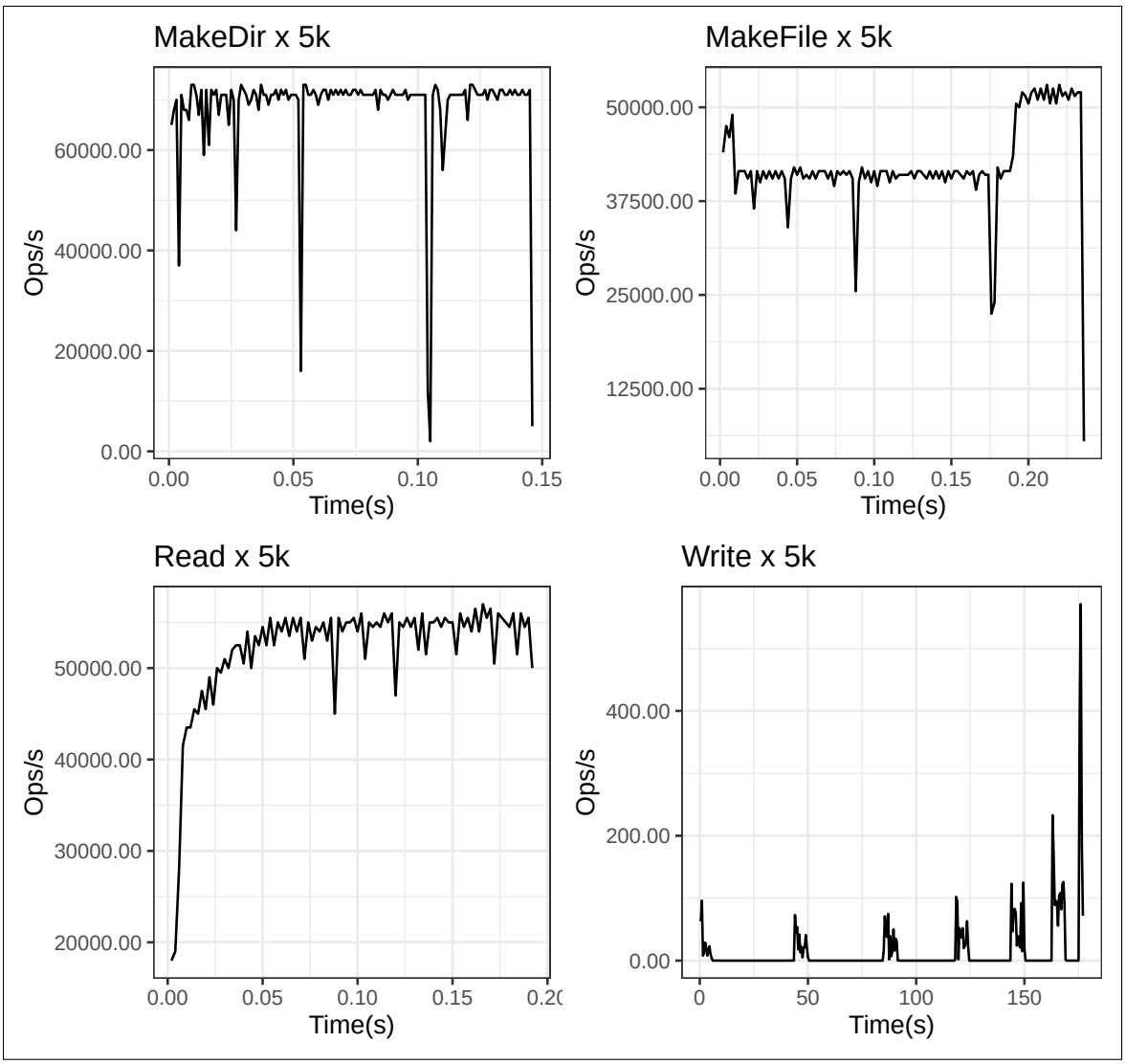
be done by `upss-fuse` as a global filesystem in comparison with S3FS and Perkeep (Figure 6.13a). In Figure 6.13b, we disabled caching and persisted the content just before the benchmark script is finished so that the content is ready to be read from the Amazon block store. Even without caching and having the content persisted to the Amazon block store, `upss-fuse` outperforms the other two filesystems by factors of 10-8000. These results show that the cryptographic foundation of UPSS provides, not just strong security properties, but a foundation for aggressive caching that would be unsafe in a system that does not use cryptographic naming.

6.3.5 UVC: UPSS Version Control System

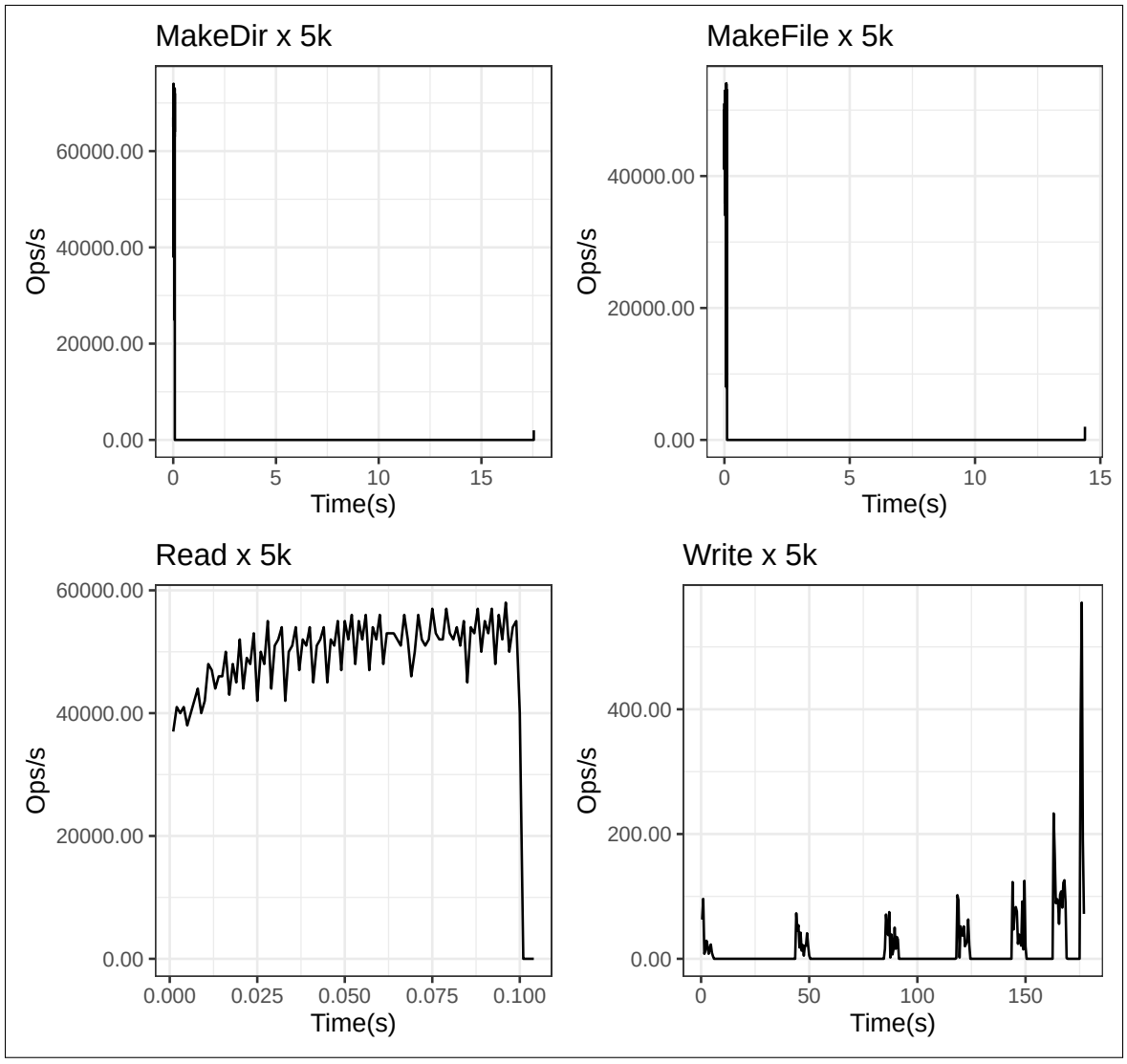
Supporting data sharing on filesystems or storages, is a broad field of study, which makes the system involved with challenges. Revision control, communication protocols, policies on user access rights, employing the underlying filesystem, are a few of involved, argumentative topics in this chapter of studies. The most motivating point of designing UVC: UPSS Version Control System is that even distributed revision control systems depend on trusted storage systems, with versioned content stored in plaintext and access control provided by third-party providers. The key point is that using UPSS, we can use an untrusted backend for bulk storage, maintaining encrypted data, along with a secure authentication and authorization mechanism.

Version control systems are a group of data sharing systems to facilitate users' contributions and collaborations on shared units of data, known as *repositories*. Git and Apache Subversion (SVN) are examples of today's widely-used version control systems. The mechanism of sharing *revisions* of data differs in each system. The most straightforward case is a centralized approach, in which data is stored on a shared remote server, and changes will be synced to it. The server manages requests, revisions, user accesses, and data storages to use. UPSS's design provides a mechanism to support revision control. on UPSS's filesystem objects, described

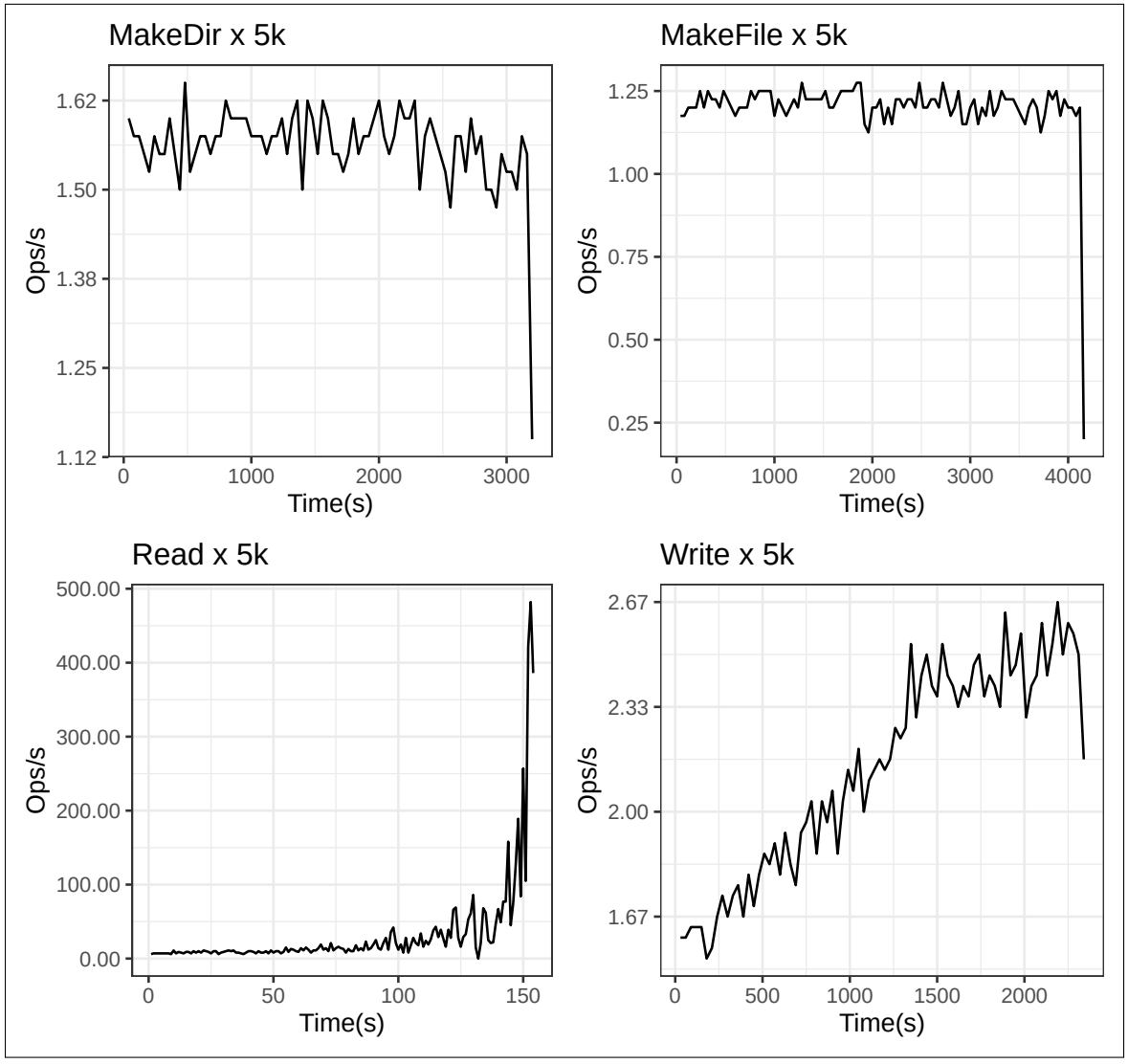
Figure 6.13: Operations that can be performed per second by `upss-fuse`-global, S3FS and Perkeep for four microbenchmarks. The behaviour of the filesystems are reported for 5k operations during time. In Figure 6.13a, the sync interval is 15000, means that the number of objects that are kept in memory before being persisted is 15000. The objects are synced to caching block stores and are journaled to an on-disk file to be synced to an Amazon block store in the background. In Figure 6.13b, the caching is disabled and the sync interval is set to 4999 to sync everything to amazon block store before the benchmark is finished.



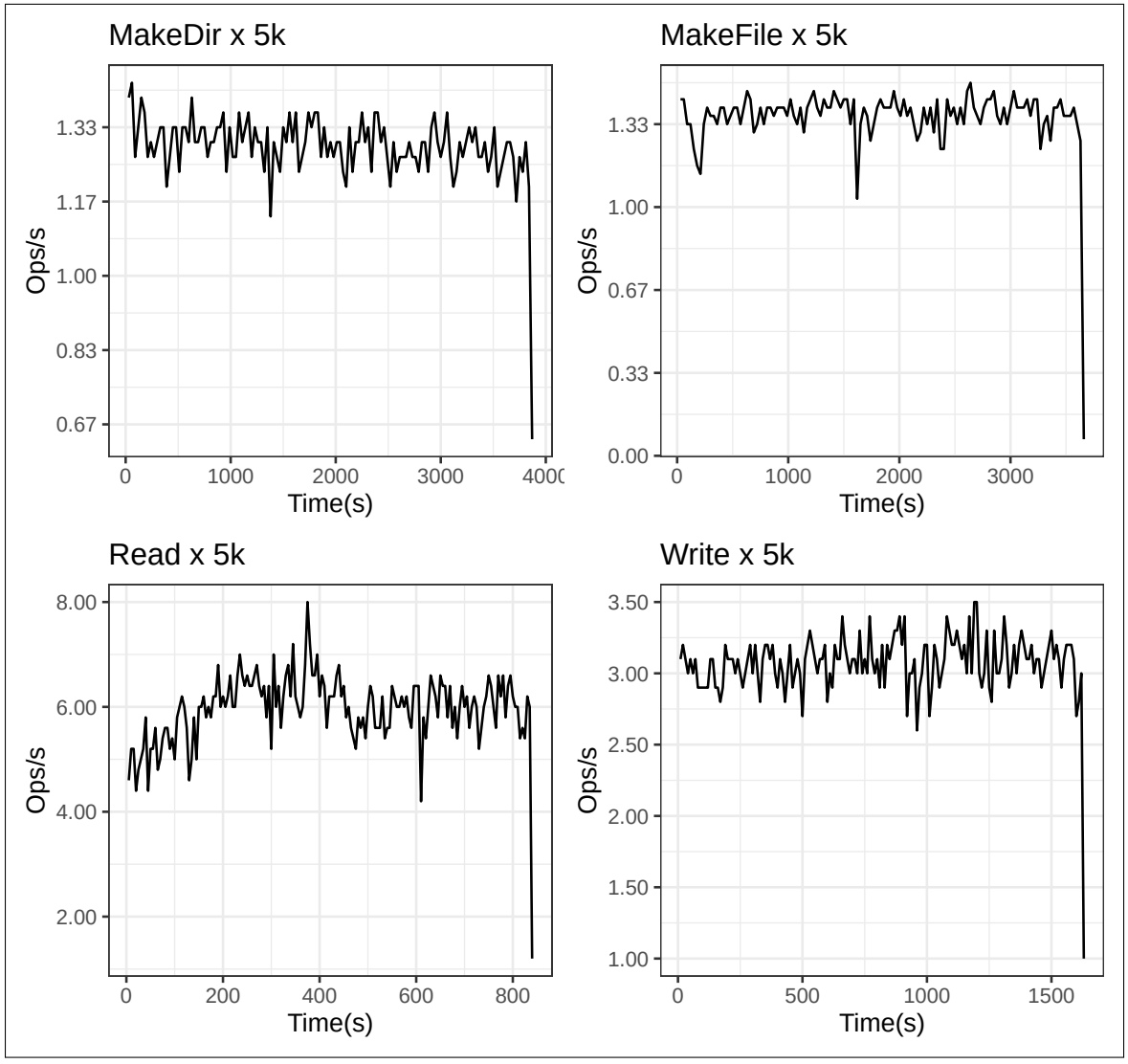
(a) upss-fuse-global



(b) upss-fuse-global-no-cache-full-sync


MakeDir x 5k
Ops/s
1.62
1.50
1.38
1.25
1.12
0
1000
2000
3000
Time(s)
MakeFile x 5k
Ops/s
1.25
1.00
0.75
0.50
0.25
0
1000
2000
3000
4000
Time(s)
Read x 5k
Ops/s
500.00
400.00
300.00
200.00
100.00
0.00
0
50
100
150
Time(s)
Write x 5k
Ops/s
2.67
2.33
2.00
1.67
0
500
1000
1500
2000
Time(s)

(c) Perkeep


MakeDir x 5k
MakeFile x 5k
Read x 5k
Write x 5k
Ops/s
Time(s)
1.33
1.17
1.00
0.83
0.67
0.33
0.00
8.00
6.00
4.00
2.00
3.50
3.00
2.50
1.50
0
200
400
500
600
800
1000
1500
2000
3000
4000

(d) S3FS

in Section 6.2.3. This makes us closer to the design of a distributed sharing system relying on our cryptographic filesystem. As the first step, we have designed and implemented a prototype of a centralized version control system, called UVC: UPSS Version Control System's prototype, UVC.

Various version control systems behave differently in detail, but they all support some initial user stories such as: cloning shared data on a local machine, pushing modifications to the sharing server, demonstrating repository statuses in terms of sequences of modifications, etc. UVC, is a client-server program supporting the mentioned initial user stories as the minimums of functionalities. Our version control system constitutes of three principle sub-modules: the *remote block store*, the *version control server*, and the *version control client*. The remote block store is a client-server application that can be integrated with `upss-fuse`'s API as a `BlockStore`, described in Section 6.2.1, to replace the local block store. In the following sections, we describe how our version control server and client, cooperate with this application to manage our data sharing scenarios.

6.3.5.1 Version Control Server

UVC's server is responsible for creating shared repositories, keeping track of changes on them, validating and handling clients requests. A repository is a *directory*, including a various number of files. As it is stated before, UPSS is a content-addressable filesystem in which each object, either a file or directory, is addressable through a *global name* called a block pointer. Any change on an UPSS's objects, results in a change in the content of the object's data or its metadata, and consequently, a new block pointer will be generated. We have benefited from this feature, as each block pointer is a global identifier for a version of the shared directory. Hence, any change on the shared directory, like adding or removing files, or altering their contents and names, results in a new revision of the shared directory. The version control server tracks and manages these revision changes to provide

users with collaboration on the shared repository. Each client's request contains a new block pointer for the modified directory, along with its previous revision's name, which is the block pointer of the previously synced revision, and finally, required timestamps, and information about the user. The server investigates the request considering the current status of the corresponding chain of revisions and decides how to respond. It should be mentioned that authorization and authentication management are not supported in the current prototype and are considered as our future developments to be added as a responsibility of UVC's server. So, UVC is designed to accomplish management policies and data transmission separately.

6.3.5.2 Version Control Client

UVC's client is responsible for creating requests, communicating with the version control server, keeping track of changes on the local machine, and retrieving modified data from, or writing it on, the remote block store. To clarify these processes and the communication between the server and client, consider a *push* procedure: Assume that the client has modified a file under a directory, which is cloned before, and then runs the *add* command to add to the remote block store. To make the stated procedure more sensible, it can be counted as Git's behavior during add and commit procedures together. However, the significant difference with Git's logic is that the modified data is synced to the remote block store before starting pushing procedure. As the confidentiality of data is already guaranteed with the approach of storing data encrypted, it is always safe to push it to the server speculatively. So, the cost of pushing changes is even less than some of the existing systems, such as Git's protocol in which the data is not transmitted before push request. Making data transmission and policy management elements separated in our prototype, makes our version control system flexible to be expanded in the future with new fine-grained access right policies, without affecting data transmission. When the modified version of data is added to the remote block store, the block pointer of

the new directory is sent to the server, wrapped in a new request associated with other information. On the other side, the server validates a new push request and investigate the revision sent, and then updates the intended chain of revisions and sends a response packet back to the client. Figure 6.14 shows this procedure.

When the new revision of the shared directory is added to the server and information is updated, other clients can *pull* it. A pull request will be sent to the server in which the last synced revision is included. The server validates the request, and returns a list of further revisions from the sent revision on, to the client. The client application is responsible for fetching the last version from the remote block store. Figure 6.15 demonstrates this process.

UVC is at the initial phases of development towards a sharing system built on a cryptographic filesystem. The system is not parallelized or optimized yet. However, we have evaluated this system in terms of execution time, comparing with Git, another version control system that is used globally today. Section 6.3.5.3 describes some of our observations.

6.3.5.3 Version Control System Evaluations

As it is mentioned in Section 6.3.5, to develop UVC, we were inspired by Git, which is a widely-used revision control system. Our version control's add command is equivalent to Git's add and commit commands together. Our push command syncs changes to the server, like Git's push command, without transferring data during the push request. At the current state, our clone and pull commands almost work in the same way, and both of them are fetching the new revision of the shared directory from the remote block store. We have examined three of our most essential procedures, including add, push, and clone commands, as our system's initial performance. Although UVC is an initial prototype, the main goal of our evaluations was to find improvement opportunities.

For the evaluations, we started with an empty shared directory created by the

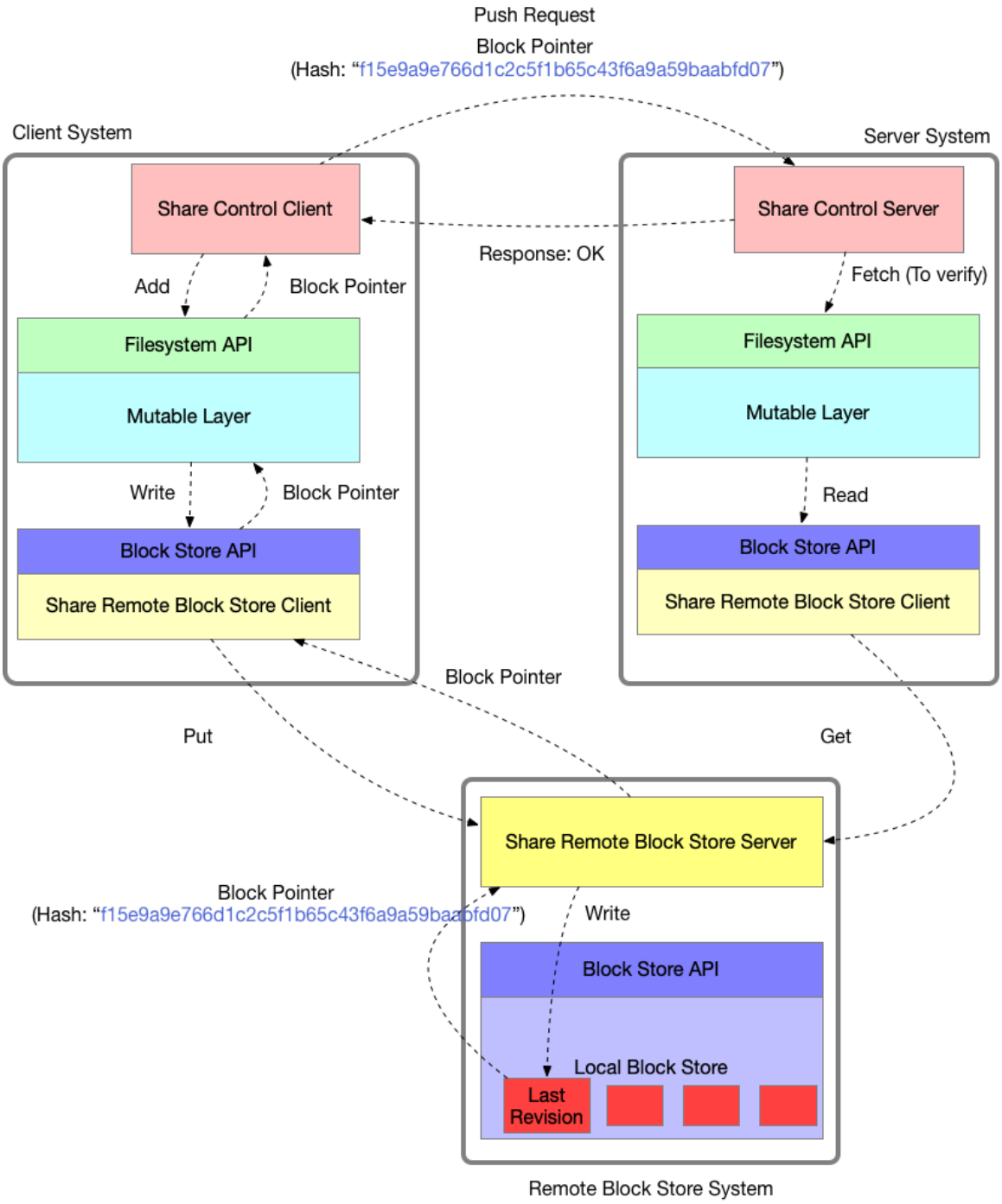


Figure 6.14: The version control client program writes changes as a new revision of the shared directory on the remote block store, and then sends a push request to the version control server. Then the server validates the client's request and will update the corresponding revision chain's information.

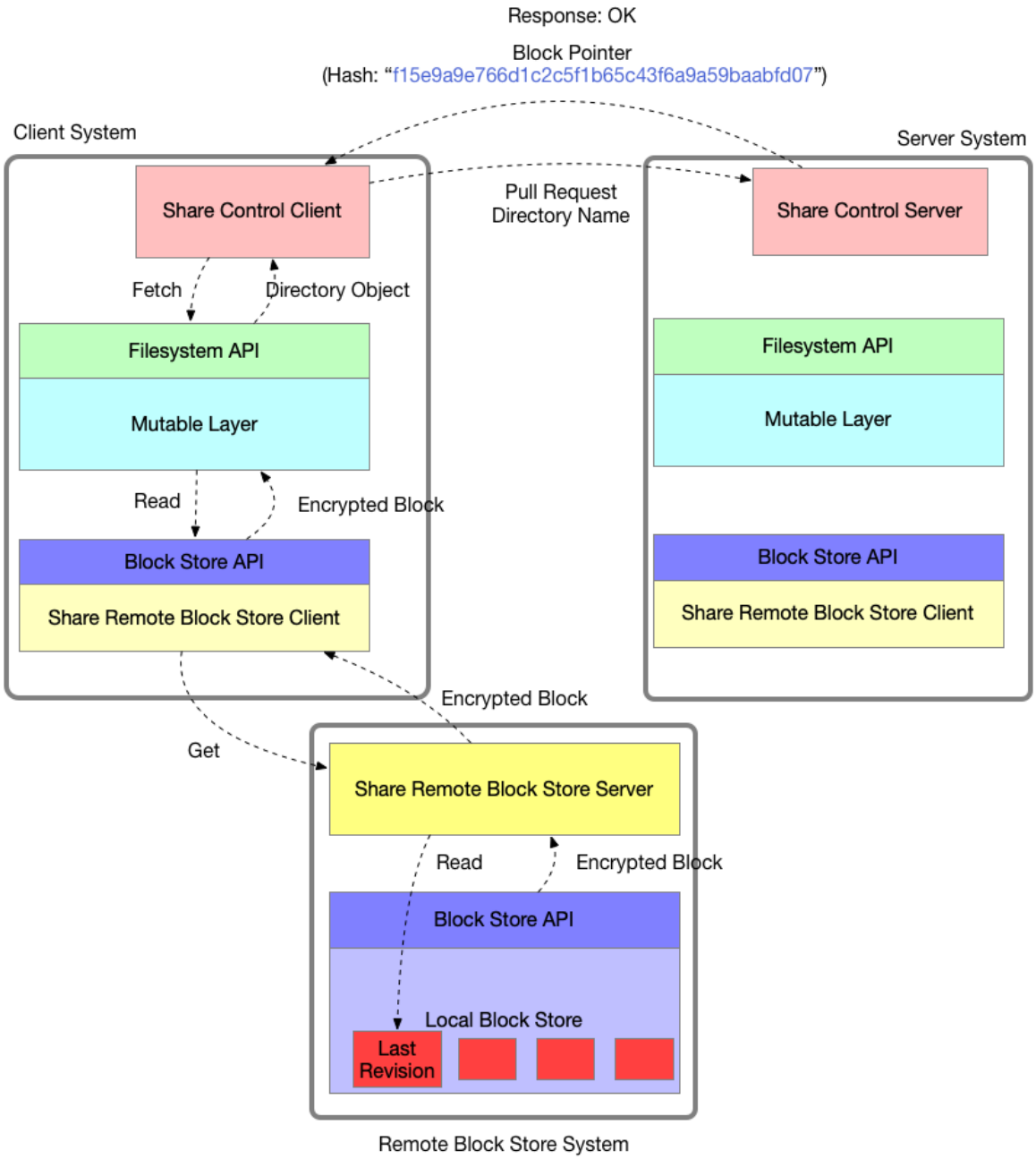


Figure 6.15: The version control server validates the client's request and responds with the block pointer of the head's revision. Then, the cryptographic hash included in the head revision's block pointer will be used to retrieve data from the remote block store.

version control server. Also, we made an empty repository on Git. In both cases, the remote store was set on a local machine, to remove the cost of networking connections. We have used the first 1024 files of the Linux kernel's source code as a pool of files. The size of the whole of the set was 18 MB. To evaluate the effect of an increasing number of files, we began with one file to be added and pushed to the server. Then we cloned the new revision of the shared directory. We repeated this scenario, increasing the number of files, each time making the previous number to the power of 2, up to 1024 files.

Figure 6.16 demonstrates the total time spent on push procedure for a different number of files, comparing Git and UVC. By total push procedure, we mean all commands run to sync new revisions to the server in both systems, which enables other clients to pull updates. This includes add, commit, and push commands for Git, and also, add and push commands for UVC. The preliminary result shows that the execution time for add and push procedures are more than Git, as expected, but we see the resulted latency is tolerable and growing linearly with a slow slope by increasing the number of files. This performance is obtained without parallelism or any performance improvements on UVC, as the initial evaluation from our system.

Beyond the comparisons between our system and Git, we tried to find the reason for the latency resulted, investigating our add command and its internal phases separately. We have found that the process of adding changes to the server, which includes data writing to the remote block store, is the most time-consuming phase of our push procedure. However, we found this delay tolerable for the current state, especially as we have not made the system multi-threaded. The cost of in-memory processes in UVC is demonstrated in Figure 6.16 as well. By *in-memory* data process, we mean processing modifications happened to the local revision, and preparing new revision, before writing changes on the remote block store. This procedure is carrying out by the add command, which writes new revisions on the remote block store.

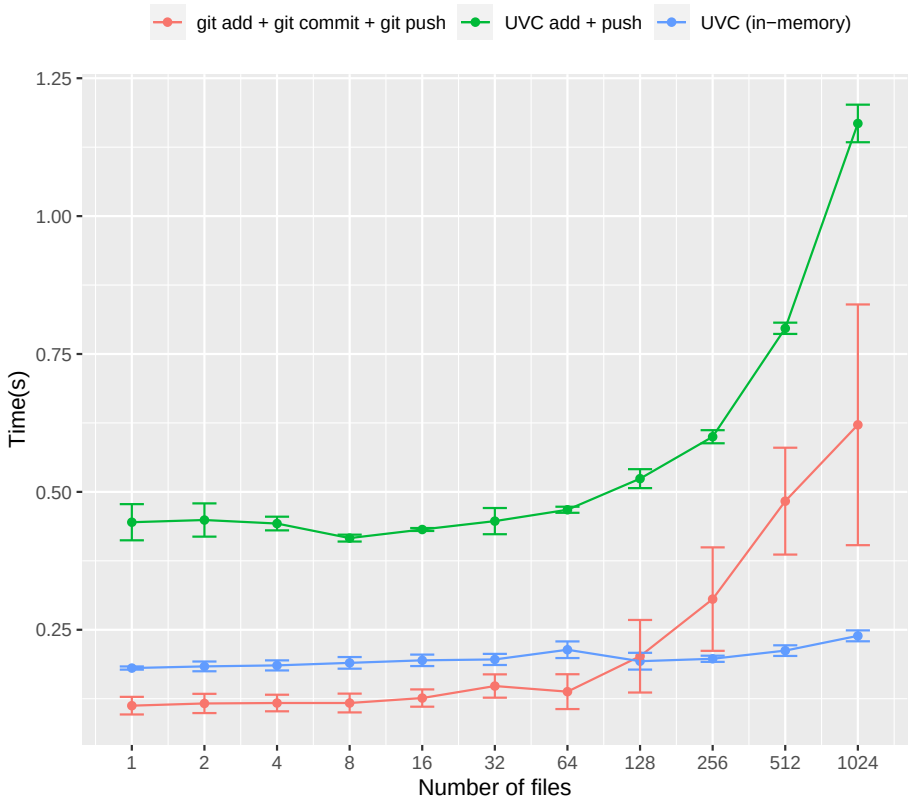


Figure 6.16: Time spent on total push procedure in UVC and Git. In UVC, writing changes to the remote block stores are included through add command. In-memory data processing shows the time spent on UVC execution before writing changes to the remote block store. Results are showing the average of 5 runs, along with corresponding standard deviations.

Also, a similar comparison has been made running the clone command on both systems, increasing the number of files. As Figure 6.17 shows, the result is similar to what we obtained from previous scenarios, as expected, which originates from the communication between principals and UPSS's filesystem read and write requests. This figure also shows how the increasing number of files impacts our client-side application performance, which is responsible for fetching the new revision from the remote block store.

To summarize our evaluation, we inferred UVC needs improvement on two

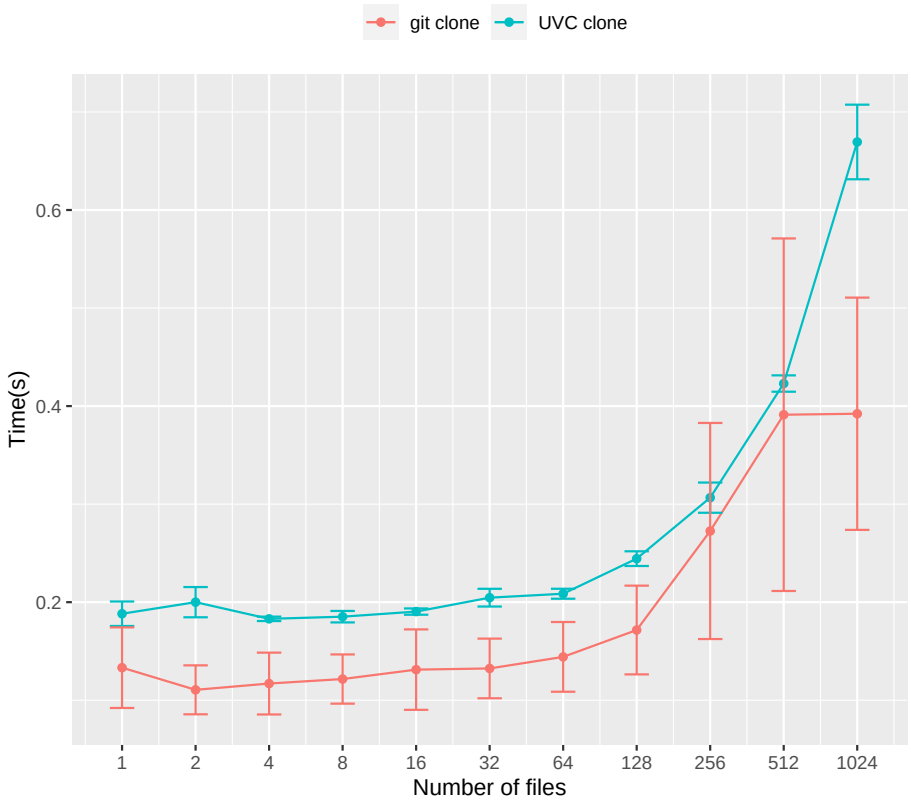


Figure 6.17: Cloning running time in UVC and Git. New version of a shared directory is retrieved from the remote block store and version control server. Results are showing the average of 5 runs, along with corresponding standard deviations.

separate layers. The first one is the client-server protocol used by the remote block store and the other one in UPSS's procedure of persisting data. As we stated before, currently, version control server and remote block store server, handle requests serially, and there is no optimized buffer or cache block store used in UVC. Addressing these issues will lead us to achieve considerable performance improvements.

6.4 Related work

The existing filesystems try to introduce a practical solution for data integrity and availability but mostly ignore confidentiality and privacy. Moreover, they do not provide a sharing mechanism that can serve users' needs in multi-user environments, such as partial sharing and subsetting and automatic conflict resolving all together.

CFS [DKK+01], Coda [SKK+90], and Ivy [MMGC02] filesystems provide availability for user data stored on dedicated servers in a distributed environment along with other features such as disconnected operations, content-addressable storage and log-structured systems. Coda introduced an automatic conflict resolution that cannot detect some classes of conflicts, such as update/update (two different updates on same objects), remove/update (removing an object from one replica and updating it in another) and name/name (creating new objects with identical names in one directory on different replicas). Ivy also introduced a conflict detector called *lc* that notifies users about the conflicts.

The more the cloud storage became popular, the more efforts have been done to introduce functional filesystems in the cloud settings, such as NCryptFS [WMZ03], EncFS [LFS16a] and CryFS [MRAMQ17]. NCryptFS and EncFS are cryptographic filesystems, which protect the content by encrypting the files, but leave the filesystem metadata, such as the directory structure unprotected. CryFS solves this problem by splitting all filesystem data into fixed-size blocks and encrypting each block individually, but with one key for all encryption. The creators of EncFS expanded their work to make it a multi-user filesystem by applying Unix local permissions to the encrypted files before being stored on remote servers [LFS16b]. However, the both approaches are not practical solutions for multi-user environments with non-local users, that need a secure, fine-grained and flexible sharing approach.

Ori [MBHM13], IPFS [Ben14] and Perkeep [LN18] (formerly known as Camlistore) try to connect different computing devices with the same filesystem and

enable users to access their files everywhere. The authors of IPFS synthesize the key ideas behind systems such as DHTs [SMK+01], BitTorrent [Coh03], Git [LM12], and self-certifying pathnames [MK98] to create a peer-to-peer version-controlled filesystem. Both Ori and IPFS reduce the data inconsistency problem to a version control problem by storing new versions of files upon their modifications and the former handles the updates with the CoW technique, which was introduced in ZFS [BAH+03]. Synchronization, failures handling, data recovery and sharing mechanism, or grafting, are the key features of the Ori filesystem. Perkeep is made on a set of open source protocols trying to create a unified storage for keeping user data from different sources such as their Twitter account or the their local hard drive. Similar to `upss-fuse`, Perkeep can be mounted backed by a memory store, a local store or a cloud account. However, none of Ori, IPFS and Perkeep provide a mechanism for sharing or subsetting file and directory hierarchies for users with different levels of access.

Tahoe [WOW08] is another cryptographic filesystem with the main goal of storing user data on untrusted storage servers. Like UPSS, Tahoe stores the content encrypted in Merkle DAGs and provides access control by cryptographic capabilities. Tahoe supports both immutable and mutable files in their design and for the later case, it signs and verifies the files with a public/private key pair. The public key and the verification key is stored as plaintext along with the files. Having mutability in the file level can cause inconsistency if Tahoe is used in a collaborative environment. The other difference between Tahoe and UPSS is that Tahoe encrypts a file with one symmetric key and erasure code the ciphertext, using Reed-Solomon codes [Riz97] into N shares to be written to N servers. However, Tahoe is designed for file sharing and archival storage; using Tahoe with POSIX-like read-write workloads can cause "its performance to crawl to a halt" [tah20]. This approach cannot provide partial sharing/subsetting and redaction, which is enabled in UPSS by having cryptographic capabilities per each encrypted block.

6.5 Future work

The very first thing that we need to do in the future is protecting the root block pointer, that is the only information needed by `upss-fuse` to retrieve the root and the entire directory tree. Currently, upon persisting the root directory, its block pointer is stored in a file as plain-text. Instead, the block pointer can be secured by user-defined credentials and be stored in a file or even in UPSS's block store.

6.5.1 FFI

Although our prototype currently requires Rust linkage and calling conventions, we are investigating the use of foreign function interfaces (FFI) to expose UPSS to code written in other programming languages. We have been exploring Rust's excellent support for C FFI, including the explicit transfer of memory ownership, to interface C code with Rust APIs. On top of this platform, we are exploring the use of CPython extensions to further expose Rust APIs to Python modules. The use of custom PyObject destructors allows the reference-counted garbage collection model of Python to be combined with the more explicit memory models of C and Rust; we have begun to explore this interface with prototype Rust code but have not yet built a complete FFI for UPSS. Also, we have begun exploring the use of Rust/WASM [was17] to provide UPSS functionality within client-side JavaScript code, e.g., in a Web browser session. In the future, this will allow us to build UPSS-based Web experiences in which user data is only decrypted within the user's browser and all communication with a remote HTTP-based block store is in terms of encrypted blocks.

6.5.2 Expansion of Version Control System

As described in Section 6.3.5, our version control system does not currently support authentication and authorization. Our next step to expand this system is to add

mechanisms that enable users to define fine-grained and flexible access rights. We will explore new types of access control and partial sharing, benefiting from our cryptographic content-addressed filesystem. Having the `MetaVersion` data structure that stores the block pointers of a file enables us to partially share a file with another user by generating a new `MetaVersion` including the block pointer (block name and its decryption key) of the blocks that we aim to share, the block name without the decryption key of the excluded blocks, and the block name of the main `MetaVersion` of the file. In this way, we can have a redacted version of the main file along with its linkage. Having all the block names in the new `MetaVersion` enables UPSS to provide the data integrity by comparing the cryptographic hashes of the block names included in the new and old `MetaVersions`.

We can expand our version control system to be a configurable centralized data sharing system in the future, towards which UVC is our first step.

6.5.3 Parallelism

Based on our evaluation results, we have observed latency that is mainly due to the single-threaded implementation of the UPSS core. Currently, blocks are read and written serially, leading to delays in higher-level operations such as directory entry iteration. Adding parallelism to UPSS and UVC, and availing of the new Rust Futures API is a way of decreasing such latency.

6.5.4 Structured files

Files and directories are meaningful in classical filesystems and are interpreted as unstructured byte arrays. However, the internal DAG structure of UPSS blocks should allow UPSS to naturally define structured files and directories. With structured files, we can guarantee data consistency for structured file content such as unserialized application data structures as well as the directory tree of the filesystem [ANMU12, TSR15]. In this way, multi-user data on different replicas are guaranteed

to be in the same state, without data loss and without requiring users to resolve conflicts manually. Automatic filesystem-level conflict resolution has been explored before in filesystems such as Coda [SKK+90], but UPSS' internal block structure naturally lends itself to a reinvigorated exploration of these ideas, defining files as Conflict-free Replicated Data Types (CRDT) [KB17, SPBZ11b, SPBZ11a]. We have started investigating CRDT structures that can be integrated into UPSS' objects and we will introduce a conflict-free version of UPSS in the future.

6.5.5 Hiding access patterns

Currently, UPSS can provide confidentiality, integrity and availability of data in the block stores. However, some information may be learned from the access patterns to the block stores. For example, an adversary would likely to be able to learn the mapping between the client's IP address and the accessed file by doing some traffic analysis. UPSS is not secured against this threat in its current implementation. We have started studying this problem and we believe that the client can be protected against this threat by using a system such as Tor [DMS04]. The full implementation is our other future work.

6.6 Conclusion

UPSS: the user-centric private sharing system is a decentralized cryptographic filesystem that provides strong confidentiality and integrity properties while relying only on untrusted backend storage. This filesystem is accessible as an embeddable Rust library, a FUSE filesystem (with local or remote storage) or as the platform for a novel confidential version control system. The performance of *UPSS* exceeds that of comparable cryptographic filesystems and is within an order of magnitude of the performance of a mature copy-on-write local filesystem ZFS. When using remote storage, UPSS achieves almost the same results for write and cached read operations

and better results for creating files and directories, compared to a mature network filesystem NFS. UPSS's performance exceeds that of an optimized global filesystem Perkeep.

UPSS uses untrusted storage backends in which data is always encrypted at rest as a *sea of blocks*: no file or directory structure can be discerned directly from the contents of an encrypted block store. However, those information can be learnt indirectly by traffic analysis, as we stated in Section 6.5.5. Along with the confidentiality supported by encrypted blocks, UPSS is a content-addressable filesystem, using cryptographic names that contain cryptographic hashes of blocks' content, as global identifiers. This mechanism allows even sensitive user content to be stored in commodity cloud storage without relying on centralized access control. *Convergent encryption* can — at the discretion of individual users — be used to enable de-duplication across mutually-distrustful users [PMÖL13, KBR13]. Cryptographically-named blocks can be combined into immutable DAGs of files and directory hierarchies, with cryptographic capabilities, used to authorize access to arbitrary quantities of data. UPSS provides a conventional filesystem API using *copy-on-write* operations around immutable DAGs; this API is accessible directly as an embedded library or proxied via a FUSE interface.

Using POSIX filesystem interfaces, we have compared the performance of UPSS/FUSE to other cryptographic filesystems, the mature copy-on-write filesystem ZFS, the mature NFS filesystem, and Google's global filesystem Perkeep. Using modern cryptographic and copy-on-write techniques, UPSS demonstrates that it is possible to achieve both strong security properties *and* high performance, even with entirely untrusted storage. Furthermore, we have demonstrated the utility of UPSS as a foundation for a novel distributed revision control system *UVC: UPSS Version Control System* with inherent confidentiality properties that are not known in contemporary revision control systems. UVC is our first step towards a secure capability-based data sharing system to provide a fine-grained and flexible user

authority management.

Whether used as a local, network or global filesystem, or the underlying filesystem for a private revision control system, UPSS: the user-centric private sharing system enables arbitrary quantities of data to be stored with strong confidentiality and integrity properties on untrusted storage backends. UPSS's performance is comparable to — or, in some cases, superior to — mature, heavily-optimized filesystems, proving that high standards of security in application or user storage need not prevent practical performance. Wide adoption of UPSS and its techniques will lay the foundation for future transformations in privacy and integrity for applications as diverse as social networking and medical data storage, providing better opportunities for users — not system administrators — to take control of their data.

Bibliography

[ANMU12] Mehdi Ahmed-Nacer, Stéphane Martin, and Pascal Urso. File system on crdt. *arXiv preprint arXiv:1207.5990*, 2012.

[ASA17] Ashish Agarwala, Priyanka Singh, and Pradeep K Atrey. Dice: A dual integrity convergent encryption protocol for client side secure data deduplication. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2176–2181. IEEE, 2017.

[AWS20] Inc. Amazon Web Services. Amazon simple storage service. "https://aws.amazon.com/s3/", (Accessed on February 28, 2020).

[BAH+03] Jeff Bonwick, Matt Ahrens, Val Henson, Mark Maybee, and Mark Shellenbaum. The zettabyte file system. In *Proc. of the 2nd Usenix Conference on File and Storage Technologies*, volume 215, 2003.

[Ben14] Juan Benet. IPFS-content addressed, versioned, P2P file system. *arXiv preprint arXiv:1407.3561*, 2014.

[BHK+91] Mary G Baker, John H Hartman, Michael D Kupfer, Ken W Shirriff, and John K Ousterhout. Measurements of a distributed file system. In *Proceedings of the thirteenth ACM symposium on Operating systems principles*, pages 198–212, 1991.

[BJA19] Arastoo Bozorgi, Mahya Soleimani Jadidi, and Jonathan Anderson. Challenges in designing a distributed cryptographic file system. In *Cambridge International Workshop on Security Protocols*, pages 177–192. Springer, 2019.

[Coh03] Bram Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72, 2003.

[DAB+02] John R Douceur, Atul Adya, William J Bolosky, P Simon, and Marvin Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Proceedings 22nd international conference on distributed computing systems*, pages 617–624. IEEE, 2002.

[DKK+01] Frank Dabek, M Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with cfs. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 202–215. ACM, 2001.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[DVH66] J. B. Dennis and E. C. Van Horn. Programming semantics for multiprogrammed computations. *Communications of the ACM*, 9(3):143–155, 1966.

[Dwo15] Morris J Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions. Technical report, 2015.

[fil16] Filebench - a model based filesystem workload generator. https://github.com/filebench/filebench, July 2016.

[fus19] Fuse (filesystem in userspace). https://github.com/libfuse/libfuse/releases/tag/fuse-3.9.0, Dec 2019.

[GNr20] Andrew Gaul, Takeshi Nakatani, and rrizun. S3fs: Fuse-based file system backed by amazon s3). https://github.com/s3fs-fuse/s3fs-fuse/releases, Feb 2020.

[KB17] Martin Kleppmann and Alastair R Beresford. A conflict-free replicated JSON datatype. *IEEE Transactions on Parallel and Distributed Systems*, 28(10):2733–2746, 2017.

[KBR13] Sriram Keelveedhi, Mihir Bellare, and Thomas Ristenpart. Dupless: server-aided encryption for deduplicated storage. In *Presented as part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13)*, pages 179–194, 2013.

[LCL+13] Jin Li, Xiaofeng Chen, Mingqiang Li, Jingwei Li, Patrick PC Lee, and Wenjing Lou. Secure deduplication with efficient and reliable convergent key management. *IEEE transactions on parallel and distributed systems*, 25(6):1615–1625, 2013.

[LFS16a] Dominik Leibenger, Jonas Fortmann, and Christoph Sorge. Encfs goes multi-user Adding access control to an encrypted file system. In *2016 IEEE Conference on Communications and Network Security (CNS)*, pages 525–533. IEEE, 2016.

[LFS16b] Dominik Leibenger, Jonas Fortmann, and Christoph Sorge. Encfs goes multi-user: Adding access control to an encrypted file system. In *2016 IEEE Conference on Communications and Network Security (CNS)*, pages 525–533. IEEE, 2016.

[LM12] Jon Loeliger and Matthew McCullough. *Version Control with Git: Powerful tools and techniques for collaborative software development*. " O'Reilly Media, Inc.", 2012.

[LN18] Paul Lindner and Wil Norris. Perkeep (née camlistore): your personal storage system for life. `https://github.com/perkeep/perkeep/releases`, May 2018.

[LZCZ86] Edward D Lazowska, John Zahorjan, David R Cheriton, and Willy Zwaenepoel. File access performance of diskless workstations. *ACM Transactions on Computer Systems (TOCS)*, 4(3):238–268, 1986.

[MBHM13] Ali José Mashtizadeh, Andrea Bittau, Yifeng Frank Huang, and David Mazieres. Replication, history, and grafting in the Ori file system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 151–166. ACM, 2013.

[MJLF84] Marshall K McKusick, William N Joy, Samuel J Leffler, and Robert S Fabry. A fast file system for unix. *ACM Transactions on Computer Systems (TOCS)*, 2(3):181–197, 1984.

[MK98] David Mazieres and M Frans Kaashoek. Escaping the evils of centralized control with self-certifying pathnames. In *Proceedings of the 8th ACM SIGOPS European workshop on Support for composing distributed applications*, pages 118–125. ACM, 1998.

[MMGC02] Athicha Muthitacharoen, Robert Morris, Thomer M Gil, and Benjie Chen. Ivy: A read/write peer-to-peer file system. *ACM SIGOPS Operating Systems Review*, 36(SI):31–44, 2002.

[MRAMQ17] Sebastian Messmer, Jochen Rill, Dirk Achenbach, and Jörn Müller-Quade. A novel cryptographic framework for cloud file systems and

cryfs, a provably-secure construction. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 409–429. Springer, 2017.

[PMÖL13] Pasquale Puzio, Refik Molva, Melek Önen, and Sergio Loureiro. Cloudedup: Secure deduplication with encrypted data for cloud storage. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, volume 1, pages 363–370. IEEE, 2013.

[RBM13] Ohad Rodeh, Josef Bacik, and Chris Mason. Btrfs: The linux b-tree filesystem. *ACM Transactions on Storage (TOS)*, 9(3):1–32, 2013.

[Riz97] Luigi Rizzo. On the feasibility of software fec. *Univ. di Pisa, Italy*, pages 1–16, 1997.

[RO92] Mendel Rosenblum and John K Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems (TOCS)*, 10(1):26–52, 1992.

[SCR+03] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. Rfc3530: Network file system (nfs) version 4 protocol, 2003.

[SKK+90] Mahadev Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on computers*, 39(4):447–459, 1990.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[SPBZ11a] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. *A comprehensive study of convergent and commutative replicated data types*. PhD thesis, Inria–Centre Paris-Rocquencourt; INRIA, 2011.

[SPBZ11b] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *Symposium on Self-Stabilizing Systems*, pages 386–400. Springer, 2011.

[tah20] Tahoe frequently asked questions. `https://tahoe-lafs.org/trac/tahoe-lafs/wiki/FAQ`, 2020.

[Tea20a] Rust Team. Rust programming language. `https://www.rust-lang.org/`, 2020.

[Tea20b] SSHFS Team. SSHFS (a network filesystem client to connect to ssh servers). `https://github.com/libfuse/sshfs/releases`, Jan 2020.

[TSR15] Vinh Tao, Marc Shapiro, and Vianney Rancurel. Merging semantics for conflict updates in geo-distributed file systems. In *Proceedings of the 8th ACM International Systems and Storage Conference*, page 10. ACM, 2015.

[VAM+19] Bharath Kumar Reddy Vangoor, Prafful Agarwal, Manu Mathew, Arun Ramachandran, Swaminathan Sivaraman, Vasily Tarasov, and Erez Zadok. Performance and resource utilization of fuse user-space file systems. *ACM Transactions on Storage (TOS)*, 15(2):1–49, 2019.

[was17] Webassembly specification. `https://webassembly.org/`, 2017.

[WMZ03] Charles P Wright, Michael C Martino, and Erez Zadok. Ncryptfs: A secure and convenient cryptographic file system. In *USENIX Annual Technical Conference, General Track*, pages 197–210, 2003.

[WOW08] Zooko Wilcox-O'Hearn and Brian Warner. Tahoe: the least-authority filesystem. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*, pages 21–26, 2008.

Chapter 7

Summary

Online Social Networks (OSNs) have a significant impact on our lives as they enable individuals to communicate with each other in a way that in real life may not be possible. They also serve as a medium for propagating ideas and thoughts and, in some cases, act as an advertising platform for product owners. Viral marketing is an example of this type, which is based on the word-of-mouth effect on OSNs. But, such networks do not always operate in users' interests, especially in cases that user data is critically important and needs to be kept confidential, one of the main requirements of a health care information system such as health care OSNs. However, by designing a system that can guarantee integrity, confidentiality and availability requirements while considering users – not system administrators – as real data owners, we can address the existing concerns of information systems while still benefiting from them.

For solving the influence maximization problem in OSNs, that was the start point of this thesis, a graph structure is extracted from the data that is stored and controlled centrally by the OSN providers. Such data can also be used for other purposes, such as advertisement, without users' explicit consent. This lack of control over data by the users and violating their privacy encouraged me to concluded my thesis by designing and implementing a privacy-preserving optionally-distributed

cryptographic filesystem. The filesystem can be used to build applications that enable rich and collaborative sharing without suffering from the cost of out-of-control sharing on untrustworthy systems.

In Chapter 2, I proposed an efficient algorithm, called *INCIM* (Influential Nodes using Community structure for solving Influence Maximization problem) for solving the influence maximization problem under Linear Threshold model, by using the community structure of OSN graphs. The spread value of each node is calculated as a combination of its local and global spread values. The local spread value is calculated per node inside the community it belongs to. In this way, I limited the search space to communities that are sub-graphs of the main graph that leads to better running time. Also, I considered the effectiveness of the communities in information spread, which was ignored by other approaches. A new graph of communities was constructed and the global spread value was calculated for each community. INCIM finds the influential nodes in a reasonable running time even for large networks. Starting the information spread from the identified nodes led to higher coverage over the whole network in comparison with the state-of-the-art approaches.

After doing extensive studies on the influence maximization problem, I started thinking about realistic scenarios that are happening in our daily life and were missed in the studies. The influence spread adoption is affected by the decisions made by others and individuals are likely to be mindful of the preferences of others. Therefore in Chapter 3, I proposed a new propagation model called *DCM* (Decidable Competitive Model) for competitive influence maximization problem, which is a extension of the Linear Threshold model. In competitive influence maximization, two adversaries are competing to gain more influence spread by choosing the minimum number of influential nodes not selected by the other. DCM enables the nodes to think about the incoming influence from the adversaries and after d timesteps, decide to adopt one of the influence spreads, which is accepted

by the majority of its neighbors. Then I presented *CI2* (Competitive Influence Improvement) algorithm for solving competitive influence improvement under the DCM model and proved its *NP*-hardness. The results of the experiments on both synthetic and real-world datasets show the applicability of the CI2 algorithm.

The other primary path that I took in this thesis was thinking about designing and implementing a secure system that can fulfill users' needs and addresses their privacy-related concerns. Such a system can be a potential replacement for current online social networks. I started by investigating the privacy-preserving approaches introduced for OSNs, that can be categorized based on the OSN architectures: centralized, decentralized or peer-to-peer, and hybrid. The existing approaches for each category are reviewed in Chapter 4. My investigations show that none of the existing approaches can address privacy, availability, and connectivity problems for OSNs, but decentralization or hybridization that also includes elements of decentralization, is the right direction toward further studies.

I also studied four motivational use cases with different and, in some cases, contradictory requirements: online social networks, censorship resistance systems, document redaction systems and health care information systems (Chapter 5). These use cases have overlapping needs for strong confidentiality, integrity, user control, reliability and performance properties. Moreover, they need a mechanism for sharing information securely and selectively without having complete trust in central servers. The result was a prototype filesystem called *UPSS: the user-centric private sharing system* in which the data storage plane is separated from the control plane. Storing data as encrypted fixed-size blocks in a content-addressable store makes UPSS functional on untrusted storage providers. Mapping the file and directory structure to immutable Merkle DAGs and convergent encryption preserves privacy and provides global deduplication.

I went beyond the ideas and implemented a functional filesystem, where its main idea was seeded in Chapter 5. The implementation details of the filesystem are

presented in Chapter 6. UPSS can be used as a traditional filesystem in userspace (FUSE), can be interacted using the provided public API by other applications, or as a platform for version control systems with fine-grained access controls. I backed UPSS with local, remote and global block stores and the extensive evaluations show that it is comparable and in some cases, superior to mature filesystems such as copy-on-write ZFS filesystem, mature NFS network filesystem and Google's global Perkeep filesystem. The distributed revision control system, *UVC: UPSS Version Control System*, which is build on top of UPSS, is the first step towards a secure capability-based data sharing system.

7.1 Future work

The `upss-fuse` can be the backbone of other systems. However, implementing such systems is beyond the time frame of this thesis and needs the effort of a team of expert people. Examples of such systems that can be built on top of `upss-fuse` are:

A secure and private online social network We can build a private Online Social Network (OSN) using UPSS as a library in which user data can be stored on local or remote block stores (see Sections 6.2.1 and 6.3.3.1), where the content is protected against adversaries, but the access patterns may be learned by traffic analysis (see Section 6.5.5). Content sharing can be supported by UVC (Section 6.3.5). Having local block stores, we can build a censor-resistant network in a peer-to-peer manner. However, there are some challenges in designing and implementing such systems that are identified and studied in Chapter 4. Also, another question is that how can we allow users to voluntarily participate in aggregation when it benefits them in such a private network for research and business purposes? Such data structures are required for solving graph-based problems such as influence maximization (see Chapters 2 and 3).

Conflict resolving in filesystem level Having a distributed read-write filesystem that supports flexible sharing (Section 6.3.5) needs a mechanism for resolving possible conflicts in both filesystem directory tree and file content. This is possible by adding data structures to the structured files in UPSS (see Section 6.5.4), but we have not integrated such structures yet.

A storage manager system with confidentiality-preserved auditing In some systems like health care information systems, it is crucial to detect the malicious behavior of authorized users. The systems that are responsible for detecting such users rely on the audited accesses of their users to the resources. Such audits are fed into the analyzer applications to detect anomalies. However, the audited logs should not reveal any information about the stored content. UPSS can provide such audits. UPSS can keep the root block name without its decryption key (see Section 6.2.1) of the accessed blocks on a block store. Those block names are meaningless outside of UPSS itself. Also, it is possible to define a list of permitted blocks that a user can access and ask UPSS to audit the accesses other than those blocks. Currently, such auditing is not implemented on UPSS core or as an auditing application that uses UPSS.