PhD Dissertation

International Doctorate School in Information and Communication Technologies

DISI - University of Trento

On Social Overlays and Their Application to Decentralized Online Social Networks

Giuliano Mega

Advisor:

Prof. Gian Pietro Picco

Università degli Studi di Trento

Co-Advisor:

Prof. Alberto Montresor

Università degli Studi di Trento

April 2013

To my family

À minha família

The following document, written under the supervision of professors **Gian Pietro Picco** and **Alberto Montresor**, was reviewed by:

Prof. Pietro Michiardi	Institut Eurecom, France
Prof. Mirco Musolesi	University of Birmingham, United Kingdom
Prof. Maarten van Steen	Vrije Universiteit Amsterdam, The Netherlands

Abstract

Over the last decade, Online Social Networks (OSNs) have attracted hundreds of millions of users worldwide, establishing themselves as one of most successful communication tools to date. Yet, the business model adopted by current centralized approaches makes them inherently prone to privacy issues and hostile to openness, as service providers rely on the commercial exploitation of their userbases' private data as their means of survival. We believe, as others do, that decentralization could represent a solution to this fundamental problem.

In this work, we propose a novel P2P approach to decentralized OSNs in which peers are organized as a *social overlay* (SO): an overlay network that effectively *mirrors* an underlying social network by constraining communication to pairs of peers whose owners are friends. SOs are special in two ways. First, by embodying friendship in their links, SOs can help us either solve or mitigate fundamental trust-related issues that arise in P2P systems. Second, SOs exhibit an inherent compatibility towards OSNs, a result of the former being shaped after human communication, and the latter being human communication tools. These give raise to an inherent potential for synergy, which we propose to reap by means of a simple approach that provides a key functionality of modern OSNs: profile-based communication.

In our approach, nodes cache the profile pages of their friends locally, and updates get proactively disseminated only by trusted nodes, over a user's *ego network*: the subgraphs of social networks composed by a user, her friends, and the connections among them. The contributions of this thesis then emerge as we tackle this seemingly simple problem of *update dissemination over ego networks* and, along the way, uncover issues that lead us to progressively deeper problems and understanding, and, ultimately, to effective solutions.

In the first part of this thesis, we explore the use of push gossip protocols as the means to achieve efficient dissemination of updates over ego networks. We show that mainstream gossip protocols cannot be applied in this context, due to the largely non-uniform structure of ego networks. By taking these structural properties into account, we develop a novel gossip protocol that is able to adapt to, and leverage this non-uniformity, providing efficient and timely dissemination of updates.

The study of these dissemination protocols under peer churn leads us to uncover the second problem we tackle in this thesis – namely, the network-induced *communication delays* that emerge from the interaction of the social graph with the underlying peer dynamics. By means of a small-scale simulation study, we find that not only these delays can be rather extreme, but that they matter more than the underlying dissemination protocol

on the long run. While this realization is in itself a contribution, we also find that evaluating the problem in more depth, as well as identifying opportunities for improvement, cannot be done by simulations alone. This is due to three factors: *i)* the size of the target networks under study, *ii)* the large parameter space inherent to availability modelling, and *iii)* the large number of repetitions required for obtaining good quality estimators. Put together, these translate into prohibitive costs. We therefore propose a novel hybrid analytical/simulation framework that enables the estimation of dissemination delays at a practical cost. In the third part of this thesis, we show how to further develop this framework by deriving analytical, closed-form expressions that describe delays as a function of a graph and availability parameters, when the underlying availability model is based on a certain class of simpler distributions.

Finally, by putting together the lessons we learnt along the way – our dissemination protocol and the knowledge we acquired about the workings of communication delays – we devise the final contribution of this thesis: a hybrid, cloud-assisted P2P architecture that enables efficient dissemination in social overlays under churn. This solution, as we show, provides performance that rivals that of centralized solutions, while incurring modest economical costs.

Keywords
[Decentralized On-line Social Networks; Social Overlays; Friend-to-Friend; Temporal Networks; Epidemic Protocols; Cloud-Assisted Peer-to-Peer]

Acknowledgments

This thesis is the result of a long and often tortuous path, along which I have received the support of a great many people. It would for sure not exist were it not for my advisors, Gian Pietro Picco and Alberto Montresor, who have devoted their time and resources in supporting me along each step of the way. I can say I was lucky to have had two advisers, since their rather different styles complemented each other, pushing me to always keep going, even during the most difficult times. I am deeply grateful to them for that.

The work of serving in a examination committee is often an arduous and thankless task. I would therefore like to express my gratitude towards professors Maarten van Steen, Mirco Musolesi, and Pietro Michiardi for taking the time from their busy schedules to review my work, and provide me with challenging questions and valuable feedback.

Living abroad has always been difficult for me, mostly due to being so far away from my family. I have been most fortunate, however, to have met many fantastic people in Trento, and these made my life much more fun, and the distance more bearable. To all of my dear friends, thank you so much for having been part of my life, and for having been my brothers and sisters abroad for all of these years. I will refrain myself from mentioning individual names, as I would for sure leave someone important out – but you know who you are.

Finally, I would like to thank my family – Stefano, Cristina, Elio, and Juliana – for having provided me with the unconditional love and support you did along these years. I do not think I will ever be able to repay that. You have been, and will always be, the irreplaceable pillars of my life, and I am thankful for that.

Thank you all!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

> No matter how many times a privileged straight white male technology executive pronounces the death of privacy, Privacy Is Not Dead. People of all ages care deeply about privacy. And they care just as much about privacy online as they do offline.
>
> — *Danah Boyd*, Making Sense of Privacy and Publicity

From the early days of Friendster to modern-day Facebook, On-line Social Networks (OSNs) have made great strides. Having hoarded hundreds of millions into the ranks of their active user bases, OSNs have surpassed the point of mere entertainment, establishing themselves as *social utilities* [13] as they take an increasingly important role in maintaining and extending social interactions among people [12].

1.1 Context and Motivation

The most successful OSNs of today are *centralized*: they run on large datacenters under the control of single entities, e.g. companies like Google, or Facebook. Centralized systems live in a controlled environment and, as such, are easier to develop, to scale and to secure, and can be made as efficient and high-performance as one can wish for. Their downside are the high development and running costs: servicing one billion users does not come cheap [68], and running an OSN platform requires a way to generate revenue. And this is where the centralized OSN conundrum begins.

The success of an OSN is highly conditioned by *network effects* [27]: without a critical mass of users, people cease to see value in the service and leave. In an environment where providers compete for having the most users, the standard practice became offering services for free. Revenue is instead generated by offering value-added services to third

parties in which providers act as mediators to the personal data of their own user base. The most well-known of such services is targeted advertisement.

The core issue with this business model is that it is based on fostering lock-in, as providers need to keep their user base at all costs, and violating privacy, as users hand in their plaintext, private data to the OSN provider, and then grant it rights to commercially exploit it. Problems are then bound to emerge as the business strategy of the provider evolves, and users start to diverge from providers over what constitutes fair use [15], ultimately being faced with the choice of either tolerating rules that they do not condone, or abandoning a service that they consider essential.

The background motivation for our work is that of providing people with an *alternative* to centralized OSNs. The ideal alternative should, in our view: *i) be open and decentralized*, allowing users to keep control over their data, while providing them with the freedom to try and switch to other alternatives if they so desire, and *ii) require little or, preferably, no extra investment,* since high costs would not only constitute a barrier to adoption, but would lead to the exclusion of an important fraction of the user base [12] (e.g. youth) who have little or no personal income.

These two requirements are naturally matched by peer-to-peer (P2P) architectures, which not only lend themselves well to decentralization, but also tend to require relatively small investments: most P2P systems are designed to run over resources that most users with a computer and an Internet connection already have.

P2P, however, presents a host of challenges of its own. First, resources in P2P systems are known – from filesharing experience – to be highly heterogeneous, both in terms of capacity and availability [99, 108, 128]. Second, although decentralization means we are effectively able to knock off the most powerful of our adversaries from the adversary list – the centralized OSN provider – it also brings with it a long list of other potential problems. At the heart of these problems lies an issue of *trust*: a P2P system works by having users contribute network and computational resources and cooperatively provide service to one another. This often means that peers have to entrust their data to a priori untrusted partners (e.g. when replicating to increase availability), or to rely on them to provide other kinds of essential services (e.g. search and message routing). Clearly, such untrusted nodes might – and often will – misbehave. This can range from selfish peers, which might simply refuse to provide resources or service to other users, to outright malicious peers, which might either individually or collectively attempt to subvert, damage, or otherwise disrupt the system (e.g. [120]). Depending on the type and scale of misbehavior, results might range from performance degradation, to privacy violation, to complete system collapse.

1.2 Social Overlays

Yet, decentralized online social networks possess one key piece of information that is normally withheld from P2P systems: the social network itself, a graph that has the power to indicate, to a large extent, the trust relationships among users in the system. It is our belief that social networks can help us either mitigate or solve fundamental trust-related issues in P2P systems. This belief led us to consider, in this work, the most radical way in which social networks and P2P systems can be mixed together: as a *social overlay* (SO) in which nodes in the P2P system are only allowed to contact each other if their owners are friends. An SO, therefore, effectively *mirrors* an underlying social network. To the best of our knowledge, no other work in the literature has ever attempted to build a decentralized OSN in this way. Further, as we will see in this thesis, the impacts of pursuing such a design are significant.

Apart from giving raise to a number of desirable security properties – some of which we discuss in Section 2.2.1 – social overlays have another characteristic that makes them attractive in our context: they are shaped after human communication patterns, a trait they inherit from the underlying social network. SOs are, therefore, bound to be in synergy with OSNs, which are essentially human communication tools. Indeed, in the context of OSNs, social overlays:

- *Connect the right people.* People tend to talk to people they know. Recent studies [8, 101] show that this also holds for OSNs, where a large fraction of user interactions take part within one-hop neighborhoods. A social overlay enables short routes and constrains traffic to small network sections at a time when handling these frequent interactions.
- *Can improve locality.* People tend to connect to people that are alike, with geographical co-location playing a key role [64]. As a result, not only routes over social overlays are likely to be short, but also physically localized.
- *Provide peer incentives.* Unlike links in commonly used P2P overlays (e.g. Distributed Hash Tables [109]), links in a social overlay represent friendship. We share the belief with other researchers [38, 58, 71] that friends are more likely to cooperate (or, conversely, less likely to misbehave) with other friends than with random strangers.

1.3 The Core Problem: Update Dissemination

The potential advantages of SOs led us to consider how to conceive a P2P OSN in a way that exploits them.

Profile-based communication. In this context, the OSN user interaction studies of Benevenuto [8] and Schneider [101] provided us with two key pieces of information that nudged us in the particular direction we took.

First, they highlight the importance of one key feature in mainstream OSNs which immediately became our focus: *profile-based communication*, the interaction process through which users publish content to each other's profile pages and receive other content, annotations, and/or comments from friends, in response. A key role in this process is played by the *newsfeed*: a mechanism that keeps track of the most recent updates posted to the profile pages of friends, and displays them in aggregate form to the user as he interacts with the system.

Second, they reveal that a high percentage (around 80%, according to [8]) of user interactions take place among direct, 1-hop friends. These two pieces of information put together allowed us to set a clear goal for what our OSN had to provide: *efficient 1-hop, profile-based communication.*

A minimalistic approach. To understand how can we entertain providing 1-hop profile-based communication in a P2P fashion over a social overlay, it helps to reason about the elements a minimalistic design would require. To simplify reasoning, we focus on the perspective of some user u. Given that P2P nodes are inherently unavailable, and u's data has to be made available, we have to replicate it. Yet, we want to avoid the privacy problems associated with replicating over untrusted nodes (which we discuss in more depth in Section 2.1.2-2), so we replicate u's data only over her friends. This has the immediate implication that u's profile page is always available to her friends, and that they can always browse it efficiently, using only local data. If u follows suit and helps replicating the profile pages of all of her friends, it follows that u can browse all of her 1-hop friend profile pages efficiently as well. Efficient 1-hop profile browsing has been dealt with.

The problem, then, shifts to that of keeping replicas up-to-date as u posts updates to her profile page, or as friends post comments in return. Once again, to avoid uncooperative, selfish, and/or malicious untrusted intermediaries, we only allow updates to be relayed by friends. This is feasible with social overlays: all we have to do is confine dissemination to u's *ego network* – the graph formed by u, her friends, and the interconnections among them. u's friends cooperate amongst themselves, relaying the update to common friends. Since the set of relay nodes is the same as the set of receivers, this works well. In topic-based P2P pub/sub terminology, an overlay exhibiting this property is called *topic-connected* [22]. As a bonus, since friends are only talking to friends and relaying an update which is of common interest (and from a common friend), we postulate that this should lead to further incentives to cooperation. Finally, newsfeed can be implemented

trivially (and based on local data) under such scheme, by simply collecting most recent updates and displaying them to the user.

The canonical example of the kind of scenario we want to support is depicted in Figure 1.1. Users u, v, and w are such that v and w are friends with u but w is not friends with v. In the interaction depicted:

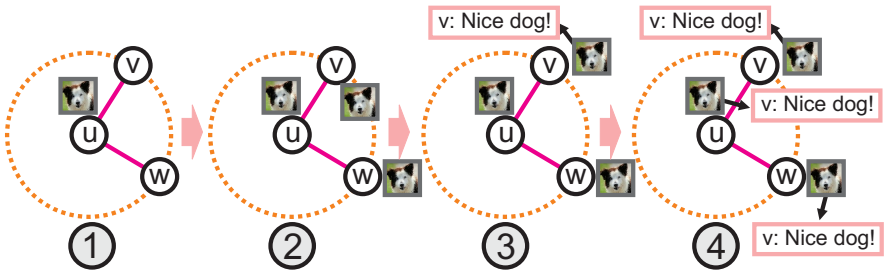


Figure 1.1: User u posts a picture, user v posts a comment, user w just watches.

1. u posts a picture to his local copy of his own profile;
2. the system disseminates the update to v and w, who update their local copies;
3. upon seeing the picture, v posts a comment to it. This goes into v's local copy of u's profile;
4. the system disseminates the comment to u and to w, updating their local copies.

The contributions of this thesis can be divided into four parts, which unfold and reveal more fundamental problems as we attempt to realize and characterize the behavior of our minimalistic design. We discuss these next, together with the story that connects them.

1.4 Roadmap and Contributions

A general overview of how the problems we tackle in this thesis fit together, as well as how they lead to the organization of our text, is depicted in Figure 1.2. This should serve as a guide to our discussion. The text starts with a presentation of background and related work in Chapter 2, together with a restatement of our key problem (update dissemination), and the base system model we adopt across the thesis. From there, we move on to the first problem we tackle in our work: *how* to actually perform update dissemination in a social overlay.

Dissemination in Social Overlays. Owing to their simplicity, scalability, and tolerance to topology changes, we study, in Chapter 3, the application of *push gossip protocols* to the timely dissemination of profile page updates over ego networks. At first, we attempt to apply a classic gossip protocol: Demers' rumor mongering [28]. We find, however, that

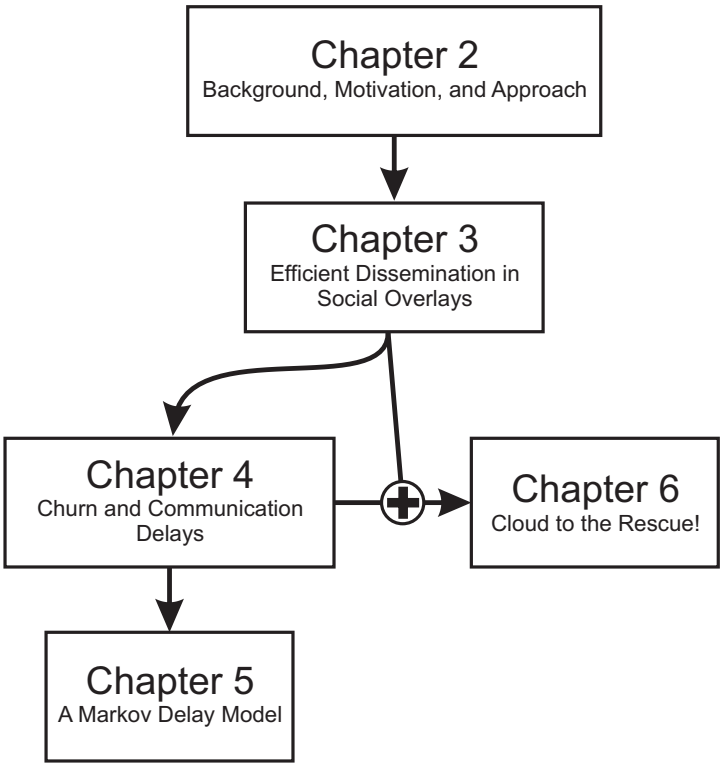


Figure 1.2: A map of our problem structure and thesis outline.

the protocol fails to operate properly in our context, mostly due to the assumption of uniform graph clustering that underpins classical protocols, and which is violated by ego networks.

We then quantify the extent to which problems arise due to this assumption mismatch and, by analysing the structural properties of ego networks, devise the first contribution of this thesis: a gossip protocol that is able to adapt to (and leverage off) such non-uniformity. The protocol in itself is a combination of known techniques and concepts – besides gossip, the use of message histories, and of a biased selection heuristic geared towards the particular properties of ego networks. Still, their combination and application to an overlay mirroring the real-world social network is, to the best of our knowledge, original.

Push protocols, however, account only for the *first phase* of the dissemination process, since it is often the case that friends might not be online when an update is posted and, as such, will be beyond its immediate reach. Indeed, these nodes would normally be taken care of in a *second phase* of the dissemination process, by means of a secondary push-pull protocol such as *anti-entropy* [28].

Churn and Communication Delays. Yet, social overlays have a rather particular

characteristic that made us suspect that, past the initial phase governed by the push protocol, dissemination performance would be driven not by our choice of secondary dissemination protocol, but by *temporal connectivity patterns* of the network itself. Our suspicion was that these give raise to network-induced delays that are more important than those incurred by any individual dissemination protocol.

To illustrate, suppose some new update were to be injected into the system by a peer u, to be disseminated to all of u's friends, including some friend v. Due to their usage habits, however, it so happens that u and v are never online at the same time. To relay the message to v, therefore, the system has to find a third peer w to broker the message, such that w is online at the same time as u, and then later at the same time as v. Further, w must be friends with both u and v. Clearly, a peer w satisfying such conditions might not exist, which means the message might have to traverse a chain of friends acting as brokers before it can actually reach v – assuming it ever does. Moreover, the longer this chain, the greater the delay experienced by the receiver v, as one broker must wait for the next one to come online before passing the message forward.

This is an important matter since, if such network-induced delays are too large, this means that the practical usefulness of social overlays – and therefore of our whole approach – is to be called into question. We therefore shift our focus, in Chapter 4, into understanding the magnitude of these more fundamental *communication delays*.

More specifically, we work on the following problem: given an arbitrary graph and an availability model (a set of probability distributions that describe the up/down behavior of nodes in this graph), what kinds of communication delays are to be expected among any arbitrary pair of nodes u and v in this graph?

This is a problem of a much more general nature, going beyond our use of ego networks for update dissemination, and touching upon issues that are relevant to all applications that rely on social overlays. It also shares deep connections with opportunistic (and delay-tolerant) networks [20], which can often be seen as some form of dynamic social network, providing a more targeted look at how performance should be over specific graphs, as opposed to the more general, network-theoretical treatment that is usually given to the topic.

Initially, we perform a small-scale simulation study that combines social network datasets with a well-known, synthetic availability model [128]. This study leads us to two realizations. First, that network-induced delays can get rather large: even in our limited study, 1% of the users would have experienced delays larger than 3 hours, which are unacceptable, and would warrant further investigation. We consider the identification of the problem, together with the initial set of results that show that it is indeed a problem, as the second contribution of our thesis.

Second, we learnt that simulations alone cannot provide us with a definitive answer on the magnitude of such delays. This is due to three main reasons: *i)* the sheer size of the networks to be considered when working with social network datasets, *ii)* the large parameter space inherent to modelling availability, e.g., the many ways in which availability can be mapped onto network nodes; and *iii)* the difficulties in working with heavy-tailed distributions characteristic of peer availability [99, 108, 128], which severely increase the number of repetitions required from simulations to obtain meaningful results [33]. These costs, which essentially get multiplied by one another, ultimately render simulations an impractical tool for studying the problem at the required scale.

We therefore set out to develop the third contribution of this thesis: a hybrid analytical/simulation framework which allows us to provide upper bounds on dissemination delays at a more practical cost. In doing so, we devise algorithms that allow the identification of key graph substructures that account for a substantial fraction of observed delays; i.e. we expose, as a side effect, the structure of delay bottlenecks.

By revealing the structure of network delays, we believe our model to be of use not only in assessing the feasibility of SOs under different availability assumptions, but also in guiding the development of mitigation strategies and system design decisions, as well as evaluating their tradeoffs.

A Markov Delay Model. Although our hybrid model allows us to reduce computational costs of delay estimates significantly, they remain nevertheless high, getting higher as more precision is required from the estimates it produces. To be really able to study larger datasets and parameter spaces, we need a model that is simulation-free.

The main hurdle to the development of such a model are, again, the heavy-tailed distributions: they are unfriendly to simulations, and difficult to manipulate analytically due to not being memoryless [93]. In Chapter 5, we modify the underlying availability model by replacing these heavy-tailed distributions with memoryless, exponential distributions. Not only is this change in line with other works in the literature [91, 107], but it also allows us to tap into Markov chain theory to make some significant headway, deriving closed-form expressions to many of the elements that previously required simulations, and effectively enabling the application of our estimation techniques to large-scale graphs. This constitutes the fourth contribution of our thesis.

At the end of Chapter 5, we present leads on future work, particularly on how to refine the model to provide higher precision delay estimates (with additional results in Appendix B), together with the sketch for a different hybrid technique developed to that end.

Cloud to the Rescue! Chapters 4 and 5 concerned themselves with showing that network-induced delays on SOs can be rather large, and with providing tools for studying

them more effectively. Without any further work, this would seem to indicate that our minimalistic design of Section 1.3 might not be feasible after all.

We therefore concern ourselves, in Chapter 6, with how to circumvent such network-induced delays (and thus bring our system one step closer to being practical) by means of a simple, yet effective hybrid architecture that augments our dissemination protocols of Chapter 3 by leveraging inexpensive, high-availability cloud resources. At its core, the idea consists in creating a out-of-band channel through the cloud that can "patch" connectivity if and when needed. Its main novelty lies in the fact that users do not need to pay for full hosting servers. Instead, our approach is entirely based on "passive" cloud services that provide storage only, like Amazon's S3. Further, the hybrid solution still leverages on the P2P network whenever possible, bringing costs down w.r.t. a solution in which only the cloud is used.

The solution represents a compromise between monetary cost and performance: by relying on a pay-per-use cloud infrastructure, we allow users to either opt for high performance and pay for it, or take the P2P network as it is, but without spending anything. Preliminary evaluation shows that it allows users to attain performance which rivals that of centralized solutions, while maintaining costs well below that of dedicated hosting. This constitutes the fifth and last contribution of this thesis.

Contribution summary. The contributions of our thesis can, then, be summarized as:

1. a novel gossip protocol which, by taking the structure of ego networks into account, is able to adapt to, and leverage on their non-uniformity, providing efficient and timely dissemination of updates;
2. the identification of network-induced communication delays as the dominating factor in dissemination processes over social overlays at larger time scales, and an initial assessment of their magnitudes by means of a simulation study;
3. a hybrid analytical/simulation framework that allows such delays to be estimated at a more practical cost, and which evidence key graph substructures that answer for a significant portion of delay;
4. a fully analytical model based on Markov chain theory for coarse estimates of network-induced delays, under the simplifying assumption that the underlying session times (time a user remains online) and inter-session times (time between two consecutive logins) are exponentially distributed;
5. a hybrid cloud/P2P architecture and a dissemination protocol which leverage on inexpensive cloud storage services to circumvent network-induced delays, providing performance that is close to centralized solutions, at modest monetary cost.

By reflecting both trust relationships and human communication patterns, social overlays would seem custom made to P2P decentralized OSNs. Our work represents the first

attempt at actually conceiving and evaluating a system that works in this way and, as we will see, the set of issues involved in the application of SOs is far from trivial.

Yet, we believe the work we develop in this thesis to make enough headway to show that the approach is credibly feasible, while building a body of work which should prove useful both as a guide and as a starting point to other researchers and practitioners who venture into their application, both to decentralized OSNs or elsewhere.

Chapter 2

Background, Motivation and Approach

> On those remote pages it is written that animals are divided into (a) those that belong to the Emperor, (b) embalmed ones, (c) those that are trained, (d) suckling pigs, (e) mermaids, (f) fabulous ones, (g) stray dogs, (h) those that are included in this classification, (i) those that tremble as if they were mad, (j) innumerable ones, (k) those drawn with a very fine camels hair brush, (l) others, (m) those that have just broken a flower vase, (n) those that resemble flies from a distance.[1]
>
> – *Jorge Luis Borges,* The Analytical Language of John Wilkins

In this chapter, we set the direction for the remainder of this thesis by putting our work into context, presenting its motivation, and discussing the basics of our approach. This is done in two parts.

In the first part (Section 2.1), we provide an overview of the OSN decentralization landscape, discussing the tradeoffs and challenges of different approaches, and broadly situating our approach of choice (P2P) as one among them.

In the second part (Section 2.2), instead, we narrow down on aspects that are unique to our approach, first by deriving it from two key pieces of related literature – OSN user studies and Friend-to-Friend networks – and then by introducing the core problem of *update dissemination over ego networks* that emerges from it, as well as the accompanying base system model, both of which will be used across this thesis.

[1] This citation was taken from [91].

2.1 Decentralized Online Social Networks

OSNs can vary significantly both in the way they work and the functionality they provide. Yet, by taking a closer look at mainstream approaches (e.g. Facebook and Orkut), it is possible to identify a key set of common features and services which not only constitute the essence of what OSNs represent from our perspective in terms of functionality, but can also be used later on as a stable basis upon which to discuss what is provided (or not provided) by current decentralization approaches. We therefore start with a discussion of these essential elements first, moving on to actual decentralization approaches after that.

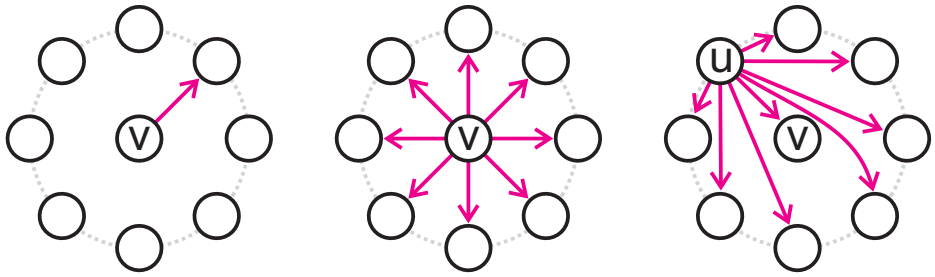
2.1.1 The Essential Elements

As we consider here, OSNs are communication tools built around the concept of *personal profile pages*: read/write personal home pages where users can build digital representations of themselves by publishing personal information and sharing content (photographs, text snippets, music, etc.) with their friends [14]. For the remainder of this text, we assume, for simplicity, that every OSN user v is associated to exactly one profile page, which she *owns*. We refer to v's profile page as $\mathbf{pp}(v)$. We say that a user u posting content to $\mathbf{pp}(v)$ *authors* such content. All contents posted to $p(v)$, however, remain under the *ownership* of v.

OSNs offer users many communication methods, including traditional ones such as e-mail and instant messaging. What makes OSNs unique, however, are not these traditional methods, but something we refer to as *profile-based communication*: the interaction process through which users publish content to each other's profile pages and receive other content, annotations, and/or comments from friends, in response. A key part of this process is the *newsfeed*: a mechanism that keeps track of the most recent updates posted to the profile pages of friends, and displays them in aggregate form to the user as he interacts with the system.

Decentralizing an OSN entails, as a first step, an analysis of their constituent building blocks – i.e. the services we would need if we were to build an OSN. These, in turn, depend on the features and services offered to end-users. Although there is considerable variation in functionality across different OSNs, it is possible, as we already mentioned, to identify a key set of such services emerging from features common to the mainstream. We divide these services into three categories: *search, messaging*, and *content access*.

- *Search* enables users in an OSN to find other users by searching a global directory of participants (e.g. by name). Although other content might end up in this global directory as well (e.g. Facebook "pages", Orkut "groups"), the ability to find other users is the common denominator.



(a) E-mail and instant messaging.

(b) v posts content to own profile page.

(c) Friend posts new content to $p(v)$.

Figure 2.1: Messaging patterns in an OSN.

- *Messaging* enables users to send e-mails and instant messages to their friends. It also enables the dissemination of *updates* from users to users as new content gets posted to their profile pages, or comments are made in return. Typical communication patterns are illustrated in Figure 2.1: e-mail and chat generally entail a straightforward unicast pattern (Figure 2.1b), whereas profile page posts entail a multicast pattern as updates are spread to friends over *newsfeed*. Perhaps surprisingly, multicast patterns do not occur only when v is the one posting: comments from users like u in Figure 2.1c also show on newsfeed, and must be disseminated to all of the friends of v, minus custom visibility rules. The set composed by v and her friends, therefore, forms a multicast group.

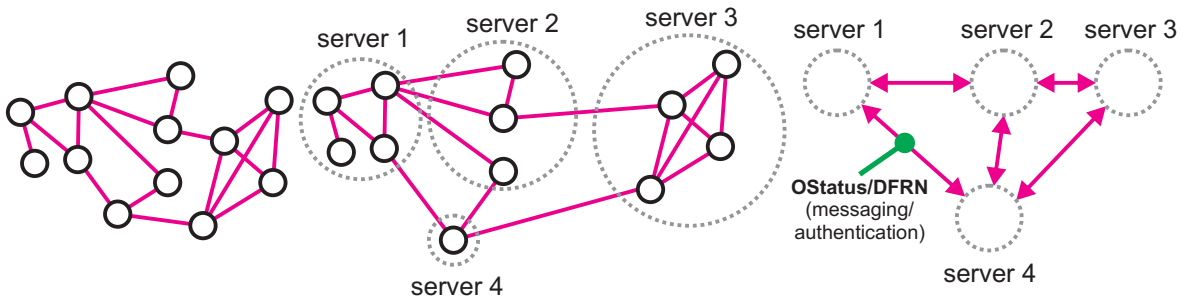
- *Content access* refers broadly to the ability to retrieve content on-demand: as a user accesses the profile page of a friend, peruses her photo albums, or accesses content posted to his own profile page, such content must be available and readily retrievable. Further, given the interactive nature of such systems, retrieval must happen with low latency, at the risk of burning out the user base otherwise [77].

These basic building blocks provide us with a set of common concepts which will be useful in our discussion of decentralization approaches next.

2.1.2 Decentralization Approaches

Approaches to decentralizing OSNs can be classified in three broad categories: *decentralized servers*, *P2P*, and *hybrid*. Each of these has its own set of challenges, offering different tradeoffs in terms of performance, cost, and security/privacy.

2.1.2-1 Decentralized servers. In *decentralized servers*, service is provided by means of a collective of independently operated servers. Profile pages are typically mapped



(a) Social network. (b) Mapping of profile pages to servers. (c) Resulting server-to-server graph.

Figure 2.2: Decentralized server approaches.

$n-$to-1 into servers (Figures 2.2a and 2.2b), and these engage into standardized, server-to-server communication protocols (e.g. OStatus stack protocols [122], or DFRN [62]) to implement the messaging services we have discussed earlier, as well as cross-server authentication (Figure 2.2c). Under such arrangements, users might either opt for running their own servers, or for registering themselves with trusted servers ran by 3rd parties. Most of the community-driven, open-source efforts [29, 35, 36, 121, 130] currently fall into this category. Server functionality might be self-contained (e.g. [29]), or might be split across a set of separate services that may run on different infrastructure (e.g. Friendica allows delegating pub/sub functionality of public posts to PubSubHubbub [34] hubs operated by 3rd parties). Search is normally provided by a set of distinguished servers, which act as opt-in user directories (e.g. [37]).

Some variants over this general description exist. *PrPl* [102] and *Vis-à-Vis* [104], for example, both assume a 1-to-1 mapping between servers and users. In *PrPl*, servers, which are assumed to reside at the user's home, are referred to as *personal cloud butlers*, and act as a gateway to a user's personal data (not only OSN data), hosting content either by themselves or with the help of remote data stores that run an access control service. PrPl further differs from other decentralized server approaches in that that it provides a language for querying distributed networks of butlers (servers) while respecting access control constraints, over which social applications can be built. Cross-butler authentication is handled by an extended OpenID [79] protocol. In *Vis-à-Vis*, instead, personal servers are assumed to run on a cloud computing platform (e.g. Amazon's EC2). Vis-à-Vis, however, is not an OSN *per se*: it provides a supporting service to location-based OSNs which enables the construction of server groups such that location queries can be answered efficiently, while allowing members to reveal as much as they want about their location information.

A whole set of new concerns emerge if we assume that the provider of the server infras-

tructure can behave in an adversarial way by peeking into sensitive content. *Persona* [6] deals with such problems by moving most functionality into clients, and relegating servers to dealing with encrypted content and managing write access.

The main advantage of decentralized server approaches is the always-on nature of servers, which makes both the problems of providing available, low-latency content access and messaging relatively simple. The downside is the requirement for somebody to operate a server, which either requires maintaining a box at home, or hiring costly dedicated hosting[2]. This cost perspective is important since, as we mentioned before, it may act as a barrier to adoption and, further, lead to the exclusion of an important fraction of the user base [12] (youth) who have little or no personal income. Indeed, this is one of the main reasons why we consider such approaches to be unattractive.

Another aspect that remains to be investigated is scalability. The server-to-server communication graphs that emerge from these systems are essentially clustered versions of the original social graphs, partitioned arbitrarily among servers – or according to laws we currently do not understand (Figure 2.2b). This is akin to horizontally scaling a centralized OSN; i.e., to having a set of servers, and then allocating users to machines. Yet, it is known [87] that scaling a system that works like that is already difficult when partitioning is done judiciously, since cut densities (the number of social edges flowing from one server to the other) can become very large. The scalability of the arbitrary partitioning induced by decentralized servers, therefore, remains to be seen.

From a security and privacy perspectives, decentralized servers – like all decentralized alternatives – are susceptible to attacks based on traffic analysis, since all server-to-server communication flows over the public Internet [81]. Adversaries able to listen to (encrypted) network traffic might be able to reconstruct edges of the social network across servers, as well as to figure out posting frequencies, and other sensitive information [39]. It is furthermore possible to disrupt individual users by launching DoS attacks directly on their servers, which could be used as an instrument to censorship. Furthermore, as we mentioned before, hosting providers themselves can be seen as adversaries, with the power to inspect not only all incoming and outgoing traffic, but also the state and memory contents of the server software itself, which in turn could allow one to recover private keys and decrypted content. It can be argued, however, that hosting providers have little incentive to act as adversaries since these are being paid for their services, with customer satisfaction being in their best interest. Yet, security breaches of large datacenters are always a possibility (e.g. [47]), at which point the infrastructure could be compromised despite the best of intentions.

Finally, users who upload their content to servers ran by 3rd parties are possibly sub-

[2]At the time of this writing, even low-cost hosting solutions (e.g. [75]) do not come for less than USD100/year.

jected to the same kind of privacy issues that exist in centralized approaches. Differently from the centralized case, however, the open nature of the system means that users always have the choice of taking their profile data somewhere else, switching to another server should problems arise.

2.1.2-2 Peer-to-Peer (P2P). In P2P approaches, users install a client software in their personal machines and form an *overlay network* amongst themselves. Users contribute spare network and storage resources, providing service to each other with little or no intervention from centralized intermediaries. Some approaches might employ distinguished peers or servers for bootstrapping and authentication, but these generally take no part in communication and storage. The main advantages of P2P are the low barrier to entrance – no fees of any kind are required – and the low running costs – the system is maintained with the sum of contributions from all users, with no single party having to finance the entirety of its operations.

A number of proposals to P2P OSNs have been made along the years. Early proposals, such as PeerSoN [18] or LifeSocial.KOM [43], attempted to conceive decentralized OSN designs by straightforward application of existing P2P techniques, most notably DHTs. The core idea of these systems is simple: nodes form a DHT and use it to store and lookup data, as well as route messages, providing basic search, messaging, and content access. Availability of user content is taken care of either by a DHT replication mechanism (e.g. PAST [95] in LifeSocial.KOM), or not at all (PeerSoN). Data is encrypted to enforce read access control. Messages are routed either directly through the DHT, as in LifeSocial.KOM, or by using the DHT as a lookup service – in PeerSoN, nodes publish their current network address on the DHT, which other nodes can then resolve and contact them directly.

One significant issue with these initial designs is that they cannot provide the kind of multicast messaging required for newsfeed efficiently – downloading the latest news for n friends, or pushing a new update towards n friends, requires n DHT lookups, leading to performance problems [45]. Acknowledging these shortcomings, Cachet [76] introduced a "social caching" scheme where nodes start by advertising their presence (in encrypted form) on the DHT. A node v wishing to retrieve the latest updates for his friends starts by greedily searching the DHT for presence objects, looking for friends who are online and, at the same time, who share the most friends with v in the social network. Whenever v finds one such node u online, it queries u for: *i)* up-to-date presence objects of all friends that u shares with v; *ii)* any recent news u might have on these friends. Such news get incorporated into the newsfeed. This simple mechanism makes Cachet orders-of-magnitude faster than naïvely querying the DHT.

Another shortcoming of these approaches is that their use of DHT replication implies that (encrypted) data gets stored at random, untrusted nodes. Apart from issues of performance [10] and availability [57] that one-size-fits-all DHT replication schemes can bring into content access, this provides ample opportunity for replicating nodes to try and make inference on encrypted content. As argued by Greschbach et al. [39], such inferences might allow an attacker to guess the type of the content that is being stored from its size or, in case of certain types of content such as large videos (which have somewhat unique sizes), even guess the what the actual contents are.

Researchers quickly realized that they could improve on the state of affairs by leveraging on the social graph available to OSNs. Starting from the assumption that social links embody trust (and, thus, that nodes run by friends can be trusted), the aforementioned security and privacy issues could be mitigated by replicating data only on trusted nodes, i.e., direct friends. The idea of using friends to store data has been around for some time [58], but the first P2P OSN to actually describe it was Safebook [25]. Safebook goes beyond storage, however, and leverages social links to provide stronger privacy guarantees: its "matryoshka" system causes messages to go through randomized paths along the social network before reaching the destination. By coupling randomization with source address rewriting at each step, Safebook obfuscates message senders and receivers, rendering their identification difficult to observers.

A Safebook *matryoshka* is a set of concentric rings put around every node v such that the first ring contains v's immediate friends, the second ring contains v's friends-of-friends, and so on. Nodes in the innermost ring (v's friends) replicate v's content to keep it available should v be offline (i.e. to provide available content access). The addresses of the nodes in the outermost ring are indexed in a DHT under v's pseudonym, and serve as entry points to v's matryoshka. To communicate with v, nodes send messages to one or more of the entry points. Messages then percolate through the matryoshka over random paths until the innermost layer, where they can be served both by v's friends (if v is offline) or by v himself. Safebook relies on a certificate authority to ensure node identities and prevent Sybil [30] attacks.

Although Safebook's design can safeguard user anonymity (or pseudonymity [83]) by rendering traffic analysis more difficult, it has its share of shortcomings. First, the paths over the matryoshkas are paths over the social graph which, as we will later see in Chapter 4, can incur high communication delays. Second, high degree nodes might face issues as they get repeatedly chosen to participate in people's matryoshkas. Third, Safebook does not specify the replication protocol for the inner ring – which is non-trivial to render efficient when nodes are not all online at the same time, and using nodes outside of the innermost ring for supporting asynchronous messaging is not allowed. Finally, node de-

grees in social networks are skewed [97], which means some nodes will have many friends, but most will have few. Replicating only over friends is therefore bound to not provide enough availability for everyone [106, 116].

SuperNova [105] and Cachet [76] deal with the availability problems of nodes with few friends by storing data *preferably* at friends, but having a fallback mechanism should friends prove to be insufficient. In SuperNova, this fallback is provided by *super-peers*, which assume a central role in the system both by directly replicating data for nodes who do not have enough friends, and by acting as brokers to replication requests among non-friend peers. In Cachet, data is kept at friends, but also at a "backup", random location in the DHT, where it remains always accessible by means of a standard DHT replication mechanism (e.g. PAST [95]), with all of the privacy caveats that this implies.

Other approaches to P2P OSNs focus on specific functionality. eXO [61] focuses on specialized search by providing efficient top-k queries for tagged content. Olteanu and Pierre [78] link the problem of providing the multicast side of OSN messaging services in a P2P context to topic-based pub/sub, and adopt the SpiderCast [22] protocol to build a dissemination-friendly overlay where nodes exploit correlated subscriptions to lower their overlay degrees.

P2P OSNs inherit all of the security and privacy issues from decentralized servers, and add some of their own. As we mentioned in Section 1.1, issues arise mostly because, in P2P systems, fundamental operations might have to be delegated to untrusted peers. We have already given an example of these problems while discussing the approaches that rely on DHTs for storage [18] or replication [43, 76].

Other issues include the use of multihop routing in P2P overlays, which employ untrusted *relay nodes* while getting a message from source to destination. This amplifies opportunities for eavesdropping, traffic analysis, as well as for attacks disrupting routes to correct nodes, such as eclipse attacks [120]. Further, nodes in a P2P network often connect directly to untrusted nodes, revealing their IP addresses. This kind of information can be used to determine the geographic location of users, or to perform correlation across services accessed from the same address. A broader analysis of the privacy threats in P2P OSNs can be found in [39].

Finally, the fact that almost every single design we mentioned in this section relies on DHTs at one level or the other means that these systems are amenable to the kinds of attacks that can affect them [120]. Recent work on trying to address security issues of DHTs in the context of P2P OSNs includes *LotusNet* [2], an OSN framework based on the Likir [1] secure DHT. Likir relies on a central certification authority to issue signed identifiers binding overlay IDs to user identifiers, thus precluding attackers from creating Sybils or picking their IDs. LotusNet, however, does not address some of the early

problems we have discussed, such as ensuring adequate availability for content access, or implementing efficient messaging for newsfeed.

2.1.2-3 Hybrid. Hybrid systems are a crossover between decentralized servers and P2P, and carry the promise of delivering better performance than pure P2P at costs that are lower than those of dedicated hosting, while offering reasonable privacy guarantees. Few actual system proposals exist for hybrid decentralized OSNs, making it the least explored portion of the design space.

In Confidant [60], authors propose keeping actual *data* at the P2P layer. Data is made available by replication over trusted nodes, i.e., nodes operated by friends. Coordination metadata – i.e., who are the chosen replicas, what is their network address and online status, as well as other relevant information – is maintained in a set of personal *naming servers* hosted on the cloud. There is a one-to-one mapping between cloud-based servers and users. Server software is so simple that it can be deployed for free on platforms such as Google's App Engine [42], making the approach cost-effective. Further, since data is located at the P2P layer, the cloud provider has no way to inspect or mine it, mitigating privacy concerns. As with all friend replication schemes, Confidant relies on having enough friends to guarantee availability. Further, if Confidant were to be deployed at a large scale, the large number of naming servers would probably catch the attention of the cloud provider, which in turn could lead to actions to ban it.

Shakimov et al. [103] propose a variation of Vis-à-Vis [104] in which users run their personal servers on their own desktop machines, activating a cloud replica on *stand-by* should the desktop machine become unavailable. As we later argue, the pricing schemes offered by providers such as EC2 [4] mean that such solutions might still incur high costs.

The position paper by Kryckza et al. [54] gives a somewhat different twist, proposing hybridization as a way to help the centralized OSN provider. The idea is to bear on the user the responsibility for serving heavier content to each other (e.g. movies, or photos), thus helping mitigate OSN server load. Social metadata, on the other hand, remains in control of the centralized OSN provider so as to enable their business model. OSN servers orchestrate the replication process and keep track of replica status, much like in Confidant. Privacy, however, is clearly not the focus.

Hybrid solutions carry the same privacy and security considerations we made for other decentralized approaches, depending on the solution. In particular, if the provider of a server infrastructure is not to be trusted, preserving privacy can be difficult. Even with approaches like Confidant – which do not trust the server infrastructure a priori – providers can still infer things like one's contact list by analysing accesses to individual naming servers and correlating IP addresses, and can infer profile page activity (and map

it to individual users) by observing naming server traffic.

2.2 Our Approach

In the last section, we have presented an analysis of the building blocks of OSNs, and have discussed decentralization approaches together with their associated tradeoffs and issues. In this section, we focus on our own approach instead, first by exposing its underlying motivation as the union of two related pieces of literature – Friend-to-Friend networks and OSN user studies – and then by introducing, in more formal terms, the problem that emerges from this approach and which permeates this thesis, namely, the dissemination of updates over ego networks. Finally, we present the base system model, with the set of a assumptions we adopt for the remainder of this text.

2.2.1 Friend-to-Friend and Social Overlays

A Friend-to-Friend (F2F) network [16] is a type of P2P network in which two nodes are only allowed to communicate directly if their owners are friends. We call the network overlay that results from a F2F network a *social overlay* (SO). It is our belief, as well as that of others [16,63,112], that, under certain application contexts, SOs might help either mitigate or solve difficult problems in P2P.

The fundamental property of social overlays is that node owners that wish to connect are assumed to be *friends*. Being friends, they can communicate either in person or through out-of-band channels to exchange cryptographic keys that allow them to authenticate their computers as being their own, without the need for a central authority. This alone acts as a powerful deterrent (though it cannot prevent) to Sybil and impersonation attacks, and gives raise to a fundamental property of SOs – what Maymounkov [63] refers to as *organic security*:

1. nodes in an SO can trust their neighbors not to reveal neither their identity nor their participation in the network to third parties. By using source address rewriting, this property enables a message to be routed over the network in full anonymity. Indeed, this is the basis to the censorship-resistance properties of systems like Freenet [112];
2. nodes in an SO will not connect, and will not reveal their connections, to untrusted nodes. This makes inferring network structure difficult, and renders SOs – which exhibit a rather irregular structure beyond certain known properties (e.g. [7,52,123]) – difficult to disrupt. Eclipse attacks also become difficult to implement, since intercepting all routes from a (set of) node(s) entails compromising and stealing the identities of a specific set of a priori unknown participants;

3. since nodes only talk directly to friends, they do not expose sensitive information such as their IP addresses to untrusted third parties, rendering the kind of IP inference attacks we described in Section 2.1.2-2 (e.g. finding a user's location by means of IP geolocalization) difficult as well.

A number of proposals for systems based on social overlays have been made along the years. The proposal that comes the closest to ours is found in the position paper by Galuba [38], in which the author indentifies that social overlays and decentralized OSNs can be a good match, for much the same reasons we advocate it (i.e. because social networks embody a map of trust that can be leveraged on). In this thesis, we take this idea forward, making the first incursion into trying to conceive and evaluate a system based on it.

Freenet [112] and Tonika [63] leverage the anonymity and disruption-resistance of social overlays to provide systems that cannot be easily censored. Both systems provide solutions to the problem of creating a greedy embedding [80] of the social graph which enables long-distance calling [49, 98], with Tonika focusing on routing, while Freenet provides, in addition to routing, a platform for publishing content and a client which makes interaction with the system similar to navigating the World Wide Web.

Similarly, Turtle [85] provides a censorship-resistant file sharing network. Searches are flooded over the social overlay, and results are propagated over the reverse paths in the broadcast tree.

Safebook [25] utilizes a social overlay over its matryoshka structures to obfuscate message sources and destinations, who expose only their pseudonyms. Safebook has been discussed in more detail in Section 2.1.2-2.

BlockParty [58] and Friendstore [118] are peer-to-peer backup systems in which friends backup their data on each other's machines. Here, the focus is not on anonymity, but on the fact that the use of friends should provide added incentives for cooperation, and less incentives to malicious behavior (e.g. deliberately corrupting backed up data).

SocialVPN [48] is a system for establishing and maintaining a social overlay which mirrors the social relationships in a pre-existing social network (e.g. Facebook). Their goal is not privacy against the centralized OSN or censorship resistance but, rather, providing automatic configuration of friend-to-friend VPN tunnels to enable support of more complex trusted collaborative systems that go beyond what OSN APIs provide (e.g. interactive multiplayer games, or desktop sharing). The centralized provider is leveraged for public key exchange, by means of a Facebook application. Other hybrid decentralized approaches for exchanging keys are also supported.

GoDisco [26] is a P2P, hierarchical, topic-based pub/sub system based on social overlays which leverages on social network *homophily* [64], i.e. the natural tendency of nodes

in a social network to connect to other nodes with similar properties/interests, to provide increased topic connectivity. Since topic connectivity is not, however, guaranteed by the social network, heuristics are adopted to cross disconnected regions while minimizing spam.

Social Overlays, however, also bear their privacy weaknesses. Like decentralized servers (Section 2.1.2-1), SOs reveal user relationships to anyone who can observe the overlay topology. This can be used as a base to launch deanonymization attacks: as Narayanan and Shmatikov [72] point out, a significant fraction of the users in an anonymized social network can be identified by matching its structure against a non-anonymized network, even if the sets of users in both networks have limited overlap.

2.2.2 Profile-based Communication

We claimed in Section 2.1.1 that the distinguishing characteristic of OSNs w.r.t. previous communication tools is that they enable profile-based communication. To substantiate that claim, as well as to provide an idea of how significant profile-based communication is, we resort to numbers from well-known studies on OSN user behavior [8, 101]. These *clickstream data* studies, which analyse the streams of server requests between the web browser and OSN servers, are currently the only public source of information on OSN user activity that can capture both *visible interactions* (e.g. posts, comments, etc.), and *silent interactions* (i.e., browsing activity).

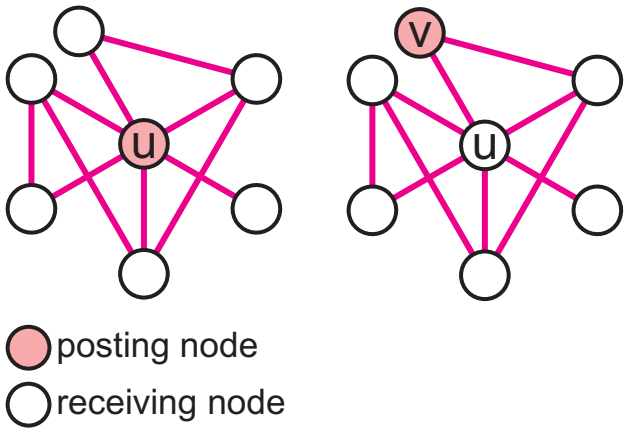
If we consider only the relevant user activities in these studies, profile-based communication constitutes anywhere between 60% [46] to 92% [8] of the requests issued by users to OSN servers, making it the single most important functionality in an OSN. Furthermore, over 80% of these interactions take place among 1-hop friends.

If we follow the lean approach to development [86] and focus on providing what brings the most value to the user first, then, as we mentioned in Chapter 1, the first thing we need to provide is efficient, 1-hop profile-based communication.

2.2.3 Our Approach: Profile-Based Communication Over Social Overlays

Social networks are a reflection of social interactions, which, in turn, are carved out from human communication patterns. It is therefore only reasonable to assume that synergy is bound to exist between social overlays – which are shaped after human communication – and OSNs – which are human communication tools.

In Section 1.2, we have presented three points in which we believe such synergy to exist: a social overlay *connects the right people*, might *improve locality*, and might *provide incentives to cooperation*. We then propose a system in which users keep local replicas (caches) of the pages of their friends. These replicas are then updated incrementally as



(a) Owner posts update. (b) Friend posts comment.

Figure 2.3: The update dissemination problem over ego network G_u.

new content gets posted, or comments are made in return – an example of the kind of scenario we want to support was given in Figure 1.1. Content posted to the profile page of a user u is disseminated exclusively by nodes that belong to the ego network of u (i.e. friends of u). This enables efficient 1-hop profile-based communication, and leads us to the update dissemination problem, which we define formally next.

Let G_u be the graph representing the ego network of some user u, and let v be a member of this ego network. The update dissemination problem consists in disseminating an update from any user v to all of the other users in G_u. The two main cases are differentiated in Figure 2.3: in Figure 2.3a, the owner of the profile page posts an update. In Figure 2.3a, a friend of u posts a comment. In both cases, the update has to be disseminated to all of the other participants of G_u.

Clearly, our minimalistic approach has significant limitations. It leaves out important functionality such as search, profile-based communication over n-hop friends, or an application platform. Further, the approach best suited to updates that are small – though, as we will later argue in Section 2.3, we do not see this as a significant limitation, as the important updates are indeed small. In any case, that is precisely the point of being minimalistic: the very essential comes first, and that is what we focus on in this thesis. Further, as we will later see, the very essential is already a difficult enough of a problem to tackle in this case.

2.3 Notation and System Model

This section states the general assumptions and notation that are valid across the thesis. More specific assumptions are added to each chapter as required.

Social and ego networks. We model a social network as an undirected graph $G = (V, E)$ where V represents its set of vertices, and E its set of edges. Following standard graph theory notation, we also denote the sets of vertices and edges of a graph G by $V(G)$ and $E(G)$, respectively. Every vertex v in $V(G)$ represents an individual. An edge (v, u) exists between individuals $v, u \in V(G)$ if and only if they are friends. We assume friendship to be a symmetric relation.

We model a *social network* as an undirected graph $G = (V, E)$, where V contains users and E is the friendship relation among them. For each user $u \in V$, function $f : V \to 2^V$ maps users to their set of friends, while f_2 maps users into their set of friends-of-friends:

$$f(u) = \{\ u \mid (v, u) \in E(G)\ \}$$
$$f_2(u) = \{\ w \mid w \in f(v) \wedge v \in f(u)\ \}$$

For convenience, we also define $f^*(u) = f(u) \cup \{u\}$.

The *ego network* (or egonet, for short) of a node u is defined to be the subgraph $G_u \subset G$ induced by u, u's friends, and the connections among them:

$$G_u = (f^*(u), \{\ (a, b) \in E(G) \mid a \in f^*(u) \wedge b \in f^*(u)\ \})$$

Mapping of users into peers. We assume that every user in the social network corresponds to exactly one node in the social overlay – i.e. we assume that users will not frequently log in from more than one device at a time. We argue that unless such phenomena is to be expected at a large scale, it is unlikely to affect the results we report in this thesis. It is in any case a simplification that is in line with other works in the literature (e.g. [26, 78, 107]).

Since mapping is one-to-one, we often refer to users and nodes interchangeably. Sentences like *"a node u and its friends $f(u)$"*, then, are to be interpreted as *"a node controlled by a user u and the nodes controlled by her set of direct friends, $f(u)$"*.

Dynamism. A social network is a dynamic concept: new friendship relationships may be formed or old ones severed. In practice, however, such changes are infrequent, at least w.r.t. to the time scale of the problems we consider in this thesis. Therefore, for the purpose of this thesis, we consider the social network to be immutable. The dynamism of social overlays cannot be, however, disregarded in the same way. Users may start and stop their nodes at will; a particular user may not be available for extended periods of time.

In P2P terminology, this phenomenon is known as *churn*. The social overlay therefore undergoes frequent reconfiguration.

Physical network. We use a simplified model for the underlying physical network in which two nodes are able to communicate as long as they are online at the same time. We assume connectivity to be symmetric (i.e. we do not model middleboxes [19] such as firewalls or NAT), and the physical network to have negligible latency. We argue that these are adequate assumptions for the kind of problem we study in our work.

Update sizes. Although users share content of different natures, the vast majority of what is shared in the profile pages of modern OSNs are small objects under 150 kB, the average size of a Facebook picture. Such objects include text snippets, messages, and low-resolution pictures and thumbnails. While users also share larger objects, such as videos or high-resolution pictures, we argue that most of the time these are not part of profile pages themselves, but rather *linked* from third-party services such as YouTube or Instagram. Updates, therefore, will normally be small, and concerns with latency take priority over bandwidth when gauging the quality of dissemination techniques.

Realizing the social overlay. Efficiently maintaining a social overlay is a non-trivial task. As Rogers [91] points out, middleboxes and dynamic network addresses are particularly problematic for social overlays, since they may prevent users from finding their neighbours after spending time offline. The overlay management problem, however, is outside of the scope of this thesis. We assume that nodes are somehow able to discover the IP addresses of their friends currently online, to enable message exchange.

Working compromises can in any case be found in hybrid solutions, such as leveraging off of existing decentralized server infrastructures like the widely available Jabber/XMPP network [96]: managing presence is already an integral part of what Jabber does, and we could simply piggyback on that. Other options include combining dynamic DNS [32] with an approach for peer discovery similar to the one in Cachet [76] (described in Section 2.1.2-2) or, at the expense of bringing all of its vulnerabilities into the system, using a DHT.

2.4 Summary

Decentralizing Online Social Networks could represent a solution to the privacy issues that plague current centralized services. The task, however, is anything but trivial: current OSNs are large, complex systems, and decentralization approaches incur new privacy, security, and performance problems. In this chapter, we reviewed the key building blocks of OSNs and, on these grounds, discussed the three main decentralization approaches in the existing literature, along with their tradeoffs and challenges.

We then formally introduced our approach as a way to provide efficient 1-hop profile-

based communication over social overlays. We presented our motivation for adopting social overlays by discussing their security properties, as well as their suitability to decentralized OSNs. Finally, we provided a more complete description of the update dissemination problem and its accompanying system model, which will be used throughout this thesis.

Chapter 3

Dissemination in Social Overlays[1]

If we all held our tongues, there wouldn't be much left to talk about.

– *English working class saying.*

In Chapter 2, we have identified the dissemination of profile page updates over ego networks as the critical problem to be solved in enabling a minimalistic, P2P, decentralized OSN based on social overlays. Unlike traditional P2P applications such as file sharing, however, OSNs are quasi-interactive systems. Although real-time interaction is not required, this means that profile updates should become rapidly available to friends currently online. In our proposal, this is achieved by relying on an efficient *profile update dissemination* protocol which pushes updates from friends, to friends, in a reliable, timely, and efficient fashion. After providing additional information about the system model in Section 3.1, in Section 3.2 we further detail the problem, our approach, and the experimental framework we use for evaluation.

Several alternatives can in principle be used to deal with the problem. We focus on gossip protocols [51] because they were originally designed for update dissemination, albeit in a slightly different context, and because of their inherent simplicity, scalability, and tolerance to topology changes. Unfortunately, gossip protocols were designed to operate on uniform random graphs, which have very different properties w.r.t. social networks. We assess quantitatively the negative impact of this assumption mismatch in Section 3.3.

The main contribution of this chapter lies in Sections 3.4 and 3.5 where we describe and, respectively, evaluate a protocol for efficient dissemination over social networks, and in Section 3.6, where we perform a more restricted evaluation over large datasets

[1] Earlier versions of this chapter have been presented in [65, 66].

to show that our results generalize. The protocol is a combination of known techniques and concepts – besides gossip, the use of message histories, and of a biased selection heuristic geared towards the particular properties of social networks. Nevertheless, their combination and application to an overlay mirroring the real-world social network is, to the best of our knowledge, original. Moreover, our evaluation shows that the approach is applicable in practice to the scale of ego networks found in real-world social networks.

Finally, Section 3.7 concisely surveys related work, and Section 3.8 ends the chapter with brief concluding remarks.

3.1 System Model

Definitions and notation. Recalling that the social network is an undirected graph $G = (V, E)$ (Section 2.3), let O be the set of all updates generated in the system. We denote by $prof : O \to V$ the function mapping an update to the owner of the profile page to which it was posted.

Knowledge of the social network. We assume user u knows not only the set $f(u)$, but also $f_2(u)$; this knowledge makes it possible, given a friend $v \in f(u)$, to obtain the set of common friends $f(u) \cap f(v)$. This assumption is in line with what is usually revealed in current OSNs, in which contact lists are completely visible to friends. This knowledge is kept up-to-date by the dissemination of new friendship events (again, as in current OSNs), which have in any case to be displayed in newsfeed.

Authentication, access control and privacy. We assume that each node u is identified by a pair of asymmetric keys $(Priv_u, Pub_u)$. Public keys are acquired when learning new friendship relations, possibly through out-of-band mechanisms. To guarantee authenticity, every update o generated by u must be signed with $Priv_u$. This also prevents users from spoofing updates (e.g. one user attempting to disseminate an update that changes the name of another user). More complex access control mechanisms are possible [6, 43, 76], although outside of the scope of our work. Even though privacy is preserved by the fact that updates are encrypted and sent only to the intended destinations, it is always possible that, once decrypted, content is disclosed to unauthorized third parties. This is not exclusive to our approach, however, and could also happen in a centralized OSN.

3.2 Problem, Approach, and Experimental Framework

In this section we state our problem, outline the proposed approach, and illustrate the metrics and experimental setup we use for the remainder of this chapter.

3.2.1 Problem Statement

As stated in Section 1.3, we envision a decentralized OSN wherein updates to user profiles are proactively disseminated to all friends and cached locally. Storing the content of all friends may seem overkill, but pretty much everything in the profile pages of modern-day OSN – comments, links, thumbnails, small pictures – are *small objects* (e.g., under 60 KB, the size of current Facebook pictures). The only exception are movies and large collections of high-quality pictures: however, these are usually not part of profiles anyway, and are linked from services such as YouTube and Intragram. Their associated "social metadata" (e.g., description, thumbnail, etc.), remain instead small objects.

The problem we want to solve is has already been described in Section 2.2.3. In particular, we envision interactions like the ones depicted in Figures 2.3 and 1.1 to be supported by a *social newscasting* service consisting of two simple primitives:

1. the operation postToProfile(PROFILE v, UPDATE o) enables u to post update o to the profile of $v = prof(o)$;
2. the callback newsReceived(list⟨UPDATE⟩ *list*) enables clients to retrieve new content upon arrival.

In the context of OSNs, realizing this social newscasting service essentially trades the problem of *fetching data* over the network with the problem of *disseminating updates* to all friends. The latter must be performed in a *timely* way, i.e., with latency comparable to that of today's centralized OSNs. In other words, for a given subgraph G_v, *any* node $u \in G_v$ may produce, at any point in time, some update o by calling postToProfile(v, o). The goal is to define a strategy to efficiently disseminate o from u to all nodes in $f^*(v)$.

3.2.2 Approach Outline

Owing to their wide recognition as robust tools for fast data dissemination, we chose to use *gossip protocols* [28] as the base for our social newscasting service. These protocols are typically run in *rounds*, in which each node selects a gossip partner using a *randomized selection heuristic* and exchanges data with it based on an *exchange strategy*. Rounds are *not* synchronized; rather, they simply play a "rate limitation" role.

Our social newscasting service employs two gossip protocols. A *rumor mongering* protocol based on a *push* exchange strategy is used to disseminate updates quickly, possibly at the expense of guaranteed delivery (e.g., for users currently not online). This fast but unreliable dissemination is therefore complemented by an *anti-entropy* [28], *push-pull* protocol. This runs in the background at a slower pace w.r.t. rumor mongering and guarantees that all nodes that become and remain online for long enough eventually receive

all updates. In a sense, anti-entropy serves as a "safety net", continuously patching up the message deliveries missed by rumor mongering.

In this chapter, we focus *exclusively* on the first protocol (i.e., rumor mongering) because, as we show in Section 3.3, its application to social networks already requires a significant departure from the mainstream and because, as we will see in Chapter 4, dissemination performance converges to protocol-independent factors beyond a certain timescale, meaning that evaluating the performance of anti-entropy is not so important. Section 3.4 discusses in detail how we improve on the base rumor mongering by choosing selection strategies that explicitly take into account topological properties of social networks, such as centrality and clustering.

3.2.3 Experimental Framework

Unit experiments. Protocols are evaluated over a large number of isolated *unit experiments*, whose results are aggregated offline to obtain global figures. A unit experiment consists of: *i)* singling out a subgraph G_v; *ii)* having v publish an update by calling postToProfile(); *iii)* waiting until the push protocol terminates; *iv)* measuring the desired parameters.

Each unit experiment e is associated with the dissemination of a single update generated by the profile owner, $root(e)$. Updates posted to profiles of friends would behave similarly and are thus excluded from the evaluation.

Performance metrics. Given a set E of unit experiments, we measure the following:

- *Residue*, i.e., the percentage of nodes who did not receive the update after the push protocol terminates. Let $undelivered : E \to \mathbb{N}$ be a function yielding the number of these nodes for a single unit experiment e. Then:

$$residue(E) = \frac{\sum_{e \in E} undelivered(e)}{\sum_{e \in E} |f(root(e))|} \tag{3.1}$$

- *Average and maximum latency*, measured in number of rounds, $t_{avg}(E)$ and $t_{max}(E)$. Defined as the number of rounds it takes for an update to reach a destination, minus the rounds for which that destination has been offline after the update was posted (measured relative to node uptime).
- *Absolute and average load*, in terms of messages sent and received. Formally, let $\ell_s : V \times E \to \mathbb{N}$ be the function yielding the number of messages sent by node $v \in V$ during a unit experiment e, and ℓ_r a similar function yielding the messages v received. The load for a given unit experiment is then $\ell(v, e) = \ell_s(v, e) + \ell_r(v, e)$.

The absolute load over the entire set E of experiments at node v, and the average load, normalized over the size of the ego network G_v, are then given as:

$$load(v) = \sum_{e \in E} \ell(v, e) \qquad \overline{load}(v) = \frac{load(v)}{|f^*(v)|}$$

It is useful to generalize this notion to sets of nodes. The average load incurred on a set of nodes $V' \subset V$ is:

$$\overline{load}(V') = \frac{\sum_{v \in V'} load(v)}{\sum_{v \in V'} |f^*(v)|}$$

- *Duplicate ratio*, i.e., the ratio between the number of messages generated and delivered. For a unit experiment e, the former is:

$$generated(e) = \frac{1}{2} \sum_{w \in f^*(root(e))} \ell(w, e)$$

and the latter is $delivered(e) = |f(root(e))| - undelivered(root(e))$. The duplicate ratio is expressed as:

$$dup(E) = \frac{\sum_{e \in E} generated(e)}{\sum_{e \in E} delivered(e)}$$

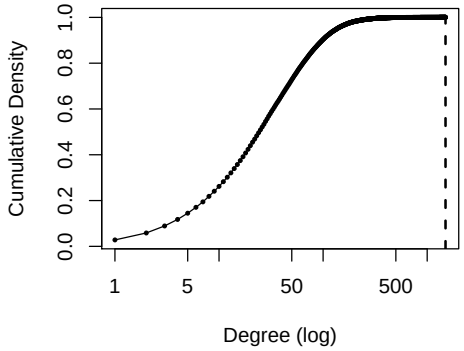
- *Unit load balance*, i.e., how balanced the load was within a single unit experiment and therefore a single message dissemination. We capture it by using a dimensionless dispersion measure known as the *coefficient of variation* CV. For a given set of observations, CV is defined as $\frac{\sigma}{|\mu|}$, where σ stands for the sample standard deviation, and μ the sample average. For a given unit experiment e, the coefficient of variation is computed over the set of load values $\{\ell(v_1, e), \cdots, \ell(v_n, e)\}$, where $v_i \in f^*(v)$, and this yields a relative measure of load balancing. The higher the CV, the less balanced is the load.

Setup. We use the PEERSIM [70] simulator, along with a social graph obtained from a popular OSN site through a snowball sampling procedure [56] over a single seed, to evaluate our protocols. Publicly-available friend connections were explored until we had a partial third level. The sample characteristics are summarized in Table 3.1a, and its cumulative degree distribution shown in Figure 3.1b. The small average degree is due to the sampling procedure: since we had to stop crawling at some point, nodes at the outermost layer—the most numerous—are inevitably missing friends. Despite the small size w.r.t. real social networks, we argue this sample is adequate for simulation purposes. We chose to use a real, albeit smaller, network instead of a synthetic one because current models are still very limited [97].

Unless otherwise noted, we run 10 unit experiments per node, yielding 723 030 repetitions for each combination of protocols and parameters.

Vertices	72303
Edges	1508283
Avg. Degree	41
Max. Degree	1500
Min. Degree	1
Clust. coef.	0.34

(a) Summary.



(b) CDF of node degrees.

Figure 3.1: Dataset characteristics.

	$p = 0.4$	$p = 0.3$	$p = 0.2$	$p = 0.1$
Observed Residue	19%	14%	10.5%	6%
Demers' Residue	$\sim 3.7\%$	$\sim 1.1\%$	$\sim 0.01\%$	$--$
Dup. Ratio (avg.)	2.53	3.37	5.04	10.03
Avg. Load	5.66	7.35	10.6	20.3
t_{max}	59	68	79	124
t_{avg}	4.3	4.5	4.7	4.94

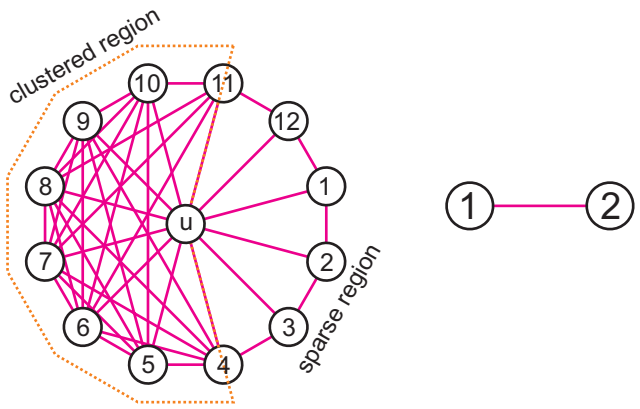
Table 3.1: Results for Demers' protocol.

3.3 A Fresh Look at Mainstream Techniques

An obvious question at this point is *"Can't we just re-use mainstream gossip protocols?"*. The answer to this question is negative, as we demonstrate quantitatively in this section, motivating the contribution described in the rest of this chapter.

Demers' rumor mongering [28] is arguably one of the most well-known gossip push protocols in the literature. We consider the *feedback/coin* variant of Demers', where each node keeps a list of "hot rumors", i.e., updates its friends are more likely not to have. In a gossip round, each node u: *i)* selects a node v from its neighborhood *uniformly at random*; *ii)* sends all, or part of its list of hot rumors to v; *iii)* collects a *response vector* from v which tells which rumors v already knew and which it did not.

Then, for each item in the response vector *i)* if the rumor was not known to v, then



(a) Irregular clustering. (b) Small egonet.

Figure 3.2: Problematic egonets for Demers' protocol.

nothing is done; *ii)* otherwise, the rumor is removed from the hot rumor list with probability p. Node v, in turn, adds to its hot rumor list the new rumors received from u. Further, whenever a new piece of content is posted locally it is immediately added to the poster's local hot rumor list and becomes eligible to dissemination.

We evaluated Demers' under our simulation framework, performing a single unit experiment per node. Aggregate results over the 72,303 unit experiments are shown in Table 3.1. The row "Demers' residue" contains the values reported in [28], for reference. Figure 3.3 shows the average load at each node as a function of node degree. Residues are orders-of-magnitude larger than expected, and average loads are large even for low degree nodes: one of the nodes with degree 1, for example, has an average load of 70. These issues arise from the fact that *social networks violate a fundamental assumption in gossip protocols, i.e., that clustering in the network is approximately uniform.* Figure 3.2a provides an example where:

1. Node u calls postToProfile().
2. As the clustered region is larger than the sparse one, it is going to be hit first with higher probability ($\frac{2}{3}$ over $\frac{1}{3}$).
3. Dissemination proceeds very quickly in the clustered region but also generates lots of duplicates.
4. Eventually, most (if not all) nodes in the clustered region get the update. The protocol, however, keeps selecting the clustered region with higher probability.
5. Every time a node selects an already-infected node it gives up on the rumor with probability p. Thus, the nodes in the clustered region (u included) are likely to give up too soon, as they "perceive" the message infection to have spread, when in fact

many nodes may still await it.

As for load, a node generates $1/p$ duplicates before giving up on a rumor, as confirmed in Table 3.1. For an ego network as in Figure 3.2b, this means node 1 will generate around 10 messages every time it publishes an update, and will receive 10 messages every time node 2 publishes an update. Since we run one unit experiment per neighbor, this amounts to an average expected load of 20, which is very high for such a small ego network. Other topologies might generate even larger averages.

3.4 Gossip in Social Overlays

Demers' rumor mongering is inefficient over social overlays as it is unable to cope with its graph properties: we propose a dissemination protocol that is aware of, and exploits them.

A simple way to solve Demers' residue issue is *flooding*: a node does not give up spreading an update until all of its friends receive it. Pure flooding, however, generates too much traffic. We therefore enhance it by piggybacking *histories* on each message, recording who received a given update. As nodes do not re-send an update to nodes known to have it, we can reduce traffic significantly.

There is, however, another relevant effect of suppressing transmissions. Our flooding protocols are gossip-based, and thus employ a randomized neighbor selection heuristic, the simplest being selecting nodes uniformly at random. However, the latter is biased towards higher degree nodes. This, perhaps contrary to intuition, hurts performance instead of helping it. The reason is in Figure 3.2a: *if higher degree nodes are packed in a cluster, random selection tends to starve regions with lower clustering.* Nevertheless, the progressive exclusion of nodes from highly clustered regions (which are the ones being selected first and thus entering the piggybacked histories first) helps us to eventually steer

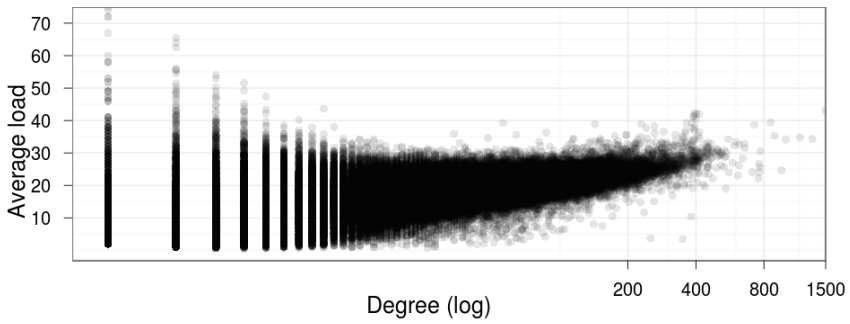


Figure 3.3: Average loads by degree.

selection towards regions containing nodes not known to have received the update.

Flooding with message histories provides the base dissemination mechanism, which we improve with a pair of selection heuristics. Indeed, even if we eventually steer selection towards less favored regions, clustered regions are still heavily flooded with messages in the early stages of dissemination, when histories did not yet propagate. This not only causes more traffic, but also slows down the protocol.

We reap further improvements with a heuristic which steers selection away from highly clustered regions nodes from the very beginning, by picking nodes with a probability inversely proportional to their degree in the ego network. Finally, ego networks are often divided into disjoint components, tied by a "central" node. The ego network in Figure 3.4b, for example, has 4 such components. We can speed up dissemination significantly if the central node spreads the update by hitting components in sequence, beginning with the largest one, down to the smallest. This way, the central node exploits its special position, keeping the ego network connected, to "parallelize" dissemination.

In the rest of this section we detail further these concepts.

3.4.1 Flooding with Histories

The baseline protocol FLOOD works as follows. When a node v learns about an update o belonging to the profile page of node $u \in f^*(v)$, it keeps sending one message per round to the common friends $f^*(v) \cap f^*(u)$ that may have not received o, stopping only once it knows that all of them have received it.

Formally, let $K_{v,o} \subseteq f^*(u)$ contain the intended destinations of o known by v to have received o, and $E_{v,o} = (f^*(u) \cap f^*(v)) - K_{v,o}$ denote the common friends *eligible* for selection at v when considering update o. Then, at each round, node v:

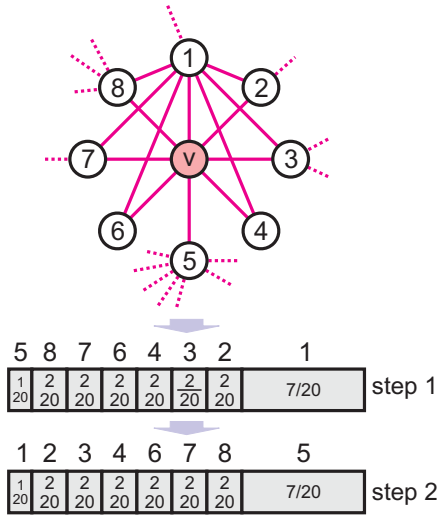
1. selects $w \in E_{v,o}$ according to a selection heuristic;
2. sets $K_{v,o}$ to $K_{v,o} \cup \{w\}$;
3. sends o to w.

Whenever v receives an update o from a node z, it:

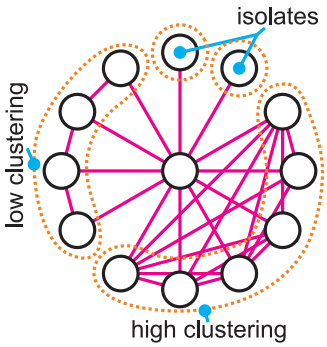
4. sets $K_{v,o}$ to $K_{v,o} \cup \{z\}$.

In the variant with message histories, called HFLOOD, nodes piggyback histories in their messages, sharing their knowledge about nodes that have received o. To get to HFLOOD, we substitute steps (3) and (4) from FLOOD as follows:

3. sends a message $\langle o, K_{v,o} \rangle$ to w, where the *message history* of o known by v is piggybacked with o;
4. when v receives a message $\langle o, K_{z,o} \rangle$ from z, it sets $K_{v,o}$ to $K_{v,o} \cup K_{z,o}$.



(a) Probability assignment in ANTICENTRALITY.



(b) Fragmented ego network.

Figure 3.4: ANTICENTRALITY and fragmented ego network.

In both protocols, dissemination terminates when $E_{v,o} = \emptyset$. Note that if u and v do not share common friends, then $E_{v,o} = \emptyset$ from the very start, i.e., v cannot assist in disseminating o.

Message histories are implemented using Bloom filters [17]. Since updates are disseminated over relatively small sets of nodes (41 nodes on average in our dataset), the overhead is small: Bloom filters incur around k bytes of overhead for an ego network of size k at a false positive rate of 1%.

3.4.2 Selection Heuristics

We consider three strategies for node v to send an update o.

Random. In the RANDOM heuristic, node v selects a node from $E_{v,o}$ uniformly at random as in Demers' [28].

Anticentrality. The ANTICENTRALITY heuristic assigns to nodes in $E_{v,o}$ a selection probability which is inversely proportional to their degree in G_v, effectively "steering selection away" from nodes with higher degree as per the reasoning of Section 3.4. We use G_v because what matters is the degree in the context of the ego network where update dissemination occurs, not the degree in the overall social network.

Figure 3.4a shows how the algorithm which assigns selection probabilities in ANTICENTRALITY works in the particular case when v starts to disseminate an update over its own ego network. We first compute the sum of the degrees into G_v of all of v's neighbors (20). We then sort the neighbors by those degree values, and assign them proportional

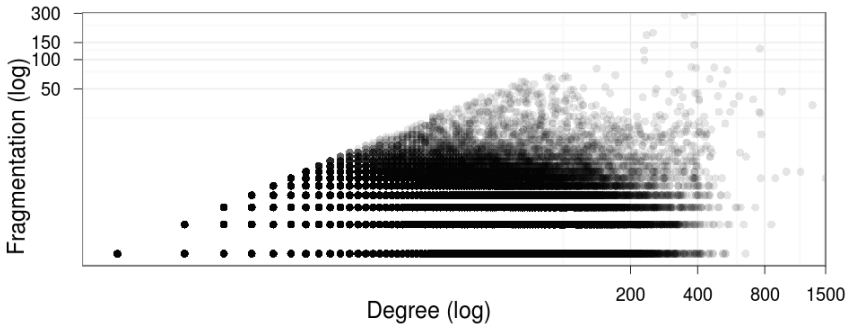


Figure 3.5: Fragmentation τ for our network crawl.

probabilities (step 1). Finally, we invert the assignments so that lower degree nodes get the high probabilities, and vice-versa (step 2).

Formally, let $d_v(w) = |f(w) \cap f(v)|$ (the degree of w into v's ego network), and $E_{o,v} = \{w_1, \cdots, w_n\}$. Suppose without loss of generality that the w_i are ordered such that $d_v(w_1) \leq \cdots \leq d_v(w_n)$. Then the probability $P_v(X = w_i)$ with which node v selects node $w_i \in E_{o,v}$ at some given round is expressed by:

$$\mathbb{P}_v(X = w_i) = \frac{d_v(w_{n-i+1})}{\sum_{k=1}^{n} d_v(w_k)} \tag{3.2}$$

Fragmentation-aware heuristics. The *fragmentation* $\tau(v)$ of a node v is the number of connected components that remain in G_v if v is removed. Formally, let G_v^* be the subgraph obtained by removing v and all its links from G_v; let $C(G_v^*)$ be the set of connected components in G_v^*. Then, $\tau(G_v^*) = |C(G_v^*)|$.

Figure 3.5 shows scatterplots of fragmentation vs. node degree in our dataset. Fragmentation varies widely at all egonet sizes, and larger egonets tend to be proportionally less fragmented than smaller ones.

Fragmentation is an important structural metric for two reasons. First, $\tau(G_u) - 1$ represents the minimum number of messages that the node u at the center of an ego network (i.e. the profile page owner) must send if an update is to reach all neighbors, regardless of who published it. Second, it provides us with a simple way of improving latency, by noticing that u should hit as many different components as possible, avoiding selecting nodes inside of the same component more than once before all components have been hit.

This suggests two new heuristics, RANDCOMP and MAXCOMP, to be applied to the profile owner only. To diffuse an update o such that $prof(o) = u$, node u does the following:

1. if a component $C_i \in C(G_u^*)$ for which no node in C_i has yet received the update

exists, then select a node $w \in C_i$ using ANTICENTRALITY;
2. otherwise, simply select $w \in f(u)$ using ANTICENTRALITY, completely ignoring the component structure.

The two heuristics differ in the way components are selected: RANDCOMP selects a random one, while MAXCOMP selects the largest among the candidate components.

Note that computing the actual connected components is inexpensive. We assume from Section 3.1 that a node u knows both $f(u)$ and $f_2(u)$. This information is all u needs to locally reconstruct the topology of its ego network, which is enough to compute its connected components.

A similar reasoning applies under churn. If we rely on the assumption made in Section 3.1 that a node u knows, with reasonable accuracy, which of its 1-hop social neighbors are on-line at any given point in time, then u is able to locally estimate the shape of its 1-hop neighborhood by excluding the nodes it thinks are off-line, and (locally) compute the connected components based on the estimate instead.

3.5 Evaluation

In this section we evaluate our protocols, first under the assumption of a static network, and then under churn.

3.5.1 Static Network

Here, we assume 100% availability. Under this assumption, all protocols yield residue zero for practical purposes, so we do not discuss it further.

Baseline. We use *direct mailing* [28] as the baseline protocol in our comparisons. In direct mailing, the node posting the update is responsible for contacting receivers directly. As we discuss in Section 3.7, this simple technique is used in some P2P OSNs. To enable comparison, we consider a round-based variant in which contacts are performed in rounds, one after the other. Although direct mailing could be run "in parallel", this would be equivalent to setting its round length to zero, which is something we can also do for our protocols. This transformation, therefore, incurs no loss of generality.

Progressive plots. Due to the nature of our experiments, to effects of the crawling procedure (e.g., border nodes all have degree 1), and to social networks themselves [97], displaying whole-network results is misleading as lower-degree nodes heavily bias figures like average latency (e.g., experiments rooted in a node with one neighbor always have average latency 1, regardless of the strategy). We therefore choose to plot these results in a way we call "progressive trimming".

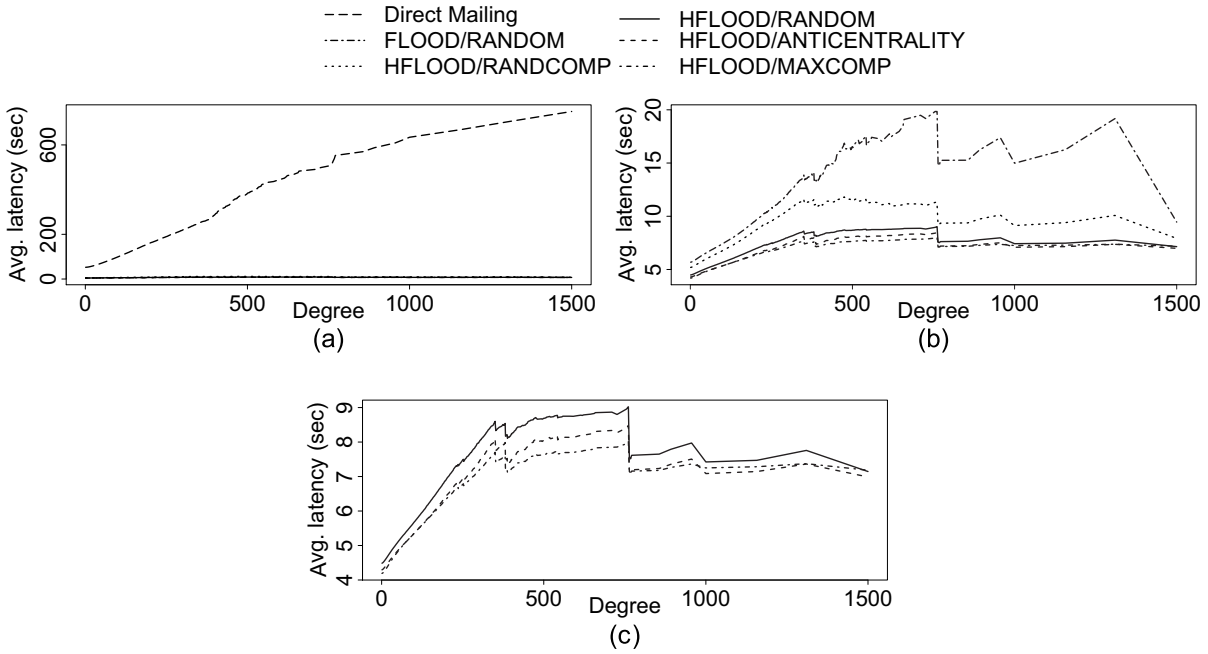


Figure 3.6: Latency.

We focus on the degree of the root node, and sweep through the set of values one at a time, in our case over the $[1, 1500]$ interval, the degree range of our crawl. Let $\delta : V \to \mathbb{R}$ be the function providing such value for a given node. For each value k, we compute the aggregate statistics (e.g. average latency) over the set of unit experiments $e \in E$ for which $\delta(root(e)) \geq k$, where E is the set of all unit experiments. We call such plot a *progressive plot*. Note that the value we would get for computing the statistics over the whole network corresponds to the first point in these graphs.

Graphs like this must be generated with care. For the statistics we compute, they work as if we were progressively removing terms from a weighted average, so we must make sure we are not removing the "relevant" terms too soon. The point is we *know* that the bias induced by lower size neighborhoods decreases the relevance of results, both because those are *much* more numerous, and because they do not afford much variability. As an example, think again about average latency: neighborhoods of size 1, 2, or 3 are likely to show very similar average latencies. Yet, their numerosity brings the average value down, making it similar over all protocols. If we show average latency by progressively excluding lower degree neighborhoods, however, the differences between the protocols become more evident.

Latency. Figure 3.6a shows the progressive plot of the average latency t_{avg} of direct mailing compared to our variants of HFLOOD. t_{avg} values for direct mailing can be computed analytically: for a neighborhood of size n, we have $t_{avg} = \frac{n-1}{2}$. This means direct

mailing does not really scale, and that can be seen from the plot: t_{avg} values grow very large, while for our protocols they seem to remain almost constant in comparison.

Note that the reason why we do not see a straight line for direct mailing in the plot is that we are doing a progressive plot by degree, not simply plotting t_{avg} by degree. The shape of the plots is heavily influenced by the skewed degree distribution of the network, and that is why the curve is irregular.

Figure 3.6b shows zoomed plots for HFLOOD variants and FLOOD. Performance benefits of using histories are clear: HFLOOD variants are faster than FLOOD across the plot, with latencies being up to three times smaller at some points of the graph (i.e. for some subsets of nodes). This is mainly due to the "anti-starvation" effect of histories described in Section 3.4.

As for the HFLOOD variants, the graph also shows the effectiveness of sorting components by size (MAXCOMP) when compared to picking them at random (RANDCOMP). Not only MAXCOMP performs better than RANDCOMP, but the latter actually performs *worse* than non-fragmentation-aware heuristics such as RANDOM.

Note that these graphs are rather "bumpy". This is a result of fragmentation: since it is the main (but not the only) bottleneck for diffusion speed in our protocols, average latency tends to correlate highly to it, particularly in those heuristics which are not fragmentation-aware. That is, in fact, the reason why bumps are located roughly at the same spot in the graphs, and why the graphs for the best performing protocols seem to resemble "flattened versions" of the worst-performing ones.

Finally, Figure 3.6c zooms into the three best-performing HFLOOD variations, and confirms both the effectiveness of ANTICENTRALITY—which performs better than RANDOM across the board—and MAXCOMP, which performs better than ANTICENTRALITY, except at the end. The reason, again, is fragmentation. From Figure 3.5 we can see that the large neighborhoods which get included at end of the graph are not very fragmented, in which case MAXCOMP performance tends to align with that of ANTICENTRALITY.

Load. Figures 3.7a and 3.7b show the progressive plots for average load over all nodes, as we trim by degree. The use of histories provide large savings in load, with HFLOOD generating on average 4.8 times less messages than FLOOD. Direct mailing is the cheapest of all protocols, since it generates no duplicates: for a neighborhood of size n, each node processes on average $\frac{2n}{n+1}$ messages. As for our approaches, there is no significant difference in load values, with MAXCOMP and ANTICENTRALITY performing slightly better than RANDOM.

An important point is that average load seems to grow with the size of the neighborhood. To get a closer look, we do a scatterplot in Figure 3.8a of average node load as a function of degree. Each point represents how much a node v pays, on average, whenever

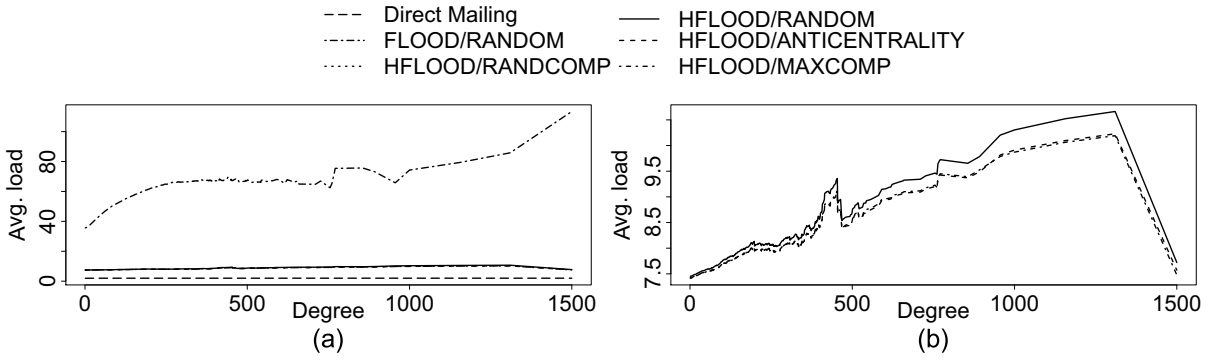


Figure 3.7: Average load.

an update emerges over $f^*(v)$. We can see that the load indeed grows fast. Fortunately, this increase comes with the size of neighborhoods, not of the network. Further, even at its maximum, the value is not too high, particularly if we consider the rate at which users post updates on OSN sites. The user with the most posts in Twitter, according to the Twitaholic website [119], posts one tweet a minute; the average user is likely to post much less. An average Facebook user posts 3 pieces of content *a day* [114].

Direct mailing is economical, but if we look at how many messages each update originator must actually push into the network, on average, to disseminate its update, we get the scatterplot in Figure 3.8b (shown for MAXCOMP). This brings us to another nice property of our protocol: it balances the load among those interested in receiving an update, shifting it away from the poster. Figure 3.8c shows a scatterplot of the coefficients of variation (Section 3.2.3) for messages sent and received by HFLOOD with MAXCOMP, as well as for direct mailing (where the coefficients of variation for messages sent and received are the same). Our approach clearly provides better balance.

Finally, we show in Figure 3.9 the progressive plots of how many message copies, on average, a single post entails. Note that this gives us a detailed account of how much redundancy our protocols generate: since direct mailing produces zero duplicates, its curve serves as a reference as well. We generate, on average, 3.79 times more traffic than direct mailing, but this value can become as high as 6.8 for some neighborhoods; if direct mailing is implemented in its naïve form, that is. If we were to implement it by using the point-to-point, DHT routing primitive provided in Graffi et al. [43], for example, this would change. While the overhead of our protocol depends on single neighborhoods and should remain stable as the system grows, the overhead for routing over a DHT grows with the size of the network, even if slowly. If we assume 100 million on-line users ($\frac{1}{5}^{th}$ of the Facebook userbase) and a Pastry DHT with settings as in [94], then $\log_{16} 10^8 = 6.64$, we would get the curve marked as "Direct Mailing/DHT" in the graph, in which the savings afforded by direct mailing disappear.

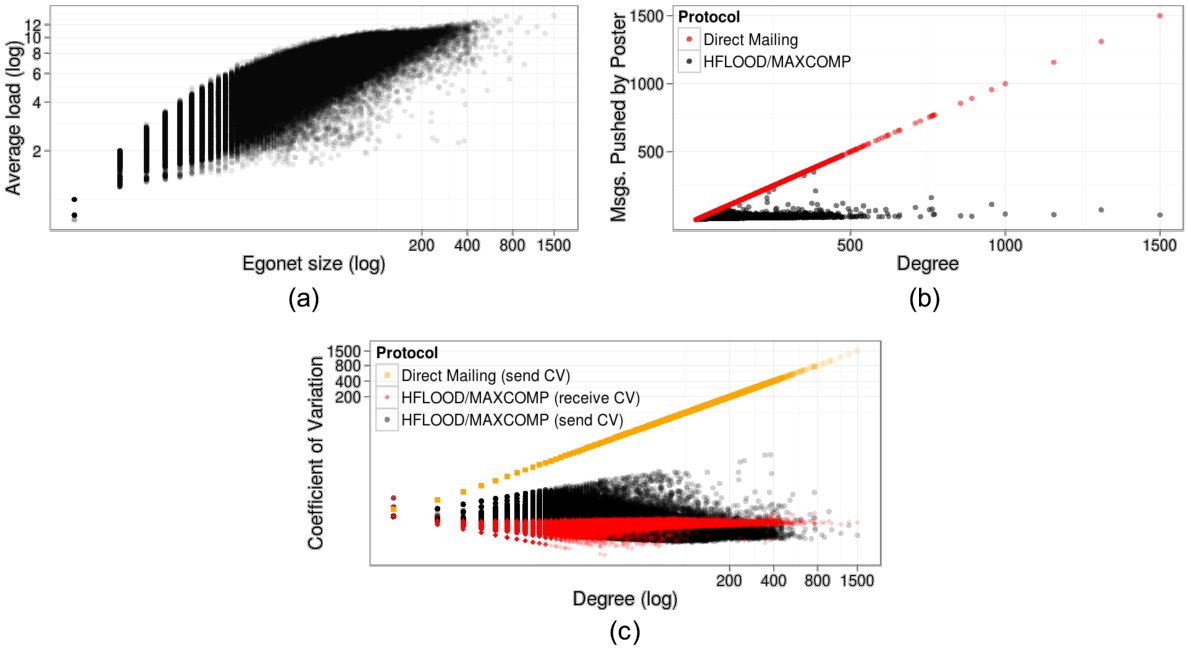


Figure 3.8: **(a)** average load, **(b)** messages posted by $root(e)$, **(c)** coefficient of variation for loads within experiments.

Our solution, therefore, incurs acceptable overhead and performance, effectively enabling the use of social overlays as dissemination media in static networks.

3.5.2 Impact of Churn

We evaluated HFLOOD with MAXCOMP and direct mailing under a simple churn model wherein we associate an on/off, discrete time stochastic process $Z_v(t)$ to each node $v \in V$, such that $Z_v(t) = 1$ if node v is alive at time t, or 0 otherwise. These processes can be modelled as a two-state Markov chain, with transition probabilities given by $p_{1,1} = p_{on}$, $p_{0,0} = p_{off}$, $p_{1,0} = 1 - p_{on}$, and $p_{0,1} = 1 - p_{off}$, where $p_{i,j}$ is the probability that the chain transitions into state j at time $t+1$ given that it was at state i at time t. Let $\mathbf{X}_v^{on}$ be the random variable defining the session length for a node v, then:

$$\mathbb{E}(\mathbf{X}_v^{on}) = \sum_{i=1}^{\infty} i p_{on}^i (1 - p_{on}) = \frac{1}{1 - p_{on}} - 1 \tag{3.3}$$

We can similarly define and derive $\mathbf{X}_v^{off}$ and $\mathbb{E}(\mathbf{X}_v^{off})$. We evaluate protocols under different p_{on} values, so as to get average session lengths of 0.5, 2, 4, and 6 hours. For the inter-session lengths we use a single p_{off} value for a 1 hour average. Asymptotic availability settings are similar to those used by Yao et. al. [128], from where our churn model derives. We use 1 second as the round duration for all protocols.

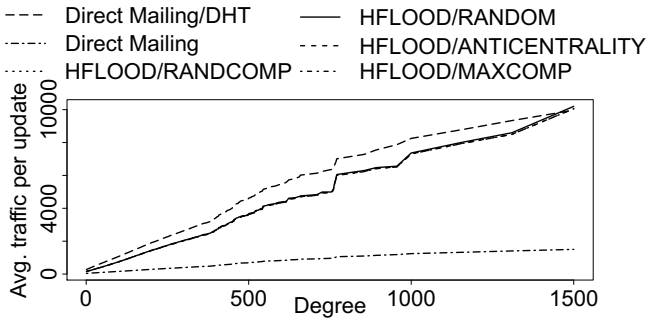


Figure 3.9: Average number of messages generated per update.

Churn introduces a new problem. Recall from Section 3.4 that a node v stops disseminating an update o only once it knows that all of its neighbors have received it. If churn is pessimistic, it might be that v has to wait for a very long time (possibly forever) for such condition to be satisfied. To remedy this situation, we introduce a *timeout parameter* t_{out} in our protocol: if a node v cannot contact any node that has not yet received o for more than t_{out} seconds, it stops disseminating o. For our simulations, we use a fixed t_{out} value of 30 rounds (i.e. 30 seconds, in this case).

Since churn simulations are much more expensive, we had to constrain our evaluation to a single unit experiment per node. We are still, however, running 72,303 unit experiments for each combination of parameters and protocols.

Residue. We argue that, contrary to intuition, residue is not a really important metric for comparing our push protocols, even under churn. The reason is that session lengths under the churn model are so large when compared to the average experiment duration that, for the purpose of a single unit experiment—focusing on a single update dissemination—it is as if the network remained static. Since over a static network the number of nodes reachable from the update originator is the same regardless of the protocol, the residue is also the same. This is confirmed in the progressive residue plot of Figure 3.10a: although graphs become noisy at larger degree values as we trim out more and more unit experiments from the average residue computation, the overall point stands, with values almost converging up until around 400.

A way to quantify if the dynamism left in the network actually produces a measurable impact on residue is by "correcting" it. We define the *corrected residue* of a unit experiment e associated to update o to be the percentage of nodes *with uptime larger than zero over* e which did not receive o by the time e finishes. Figure 3.10b shows a progressive plot of corrected residues for HFLOOD and direct mailing under the various churn settings we tried. HFLOOD produces less corrected residue than direct mailing *over all settings*. Other churn and timeout settings are likely to exacerbate these effects, and

that is something we intend to investigate next.

Latency and Load. Churn transforms the underlying social network by removing nodes from it. Therefore, we need to assess whether these transformed networks create adverse effects, be it on load (by causing certain nodes to grow in importance) or latency (by increasing the lengths of dissemination paths). Figure 3.11a shows progressive plots for latency over all average session length settings, trimmed by degree, comparing HFLOOD and direct mailing. The overall point of this graph is: churn is effectively shrinking the neighborhoods, and this translates into a considerable improvement to direct mailing, but our protocol remains faster.

We zoom into the progressive latency plots for our protocol in Figure 3.11b, and compare its latency figures with what we had before churn. Even if the initial averages under churn are smaller, numbers grow larger at some points, likely due to some unlucky or unforeseen structural changes. Performance remains however generally consistent, with increases only at the lowest availability levels.

Finally, Figure 3.11c, shows progressive plots for load for direct mailing and our protocol, as well as for our protocol without churn. The curves for our protocol are similar, resembling translated and slightly flattened versions of the static load curve at the top. By and large, the roles played by the nodes in dissemination in terms of relative importance over their neighborhoods remain similar, with curves flattening as average uptime plummets. This is to be expected: the lower the availability, the less nodes in the network, and the more fragmented the remaining neighborhoods become. As fragmentation increases, our protocol smoothly converges into direct mailing, both in terms of latency and load.

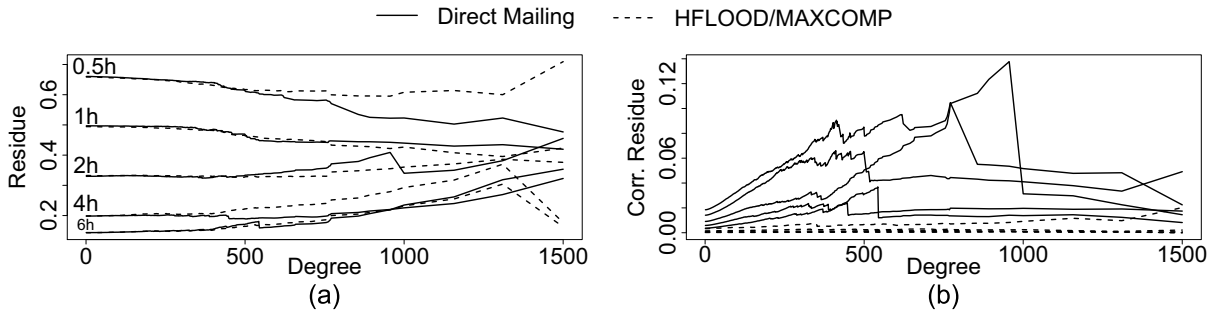


Figure 3.10: **(a)** residue and **(b)** *corrected residue* for HFLOOD and direct mailing.

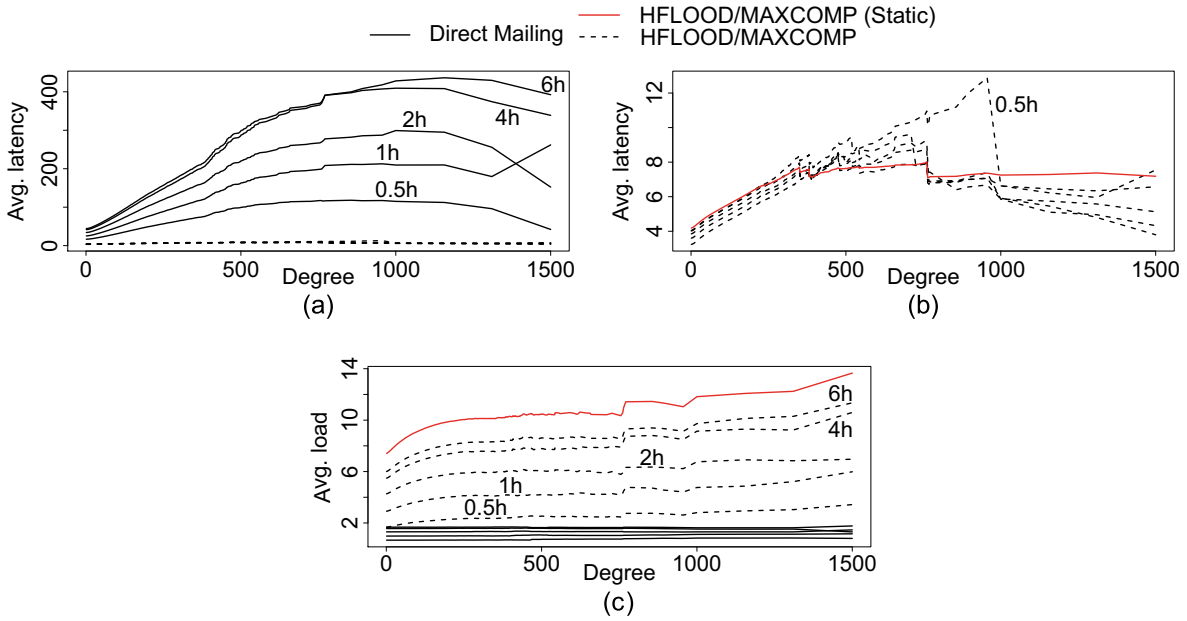


Figure 3.11: **(a), (b)** average latency and **(c)** load of HFLOOD and direct mailing.

Dataset	**Vertices**	**Edges (undirected)**
Our crawl	72 303	1 508 283
Orkut	3 072 441	117 185 083
Regional Network A	3 097 166	28 377 481
Regional Network B	2 937 614	24 236 701

Figure 3.12: Summary statistics of large datasets.

3.6 Evidence for Generalization

In this section, we provide further evidence that our findings on ego network structure and their interaction with our dissemination protocols – particularly those leading to improved latencies – generalize to datasets beyond those of our small Facebook crawl. To that end, we reproduce the same static experiments of Section 3.5 in three large datasets, and analyse the differing latencies across progressive plots. The datasets we use are the Orkut crawl of Mislove et al. [69], and the two Facebook regional network crawls released by Wilson et al. [125], which we refer to as regional networks A and B.

All datasets have been originally acquired by snowball sampling, with the regional network samples being complete crawls. Size statistics are provided in Table 3.12 (our crawl is included for comparison). In order to increase the significance of results, unit experiments are repeated 100 times instead of 10 as before. Figure 3.13 shows progressive

plots for all datasets and, in every account, they confirm our results. The performance gains obtained by the use of our custom heuristics as opposed to random selection vary significantly, but are either in line with, or significantly exceeding the gains we have observed with our own small crawl. Indeed, over the Orkut dataset MAXCOMP provides latency improvements of up to 120% for some subsets of ego networks, whereas with the Facebook crawls figures are more modest, at a maximum improvement of 68% for network A, and 22% for Network B. In comparison, savings for our crawl were no larger than 23%. Further, in all datasets, we have HFLOOD coming last, with ANTICENTRALITY in second, and MAXCOMP as the best all-around performer.

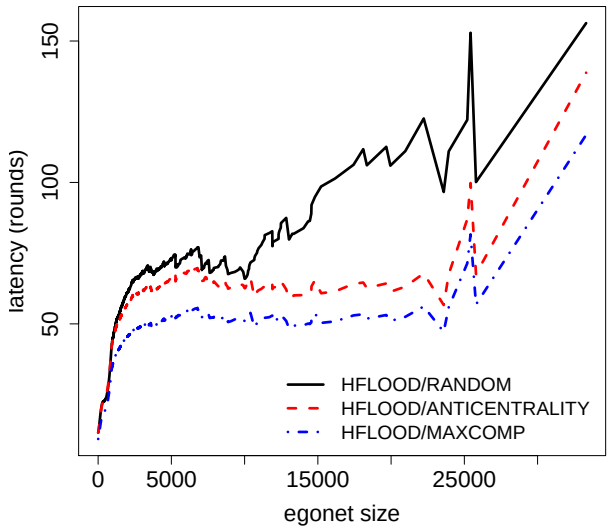
Differences in gains can be explained by the presence of larger ego networks in the Orkut graph (Facebook caps users at 5 000 friends, while Orkut had users with more than 30 000 friends), which favours fragmentation, and a much higher edge density (the Orkut graph is almost 5 times more dense than the Facebook graphs), which we hypothesize translates into more uneven clustering.

3.7 Related Work

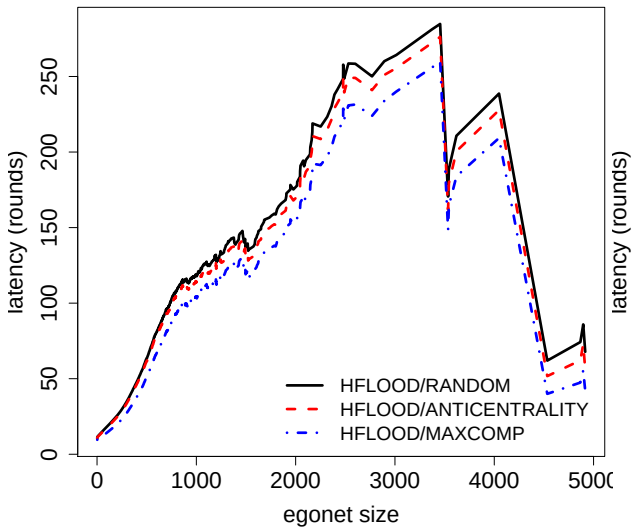
Decentralized OSNs. The *decentralized server* OSNs we have discussed in Section 2.1.2-1 employ direct mailing when pushing updates across servers. Since server-to-server communication graphs are essentially clustered versions of social graphs – i.e., they bear similar topological characteristics to social overlays – our approach could serve as a faster, drop-in alternative for data dissemination among servers (i.e. for newsfeed) which allows data to flow only along trusted servers.

Our protocols could also find application in existing P2P OSNs. PeerSoN [18] and LifeSocial.KOM [43], for example, do not offer protocols for fast data dissemination among nodes that are online. Safebook [25] replicates data at friends to increase availability and preserve privacy. The protocol which is used to implement replica consistency, however, is not described. Again, our protocols could be used to that end. Finally, Cachet [76] specifies a fast bootstrapping procedure for downloading news from friends once a node logs in, but it does not specify how such news are to be handled when nodes are already online. Our protocols could, once again, be a good fit for that role.

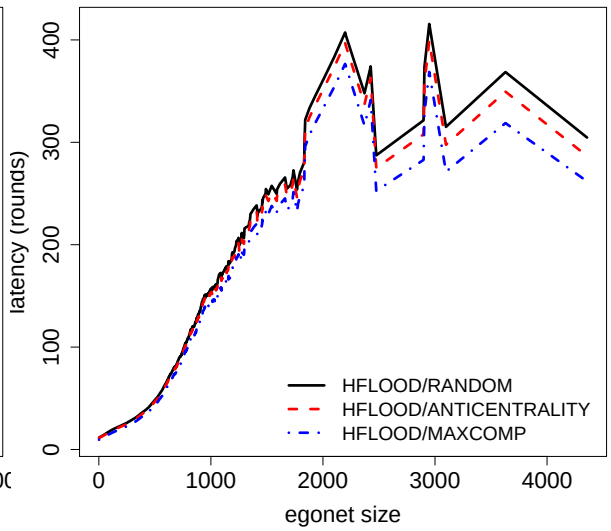
P2P pub/sub. The dissemination of updates can be seen as a pub/sub problem wherein every user v defines a topic, to which both v and her friends are subscribed, and in which all participants are potential publishers. An alternative to using social overlays would be, then, the use of topic-connected overlays [22]: while not respecting social constraints in general, these overlays would at least allow information to be disseminated without leaving the circle of subscribers (i.e., updates can be disseminated from a user to its



(a) Orkut.



(b) Network A.



(c) Network B.

Figure 3.13: Large dataset latency plots.

friends using only friends).

Unfortunately, efficiently building and maintaining these overlays in a decentralized fashion is known to be difficult. Recent solutions such as Vitis [88] and Magnet [40] still suffer from limitations that are significant in our context, as they are unable to guarantee topic connectivity. Olteanu and Pierre [78] show that poor topic connectivity in SpiderCast [22] can be mitigated (but not overcome) by subscribing a small number of random nodes to topics. Although this mitigates the problem, it introduces untrusted nodes into dissemination, and we therefore do not consider it to be an acceptable solution as well.

Quasar [127] is an overlay-independent pub/sub system for social networks. Nodes use attenuated Bloom filters to create "gravity wells" for topics of interest. Publishing amounts to performing parallel random walks on the overlay, with walkers being "pulled" into and then "expelled" from gravity wells. It relates to our approach in that random walks and push gossip protocols are similar, and because it piggybacks histories of previously visited subscribers into walkers. However, the latter is done in Quasar to avoid loops, while we do it to increase efficiency under differing clustering conditions.

GoDisco [26] is a hierarchical topic-based pub/sub system which exploits social communities to route messages and, as in our approach, uses social overlays. The authors share our view on their potential benefits, but are most interested in homophily: nodes with similar interests tend to cluster, which means that social overlays might provide an efficient dissemination medium for their pub/sub system. Like our approach, GoDisco embodies topology awareness in its routing protocol: it uses social triads [89] to counter duplicates. We instead rely on message histories, which we believe provides more robustness, and embody different topological awareness techniques to speed up dissemination.

Gossip protocols. The ANTICENTRALITY selection heuristic we put forth on Section 3.2.2 is similar to directional gossip [59], particularly in that nodes with reduced connectivity are given priority. Apart from the details of computing weights, the main difference w.r.t. our approach is that whereas directional gossip assigns a "thresholding weight" above which it switches from gossip to flooding, our heuristic biases selection based on such weights instead. The rationale is that if all nodes have similar characteristics, our protocol behaves like uniform gossip; otherwise it adapts, subject to the particular topological conditions of the neighborhood over which it disseminates.

Our protocols can be related to *biased gossip* approaches like BEEP [11]. The inputs and rationale for biasing are however different: BEEP heuristically disseminates news to nodes that might find them interesting, while adapting the fan-out so that popular news spread and unpopular ones die. Our biasing is instead to reduce duplicates and latency, and to compensate for the non-uniformity of social graphs.

Finally, while our protocols share their general operation with traditional gossip [28], they are particular in that nodes are only allowed to talk to friends. This constrains the protocol to a set of arbitrary, richly varying graphs—the social neighborhoods. General reliability results [51] are then not expected to hold and, given the complexity and variability of these graphs, extensive empirical evaluation is required instead.

3.8 Discussion and Outlook

This chapter has discussed the use of push-based gossip protocols for the dissemination of updates over social networks. The problem is of interest not only because it fits our vision for a P2P OSN based on *social overlays*, but also because it might find broader application into existing decentralized OSN proposals which, up until now, have relied either explicitly or implicitly on the use of direct mailing, or the naïve use of DHTs.

We have shown the caveats of applying gossip protocols to social networks by quantitatively demonstrating the extent to which classical protocols such as Demers' rumor mongering become inefficient under their widely non-uniform clustering characteristics. We then introduced a novel gossip protocol capable of adapting to (and leveraging off) such non-uniformity. This protocol is based on three key principles: the use of *message histories*, an *anticentrality* selection heuristic, and *fragmentation awareness*. We have shown through simulations that these principles yield benefits, given that our protocol significantly improves over mainstream gossip protocols and direct mailing.

Although our initial evaluation was limited in size, we provided, at the end of the chapter, further evidence that our findings on the interplay between egonet structure and gossip protocols under a static network assumption generalize, by showing that our custom heuristics provide performance benefits under larger datasets as well.

The space for exploration of dissemination heuristics is far from over. Promising avenues for future exploration include the generalization of the notion of fragmentation to community structures [55] (communities, which can be seen as disjoint sets of densely connected nodes for which cuts across sets are sparser in comparison, generalize the notion of connected components to an extent), and the use of Metropolis-Hastings unbiased random walks [41], which could be effective in countering the effects of clustering.

Finally, we have performed some limited analysis under churn, showing that our protocol performs acceptably under a number of churn conditions. Yet, this last part of our study is rather limited, both in modelling and analysis, for two main reasons:

1. *Simplistic churn model.* The churn model we adopt is rather simplistic, in that it fails to capture both peer heterogeneity and the characteristic heavy-tailed nature of session and inter-session lengths of P2P systems [99, 108]. A more realistic

model which caters for such characteristics would likely lead to massive ego network partitioning and performance degradation or, at the very least, much larger residues.

2. *Large residues.* Our simulations stop at the moment at which protocol timeouts expire. But at that point residues are rather large, even with our optimistic churn model. Indeed, for the most realistic setting for average inter-session lengths we used in Section 3.5.2 – 0.5h, which in accordance with the measurement study of Saroiu and Gummadi [99] – residues are near 70%. This means that we have no idea of what kinds of latencies are to be expected for 70% of the userbase under churn. These issues give raise to a number of questions:
   (a) what happens to these nodes? Would they be reached if we were to set the timeouts to larger values?
   (b) How would residue decay as we increase timeouts? What happens to latency then?
   (c) If we were to set our timeouts to infinite, how long, at most, would nodes have to wait before getting their updates? Does latency become unacceptable to anyone?
   (d) If the churn model were more realistic and that indeed led to massive partitioning, would there be any nodes that simply never get their update?

As we will see in the next chapter, these questions lead to the unravelling of a set of more fundamental and general problems regarding the behavior of social overlays under churn, and which reveal characteristics of social overlays that go beyond the application of any given protocol.

Chapter 4

Churn and Communication Delays[1]

Omnis nimium longa properanti mora est.
(Every delay is too long to one who is in a hurry.)

– *Seneca*, Agamemnon.

In the last chapter, we have explored the problem of disseminating updates over ego networks. The chapter ended with some important open questions about the behavior of our protocols under churn, and these questions, in turn, have led us to the discovery of the more important, and more general problem we tackle in this chapter. Essentially, instead of looking at the performance characteristics of a particular protocol (ours or otherwise), we focus on a simpler, yet more fundamental question: *how fast can information flow among participants of a social network when it is subjected to churn?*

Our insight to the study of this problem comes from the realization that SOs are quite different from other kinds of overlays, in that they cannot be reorganized to cope with churn, and this makes them susceptible to transient partitioning. Indeed, in "synthetic" overlays such as DHTs, node joins and leaves are dealt with by a "stabilization protocol" [109] which causes connections among neighbors to be rewired. In performing this rewiring process, however, nodes are free to choose how to establish connectivity between the remaining nodes: constraints are imposed only for optimizing system parameters, or respecting certain overlay invariants. Social overlays do not enjoy the same luxury, as their defining property is that connectivity is allowed only among 1-hop friends. Breaking this constraint would break their essence. However, this very same constraint severely reduces the "opportunities for healing" in the presence of churn, and might lead to inherently slow, inefficient communication.

[1] An earlier version of this chapter has been presented in [67].

And this is there where it all connects: as we will show at the end of the chapter, beyond the initial set of connected nodes that are online when an update gets posted, it is the overlay dynamics (social graph plus churn) that dictates the speed at which dissemination occurs, not the dissemination protocol. Yet, the problem we tackle here is deeper than just update dissemination, in that we establish lower bounds on pairwise communication speed for *any* protocol.

Using our running problem – the dissemination of updates over ego networks – the general problem we study can be illustrated as follows. Suppose some new update were to be injected into the system by a peer u, to be disseminated to all of u's friends, including some friend $v \in f(u)$. Due to their usage habits, however, it so happens that u and v are never online at the same time. To relay the message to v, therefore, the system has to find a third peer w to broker the message, such that w is online at the same time as u, and then later at the same time as v. Further, w must be friends with both u and v. Clearly, a peer w satisfying such conditions might not exist, which means the message might have to traverse a chain of friends acting as brokers before it can actually reach v – assuming it ever does. Moreover, the longer this chain, the greater the *delay* experienced by the receiver v, as one broker must wait for the next one to come online before passing the message forward.

In the study we conduct in this chapter, we focus on the notion of delay as the metric determining the practical usability of the system. Intuitively, if a message from a user takes "too long" to be delivered, the system may become unusable for practical purposes. In Section 4.1 we add some information to our system model, while in Section 4.2 we formally define the problem and the metrics – end-to-end delay and receiver delay – we use.

In this context, the research questions at the heart of this chapter are two:

1. *Does the constraint of using only "friend links" limit the applicability of social overlays?*
2. *If so, how can we identify strategies to mitigate the effects of churn on social overlays?*

We investigate the answer to these questions by putting forth two contributions.

First, we analyze and compare a pure SO-based solution against a mainstream server-based solution (e.g., representative of those used by online social networks), using simulation from real datasets combined with a well-known availability model [128], which is much more realistic than the simple model we adopted in Section 3.5.2. As we show in Section 4.3, 1% of the receivers experience a delay greater than 3 hours. This result may appear good, and it is for applications that are not sensitive to delay. Also, 1%

may seem at first a small percentage. However, when considering large-scale, interactive applications such as Facebook, this result would translate into an unacceptable user experience for millions of users. Since the culprit is churn, we also compare against a solution where a variable fraction of the nodes composing the SO are assumed to be fixed. This allows us to glance at opportunities for improvement that could be easily harvested by, say, assuming that some of the peers are willing to rely on cloud computing services to provide a continuous online presence. Note that this would still preserve decentralization, as well as many of the desirable security properties provided by SOs (i.e. nodes would still connect only to authenticated friends, and would still be able to trust those).

We argue that the simulation study above is a valuable indication that indeed churn *is* a concern for social overlays, therefore partially answering question 1. Nevertheless, a more articulate answer, as well as the answer to question 2, both require a level of analysis that can hardly be supported by a simulation study alone, due to *i)* the sheer size of the networks to be considered when working with social network datasets, *ii)* the large parameter space inherent to modeling availability, e.g., the many ways in which availability can be mapped onto network nodes; and *iii)* the difficulties in working with heavy-tailed distributions, which severely increase the number of repetitions required to obtain meaningful results.

Therefore, our second contribution, detailed in Section 4.4, revolves around an analytical model of dissemination over social overlays. Specifically, we present:

1. a hybrid analytical/simulation framework which enables us to determine, at a practical cost, an upper bound to dissemination delays.
2. algorithms for identifying key graph substructures that, as we show, are responsible for a substantial fraction of the observed delays.

The chapter is completed by a concise overview of related works in Section 4.6, and by final remarks in Section 4.7.

4.1 System Model

In this section, we complement the system model of Section 2.3 with information relevant to this chapter.

4.1.1 Availability Model

Our system is a P2P network with $|V|$ participants, where each user u can be either logged in (online) or out (offline) at any given time instant t. In this context, we were faced with two options when it came to specifying online/offline behavior.

Since, as we mentioned in Section 3.8, adopting a realistic availability model is important, the first option we considered was to use one of the publicly-available traces from measurement studies of P2P systems (e.g., [108]) and, more recently, from instant messaging applications (e.g., [117]). Although directly applying real datasets is appealing, it is also a challenge in itself in our context, since: *i) size*: the social graphs required for our experiments are much larger than the networks captured by availability traces, and it is unclear how to employ the latter to simulate the former without introducing subtle correlations (and therefore bias) into the results; *ii) duration*: even the longest traces available are too short to accommodate a sufficient number of decorrelated experiments; *iii) noise*: most traces contain high rates of permanent departures (e.g., due to peer aliasing [108]), which affect the statistical properties of peer sessions. These factors taken together have the power to introduce enough bias into our experiments so as to render results questionable.

We therefore chose a second option, namely, the use of a well-known synthetic churn model, hereafter referred to as the Yao model [128]. This model is derived from the statistical behavior of measurement studies in the context of P2P systems, and therefore provides us with a reasonable approximation of real peer behavior, while avoiding the aforementioned problems associated with the use of the actual real data.

The Yao model associates an *alternating renewal process* to each node $u \in V$. In these processes, both session lengths (online time between a login and the next logout) and inter-session lengths (offline time between a logout and the next login) of a node u are given by random variables $\mathbf{X}^u_{on}$ and $\mathbf{X}^u_{off}$, drawn from node-specific distributions F^u_{on} and F^u_{off}.

Although the analysis tools we later develop are independent of these distributions, we consider here one of the instantiations of the model proposed by [128], in which both F^u_{on} and F^u_{off} are shifted Pareto distributions $\mathsf{pareto}[\alpha, \beta]$ with parameters α and β and average $\beta/(\alpha - 1)$. Their cumulative distribution function (CDF) is given by:

$$\mathsf{pareto}[\alpha, \beta](x) = 1 - (1 + x/\beta)^{-\alpha}, x > 0, \alpha > 1$$

This distribution is heavy-tailed, being known for accurately modeling the skewed availability patterns characteristic of P2P systems [108]. Let the average session and intersession lengths for node u be $\ell^u_{on} = \mathbb{E}(\mathbf{X}^u_{on})$ and $\ell^u_{off} = \mathbb{E}(\mathbf{X}^u_{off})$, respectively. Then, the *asymptotic availability* of u is [128]:

$$a_u = \lim_{t \to \infty} \mathbb{P}(u \text{ is on-line at time } t) = \frac{\ell^u_{on}}{\ell^u_{on} + \ell^u_{off}} \tag{4.1}$$

4.1.2 Heterogeneity

Heterogeneous user availabilities are modelled once again as in Yao et al. [128], by drawing the ℓ_{on} and ℓ_{off} shown in Equation (4.1) from yet another pair of shifted Pareto distributions. For session lengths, we use $\mathsf{pareto}[3, 1]$ yielding $\mathbb{E}(\ell_{on}) = 0.5)$ hours. For inter-session lengths we use $\mathsf{pareto}[3, 2]$, yielding $\mathbb{E}(\ell_{off}) = 1$ hour. These averages, again, coincide with those of measurement studies found in the literature [99].

With those in place, we set a session length distribution for each node u to be $F^u_{on} = \mathsf{pareto}[3, 2 \cdot \ell_{on}]$ and the inter-session length distribution $F_{off} = \mathsf{pareto}[3, 2 \cdot \ell_{off}]$. Note that, under those settings, $\mathbb{E}(\mathbf{X}_{on}) = \ell_{on}$ and $\mathbb{E}(\mathbf{X}_{off}) = \ell_{off}$.

Each node $u \in V$ is then associated to a pair of distributions (F^u_{on}, F^u_{off}). We call the set A_G of all of these pairs, one per node, the *availability assignment of graph* G.

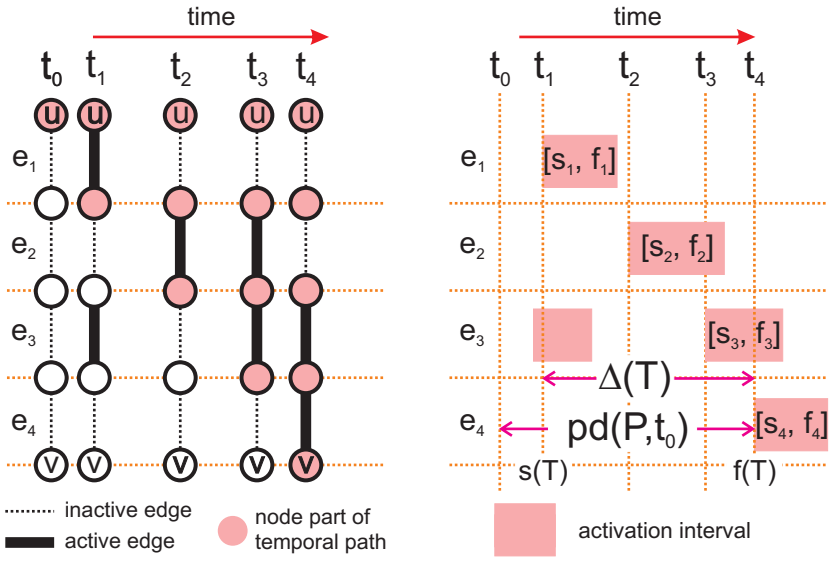
4.2 Problem Statement

Given an arbitrary connected graph $G = (V, E)$ and its availability assignment A_G, we want to understand how long it takes for a message to propagate between an arbitrary pair of vertices u, v. A lower bound for such *communication delay* can be obtained by studying the *temporal connectivity* of G under A_G. The problem is deeply connected to the study of *temporal paths*, defined next. Note that this problem formulation is very general, as G can represent an entire social network just as well as some subgraph of it, such as an ego network.

We say that an edge $e = (u, v) \in E$ is *active* when both u and v are online at the same time. Each edge e is associated to a infinite set $I(e)$ of disjoint *activation intervals*, which represent the periods of time in which e is active. An interval $[s, f] \in I(e)$ starts at time s and finishes at time f.

Let $P = \{e_1, \cdots, e_n\}$ be a path between u and v in G. We say that a *temporal path* $T = (P, S)$ exists between nodes u and v if and only if there exists a sequence of intervals $S = [s_1, f_2], [s_2, f_2], \ldots, [s_n, f_n]$ such that $[s_i, f_i] \in I(e_i)$ $(1 \leq i \leq n)$ and $s_i \leq f_{i+1}$ $(1 \leq i < n)$. In other words, a temporal path only exists if there is a sequence of activation intervals for which the edges of P are activated one after the other, allowing a message to be propagated from u to v. The idea is illustrated in Figure 4.1a: as the edges become active in sequence, a message can flow from node u towards node v. We say that the temporal path T *materializes* the underlying path P or, equivalently, that T is a *materialization* of P.

For a given temporal path T, we define $s(T) = s_1$ and $f(T) = s_n$ as the time instants at which the first and last edges of T get activated, respectively. The *duration* $\Delta(T)$ of a temporal path can then be defined as $f(T) - s(T)$. The idea is illustrated in Figure 4.1b,



(a) Materialization. (b) Corresponding time diagram.

Figure 4.1: Two views of a temporal path.

which shows the activation intervals of Figure 4.1a as they develop in time. The duration of the temporal path in the diagram is given by $\Delta(T) = t_4 - t_1$.

Note that many temporal paths form between u and v over time, but we are only interested in those that form after u has actually something it wants to transmit to v. The *earliest* time instant when this may happen is when u logs in. Therefore, we choose to (pessimistically) measure the communication delay w.r.t. the login events of u, rather than the duration of the temporal paths connecting u and v. This is equivalent to assuming that a node u sends a message as soon as it logs in.

The idea is illustrated again in Figure 4.1b, where $\mathbf{pd}(P, t_0)$ represents the communication delay along the temporal path materializing P, measured w.r.t. the login event of u at t_0 instead of the beginning of the first activation interval at t_1. The communication delay, *in this case*, is given by $t_4 - t_0$.

Formally, let t_0 be a time instant at which u has logged in. For a given path $P \in G$ between u and v, let $\mathcal{M}(P, t_0)$ represent the set of all temporal paths T that materialize P, such that $s(T) > t_0$. We define the *path delay* of P w.r.t. t_0 as:

$$\mathbf{pd}(P, t_0) = \min\{f(T) - t_0 \colon T \in \mathcal{M}(P, t_0)\}$$

This notion captures the communication delay among u and v over a single path. In general, however, if several temporal paths materialize between u and v, the path that actually carries the message sent from u is the one that forms first. Based on this intuition, we can now define the *end-to-end delay* $\mathbf{ed}(u, v, t_0)$ among two nodes u and v w.r.t. t_0 as

the smallest path delay, also w.r.t. t_0, over the set $\mathcal{P}(u, v)$ of all paths connecting u and v in G:

$$\mathbf{ed}(u, v, t_0) = \min\{\mathbf{pd}(P, t_0) \colon P \in \mathcal{P}(u, v)\}$$

For a given set of time instants $L = \{t_0, \cdots, t_n\}$ representing logins of u, we can compute the *average end-to-end delay* w.r.t. v as:

$$\mathbf{aed}(u, v, L) = \frac{1}{|L|} \sum_{t_i \in L} \mathbf{ed}(u, v, t_i) \tag{4.2}$$

The end-to-end delay is, by nature, a network-level metric. From an application and user perspective, however, what really matters is the delay *experienced by the receiver* of a message. For instance, a receiver that logs in very infrequently would not really care whether a message was sent a long time ago, as long as it is received shortly after login.

We therefore define a second, application-level metric based on the fraction of time that the receiver spends online, i.e., with a chance of getting the message. Let $\mathsf{up}(v, t) : V \times \mathbb{R} \to \mathbb{R}$ be the function yielding the *total uptime* of node $v \in V$, i.e., the accrued time v spent online until some time instant t. Now, let t_0 be an instant when a sender u logs in. We define the *receiver delay* $\mathbf{rd}(u, v, t_0)$ between u and v w.r.t. t_0 as the total time that v spends online between the sending and the receipt of the message sent by u, i.e., in the interval $[t_0, t_0 + \mathbf{ed}(u, v, t_0)]$. Using the up function, we can express this as:

$$\mathbf{rd}(u, v, t_0) = \mathsf{up}(v, t_0 + \mathbf{ed}(u, v, t_0)) - \mathsf{up}(v, t_0) \tag{4.3}$$

Similarly to end-to-end delay, we can define an *average receiver delay* w.r.t. a set of login events $L = \{t_0, \cdots, t_n\}$ of node u as:

$$\mathbf{ard}(u, v, L) = \frac{1}{|L|} \sum_{t_i \in L} \mathbf{rd}(u, v, t_i)$$

The two metrics, **ed** and **rd**, are related. By recalling from Equation (4.1) that the asymptotic availability a_v of node v expresses the average fraction of time that v spends online, we can approximate **rd** as:

$$\mathbf{rd}(u, v, t_0) \sim a_v \cdot \mathbf{ed}(u, v, t_0) \tag{4.4}$$

The practical impact of this relation is that the analytical model in Section 4.4 can focus on end-to-end delay, as receiver delay can be derived from it. Before illustrating our model, however, we turn our attention to determining whether churn poses a threat to the practical application of social overlays.

4.3 Can Social Overlays Support Communication Under Churn?

A simple approach to answering this question is by means of plain, discrete-event simulations. If we simulate the churn model and collect enough **ed** and **rd** values, we can obtain **aed** and **ard** values that are "close enough" to the "population averages" for these values. These simple simulations entail simulating all events in all nodes in the graph. We call them *full simulations*, in contrast with the smaller-scale simulations we exploit in Section 4.4.

We now describe in detail how we designed these simulations, along with our specific experimental setting. Although the latter is limited by the relatively small scale of the experiment and the specific choice of parameters, the results point to the fact that a pure P2P social overlay fails to perform acceptably under churn. This calls for a more comprehensive study, for which full simulations do not represent an adequate tool, motivating the contribution put forth in Section 4.4.

4.3.1 Simulation Design

To simplify the discussion, we focus this subsection on the simulations for the average end-to-end delay **aed**. Later, we discuss briefly how to use the very same method to directly obtain values for the average receiver delay **ard**.

Estimating **aed** by full simulations amounts to running the churn model in a discrete event simulator and directly observing a large number of **ed** values. The way this simulation works is described in Algorithm 1. Initially, we set all nodes to "logged out" (line 4). We then remove the bias introduced by this initial state by running the churn simulation for a *burn-in* time γ (line 11) until the number of nodes logged in stabilizes. We have empirically established $\gamma = 48$ hours to be sufficient for most practical cases, where we observe stable-state parameters similar to what described in [128].

The simulation progresses by identifying which nodes are "reachable" from the source u. A node v becomes reachable when the first temporal path starting from the source reaches it. The first node to become reachable is the source u itself. This happens when u logs in for the first time (line 12–13).

Then, for each subsequent login event (from any node), we check whether there are any nodes that, previously unreachable from u, are now reachable, using Algorithm 2. We start a breadth-first-search (BFS) from each member v of the set of nodes $Q = R \cap U$ reachable from u and currently online. The BFS visits a node w if it is online ($w \in U$) and has not been visited before ($ed[w] = \perp$). When w is visited, $ed[w]$ is set to the current time and w is added to both Q and R.

The **ed** values generated for each experiment are our samples taken from the under-

Algorithm 1: FullSimulation

Input: Graph $G = (V, E)$, number of repetitions n, burn-in time γ, source node u.

```
1  forall the v ∈ V do aed[v] ← 0                                  % avg. comm. delay
2
3  for i ← 1 to n do
4      U ← ∅                                                       % nodes currently logged in
5      R ← ∅                                                       % nodes reached from u so far
6      forall the v ∈ V do ed[v] ← ⊥                               % comm. delay
7      while R ≠ V do
8          e ← nextSimulationEvent()
9          if e.type = Login then U ← U ∪ {e.node}
10                           else U ← U − {e.node}
11         if e.time ≥ γ and e.type = Login then
12             if e.node = u and u ∉ R then
13                 R ← {u}
14                 ed[u] ← e.time
15             if u ∈ R then
16                 Reachable(G, R, U, ed, e.time)
17     for v ∈ V do
18         aed[v] ← aed[v] + (ed[v] − ed[u])
19 forall the v ∈ V do aed[v] ← aed[v]/n
20 return aed
```

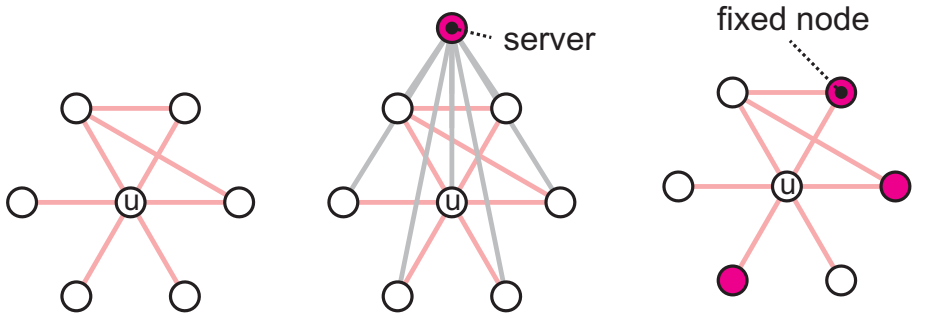
Algorithm 2: Reachable

Input: $G = (V, E)$, R, U, ed from Alg. 1 and time t.

```
1  Q ← R ∩ U
2  while Q ≠ ∅ do
3      v ← extract(Q)                                              % Remove in FIFO order
4      forall the (v, w) ∈ E do
5          if w ∈ U and ed[w] = ⊥ then
6              ed[w] ← t
7              R ← R ∪ {w}
8              Q ← Q ∪ {w}
```

Metric	Value
Number of Vertices	137892
Number of Edges Edges	1460043
Avg. Clustering Coefficient	0.112
Average Egonet Size	197.5

Table 4.1: Statistics for ego network sample.



(a) Ego network G_u. (b) G_u with server node. (c) G_u with 3 fixed nodes.

Figure 4.2: The three types of experiments.

lying, unknown distribution of **ed** values between u and v. These get accumulated in an array (line 17) which, at the end of the whole procedure, is divided by the number of repetitions to yield the **aed** values.

The average receiver delay **ard** can be obtained by keeping track of the accrued uptimes for each node, so that we can have an implementation of the $\mathsf{up}(v,t)$ function required by Equation (4.3). When the source u logs in for the first time (line 13), we take a snapshot of the uptimes of all nodes, and store them in an auxiliary array. These are the values of $\mathsf{up}(v,t_0)$ of Equation (4.3). Then, whenever a node v becomes reachable at instant t_1 we compute $\mathbf{rd}(v,t_0) = \mathsf{up}(v,t_1) - \mathsf{up}(v,t_0)$.

4.3.2 Experimental Setting

We perform full simulations over ego networks extracted from the Orkut crawl in [69]. The original graph contains 3 million vertices (3 million ego networks), 223 million (undirected) edges, and has an average clustering coefficient of 0.171. We single out 700 of these ego networks uniformly at random. This sample includes around 5% of the vertices in the graph, and its average clustering coefficient is slightly smaller, as shown in the statistics in Table 4.1.

We chose ego networks because *i)* simulating a whole social network would be pro-

hibitively expensive. Ego networks allow us to focus on a relatively smaller problem, for which simulations are nonetheless already expensive, and *ii)* they allow us to investigate what the answers to the questions we posed in Section 3.8 could be.

For each ego network G_u in our sample, we:

1. select a node $v \in G_u$ uniformly at random;
2. perform 100 000 full simulation runs taking v as a source;
3. estimate **aed** and **ard** from v to all other nodes in G_u.

This yields delay estimates for a total of 137 260 source/destination pairs, coming from 7×10^8 full simulation runs. The high number of repetitions – another complication of the approach based purely on simulations – is required because the underlying distribution of the **ed** and **rd** metrics is heavily skewed (possibly heavy-tailed), meaning that it is not possible to obtain an accurate approximation of the average unless we run a *very large* amount of repetitions, so that "rare events" that impact the average have the opportunity to unfold [33].

As a baseline for comparison, we adopt a hypothetical "server-based" approach which mimics current centralized solutions like Facebook. To that end, we add a special node to the graph which is connected to all other nodes, and which is also available (up) 100% of the time. This allows us to understand, for a given availability assignment, what is the best achievable end-to-end delay. Needless to say, the receiver delay under such a setting will always be equal to zero. This is illustrated in Figure 4.2b, where we add the special node to the ego network depicted in Figure 4.2a.

We also perform another kind of experiment, where we select a percentage of the nodes in the overlay uniformly at random and make them *fixed*; i.e., we make them always online. Note that, as illustrated in Figure 4.2c, this is different from the "server" approach – we do not add any new nodes here, we change *existing* nodes so that they are always available. This allows us to understand whether and how performance numbers change as we "patch up the holes" in the network, as well as how much "patching" we need before performance becomes acceptable. Also note that, unlike the server-based approach, this experimental setting mimics possible configurations for decentralized server systems such as Diaspora [29], in which a percentage of the nodes might decide to run their servers in their own machines, while another percentage runs their servers in paid, cloud-hosting services.

4.3.3 Results and Discussion

Figure 4.3a shows the cumulative distributions for the values we obtained for **aed** and **ard**, against their baselines, as well as for varying proportions of fixed nodes (15%, 50%,

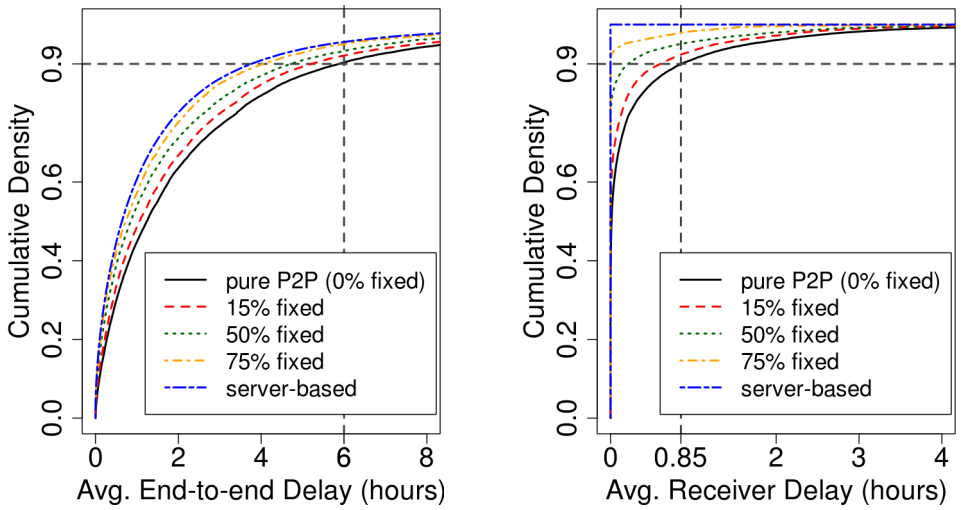


Figure 4.3: CDFs for average end-to-end and receiver delay.

	Average		99^{th} **Perc.**		**Maximum**	
	aed	**ard**	**aed**	**ard**	**aed**	**ard**
0% **fixed**	2.36h	14 mins	17.8h	3.6h	724h	10h
15% **fixed**	2.35h	14 mins	16.1h	3.0h	719h	9.5h
50% **fixed**	1.88h	7 mins	14h	2.7h	262h	8.9h
75% **fixed**	1.61h	2 mins	12h	1.4h	167h	7h
server	1.48h	0	11.7h	0	111h	0

Table 4.2: Statistics for communication delay.

and 75%). Plots are truncated near the largest 90^{th} percentile, with complementary information provided in Table 4.2.

The experiment, despite the aforementioned limitations, reveals a number of important properties and issues. The first one is the generally high values for **aed**, which can be seen all across the board. Averages are no less than 1 hour and 48 minutes, and maximum values are as high as 724 hours (a month). The fact that even the server-based approach exhibits significantly high values means, however, that most of this delay is likely due to receivers that come online very infrequently. This is confirmed by the comparatively much lower values of **ard**, which remain in the order of minutes, on average, and hours, in the worst case.

By examining **ard** values closely, however, a number of observations can be made. First, the maximum values are unacceptably high (10 hours for a purely P2P solution),

and seem to decrease quite slowly, even as we add a large percentage of fixed nodes. Second, as Figure 4.3b shows, although performance is acceptable for a large portion of the users (on the order of tens of seconds for around 50% of the users), curves flatten significantly as we approach the 90^{th} percentile, with receiver delays never lower than 1h for the slowest 1%, even with 75% of fixed nodes. The pure P2P approach performs particularly poorly, with receiver delays nearing 1 hour already at the 90^{th} percentile, and climbing to 3.6 hours at the 99^{th} percentile.

Given that this constrained experiment already reveals significant performance issues, we conclude this section by pointing out that a deeper, more comprehensive investigation is required if we are to understand under which availability conditions social overlays are viable. Yet, to base this investigation on full simulations is simply not feasible, because:

1. *Social network datasets are large.* Claiming something meaningful about social networks under churn entails studying a large number of them, which translates into prohibitively high simulation costs.
2. *Parameter choices are large.* The experiments in this section are run with a single, randomly-generated availability assignment, as described in Section 4.1. Properly exploring the parameter space would entail experimenting with more of these assignments, including ones with different distribution parameters, varying assumptions of correlation/no correlation among graph-structural (e.g., node degree) and node properties (e.g., availability), among others. The costs of each assignment gets multiplied by the size of the datasets, yielding even higher simulation costs.
3. *Distributions are heavy-tailed.* Adding to the previous issues, skewed and heavy-tailed distributions both in the availability model and the observed metrics mean a large number of repetitions is required before reliable results can be obtained, further increasing the costs.

The combination of these issues ultimately renders full simulations an impractical approach for studying the problem at hand, motivating a quest for more efficient alternatives.

4.4 Towards an Analytical Model

In this section, we present our first results towards such an alternative: a partial, hybrid model which uses simple analytical results at its core, and fills in the remaining gaps with simulations. Our current model allows us to determine an upper bound for the average end-to-end delay, as well as to identify key graph substructures that, as we will show, determine a significant fraction of the observed delay values.

Given the relation between **ed** and **rd** expressed by Equation (4.4) we focus our model only on the end-to-end delay **ed**.

4.4.1 Formation of Temporal Paths

The key observation behind our model lies in something we briefly mentioned in Section 4.1: that the only requirement for nodes u and v to be able to communicate over a graph G is that *some* temporal path forms between them – we do not really care which one. We can gain insight on how to model the end-to-end delay between two vertices u and v, then, by reasoning probabilistically about the time it takes for some temporal path T to materialize a path P between them.

Let $P = \{e_1, ..., e_m\}$ be a path between u and v in $G = (V, E)$. For each edge $e_i = (w_i, w_{i+1}) \in P$, let $\mathbf{X}_{e_i}$ be the random variable representing the time elapsed between a (randomly picked) login event of w_i and the beginning of the next activation interval of e_i. We refer to $\mathbf{X}_{e_i}$ as the *edge delay* of e_i. Given that our churn model is probabilistic, we can also represent the duration of a temporal path – i.e, the time it takes for a temporal path T to materialize P – as a random variable Δ_P. We can then express the path delay $\mathbf{D}_P$ of a path P as:

$$\mathbf{D}_P = \mathbf{X}_{e_1} + \Delta_P \tag{4.5}$$

Note that Equation (4.5) is just a restatement of the notion of path delay **pd** laid out in Sec. 4.2, but expressed this time as a sum of random variables. Now, we claim that:

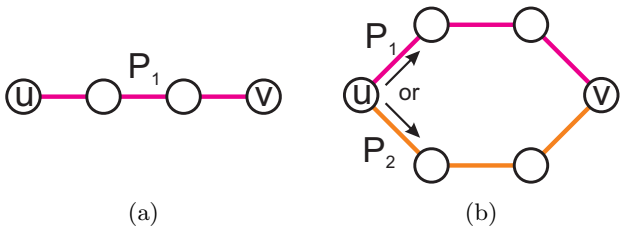
$$\Delta_P \sim \sum_{i=2}^{m} \mathbf{X}_{e_i} \overset{(4.5)}{\Rightarrow} \mathbf{D}_P \sim \sum_{i=1}^{m} \mathbf{X}_{e_i} \tag{4.6}$$

In other words, we claim that the duration of a temporal path is approximately equal to the sum of the delays of its edges, except for the first, which always contributes with zero. To understand why, think of a temporal path with a single edge, and note that it will always have duration zero, since its materialization happens instantaneously at the instant of activation of its single edge.

Substituting this claim back into Equation (4.5) yields the final result of the equation, namely, that the delay of a path is approximately equal to the sum of the delay of its edges.

We are not yet able to prove whether Equation (4.6) holds as an equality in general (and we suspect it does not), but we could experimentally verify that:

$$\mathbb{E}(\mathbf{D}_P) \sim \sum_{i=1}^{m} \mathbb{E}(\mathbf{X}_{e_i}) \tag{4.7}$$



(a) (b)

Figure 4.4: Two simple graphs.

that is, even if there are effects (e.g., residuals) unaccounted for in Equation (4.6), these are empirically negligible from the point of view of the expectation. An explanation of how we conduct this empirical validation, as well as numerical evidence supporting our assertion, is given in Sec. 4.4.4-1.

4.4.2 Bounding from Above

Given the previous results, we can provide an upper bound to the average end-to-end delay. For a pair of nodes u, v, let $\mathbf{C}_{u,v}$ be the random variable representing the end-to-end delay between them. The upper bound follows from the observation that the following holds for any path P between u and v:

$$\mathbb{E}(\mathbf{C}_{u,v}) \leq \mathbb{E}(\mathbf{D}_P)$$

This inequality states that the expected end-to-end delay among two nodes is always smaller than the expected delay of any single path connecting them. To see why this is true, it helps to think of the delay for a path P in terms of probabilities. Suppose we had a single path P_1 connecting u and v, as in Figure 4.4a, and that u were to log in the system at time $t_0 = 0$. The probability that *at least one* path gets materialized between u and v by time t is thus given by the probability that the delay for this path is smaller than t, or $\mathbb{P}[\mathbf{D}_{P_1} \leq t]$.

Now, without loss of generality, assume that we had two disjoint paths P_1, P_2 connecting u and v in G instead of just one (Figure 4.4b), while everything else remains the same. In this case, the probability that *at least one* path gets materialized by time t is given by $\mathbb{P}[\mathbf{D}_{P_1} \leq t \vee \mathbf{D}_{P_2} \leq t]$ (i.e., the probability that the delay of P_1 or P_2 is smaller than t), which is obviously higher than the probability of one single path being materialized by time t.

This intuition extends to $\mathbb{E}(\mathbf{C}_{u,v})$: the average end-to-end delay between nodes u and v is always less than or equal to the expected delay time $\mathbb{E}(\mathbf{D}_P)$ of *any* individual path P with equality holding if and only if P is the only path in the graph that connects u

and v. This also implies that computing the "true" value of $\mathbb{E}(\mathbf{C}_{u,v})$ would require us to account for the effects of *all* paths connecting u and v.

There is one particular path, however, that is "closest" to $\mathbb{E}(\mathbf{C}_{u,v})$ than any other: the path P_{min} for which $\mathbb{E}(\mathbf{D}_{P_{min}})$ is the smallest (i.e., the "fastest path" between u and v). Using Equation (4.7), we can find P_{min} through the following procedure:

1. *Assign edge weights.* Given the graph $G = (V, E)$ and its availability assignment A_G, we assign to each edge $e \in E$ a weight $h_e = \mathbb{E}(\mathbf{X}_e)$, i.e., equal to its expected delay. As we discuss later, we can compute an estimate for $\mathbb{E}(\mathbf{X}_e)$ by means of simulations which are much simpler than the full simulations of Section 4.3.1. Once we have the weights h_e for all edges in the graph, computing an estimate for the expected path delay $\mathbb{E}(\mathbf{D}_P)$ for any arbitrary path P in G becomes, by virtue of Equation (4.6), a matter of summing the weights of its edges. In other words, for a path $P = \{e_1, \cdots, e_n\}$, we have:
$$\mathbb{E}(\mathbf{D}_P) \overset{(4.7)}{\sim} \sum_{e \in P} \mathbb{E}(\mathbf{X}_e) \sim \sum_{e \in P} h_e$$
2. *Compute shortest paths.* Using a shortest path algorithm (e.g. Dijkstra's) and the weight assignment of the previous step, we compute the minimum cost path between u and v. The resulting path, as per the previous discussion, is the *fastest path* P_{min} between u and v.

The pseudocode for the simulations estimating $\mathbb{E}(\mathbf{X}_e)$ is given in Algorithm 3. We generate samples of $\mathbf{X}_e$ by buffering all login instants $\{l_1, \cdots, l_n\}$ of node w_1 in set L (line 8), until the time t at which node w_2 logs in (line 10). At that point, we are able to generate $|L|$ samples $s_1, \cdots, s_n$ by taking $s_i = t - l_i$. These samples are accumulated in variable s (line 11) and, once a sufficient amount of samples are generated, we compute the average as the estimator for $\mathbb{E}(\mathbf{X}_e)$.

These simulations are much simpler than the ones in Section 4.3.1, for three reasons. First, when estimating the weight for an edge, we need to simulate only two nodes at a time instead of the entire network. Second, the quantity we estimate can be sampled by simply looking at the state of the processes, without an expensive breadth-first search over the entire graph. Third, due to the simplicity of these simulations, running the simulation longer is sufficient to trust the resulting average is not biased, without a costly per-repetition burn-in period.

4.4.3 Improving on the Bound

We expect the upper bound we just derived to be tight in some cases, and not so tight in others (e.g., when there are many similar-weight paths connecting source and destination).

Algorithm 3: EstimateEdgeDelay

Input: An edge $e = (w_1, w_2)$; the number of samples n.

```
1  s ← 0                                          % Sum of all samples
2  r ← 0                                          % Total number of samples
3  L ← ∅                                          % Sample buffer
4  on ← {false, false}                            % Online status of w1, w2
5  while r ≤ n do
6      e ← nextSimulationEvent()
7      on[e.node] ← (e.type = LOGIN)
8      if e.type = LOGIN and e.node = w1 then
9          L ← L ∪ {e.time}
10     if on[w1] and on[w2] then
11         s ← s + Σ_{l∈L}(e.time − l)
12         r ← r + |L|
13         L ← ∅
14 return s/r
```

Intuitively, one can improve it by considering the top-k "fastest paths" between source and destination instead of taking just one. The main problem, however, is that reasoning about the expected delay of at least one among a set of (possibly overlapping) paths is significantly more complex than computing this expected delay for a single path, since we can no longer simply compose edge delay expectations as before.

We can, however, verify the extent to which the top-k fastest paths between u and v are actually enough to approximate $\mathbb{E}(\mathbf{C}_{u,v})$ by means of the following procedure:

1. *Compute the top-k fastest paths.* Using the same edge weights $h_e = \mathbb{E}(\mathbf{X}_e)$ as we did in Sec. 4.4.2, we run a top-k least-cost paths algorithm between u and v.
2. *Simulate over the resulting sub-graph.* The k least-cost paths between u and v induce a subgraph $G_{k,u,v}$ over G. By modeling churn with the same availability assignment of G for the corresponding vertices in $G_{k,u,v}$, we perform full simulations on $G_{k,u,v}$. We then compare the results for the source/destination pair u, v with those obtained by running the full simulations on G.

For our study, we consider two top-k least-cost paths algorithms. The first one is due to Yen [129], and it finds a set of top-k least-cost paths which are allowed to share both edges and vertices. We refer to it as *Yen's top-k*. The second algorithm, instead, finds a set of paths which are edge-disjoint, and we refer to it as *edge-disjoint top-k*.

It consists of a simple iterated application of Dijkstra's shortest-path algorithm, similar Dunn's heuristic [31], and works as follows. Starting with a graph G', we repeat for k iterations:

1. compute the shortest path P between u and v by using Dijkstra's algorithm;
2. remove the edges used in P from G'.

Again, estimating $\mathbb{E}(\mathbf{C}_{u,v})$ over $G_{k,u,v}$ is in general significantly cheaper than doing so over G since, as we discuss in Section 4.4.4-2, $G_{k,u,v}$ tends to be much smaller than G.

4.4.4 Evaluation

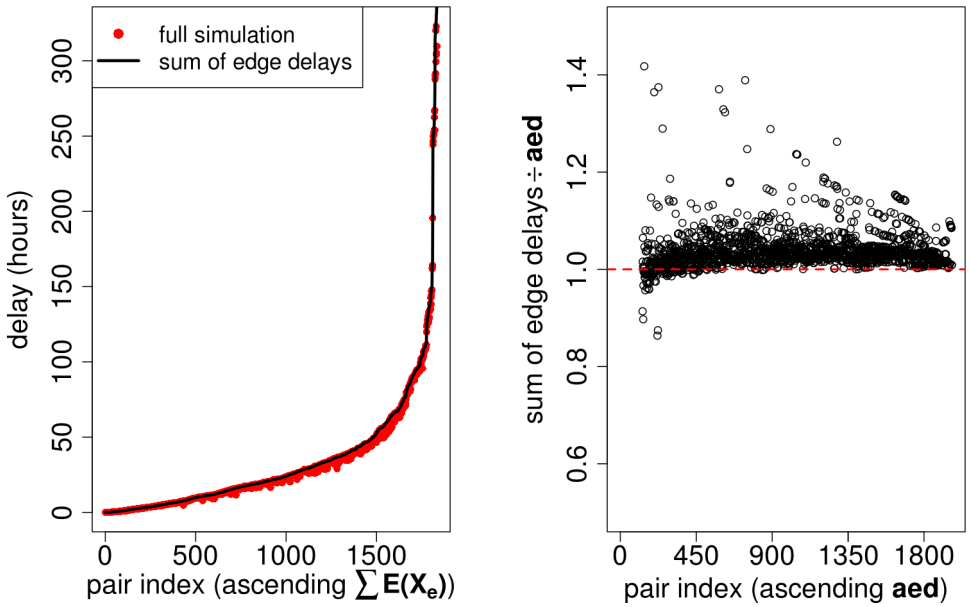
Our evaluation is divided in two parts. Section 4.4.4-1 provides numeric evidence to support the claim made in Equation (4.7) that the expected delay for a path P is approximately the same as the sum of the expected delays of its edges. Section 4.4.4-2, instead, focuses on the end-to-end delay in ego networks, providing evidence that *i)* the upper bound from Section 4.4.2 holds, and *ii)* the subgraph $G_{k,u,v}$ induced by top-k least-cost paths connecting a pair of vertices u and v accounts for a significant part of the end-to-end delay between them. We also show the differences in tightness for the bounds produced by the two top-k approaches we adopt.

Again, we use ego networks, for the same reasons outlined in Section 4.3 and to allow comparison of the results obtained here with those of the full simulations discussed there.

All the experiments we run in this section provide us with a value – either an upper bound or an estimate – on the end-to-end delay for a *pair* of source/destination nodes u and v. A single pair of nodes is, therefore, associated to different delay values, which are generated by different methods. To avoid confusion about the type of delay we are referring to, and how it has been obtained, we establish that:

1. end-to-end delay estimates produced by full simulations are referred to as **aed** (average end-to-end delay);
2. upper bounds produced by summing the estimates for expected edge delays along the least-cost path connecting u and v (Section 4.4.2) are referred to as $\mathbf{aed_1}$;
3. upper bounds produced by full simulations over the subgraph induced by the top-k shortest paths connecting u and v (Section 4.4.3) are referred to as $\mathbf{aed_k}$.

4.4.4-1 Numeric Evidence for Expectation To support the claim of Equation (4.7), we emulate the behavior of a path of size n by using a *list graph* of size n – a graph in which nodes are linked together in a doubly-linked list. An example for $n = 4$ is given in Figure 4.4a. The experiment consists in measuring the end-to-end delay from node u (the leftmost vertex) to all nodes to the right. Note that the end-to-end delay is the same



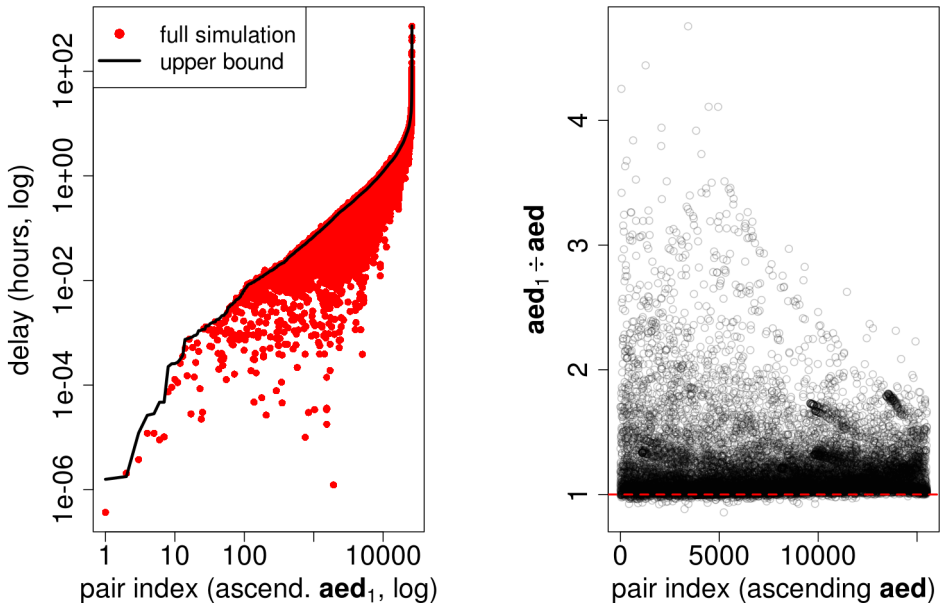
(a) (b)

Figure 4.5: Delay estimates of full simulations (**aed**) vs. sums of edge delays.

as the path delay in this case, since there is only one path connecting the source to each destination. For each node along the path, we compare the values of **aed** obtained by running full simulations with u as a source versus those obtained by summing the estimates for the expected edge delays along P. For our experiments, we consider $n = 15$, which is already more than twice the widely-accepted diameter of 6 for social networks [21]. We test 130 random availability assignments for this long path, which are generated according to the model described in Sec. 4.1. Furthermore, for each assignment:

1. full simulations are ran 700 000 times;
2. edge delay estimates are obtained by averaging 500 000 samples per edge – in other words, we run Algorithm 3 with $n = 500\,000$.

Figure 4.5a shows the delay values produced by the two methods for all pairs u, w, where u is the leftmost vertex in the list graph. Pairs are sorted in ascending order of their sum of expected edge delay values, shown as a curve. Points fall close to the curve, showing that Equation (4.7) indeed holds in practice. A quantitative measure of error can be seen in Figure 4.5b, which shows the ratio between the value produced by summing the edge delay estimates and **aed**, with pairs sorted by their **aed** values this time. We put a dashed line at $y = 1$ where the values coincide. The error stays on average below 4%, with 95% of the points staying under 10%. Most of the outliers with large relative errors (above 20%) lie in a zone where **aed** is small, meaning that the high percentage error



(a) Simulations and upper bound.

(b) Relative error of the upper bound.

Figure 4.6: Upper bound $\mathbf{aed_1}$ vs. full simulations **aed**.

translates into a small absolute error. Further, values in that area are more susceptible to random noise.

4.4.4-2 Delay on Ego Networks We analyse, in this section, the same set of 700 ego networks we described in Section 4.3.2. To validate our model, we compare the **aed** values from the full simulations which we obtained previously against the values produced by our model. The expected edge delay estimates required by the model (Algorithm 1) are again obtained by averaging 500 000 samples per edge.

Upper Bound Holds. We validate our claim that the expected delay $\mathbb{E}(\mathbf{D}_{P_{min}})$ for the least-cost path connecting two nodes u and v serves as an upper bound to the expected end-to-end delay $\mathbb{E}(\mathbf{C}_{u,v})$. Figure 4.6a shows, for each pair u, v in our sample, a log-log plot of the values of **aed** (dots), together with the corresponding $\mathbf{aed_1}$ values for the least-cost paths (thick black line). Pairs are ordered in the x axis in ascending order w.r.t. $\mathbf{aed_1}$.

As expected, dots fall consistently below the black line, providing solid evidence for our claim. The fact that some dots fall slightly above the black line is not really an issue, rather a result of the fact that we are using sample averages as estimators both for $\mathbb{E}(\mathbf{D}_{P_{min}})$ and **ard**, and those are subject to noise, even with the many repetitions we run.

As for how "tight" the upper bound is, we again provide a plot which shows, for each

	Average	95^{th} **percentile**
Least-cost path (r_1)	21%	258%
Yen's Top-k (r_k)	7%	151%
Edge-Disjoint Top-k (r_k)	2%	35%

Table 4.3: Error (r_1 and r_k) statistics for bounds.

pair, the ratio between $\mathbf{aed_1}$ and **aed** (Figure 4.6b). To avoid the effects in the low **aed** region we observed in Figure 4.5b, we constrain the pairs in Figure 4.6b to those in which the **aed** ≥ 0.5 hour (72% of the pairs).

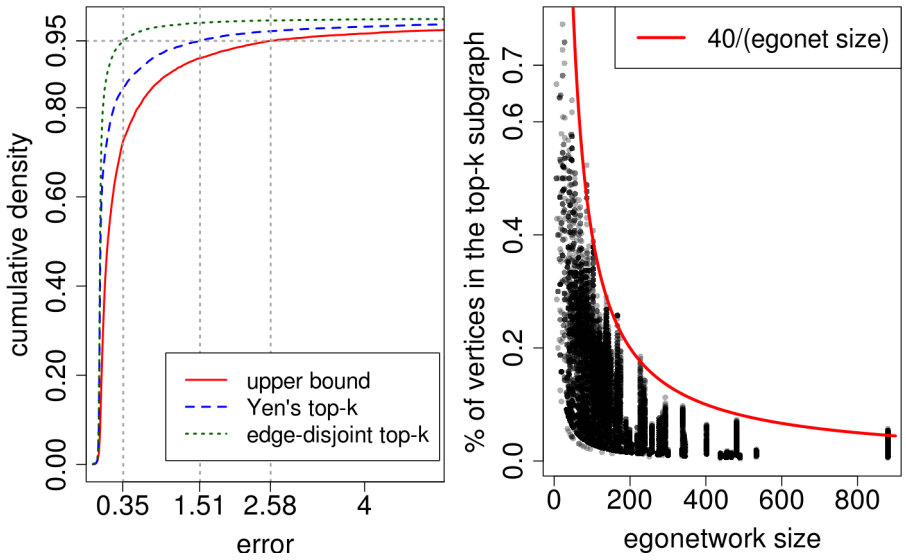
The upper bound provides a good estimate of **aed**, falling within 16% of the full simulation values, 21% if we consider the worst 25% points, and 75% if we consider the worst 5% points. The bound becomes more accurate in relative terms for larger values of **aed**, meaning it is better at helping identify cases for which we can expect poor performance.

Top-k Paths Provide a Better Bound. By using the procedure described in Section 4.4.3, we investigate how much we can improve on our bound by considering not only the single least-cost path connecting a pair u, v, but the k least-cost paths. We carry out this evaluation on a subset of the 137 260 pairs we considered in Section 4.3.2, by drawing 14 000 source/destination pairs at random and without replacement from the original sample. For the experiments carried out in this section, we use a value of $k = 10$.

We examine the metrics $r_k(u, v) = \frac{\mathbf{aed_k}}{\mathbf{aed}} - 1$, and $r_1(u, v) = \frac{\mathbf{aed_1}}{\mathbf{aed}} - 1$. These give an error measure, in percentage terms, of how much $\mathbf{aed_k}$ and $\mathbf{aed_1}$ deviate, respectively, from the **aed** estimate produced by the full simulations.

We plot in Figure 4.7 the cumulative distributions for both r_k and r_1, with the 95% percentiles marked by lines. Further statistics are summarized in Table 4.3. The top-k estimate stays significantly closer to the actual **aed** estimate than the upper bound (2% vs. 21% on average for the upper bound), with the top-k, edge disjoint paths performing better than Yen's for the value of k we consider. This can be attributed to the fact that the paths selected by Yen's algorithm tend to overlap heavily when, intuitively, the discussion in Section 4.4.3 points to the fact that selecting disjoint paths increases the probability that at least one of them materializes by some time instant t.

Finally, we show that although the delay estimate measured over the subgraphs $G_{k,u,v}$ (the subgraphs induced by the top-k edge-disjoint paths connecting u and v) is close to the delay observed over the full graph G, these subgraphs are in fact significantly smaller. This is shown in Figure 4.7b, where we plot the percentage of the vertices carried over from each ego network G to $G_{k,u,v}$ as a function of how many vertices G has. We see



(a) (b)

Figure 4.7: Least cost vs. top-k least cost paths, and size of top-k subgraphs.

that, the larger the graph, the smaller the percentage of vertices carried over, providing evidence that $G_{k,u,v}$ is indeed a smaller substructure that is capable of explaining the end-to-end delay.

Indeed, by looking carefully, we see that their size is bound by a constant: paths that makes up $G_{k,u,v}$ have lengths that are proportional to the diameter of the graph, which is 2 in the case of ego networks. The resulting graphs, therefore, have a size that is proportional to k times the diameter, which is evidenced by the curve $2 \cdot k \cdot diameter/(\text{egonet size}) = 40/(\text{egonet size})$ we overlay in the graph. Although we are not entirely sure of how well this holds for larger datasets, it means that even if we face NP-complete problems in trying to model delays over these graphs, such problems, over small-world networks, might nevertheless remain almost constant in size, at $\mathrm{O}(k \log n)$. $G_{k,u,v}$, therefore, is where our next analysis efforts should be focused on.

4.5 Implications to the Study of Dissemination Protocols

The results presented in this chapter bear direct consequence on the dissemination problem we studied in Chapter 3. They essentially entail that, under churn, distribution of dissemination delays follows a two-phase process. In the first phase, which happens at time scales of seconds to minutes, dissemination performance is driven by the actual dissemination protocol. In the second phase, instead, dissemination times are dominated by graph dynamics.

The most obvious evidence for the existence of this phenomenon is that the network-induced delays we have measured in this chapter are often in the order of hours, whereas delays induced by any properly designed dissemination protocol will usually be much lower (e.g. seconds to minutes), network allowing. Indeed, even delays caused by slower protocols such as anti-entropy will tend to be dwarfed by the extreme timescales of network-induced delays as dissemination progresses.

An important implication of this phenomenon is that, unless we can separate the two phases, contributions of network dynamics and dissemination protocols get mixed together when evaluating protocols under churn, and differences between protocols disappear. Indeed, looking at larger timescales, protocol delay distributions will tend to converge into a common curve – the delay curve given by network dynamics. This reveals yet another inadequacy of the churn study in Section 3.5.2 – the timeouts we used are an arbitrary stopping point for dissemination protocols which provide no guarantees whatsoever as to which phase the measured delays (or latencies, as we called them in Chapter 3) belong to.

To illustrate the effect of convergence, we take a subset of 162 ego networks from Section 4.3.2 (83 021 source/destination pairs) and, under the same availability assignments, we run simulations of both HFLOOD and ANTICENTRALITY under churn. As before, we warm up the network by letting it run for a burn-in period γ of 48 hours, and then cause a message to be posted at the first login of the source after burn-in. We then let the protocols run until they go quiescent. We set the timeout parameter of the protocols to infinity, so that they do not give up on disseminating until they are sure that every neighbour they have has gotten the message. Experiments are repeated 100 times.

The performance of our protocols vary depending on network-structural properties, and churn essentially transforms network structure. To render comparison meaningful (and possible), therefore, we need to ensure that the underlying ego networks get transformed similarly as we evaluate the different protocols. To that end, we identify each experiment e and each repetition r by a unique id (e, r). We then assign the same seed $s(e, r)$ to the random generator governing the up/down distributions of processes with the same values for both protocols. The net effect is that the up/down "schedule" for both protocols is the same.

Progressive plots of protocol latency (receiver delay) by egonet size for both HFLOOD and ANTICENTRALITY are shown in Figure 4.8. From these plots, one could conclude that the two protocols exhibit no discernible difference in performance because, say, the structural features that made them perform differently are lost under churn. Indeed, if anything, HFLOOD seems to exhibit slightly better performance than ANTICENTRALITY (3% better, at most). Yet, if we look at scatterplots for end-to-end delays for both

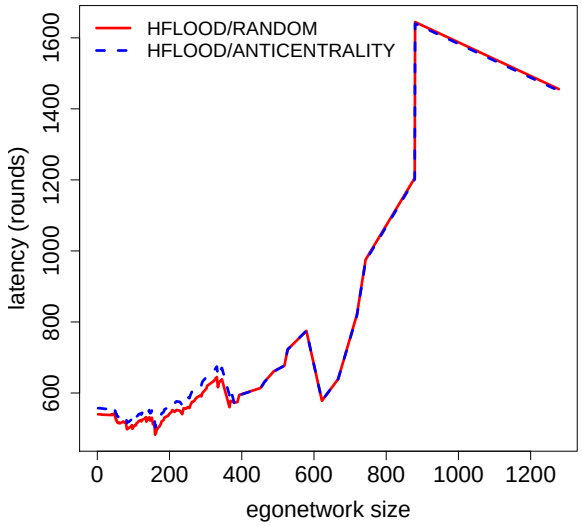
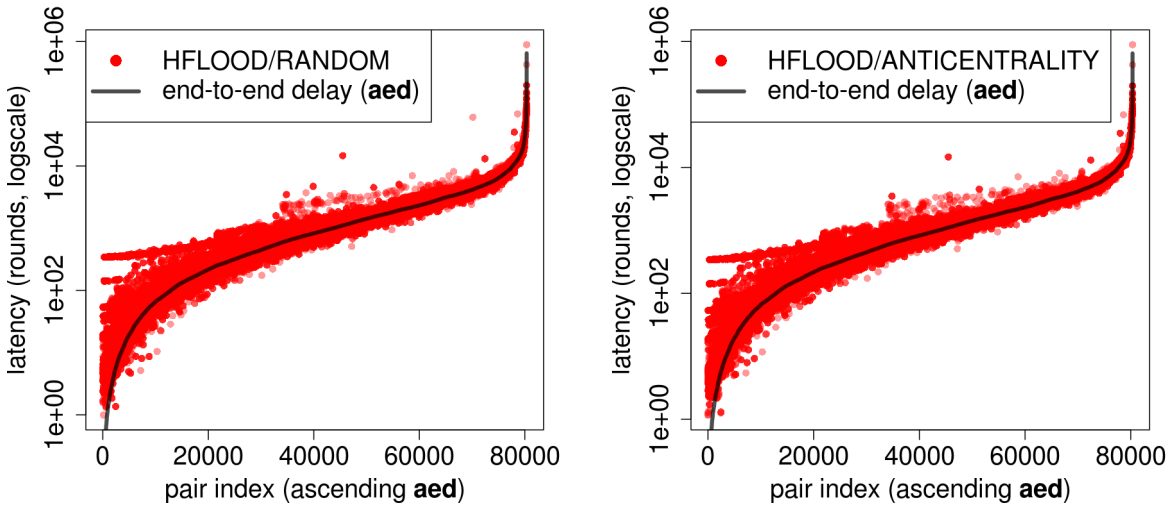


Figure 4.8: Latency measurements contaminated by phase 2 delays.

protocols, which are shown in Figure 4.9, what we see is that the two protocols are simply reflecting the curve for network-induced delay, given by our **aed** estimates from Section 4.4.4 – our measurement has overshot into phase two, and we can no longer see any difference.

A few other notes are in order for these scatterplots. First, note that the protocols do not follow the curve *exactly*. This is because, at 100 repetitions, there is still a significant amount of noise in the estimators. Indeed, our **aed** estimates are based on 3 orders of magnitude more repetitions which, as we mentioned before, are required to obtain "good" values for the estimators under heavy-tailed distributions. Second, note that the protocols sometimes perform better than the **aed** curve, when we claimed that **aed** should be a lower bound on performance. Again, the reason is noise – **aed** is a bound on the average end-to-end delay so, as we run more repetitions, red dots should slowly move either above or on top of the curve. Finally, note that, even though there is noise, dots have a very similar positioning (but not always exactly the same) in both graphs – i.e. the "noise" is similar. This is because we seeded the underlying churn model identically for each experiment repetition, and this actually makes our point stronger: performance is *really* being driven by phase two delays, and what we observe, indirectly, is the variation of a point estimator for phase two delays for which the sample size is small.

We now describe a methodology to compare dissemination protocols under churn. The way to separate phase 1 delays is to identify the set S of nodes that can be directly reached from the source at the moment in which an update gets posted. This can be done by



(a) HFLOOD (b) ANTICENTRALITY

Figure 4.9: Dissemination protocols and the end-to-end delay bound (**aed**).

running Algorithm 4 at the first post of the source, which then calls Algorithm 5.

Algorithm 5 then performs a breadth-first-search from the source, following only links that point to nodes that are online, and collecting these reachable nodes in set C (line 7). At the end of the search, lines 9–10 will result in S being set to C.

Then, to avoid "contaminating" measurements, we need to continuously update S so as to remove nodes that become unreachable from the source before getting the update due to network changes. Again, we can simply call Algorithm 5, which again collects the nodes currently reachable from the source in set C. This time, however, it sets D to $V - (C \cup R)$ (line 9) – i.e. the set of nodes that have not yet received the update and are not reachable from the source. These nodes are then subtracted from S (line 10).

At the end of the simulation, S will contain only the set of nodes that had a direct path from the source when they got their update, and this is a conservative estimate of nodes uncontaminated by phase 2 delays. If we account only to delays towards these nodes, then we expect that differences between the protocols will still exist.

Algorithm 4: OnSourcePost

Input: Graph $G = (V, E)$, source u, set of nodes that are online U

1 $S \leftarrow \text{ComputeCore}(G, u, V, U, \{u\})$

Indeed, Figure 4.10 shows again progressive plots for receiver delay in both protocols,

Algorithm 5: ComputeCore(G, u, S, U, R)

Input: Graph $G = (V, E)$, source u, core set S, set of nodes that are online U, set of nodes reached by the update R.

```
1  Q ← {u}
2  C ← ∅                                   % Nodes currently reachable from source u.
3  while Q ≠ ∅ do
4      v ←extract(Q)                       % Remove in FIFO order
5      forall the (v, w) ∈ E do
6          if w ∈ U then
7              C ← C ∪ {w}
8              Q ← Q ∪ {w}
9  D ← V − (C ∪ R)                         % Nodes to remove from core.
10 S ← S − D
11 return S
```

trimmed by the size of the connected core. This time, not only ANTICENTRALITY comes out as the best performer, but differences are noticeable: from a 10% improvement if we consider cores of size 200 and up, to a maximum of 645% for cores of size 590 and up, meaning that heuristics still make a difference, even under churn. A more comprehensive evaluation of these differences, however, as well as MAXCOMP, is left as future work.

4.6 Related Work

Connectivity of social graphs. The work that is perhaps most similar to ours in terms of intent is the study by Singh et al. [107], in which authors analyse the connectivity of social overlays under churn. They argue that SOs are not adequate as dissemination media, by showing that individual snapshots of a social graph might be partitioned, or exhibit dilated average path lengths. Although our conclusions are similar, our work differs in that snapshot analysis fails to capture that communication is a process that develops in time, and that individual snapshots of a graph reveal very little about their ability to carry information. Indeed, as shown by Clementi et al. [23], there are models of temporal random networks which exhibit logarithmic flooding time w.h.p. , even though individual snapshots appear to be completely disconnected (i.e. w.h.p. they exhibit no giant component).

Delay-tolerant networks (DTNs). Unlike the systems we target, where the network dynamics are determined by *node* churn, DTNs are characterized by *links* appearing and

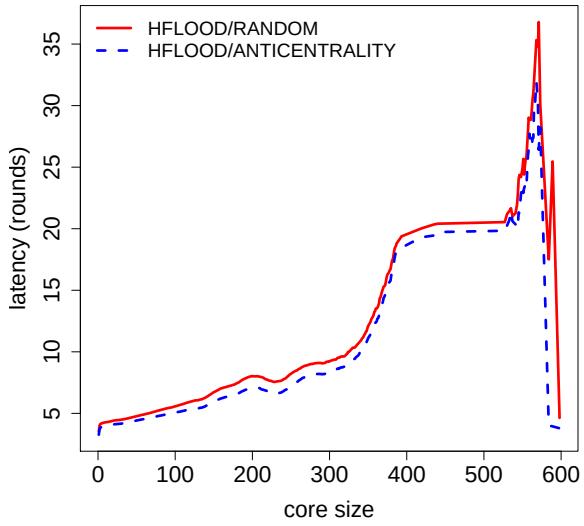


Figure 4.10: Uncontaminated phase 1 measurements.

disappearing (e.g., due to mobility). The temporal networks that arise in both contexts are, however, similar.

Tang et al. [111] establishes a bridge between the realms of social network analysis and DTNs by considering mobile social networks. Their focus, however, is on the characterization of temporal graphs by appropriate "temporal metrics", which notably include a notion of temporal distance similar to our definition of end-to-end delay. In contrast, we consider only two metrics, but with the goal of developing techniques for analysing asymptotic behavior.

Chaintreau et al. [20] tackle problems similar to ours while studying the diameter of temporal networks. They analyse the conditions under which paths with logarithmic delay and hop count emerge as the network size approaches infinity, thus establishing bounds on the expected end-to-end delays. Their theoretical results are, however, based on a simple temporal generalization of Erdös-Rényi graphs, with the purpose of providing insight into the behavior of real opportunistic networks, along the lines of what Watts and Strogatz [123] did for social networks. In contrast, our goals are driven by the immediate need to provide system designers with latency bounds over real, arbitrary social graphs, given an availability model. The need to take a predefined structure and heterogeneous parameters into account makes the nature of the problems rather different. These two differences—network model and overall goals—prevent a straightforward reuse of the results in [20] in our context.

Scheduled and temporal networks. Our work can be related on the surface to previous

literature on scheduled [9] and temporal networks [50]. Although some of the formalisms found in these papers are similar to ours (e.g., Berman's [9] characterization of temporal paths) these works concern themselves mostly with combinatorial problems over temporal graphs for which the entire edge schedule is known in advance (e.g. generalizing Menger's theorem to temporal paths). Our work, instead, concerns itself with the asymptotic behavior of simpler properties of graphs for which we have a generating model, but for which the observation window is not finite.

Clementi et al. [23] studies the flooding time of *edge-Markovian graphs*, which are graphs in which edge activation/deactivation times are governed by a discrete, time-homogeneous Markov chain similar to the chains that describe the behavior of individual nodes in our model. These graphs are rather different from ours, however, in that all edge Markov chains are independent and identical, and that edges are allowed to form between any pair of nodes in the graph, giving raise to rather regular structures over which authors then prove bounds. Again, this work is in the same spirit as the work of Chaintreau et al. [20] – providing models for explaining the governing laws of real-world temporal networks.

Partly based on the work of Tang et al. [111], Scellato et al. [100] presents results on the distributions of temporal distances – which are equivalent to our end-to-end delays – in temporal Erdös-Rényi graphs (i.e. sequences of Erdös-Rényi graphs). Again, the simple nature of these random enables the derivation of general results, but these results cannot be applied in our context.

Gossip protocols can be seen as temporal sequences of graphs in which an edge exists between two nodes if and only if one of them selects the other. It has been long established that flooding times for graphs induced by uniform gossip are logarithmic w.h.p. [44, 84].

4.7 Discussion and Outlook

This chapter makes two main contributions. First, we have shown through simulations over real datasets that churn might induce non-negligible delays on information dissemination over social overlays. Second, we have introduced a hybrid analytical model for the communication delay between two vertices in general graphs and, based on this model, we have shown how to obtain reasonably accurate upper bounds by using lighter-weight simulations.

Then, based on the knowledge we have acquired by studying delays, we have shown that evaluating dissemination protocols such as the ones from Chapter 3 under churn is more complex than originally thought, and that acknowledging that dissemination is a two-phase process – and being able to identify and separate these phases – is an important

piece of a proper evaluation.

Finally, this work opens up a large research space. While it is clear that, in practical terms, a real P2P system based on a social overlays might provide adequate performance to the majority of its users, a small percentage of that user base might experience delays on the order of hours or more. Given the targeted scale of these systems, that small percentage can translate into millions of users. Understanding exactly what are the conditions under which this problem happens and how it can be mitigated are open research questions. Our model might help with the former, while the work we develop in Chapter 6 might set a direction for the latter.

Before delving into how to mitigate the effects of network-induced delays, however, we take a short detour in Chapter 5, in which we simplify our model in a way that still conserves its representativeness, while allowing us to solve important parts of it analytically.

Chapter 5

A Markov Delay Model

All models are wrong, but some are useful.

– *George E. P. Box*, Empirical Model-Building and Response Surfaces

The main analytical hurdle we face in trying to extend the hybrid model of Chapter 4 into one that does not rely on simulations is that heavy-tailed distributions are not memoryless. This renders the underlying processes governing our system non-Markov, precluding us from applying the widely available results on Markov chain theory. In this chapter, we sacrifice model realism for analytical simplicity, and replace these heavy tailed session/inter-session distributions by a pair of memoryless, exponential distributions.

Then, by applying Markov chain theory, we develop analytical counterparts to a number of elements in our model which previously required simulations. We start by formally presenting the modifications we apply to the underlying churn model in Section 5.1. We then derive, in Section 5.2, a closed-formula expression for the lowest-level building block of our model: expected edge delays.

From this basic block we work our way up towards more complex elements. In Section 5.3 we deal with paths. We prove that a fundamental result relating edge and path delays, namely, Equation (4.6), holds as an equality when session and inter-session lengths are exponentially distributed. Using these results, we proceed to derive a closed-formula expression for expected path delays which allows us to compute the single-path upper bounds of Section 4.4.2 at a low cost, effectively enabling the application of the technique to large graphs.

We then discuss how to tackle the highest-complexity structure in our modelling chain – the graphs composed by the top-k, least-cost paths connecting a pair nodes, which we described in Section 4.4.3. Unfortunately, we are not yet able to model those accurately.

We therefore present the main modelling challenges, referring the interested reader to a detailed account of our partial results in Appendix B.

Finally, leveraging on the analytical tools we develop in this chapter, we propose a new hybrid technique which starts off by giving us a coarse, low-cost view on delays which is based entirely on analytical techniques. Although imprecise, this coarse view allows us to target our efforts on a smaller subset of our graph, for which better estimates are then progressively obtained by means of more expensive techniques.

5.1 System Model

We adopt the same system model of Chapter 4, but replace the Pareto-distributed session and inter-session lengths by exponential distributions. We denote an exponential distribution with parameter lambda λ and expectation $1/\lambda$ by $\exp[\lambda]$. The cumulative distribution function of $\exp[\lambda]$ is given by:

$$\exp[\lambda](x) = 1 - e^{-\lambda x} \tag{5.1}$$

The rest of our availability model remains unchanged. Heterogeneity, in particular, is modelled exactly as in Chapter 4: for each node u, we draw two parameters ℓ^u_{on} and ℓ^u_{off} from a pair of shifted Pareto distributions $\mathsf{pareto}[3,1]$ and $\mathsf{pareto}[3,2]$, respectively. We then set the session and inter-session length distributions to $\exp[1/\ell^u_{on}]$ and $\exp[1/\ell^u_{off}]$, so as to yield system-wide averages for session and inter-session lengths of 0.5 and 1 hours.

Although the assumption of exponential session and inter-session times might seem like a crippling simplification, we note that this is the standard practice in the related literature (e.g. [91, 107]), precisely because heavy-tailed distributions are so problematic.

5.2 Edge Delays

We start by tackling the fundamental building block for our model, which are edge delays. We can derive their distributions by modelling edge behavior as a continuous-time Markov chain (CTMC). For every node w in the system, let $\{\mathbf{E}_w(t), t > 0\}$ represent the stochastic process describing its up/down behavior, such that:

$$\mathbf{E}_w(t) = \begin{cases} 1 & \text{if } w \text{ is online at time } t, \\ 0 & \text{otherwise.} \end{cases}$$

If session and inter-sessions lengths are exponentially distributed with parameters μ_w and γ_w, respectively, then $\mathbf{E}_w(t)$ is Markov and can be described by the CTMC depicted in Figure 5.1. To avoid complicating notation, we will loosely refer to both a CTMC and

its associated Markov processes by its indexed random variable (e.g. $\mathbf{E}_w(t)$ denotes both the chain and process).

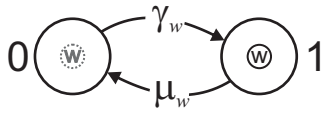


Figure 5.1: Continous-time Markov chain $\mathbf{E}_w(t)$ for process w.

Now, let $e = (u, v)$ be an arbitrary edge in the system. Since $\mathbf{E}_u(t)$ and $\mathbf{E}_v(t)$ are CTMCs, we can describe the states of edge e as a four-state Markov chain that arises from their composition. The resulting chain, depicted in Figure 5.2, is named $\mathbf{E}(t)$.

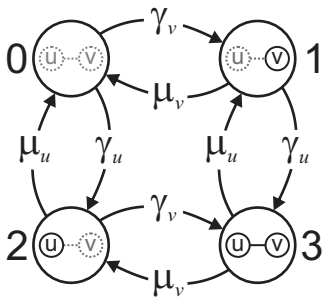


Figure 5.2: Continuous-time Markov chain $\mathbf{E}(t)$ for edge $e = (u, v)$.

Clearly, the *delay* incurred by e on some message m is given by the time elapsed from the instant s at which m reaches u to the instant $t+s$ at which both u and v are online at the same time. Since u can only receive m if it is online, it is necessarily true that when u receives m, either $\mathbf{E}(s) = 2$, or $\mathbf{E}(s) = 3$. We look at these two cases separately.

Case 1: $\mathbf{E}(s) = 3$. In this case, both nodes are online, so the delay incurred by e is zero.

Case 2: $\mathbf{E}(s) = 2$. In this case, delay is given by the time it takes for the chain to transition into state 3 *for the first time* after having started, at instant s, at state 2. This is captured by the notion of *first hitting time* of a Markov chain – the random variable $\mathbf{T}_{ij}^{\mathbf{E}}$ that expresses the time it takes for $\mathbf{E}$ to enter state j *for the first time* given that it started, at some time instant s, at state i. Formally:

$$\mathbf{T}_{ij}^{\mathbf{E}} = \min\{t : \mathbf{E}(t+s) = j \mid \mathbf{E}(s) = i\} \overset{\star}{=} \min\{t : \mathbf{E}(t) = j \mid \mathbf{E}(0) = i\} \tag{5.2}$$

where the starred equality, which essentially renders the model tractable by saying that whatever happened until time instant s is not relevant to the analysis, follows from the fact that $\mathbf{E}(t)$ is Markov and time-homogeneous [90], i.e. the rate parameters do not change in time. To compute the average delay of edge e, then, we need to understand what is the expected value of $\mathbf{T}_{23}$.

This is not, however, the whole problem: we also need to understand how often e will actually incur any delay; i.e., how often should we expect chain $\mathbf{E}$ to start at states 2 (Case 1) or 3 (Case 2) when a message reaches u. For example, if someone told us that for 80% of the time the chain starts in state 3, then we would expect only 20% of the messages to be delayed by $\mathbb{E}(\mathbf{T}_{23})$. All the other messages would suffer zero delay. The grand average delay would, therefore, be $0.2 \times \mathbb{E}(\mathbf{T}_{23})$.

If we assume the system to be running for "long enough", then the probability of catching a Markov chain at some state i at some arbitrary (but large) time instant s is given by π_i, the stable-state probability that the chain is at state i as t goes to infinity [90]. As per our previous reasoning, we know that u had to be online when it got message m (i.e. $\mathbf{E}_u(s) = 1$). We also know that m will be delayed if and only if v is offline when that happens; otherwise, delay will be zero. We can therefore focus on the state of v – i.e., the chain $\mathbf{E}_v(t)$ – to determine when there will be any delay. Let π_0 and π_1 denote the stable-state probabilities that $\mathbf{E}_v(t)$ is in states 0 and 1, respectively, at some arbitrary but large time instant s. Then:

$$\begin{aligned}\mathbb{P}[\mathbf{E}(s) = 2 \mid \mathbf{E}_u(s) = 1 \wedge s \gg 0] &= \mathbb{P}[\mathbf{E}_v(s) = 0 \mid s \gg 0] = \pi_0 \\ \mathbb{P}[\mathbf{E}(s) = 3 \mid \mathbf{E}_u(s) = 1 \wedge s \gg 0] &= \mathbb{P}[\mathbf{E}_v(s) = 1 \mid s \gg 0] = \pi_1 \end{aligned} \tag{5.3}$$

As before, let $\mathbf{X}_e$ denote the random variable representing the delay incurred by edge e. We have that the expectation we want to compute can then be expressed as:

$$\mathbb{E}(\mathbf{X}_e) = \pi_{v,0} \cdot \mathbb{E}\left(\mathbf{T}_{23}\right) + \pi_1 \cdot 0 = \pi_0 \cdot \mathbb{E}\left(\mathbf{T}_{23}\right) \tag{5.4}$$

For the remainder of this section, we focus on obtaining the formulae for π_0 and $\mathbb{E}\left(\mathbf{T}_{23}\right)$ as a function of the up/down parameters of nodes u and v.

5.2.1 Stable-State Probabilities

The stable-state probabilities π_i for $\mathbf{E}_v(t)$ can be easily obtained by solving the balance equations [90]:

$$\begin{cases} \pi_0 \cdot \gamma_v = \pi_1 \cdot \mu_v \\ \pi_0 + \pi_1 = 1 \end{cases} \iff \pi_0 = \frac{\mu_v}{\mu_v + \gamma_v}, \pi_1 = \frac{\gamma_v}{\mu_v + \gamma_v} \tag{5.5}$$

5.2.2 First Hitting Time Distributions

First hitting times are more complex, but these can be obtained as follows. Let $P_{ij}^{\mathbf{E}}(t)$ denote the probability that chain $\mathbf{E}$ is at state j at time t, conditioned on it having started at state i. Formally:

$$P_{ij}^{\mathbf{E}}(t) = \mathbb{P}[\mathbf{E}(t) = j \mid \mathbf{E}(0) = i] \tag{5.6}$$

Markov chain theory provides us with a way of computing $P_{ij}^{\mathbf{E}}(t)$ by specifying and solving a set of equations for $\mathbf{E}$ known as *Kolmogorov's forward equations*. But $P_{ij}^{\mathbf{E}}(t)$ is not the probability we seek, since the probabilities provided by Equation (5.6) also take into account sample paths in which the chain is allowed to transition in and out of j arbitrarily many times before time t. To compute first-hitting times, we have to do a small adjustment: namely, we need to create a new chain $\mathbf{Y}(t)$ from $\mathbf{E}(t)$ in which we take the states for which we want first-hitting probabilities – state 3, in our case – and make them absorbing states. The modified chain is shown in Figure 5.3.

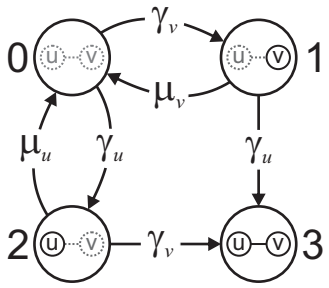


Figure 5.3: Modified Markov chain $\mathbf{Y}(t)$, in which state 3 is absorbing.

Since 3 is an absorbing state, the chain will be trapped once it gets in, which means that:

$$\begin{aligned} P_{ij}^{\mathbf{Y}}(t) &= \mathbb{P}[\mathbf{Y}(t) = j \mid \mathbf{Y}(0) = i] = \\ &= \mathbb{P}[T_{ij}^{\mathbf{E}} \leq t]. \end{aligned} \tag{5.7}$$

where Equation (5.7) is what we want to compute (for $i = 2, j = 3$). The reason this equivalence holds is simple: since j is absorbing, the modified chain enters it only once. The probability $\mathbb{P}[\mathbf{Y}(t) = j \mid \mathbf{Y}(0) = i]$, therefore, expresses the probability that, starting from state i, $\mathbf{Y}$ has transitioned into the absorbing state for the first (and only) time by instant t. Since $\mathbf{Y}$ and $\mathbf{E}$ are identical in every other aspect, this is the same as the probability for $\mathbf{E}$ to enter j for the first time by time instant t, and equality follows. To get the distribution for T_{23} in chain $\mathbf{E}(t)$, therefore, we need to obtain $P_{23}^{\mathbf{Y}}(t)$.

Given that $\mathbf{Y}$ is a rather small chain, and given the existence of published results [124] in which Kolmogorov's forward equations are successfully solved for such small models, we first attempted to specify and solve the system of equations for $\mathbf{Y}$. The main advantage of getting a direct solution is that it can provide us with closed-form expressions for $P_{23}^{\mathbf{Y}}(t)$ and, therefore, for the distribution of edge delays. Unfortunately, even if $\mathbf{Y}$ satisfies the

required regularity conditions for a solution to the system of equations to exist, we could not manage to find it with the automated solver we had available [126]. We suspect this is so because $\mathbf{Y}$ is absorbing. The system of equations is, in any case, described in Appendix A.

5.2.3 Expectation for Edge Delays

Given the difficulties with tackling Kolmogorov's equations directly, we searched for alternatives which would allow us to obtain the distribution's moments (expectation, variance) in closed form instead. We found our answer in the theory of *phase-type distributions* [74]: a general family of probability distributions that describe the time-to-absorption in Markov chains that have only one absorbing state, like our chain $\mathbf{Y}(t)$.

In this section, we derive the closed-form expression for $\mathbb{E}(\mathbf{T}^{\mathbf{E}}_{23})$ by specifying the phase-type distribution for the time-to-absorption of $\mathbf{Y}(t)$ and computing its central moment. We start by introducing phase-type distributions and related concepts in terms of general continuous-time Markov chains Section 5.2.3-1. We then apply these concepts to our chain $\mathbf{Y}(t)$ in Section 5.2.3-2, from where we obtain $\mathbb{E}(\mathbf{T}^{\mathbf{E}}_{23})$ and, finally, the formula for the expected edge delay $\mathbb{E}(\mathbf{X}_e)$.

5.2.3-1 Phase-type distributions. Before we can describe the general form of a phase-type distribution and its relationship to the underlying Markov chain, we need the concept of the *generator matrix* of a Markov chain.

Definition 1 (Generator Matrix of a CTMC)**.** *Let $\mathbf{M}(t)$ be a continuous-time Markov chain. Then, the generator matrix A of $\mathbf{M}(t)$ is defined as:*

$$A_{ij} = \begin{cases} -\lambda_i & \text{if } i = j \\ \lambda_i Q_{ij} & \text{if } i \neq j \end{cases} \tag{5.8}$$

where Q_{ij} is the transition probability from i to j in the *embedded jump chain*[1] of $\mathbf{M}(t)$, and λ_i is the is the derivative of the holding time distribution for state i, i.e. λ_i is the parameter of the exponential distribution that governs the time that the chain spends in state i.

Let $\mathbf{M}(t)$ be an irreducible, continuous-time Markov chain with m states, and one single absorbing state, like our chain $\mathbf{Y}(t)$. For simplicity, let us assume that the absorbing state is m. As before, let A denote the generator matrix of $\mathbf{M}(t)$. Then A can be rewritten

[1] The corresponding discrete-time Markov chain that transitions amongst states with the same probabilities as $\mathbf{M}(t)$ does.

as:

$$A = \begin{bmatrix} \mathbf{S} & \mathbf{s} \\ \mathbf{0} & 0 \end{bmatrix} \tag{5.9}$$

where $\mathbf{S}$ is a $(m-1) \times (m-1)$ matrix, $\mathbf{s}$ is the vector of entrance rates into absorbing state m, and $\mathbf{0}$ is a $(m-1)$ vector of zeroes.

Assuming without loss of generality that the chain cannot start at the absorbing state, let $\boldsymbol{\tau} = \{\tau_1, \cdots, \tau_{m-1}\}$ represent the probabilities of starting at each of the $m-1$ transient states. Finally, let $\mathbf{1}$ represent the $m \times 1$ vector which contains only ones. Then, the absorption time T follows the phase-type distribution with parameters $\mathbf{S}$ and $\boldsymbol{\tau}$, denoted as $\text{PH}(\mathbf{S}, \boldsymbol{\tau})$, and has the following cumulative density function:

$$\mathbb{P}[T < t] = 1 - \boldsymbol{\tau} e^{\mathbf{S}t} \mathbf{1} \tag{5.10}$$

where $e^{\mathbf{S}t}$ denotes the matrix exponential:

$$e^{\mathbf{S}t} = \sum_{n=0}^{\infty} \frac{t^n \mathbf{S}^n}{n!} \tag{5.11}$$

Although Equation (5.10) effectively expresses the distribution of the time-to-absorption of $\mathbf{M}(t)$, it has a number of shortcomings: *i)* it is not a closed-form expression, *ii)* it is numerically unstable [115], and *iii)* it is cumbersome to handle analytically. Fortunately, however, the moments for phase-type distributions are much simpler to handle. Indeed, if T is a phase-type distributed random variable with parameters $\boldsymbol{\tau}$ and $\mathbf{S}$, then:

$$\mathbb{E}(T^n) = (-1)^n n! \boldsymbol{\tau} \mathbf{S}^{-n} \mathbf{1}. \tag{5.12}$$

for which of particular interest to us is the central moment:

$$\mathbb{E}(T) = -\boldsymbol{\tau} \mathbf{S}^{-1} \mathbf{1} \tag{5.13}$$

We can now apply these tools to $\mathbf{Y}(t)$ to compute its average absorption times, thus obtaining the expectation for the first hitting time of $\mathbf{E}(t)$, $\mathbb{E}(\mathbf{T}_{23}^{\mathbf{E}})$ and, ultimately, to $\mathbb{E}(\mathbf{X}_e)$.

5.2.3-2 Computing $\mathbb{E}(\mathbf{T}_{23}^{\mathbf{E}})$ and $\mathbb{E}(\mathbf{X}_e)$. Recalling that to specify the phase-type distribution that describes the time-to-absorption of $\mathbf{Y}(t)$, we first need to compute its generator matrix.

To compute its generator matrix, in turn, we need both the holding times and the transition probabilities of the embedded jump chain of $\mathbf{Y}(t)$. The model of Figure 5.3 does not give us those directly, so we need to compute them ourselves. For all states in

Figure 5.3 – except for state 3 – we have two exponential distributions with parameters λ_j and λ_k which may cause the chain to change state. This is illustrated in the "piece" of chain $\mathbf{Y}(t)$ depicted in Figure 5.4. Let Y_j and Y_k be two random variables drawn from these two distributions. We now present two simple theorems for the chain in Figure 5.4 which allow us to compute what we need for chain $\mathbf{Y}(t)$.

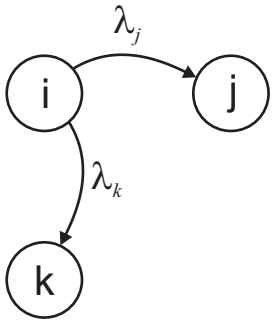


Figure 5.4: A "piece" of the chain that governs $\mathbf{Y}(t)$.

Theorem 1 (Sojourn Time)**.** *The sojourn time in state i is given by* $\exp[\lambda_j + \lambda_k]$.

Proof. The amount of time $\mathbf{Y}(t)$ spends in state i is given by $\min\{Y_j, Y_k\}$ – the smallest time until one of the exponentials make the process jump to another state. It is a known result [90] that the minimum of two exponential distributions with parameters λ_j and λ_k is also distributed according to an exponential distribution with parameter $\lambda_j + \lambda_k$, which gives us our proof. □

Theorem 2 (Transition Probability)**.** *The chain transitions from state i to state j with probability* $\frac{\lambda_j}{\lambda_j+\lambda_k}$.

Proof. The chain transitions from i to j if and only if $Y_j < Y_k$ – otherwise, it would transition to k. Further, we can disregard the case $Y_j = Y_k$ since $\mathbb{P}[Y_j = Y_k] = 0$. It is a known result [90] that if Y_j and Y_k are independent exponentially distributed random variables with parameters λ_j and λ_k, then $\mathbb{P}[Y_j < Y_k] = \frac{\lambda_j}{\lambda_j+\lambda_k}$, and the result follows from there. □

The transition matrix for our chain is therefore given by:

$$Q = \begin{bmatrix} 0 & \frac{\gamma_v}{\gamma_u+\gamma_v} & \frac{\gamma_u}{\gamma_u+\gamma_v} & 0 \\ \frac{\mu_v}{\mu_v+\gamma_u} & 0 & 0 & \frac{\gamma_u}{\mu_v+\gamma_u} \\ \frac{\mu_u}{\mu_u+\gamma_v} & 0 & 0 & \frac{\gamma_v}{\mu_u+\gamma_v} \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.14}$$

and the generator matrix by:

$$A = \begin{bmatrix} -(\gamma_u + \gamma_v) & \gamma_v & \gamma_u & 0 \\ \mu_v & -(\mu_v + \gamma_u) & 0 & \gamma_u \\ \mu_u & 0 & -(\mu_u + \gamma_v) & \gamma_v \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.15}$$

With the generator matrix of $\mathbf{Y}(t)$ in hands, we now use the formula for the central moment of a phase-type distribution (Equation (5.13)) to derive $\mathbb{E}\left(\mathbf{T}_{23}^{\mathbf{Y}}\right) = \mathbb{E}\left(\mathbf{T}_{23}^{\mathbf{E}}\right)$. Our model always starts at state 2, so $\boldsymbol{\tau} = (0, 0, 1)$. Since $\mathbf{S}$ is 3×3, computing the inverse analytically is feasible:

$$\mathbf{S}^{-1} = \begin{pmatrix} -(\gamma_u + \gamma_v) & \gamma_v & \gamma_u \\ \mu_v & -(\gamma_u + \mu_v) & 0 \\ \mu_u & 0 & -(\gamma_v + \mu_u) \end{pmatrix}^{-1} =$$

$$= \begin{pmatrix} -\frac{(\gamma_v+\mu_u)(\gamma_u+\mu_v)}{\gamma_u\gamma_v(\gamma_u+\gamma_v+\mu_u+\mu_v)} & -\frac{\gamma_v+\mu_u}{\gamma_u(\gamma_u+\gamma_v+\mu_u+\mu_v)} & -\frac{\gamma_u+\mu_v}{\gamma_v(\gamma_u+\gamma_v+\mu_u+\mu_v)} \\ -\frac{(\gamma_v+\mu_u)\mu_v}{\gamma_u\gamma_v(\gamma_u+\gamma_v+\mu_u+\mu_v)} & -\frac{\gamma_u+\gamma_v+\mu_u}{\gamma_u(\gamma_u+\gamma_v+\mu_u+\mu_v)} & -\frac{\mu_v}{\gamma_v(\gamma_u+\gamma_v+\mu_u+\mu_v)} \\ -\frac{\mu_u(\gamma_u+\mu_v)}{\gamma_u\gamma_v(\gamma_u+\gamma_v+\mu_u+\mu_v)} & -\frac{\mu_u}{\gamma_u(\gamma_u+\gamma_v+\mu_u+\mu_v)} & -\frac{\gamma_u+\gamma_v+\mu_v}{\gamma_v(\gamma_u+\gamma_v+\mu_u+\mu_v)} \end{pmatrix}$$

from where we get:

$$\mathbb{E}(\mathbf{T}_{23}) = -\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \cdot \mathbf{S}^{-1} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \frac{(\gamma_u + \mu_u)(\gamma_u + \gamma_v + \mu_v)}{\gamma_u\gamma_v(\gamma_u + \mu_u + \gamma_v + \mu_v)} \tag{5.16}$$

Finally, by putting together Equations (5.4), (5.5), and (5.16), we obtain the closed-form expression for expected edge delays:

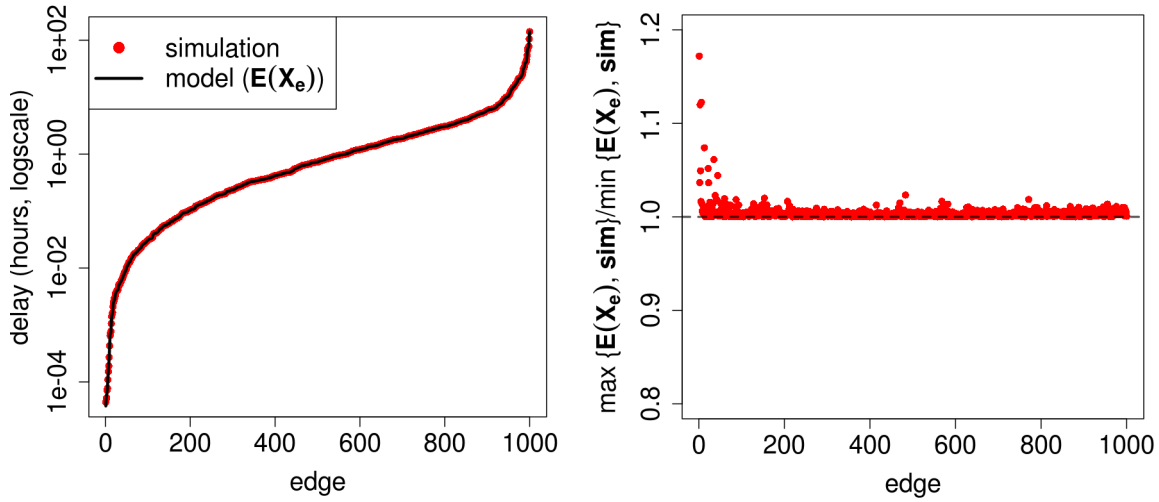
$$\mathbb{E}(\mathbf{X}_e) = \pi_{v,0} \cdot \mathbb{E}(\mathbf{T}_{23}) = \frac{\mu_v}{\mu_v + \gamma_v} \cdot \frac{(\gamma_u + \mu_u)(\gamma_u + \gamma_v + \mu_v)}{\gamma_u\gamma_v(\gamma_u + \mu_u + \gamma_v + \mu_v)} \tag{5.17}$$

5.2.4 Validation

We validate Equation (5.17) by comparing it against numerical edge delay estimates over 1 000 different edges. Availability parameters are generated according to the Yao model, as described in Section 5.1. Numerical estimates are obtained by running the simulation described in Algorithm 3 for each edge, with sample averages computed from 1 000 000 samples.

Figure 5.5a shows a scatterplot of the numerical estimates for all edges, sorted by value. Model estimates obtained from Equation (5.17) are shown as a black line. We can see that the match is quite accurate. To quantify it numerically, we produce an error estimate by taking each simulation estimate (**sim**) and each model estimate ($\mathbb{E}(\mathbf{X}_e)$) and

computing the ratio $\max\{\mathbb{E}(\mathbf{X}_e), \mathbf{sim}\}/\min\{\mathbb{E}(\mathbf{X}_e), \mathbf{sim}\}$. This is shown in Figure 5.5b. Points, which are sorted by their value of **sim**, land mostly near 1. Deviations hardly go above 3%, except at the lowest delay region (below 30 seconds) where roundoff and very small absolute deviations end up translating into large relative errors, which nevertheless do not go above 17%.



(a) Absolute values.

(b) Relative error.

Figure 5.5: Expected edge delays in simulation and model.

5.3 Path Delays

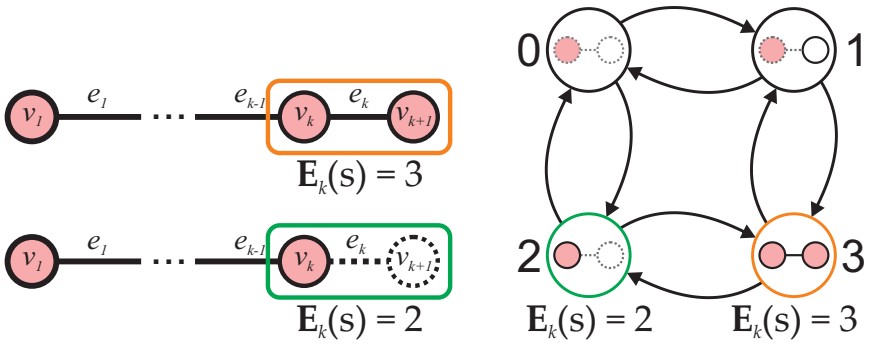
In this section, we proceed towards the next building block of our model: paths. We begin by proving that the fundamental result relating edge to path delays of Equation (4.6) holds as an equality when session and inter-session lengths are exponentially distributed.

5.3.1 The Path Delay Theorem

Recapping, let $\mathbf{D}_P$ be the random variable representing the delay incurred by a path $P = \{e_1, \cdots, e_n\}$, and let $\mathbf{X}_{e_i}$ be the random variable representing the delay incurred by edge e_i. Then:

Theorem 3 (Path Delay Theorem)**.** *For any* $P = \{e_1, \cdots, e_n\}$*:*

$$\mathbf{D}_P = \sum_{1}^{|P|} \mathbf{X}_{e_i} \tag{5.18}$$



(a) Possible states for e_k at transition point.

(b) Edge e_k's CTMC, $\mathbf{E}_k(t)$.

Figure 5.6: The transition between v_k and v_{k+1}.

Proof. We will prove that Equation (5.18) holds by induction on prefixes of P. Let $R(k)$ represent the (unique) prefix of P of length k, i.e. $R(k) = \{e_1, \cdots, e_k\}$, $k \leq |P|$.

Base case. Equation (5.18) holds for $R(1)$.

Proof. Let $e_1 = (v_1, v_2)$. The delay incurred by path $R(1) = \{e_1\}$ is given by the time elapsed from the login of v_1 to the time at which v_1 and v_2 are online at the same time. But this is the definition of the delay of edge e_1. It therefore follows that $\mathbf{D}_{R(1)} = \mathbf{X}_{e_1}$.

Induction step. If Equation (5.18) holds for $R(k-1)$, then it holds for $R(k)$.

Proof. Denote by v_i, $i \leq k$, the i-th vertex along $R(k)$, such that $e_i = (v_i, v_{i+1})$. We derive our proof by examining what happens as a message transitions from v_k – the last vertex of $R(k-1)$ – to v_{k+1} – the last vertex of $R(k)$.

Recall, from Section 5.2, that edge $e_k = (v_k, v_{k+1})$ can be described as a four-state, continuous-time Markov chain (Figure 5.6b). Here, we denote this Markov chain by $\mathbf{E}_k(t)$. We now repeat a reasoning very similar to the one made in Section 5.2.

When a message reaches v_k from v_{k-1} at some arbitrary time instant s, we know that v_k has to be online to receive it. But there are only two states in $\mathbf{E}_k(s)$ for which v_k is online, namely, states 2 and 3. Therefore, at instant s, either $\mathbf{E}_k(s) = 2$, or $\mathbf{E}_k(s) = 3$. This is illustrated in Figure 5.6a. Since we assume the system to be running for "long enough", the probabilities of having $\mathbf{E}_k(s)$ at states 2 or 3 are given by π_0 and π_1, the stable-state probabilities for v_{k+1} to be either logged off or on (i.e. Equation (5.3)) applies).

If $\mathbf{E}_k(s) = 3$, then delay is zero. Otherwise, since $\mathbf{E}_k$ is Markov, Equation (5.2) says that delay is distributed according to $\mathbf{T}_{ij}^{\mathbf{E}_k}$. But this is exactly the distribution of $\mathbf{X}_{e_k}$. It follows that $\mathbf{D}_{R(k)} = \mathbf{D}_{R(k-1)} + \mathbf{X}_{e_k}$. By the inductive hypothesis, $\mathbf{D}_{R(k-1)} = \sum_{i=1}^{k-1} \mathbf{X}_{e_i}$, so that $\mathbf{D}_{R(k)} = \sum_{i=1}^{k-1} \mathbf{X}_{e_i} + \mathbf{X}_{e_k} = \sum_{i=1}^{k} \mathbf{X}_{e_i}$, and this completes the proof.

The key here is that, because of exponential distributions, $\mathbf{E}_k$ is Markov. Were it not Markov, we would have to consider the instant s at which the message reaches v_k, and this would complicate the analysis significantly. □

5.3.2 Expected Path Delays

An immediate implication of Theorem 3 is that (4.7) also holds as an equality:

Corollary 1. *For any $P = \{e_1, \cdots, e_n\}$:*

$$\mathbb{E}(\mathbf{D}_P) = \sum_{1}^{|P|} \mathbb{E}(\mathbf{X}_{e_i}) \tag{5.19}$$

By combining this Corollary 5.19 with Equation (5.16), we are able to provide a closed-form expression for the expected delay of a path $P = \{e_1, \cdots, e_n\}$. For each vertex v_i along P ($1 \leq i \leq n+1$), denote by μ_i and γ_i the parameters of the exponential distributions governing its session and inter-session lengths, respectively. Then:

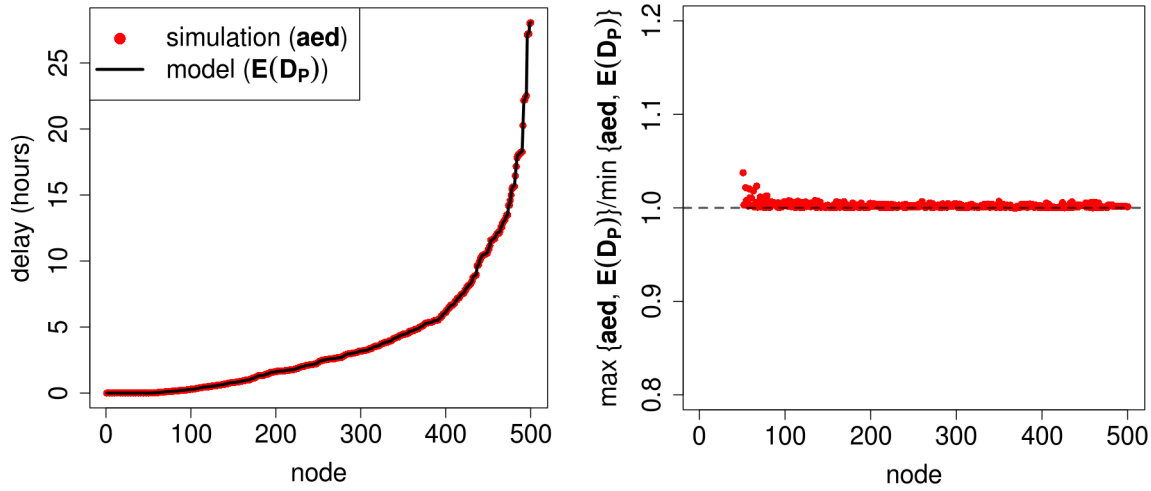
$$\mathbb{E}(\mathbf{D}_P) = \sum_{i=1}^{|P|} \frac{\mu_{i+1}}{\mu_{i+1} + \gamma_{i+1}} \cdot \frac{(\gamma_i + \mu_i)(\gamma_i + \gamma_{i+1} + \mu_{i+1})}{\gamma_i \gamma_{i+1}(\gamma_i + \mu_i + \gamma_{i+1} + \mu_i)}$$

The implication of this formula is significant, in that it essentially enables the computation of the upper bounds of Section 4.4.2 without any simulations, effectively rendering the application of the technique to large graphs feasible.

5.3.3 Validation

We validate Equation (5.3.2) by comparing it against numerical estimates obtained by running full simulations over 50 distinct availability assignments in list graphs of size 10. Recalling that list graphs, which were introduced in Section 4.4.4-1, are graphs in which nodes are organized as doubly-linked lists, and essentially emulate paths of equal length. Full simulations are ran by setting the leftmost node as the source, and then measuring the end-to-end delays (**ed**) towards all of the nodes to the right. The rightmost node, in particular, will be 9 hops away from the source. The numeric average **aed** is then computed for each node, and compared with the value yielded from Equation (5.3.2) for the same parameters, which we refer to as $\mathbb{E}(\mathbf{D}_P)$.

Figure 5.7a shows a scatterplot of **aed** values ordered by increasing magnitude, along with $\mathbb{E}(\mathbf{D}_P)$ values depicted as a black line. Again, visually, the two overlap almost perfectly. We again quantify deviations with the ratio $\max\{\mathbf{aed}, \mathbb{E}(\mathbf{D}_P)\} / \min\{\mathbf{aed}, \mathbb{E}(\mathbf{D}_P)\}$, shown in Figure 5.7b. Again, deviations tend to be rather small, going above 3% only in the lowest delay region, and staying nevertheless below 5%.



(a) Absolute values.

(b) Relative error.

Figure 5.7: Expected path delays in simulation and model.

5.4 Towards Practical and Precise Delay Estimates

The main use of Equation (5.3.2) is that it can help us compute, in a very efficient way, the expected delays over the "fastest" path connecting an arbitrary pair of nodes u and v over a graph G. This, in turn, provides us with an upper bound on the "real" expected delay over G. As we have seen in Section 4.4.4, however, these single-path estimates – although reasonable – can be rather coarse.

In Section 4.4.3 we have suggested a way to improve on those estimates: by considering the top-k fastest paths connecting nodes u and v, instead of only the single fastest path. Estimates, in this case, are obtained by measuring delays over smaller graphs, which are formed as we put these paths together. We refer to these graphs as $G_{k,u,v}$, or "top-k graphs".

The natural next step in our modelling effort would be analytically deriving the distributions and moments for delays over top-k graphs, as we did with edges and paths. Yet, as it turns out, top-k graphs are much more elusive. We briefly discuss the key issue behind modelling delays over top-k graphs in Section 5.4.1. The interested reader can find a detailed account of our efforts in Appendix B.

This would seem to indicate that practical and accurate estimation of delays over large graphs is currently not possible. Yet, by realizing that accuracy is only important for pairs of nodes that exhibit high delays, we derive a different hybrid technique from our current

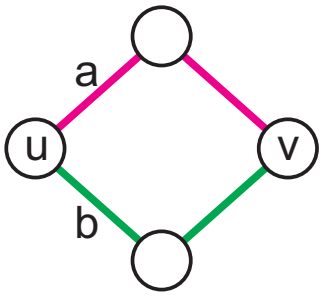


Figure 5.8: A simple top-k graph made of two paths.

toolset which, we believe, can help us estimate delays at a practical cost where it matters the most. We sketch this technique – which we do not validate – in Section 5.4.2.

5.4.1 Analytical Modelling of top-k Graphs

The core idea behind our model for delays over top-k graphs is that the delay between a pair of nodes u and v that is connected by a set of k paths $P_1, \cdots, P_k$ should be equal to the delay of whichever path "arrives first" at the destination v. In other words, if we denote by $\mathbf{D}_{P_i}$ the random variable representing the delay of path P_i, we have that the delay between u and v should be given by $\min\{\mathbf{D}_{P_1}, \cdots, \mathbf{D}_{P_n}\}$.

Yet, things are far from being that simple. Indeed, this model carries and important implicit assumption that fails to hold over top-k graphs – namely, that edge delays are *independent*. To understand the issue, suppose we had the simple top-k graph depicted in Figure 5.8, and that we were looking at the delay distributions of edges a and b – i.e. at the distributions of the random variables $\mathbf{X}_a$ and $\mathbf{X}_b$. Further, to make the discussion simpler, assume, without loss of generality, that at time instant $t = 0$:

1. node u already has a message to send, but it is currently offline (i.e. the chains $\mathbf{E}(t)$ associated to both edges a and b start at state 2);
2. from time instant t onwards, nodes x and y are always online.

For our simple model to hold, we need edge delays to be independent. In other words, we need to have, for $x, y \in \mathbb{R}$, $x > 0, y > 0$, that $\mathbb{P}(\mathbf{X}_a < x \mid \mathbf{X}_b = y) = \mathbb{P}(\mathbf{X}_a < x)$ (i.e. $\mathbf{X}_b$ bears no influence on X_a). Yet, this is clearly not true in the scenario we depict. Indeed, since x and y are always online, we know that the only way for $\mathbf{X}_b$ to be equal to y is if node u remained down for y time instants. Therefore, $\mathbb{P}(\mathbf{X}_a < x \mid \mathbf{X}_b = y) = 0$, if $x < y$, or 1, otherwise. It follows that $\mathbf{X}_a$ and $\mathbf{X}_b$ are not independent.

The key here is that edges a and b share a vertex – vertex u – which couples their behavior. Indeed, if node u remains offline for a period of time δ, this will increase the delays of both edges a and b by δ, and not by two independent, randomly distributed

amounts. And this reflects itself when we take the minimum among all paths. Yet, this dependence is rather difficult to model. A detailed account of all the issues involved, as well as our partial results on a model for top-k graphs, can be found in Appendix B.

5.4.2 A More Practical Technique

Direct analytical modelling of delays over top-k graphs proved to be elusive, with the implication that the exhaustive, accurate assessment of delays over all pairs of nodes in large graphs remains impractical. Yet, when the goal is assessing feasibility and performance of social overlays, we argue that such accurate estimates might not required for that many pairs of nodes in the graph after all. Indeed, if for a given pair of nodes u and v we obtain coarse, single-path upper bounds on the order of seconds, then we can be confident that performance for this particular pair is more than reasonable, no matter how inaccurate estimates are.

And this reasoning leads us to our technique. We start by computing coarse, cheap upper bounds for all pairs of nodes in the network. We then improve these delay estimates by running simulations to refine them, *but only for pairs of nodes for which upper bounds yield values that are too high*. This refinement step effectively reduces upper bound uncertainty for the pairs of nodes for which such uncertainty matters.

The notion of a delay that is "too high" is application-specific, but we could, for instance, use Equation (4.4) to map end-to-end delays into receiver delays, and exclude from further consideration any node for which the estimated receiver delay is less than, say, 10 minutes. In more detailed terms, the technique we propose works in three phases:

1. compute the coarse, single-path delay estimates of Section 4.4.2 for all node pairs by applying Equation (5.3.2). This is equivalent to solving the all-pairs shortest path problem in a weighted graph, for which efficient algorithms are well-known [24];
2. based on a *significance criterion*, identify the set of node pairs $S \subset V \times V$ for which delay estimates produced in phase 1 are "too high";
3. refine the measurements for pairs $(u, v) \in S$ by constructing $G_{u,v,k}$ and running simulations over it.

Clearly, if set S is too large, then the technique is not cost-effective. Yet, our experiments so far show that pairs with high receiver delays constitute only a minority of the total set. As an example, if we adopt the same set of 137 260 source/destination pairs and the same availability assignments of Section 4.4.4, but under an exponential model, then less than 10% of our source/destination pairs exhibit receiver delays of more than 10 minutes. This means we might be able to reduce estimation costs significantly. The magnitude of this reduction, however, remains to be seen.

5.5 Discussion and Outlook

In this chapter, we have replaced the heavy-tailed distributions that made the analytical model from Chapter 4 difficult to treat analytically and, based on Markov chain theory, have developed closed-form expressions for expected path and edge delays, as well as shed light on their distributions. These expressions effectively enable the computation of delay bounds on large graphs, and should lead to a better understanding of their delay characteristics.

The difficulties related to modelling delays analytically over top-k graphs mean that we cannot, currently, provide low-cost/high precision estimates in a purely analytical way. Yet, by putting together the tools we have developed so far, we propose a mixed technique in which low delay pairs are first filtered out by means of cheaper techniques, and then delay estimates are progressively refined for the remaining pairs, by means of more expensive techniques.

We believe that these results, taken together, will prove useful to researchers who wish to understand the delay characteristics of social overlays under churn. As for ourselves, we intend to continue pursuing a viable analytical model, as well developing and refining the techniques made available in this work as we conduct studies over larger-scale graphs.

Chapter 6

Cloud to the Rescue!

> Clouds come floating into my life, no longer to carry rain or usher storm, but to add color to my sunset sky.
>
> –*Rabindranath Tagore*, Stray Birds

In the last two chapters, we have focused on analysing the disruptions in communication introduced by churn on social overlays, with the key conclusion that delays might grow very large and impact system usability. We now turn ourselves once again to the minimalistic, SO-based, decentralized OSN design of Section 2.2, and to the problem that emerges from this new knowledge: *how to disseminate updates efficiently when the overlay network does not allow us to do so?* As it is often the case, the key is to be willing to compromise: while a pure P2P solution might not work as well as we would like to, we will show that the delay tail can be mitigated by exploring a novel, hybrid approach.

P2P and the cloud: getting the best of both worlds. The main issue with social overlays as a dissemination medium is that they do not provide enough "options" to propagate updates in the presence of churn. To address this shortcoming, we propose a simple, and yet effective idea: to create an out-of-band channel that can "patch" connectivity if and when needed, by leveraging the persistence and ubiquity of cloud services.

In a nutshell, the scheme works as follows. The bulk of profile update dissemination is still carried out in a fully decentralized, P2P fashion on the social overlay. Specifically, we continue to use the push protocol (MAXCOMP) of Chapter 3. In addition, each node u is associated to a *profile store*, which is hosted on the cloud. Updates to u's profile (by u or some of its friend nodes) are always performed first on the profile store, and then disseminated via the social overlay. Therefore, the profile store of u contains an always-available, consistent copy of her profile. The availability of the profile store is key in overcoming the aforementioned delays in the propagation of profile updates. When a

node v, friend of u, has not heard any update from u for a predefined time interval, it assumes that the update has been delayed, and verifies whether this is the case by polling the profile store. In principle, this naïve solution is enough to overcome the aforementioned limitations. However, cloud access has a monetary cost, and we show that this solution has a poor performance/money tradeoff. We improve over this baseline by disseminating the outcome of polling the profile store back on the social overlay. This technique has always the beneficial effect of quenching cloud access from other nodes (i.e., saving money) and, in the case where an update exists, of speeding up dissemination.

These two metrics, dissemination delay and monetary cost, make up the core of our evaluation. Results show that, compared with a fully decentralized solution, our hybrid solution brings maximum delays from hours to minutes, and average delays from minutes to seconds.

Roadmap. The remainder of the chapter is organized as follows. Section 6.1 introduces complementary assumptions on the system model. Section 6.2 states the problem, namely, mitigating dissemination delays over social overlays in the presence of churn. Section 6.3 illustrates our hybrid solution leveraging the combination of P2P dissemination on the social overlay with occasional cloud access, which is then evaluated in Section 6.4. Section 6.5 places our work in the context of related efforts. Section 6.6 draws conclusions and points at opportunities for future work.

6.1 System Model

In this section, we detail the extra assumptions required by the problem we tackle in this chapter.

Clock synchronization. We assume clocks to be loosely synchronized, within the order of minutes. This can be easily achieved by NTP [110].

Availability. As before, we adopt the availability model of Yao et al. [128]. We adopt, however, the version described in Chapter 5, in which both session and inter-session times follow exponential distributions $\exp[\lambda]$. The reason is a practical one: the simulations we perform in this chapter are more complex, and more expensive, than the ones of Chapter 4. We therefore cannot afford nearly as many repetitions, and have to trade off precision for fast convergence of estimators instead. As we will see in Section 6.4.4, however, delays remain a problem under exponentials as well. And, once again, we note that this simplification is in line with other works in the literature [91,107]. Heterogeneity is also modelled as in Chapter 5.

Cloud Access. We assume the existence of a highly available, cloud-based infrastructure which nodes can access to store and retrieve data. Such a cloud infrastructure allows users

to create personal storage areas under their control, i.e, they can selectively allow or deny read and write access to other users.

Furthermore, we adopt the *requester-pays* billing scheme of Amazon S3 [5], in which users who access data are charged for it. In other words, if a user v decides that she wants to download the new updates from her friend u directly from u's personal storage area (and is authorized to do so by u), it is up to v to pay for the download costs. This is important as it establishes the basis for the fair cost model of our solution: a user might opt to either go directly to the cloud and pay to immediately download updates from his friends, or use the free P2P network instead, but potentially face large delays.

In S3, costs can be broken down into three components:

1. *Cost per request.* In S3, a user pays 0.01¢ for every $10,000$ GET requests, or every $1,000$ PUT requests. We assume a similar cost model in which read requests are cheaper than writes;
2. *Bandwidth costs.* The more a user downloads from the cloud, the more she pays;
3. *Storage costs.* Keeping data in the cloud has a fixed monthly cost which increases with the amount of data stored, and is independent of whether or not it is accessed.

Storage costs are expected to be very small in our case. First, profile pages are made up of small bits of content, so that the total amount of data stored by a user should also be small (e.g., a few gigabytes). Second, storage is cheap: at the time of this writing, the yearly cost for a gigabyte is around one US dollar. Since storage costs do not impact our figures to a measurable extent, we choose to abstract them away altogether.

Similar considerations hold for bandwidth: as we have stated in Section 2.3, updates are expected to be small in size and, as we later show, our protocol works by favouring many lightweight requests (a few hundred bytes per request) over fewer, larger requests. The dominant cost component of our protocol is, therefore, the cost per request, and is the one we focus on for our evaluation.

6.2 Problem Statement

We adopt, in this chapter, the same notions (and notation) of end-to-end (**ed**) and receiver delay (**rd**) we have defined in Section 4.2. Our main goal is that of solving the problem of update dissemination while providing an acceptable user experience. In formal terms, we can express this as the enforcement of a *target delay bound* δ on update delivery. In other words, we would like to guarantee that:

$$\mathbf{rd}(v, w, t_0) \leq \delta \tag{6.1}$$

for every source v, receiver w, and time instant t_0.

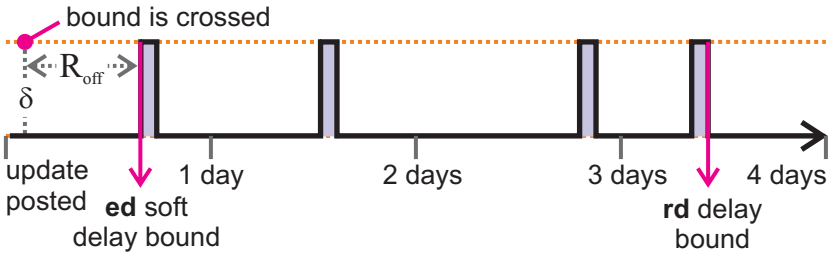


Figure 6.1: Delay bounds for cloud-assisted dissemination.

Putting a cap on **rd**, however, is not enough, as it allows for some undesirable situations to emerge. Suppose we were to set δ to 20 minutes and, for the sake of argument, we had a receiver like the one in Figure 6.1, who logs in for 5 minutes every day. From the point of view of Equation (6.1), the bound is honored as long as we deliver the update before the end of the 4^{th} day.

But this is too long: an update that is four days old is not as useful as one that is one day old. Further, the receiver did log in on a number of occasions over the course of these 4 days, which means that there were windows of opportunity to deliver it. And this shows the main weakness of using a pure **rd** bound: it allows end-to-end delays to become excessively, unnecessarily large. We need a stronger bound. Yet, bounding **ed** directly is in general not possible, as the situation depicted in Figure 6.1 shows – if the receiver were to be offline at the instant at which we cross the bound, there would be no way the system could deliver the update on time.

We reach a compromise by allowing a *soft delay bound* on **ed**. The idea is that as soon as the *target delay bound* δ is crossed, the system must deliver the update *at the next login of the receiver*. We express this by adding some slack time to the bound in case w is offline. This slack time represents the residual offline time $\mathbf{R}_{off}$ of w until its next login. Formally:

$$\mathbf{ed}(v, w, t_0) \leq \begin{cases} \delta & \text{if } w \text{ online at } t_0 + \delta \\ \delta + \mathbf{R}_{off} & \text{otherwise} \end{cases} \tag{6.2}$$

Note that the slack time is actually a random variable, for which the probability distribution results directly from the availability model. The bound in Equation (6.2), therefore, is no longer an exact, one-size-fits-all bound, but rather a probabilistic bound which exhibits different statistical behavior for every receiver. And this makes sense, since different availabilities lead to different guarantees. Finally, note that since the slack is composed entirely of offline time, it does not count as receiver delay. Therefore, by honouring Equation (6.2) we are also automatically honouring Equation (6.1).

The goal we set for this chapter is that of providing a hybrid cloud/P2P dissemination protocol which can honor the soft latency bounds of Equation (6.2) while being efficient,

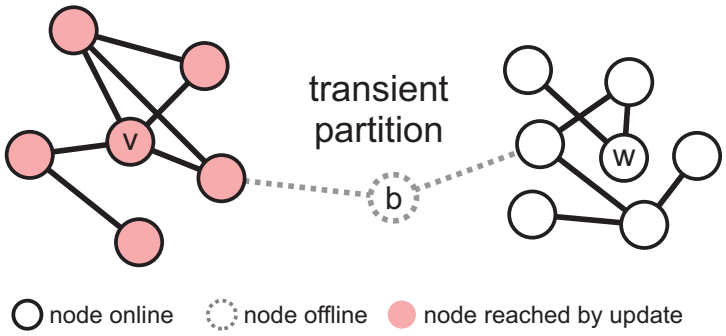


Figure 6.2: Transient partitions in social overlay.

low-cost, and offering a good compromise on user privacy w.r.t. cloud providers.

6.3 Cloud to the Rescue!

The need for cloud resources arises because the social overlay cannot provide acceptable delays to all source/destination pairs in the network. As Figure 6.2 shows, this happens because the absence of certain nodes can create transient partitions that disrupt communication paths and introduce delay.

What we need, then, is a way to "patch" such partitions. A simple way of achieving this goal would be by associating each node u to an *alias* $\tilde{u}$ of itself in the cloud. $\tilde{u}$ would ideally be indistinguishable from u to interacting nodes, and would be activated on-demand to satisfy requests on behalf of u, should u be offline. The net effect of having cloud aliases is that the cause of the transient partitions in Figure 6.2, and their associated delays, would cease to exist, thus solving the problem.

Unfortunately this solution has a number of drawbacks:

1. *Cost.* Cloud providers which allow instances to be brought up on-demand, such as EC2 [4], charge significant prices-per-hour and, as a result, activating an alias for extended periods of time can rapidly become economically unattractive. The alternative – hiring permanent, always-on hosting – is also expensive, with even the lower-cost providers (e.g. [75]) charging in excess of $100 for one-year contracts.
2. *Privacy.* We do not want the cloud provider to be able to inspect the full memory state of the running software: although we believe providers to have no incentive to behave in an adversarial way, they might have their own internal security issues. We therefore share the guideline of Liu et al. [60] that users should minimize information exposure to providers.
3. *Effectiveness.* We have seen in Section 4.3 that delays can remain high even when a significant fraction (70%) of the user population adopts always-on hosting. While

one could argue that results could have been different had we been judicious with the selection of nodes to be made always-on instead of picking them at random, it is in general unreasonable to expect that we can convince individual users to adopt always-on hosting. Indeed, in practice, we might not even be able to identify who are the users causing the delay bottlenecks.

6.3.1 Update Dissemination with Profile Stores

The aforementioned problems led us to consider an alternative solution, in which $\tilde{u}$ is no longer required to be a full clone of u. Instead, we make $\tilde{u}$ a simple high-availability *profile store* in which $\mathbf{pp}(u)$ is kept.

Publishing an update to a profile store is simple—if user v wants to post something to $\mathbf{pp}(u)$, all it has to do is write this update directly to $\tilde{u}$. Access control is also not a problem, since the primitives provided by services such as Amazon's S3 [5] make it straightforward to ensure that only authorized users get to write to $\tilde{u}$. By adopting the "requester-pay" model of S3, we ensure that each user has complete control on her expenses, as discussed in Section 6.1. To prevent the cloud provider from accessing sensitive data, all data stored in $\tilde{u}$ is encrypted before upload. This is in stark contrast with centralized solutions such as Facebook, in which the advertisement-based business model effectively precludes storing encrypted data from being acceptable practice.

However, differently from cloud aliases, profile stores are passive in that they cannot initiate interactions with other nodes. Therefore, to actively overcome the transient partitions that cause delays, we have to resort to periodic *polling*.

Naïve approach. In its simplest form, this polling procedure works by having each node $w \in V(G_u)$ independently poll $\tilde{u}$ every δ time instants, and retrieve any new updates to $\mathbf{pp}(u)$ that might have been posted in the meantime. If w happens to be offline after δ time instants have passed, then w accesses the cloud immediately once it logs back in. This simple protocol, which we refer to as PurePoll, essentially allows us to circumvent the transient partitioning caused by node b through an out-of-band channel, and satisfy Equation (6.2).

Hybrid approach. While simple, PurePoll is wasteful in that it disregards the existence of low-delay paths in the social overlay which could be used to our advantage. To understand how, note that if we were to discard all paths in the social overlay that have delays above or close to the delay bound δ we wish to maintain – such as the paths that go through node b in Figure 6.2 – we would be left with a set of disjoint groups for which the *internal* delays are low, as illustrated in Figure 6.3. We call these *delay groups*. To get a message disseminated over a set of delay groups while respecting the delay bound δ, then, all we have to do is to ensure that *at least one* node in each of these groups actually

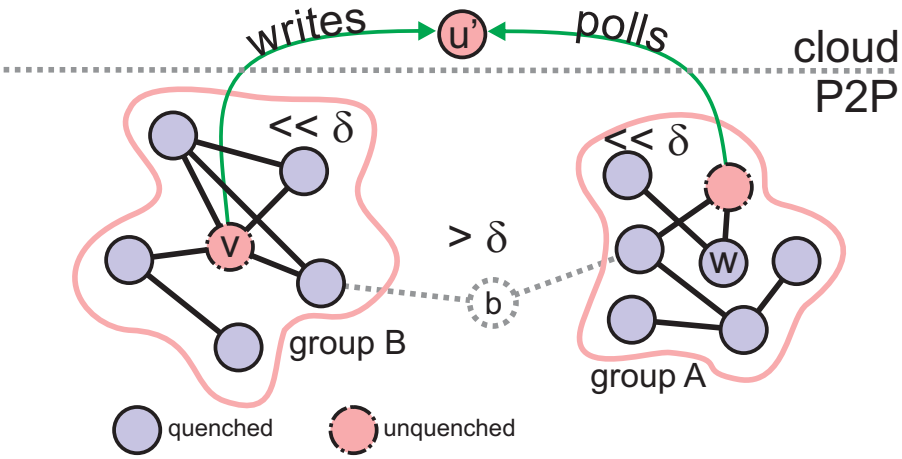


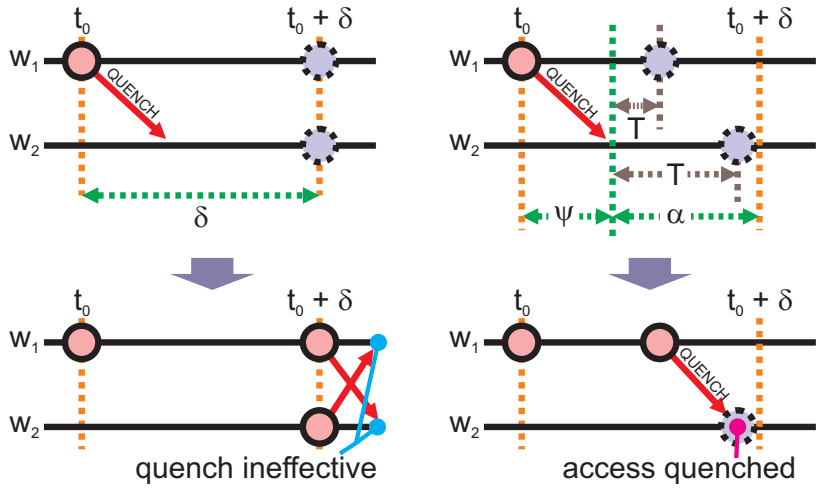
Figure 6.3: Delay groups and HYBRID.

accesses the cloud every δ time instants. The other nodes can then get the update from this single accessing node directly from the social overlay – which can provide fast enough dissemination inside of each group – and avoid accessing the cloud themselves.

The second method we propose, named HYBRID, does just that. In simple terms, every node $w \in V(G_u)$ keeps track of the last instant in time at which it heard any news from u. Whenever w goes for more than δ time instants without hearing from u, it polls the profile store of u to see if there are new updates. In case there are, w downloads these updates from the cloud and pushes them into the social overlay by means of the push gossip protocol from Chapter 3 (MAXCOMP). Otherwise, w pushes a special QUENCH message that contains the time t_0 at which w accessed the cloud and found nothing new. This message serves to inform other nodes that, as of t_0, there are no new updates, and that they can therefore refrain themselves from accessing the cloud for an extra δ time instants. We call this mechanism *access quenching*.

To see how this protocol approximates the ideal situation of dissemination over delay groups that we described previously, note that if two nodes belong to the same delay group, then it is quite likely that they will hear the accesses from one another, resulting in access quenching. If two nodes do not belong to the same delay group, on the other hand, it is unlikely that they hear each other in time to promote quenching, and two separate cloud accesses will ensue. This simple protocol, therefore, effectively adjusts to the delay characteristics of the surrounding network, and provides a self-organizing mechanism for bridging transient partitions by polling.

Randomizing cloud accesses. The scheme just described would work fine were it not for the fact that, as it is now, the protocol induces nodes belonging to the same delay group to synchronize their accesses to the profile store.



(a) Fixed. (b) Randomized.

Figure 6.4: Quenching.

The issue is illustrated in Figure 6.4a, in which node w_1 initially accesses $\tilde{u}$ at time t_0 and, having found no new updates, schedules its next access to $t_0 + \delta$ to respect the soft delay bound. At the same time, w_1 propagates this knowledge over the P2P network by means of a QUENCH message. Upon receiving the QUENCH message, node w_2 learns that, as of time t_0, there are no new updates to $\mathbf{pp}(u)$, and therefore it can refrain itself from accessing the cloud until $t_0 + \delta$. The two nodes are now synchronized. When we get to instant $t_0 + \delta$ (Figure 6.4a), the two nodes access $\tilde{u}$ at the same time. Having again found no updates, they disseminate their QUENCH messages, but to no effect: since the accesses are too close to one another, quenching is ineffective.

To solve this problem, we scatter access times near $t_0 + \delta$ as follows. First, we break δ down into a *fixed* component ψ, and a *random* component α, such that $\delta = \psi + \alpha$.

Then, let $\mathbf{T}$ be a random variable drawn from a uniform distribution $\mathcal{U}(0, \alpha)$. Now, when node w_1 accesses the cloud and, later, when w_2 receives the QUENCH message, they set their access times to $t_0 + \psi + \mathbf{T}$. This causes w_1 and w_2 to have slightly different access times after $t_0 + \psi$ which, provided α is sufficiently large, should be enough to make quenching effective (Figure 6.4b). Note that since $0 \leq \mathbf{T} \leq \alpha$, we have that $\psi + \mathbf{T} \leq \psi + \alpha = \delta$. The modified protocol, therefore, still honours the soft bounds of Equation (6.2).

6.3.2 Hybrid in Detail

In HYBRID, every node $w \in V(G)$ keeps track of a *last seen* timestamp $last[u]$ which represents the last time instant at which w has heard any news from $u \in f(w)$. It

also keeps track of a version number *version*[u] for $\mathbf{pp}(u)$, which it uses to understand whether or not its local version of $\mathbf{pp}(u)$ is up-to-date w.r.t. the one in the profile store $\tilde{u}$. In addition, w keeps track of its current, randomized *target*[u].

Whenever the target delay bound is crossed and w has not heard any news from u, it accesses the cloud. The description of the access protocol is given in Algorithm 6. The first action w takes, since it is about to acquire fresh first-hand information on the status of $\mathbf{pp}(u)$, is to suspend the dissemination of any QUENCH messages that it might be previously disseminating (lines 1–2), as these are effective immediately stale. Next, w updates its last seen timestamp *last*[u] to reflect the fact that it is about to get the latest updates concerning u.

Then, w downloads the set U containing all the identifiers of u's updates that are more recent than *version*[u] (line 4). If U is not empty (line 5), *version*[u] is set equal to the largest update version number contained in U, and all the updates that have not been received yet are downloaded from the cloud. The precise nature of the cloud operations depend on the API given by the provider. For example, in S3 [5]: *i)* updates could be stored as different *versions* of the same object; *ii)* the list of identifiers of versions that are more recent than *version*[u] could be obtained through the "GET bucket object versions" primitive; and *iii)* once w gets the list of version identifiers, it can download specific versions.

After download, w delivers the updates to the application layer (lines 9–10) and to the P2P dissemination layer (line 10), from where these are spread over the ego network G_u. As we can see from line 10, update messages contain five fields: a type descriptor (UPDATE), the identifier of the owner of the profile page to which the update is addressed (u), the timestamp of when the update was downloaded (*last*[u]), the identifier of the update (*id*), and the update itself (*upd*).

If instead no updates are found (line 11), node w disseminates a QUENCH message with the identifier of the profile page owner and an access timestamp (lines 11–12). Finally, w updates its target delay bound *target*[u] by randomizing it, as we have described in Section 6.3.1. Upon receiving a message m, a node w executes the code in Algorithm 7. If the message is an update (line 2) and the content is unknown to the receiver, the update gets delivered to the application layer (line 4).

If m is a QUENCH message, but either its timestamp is older than *last*[u], or the difference between the current local time and m's timestamp is larger than δ, this means that the message is too old to be helpful, and its dissemination is therefore interrupted (line 7).

Regardless of the type of message received, w updates *last*[u] and its target delay bound whenever the timestamp of the message is more recent than its own – which will

Algorithm 6: OnTimeout

Trigger: Target delay bound is crossed ($\mathsf{clock}() - \mathit{last}[u] > \mathit{target}[u]$).

```
1  M ← P2P.query(⟨QUENCH, u, ·⟩)
2  forall the m′ ∈ M do P2P.remove(m′)
3  last[u] ← clock()
4  U ← cloud.list(u, version[u])
5  if U ≠ ∅ then
6      version[u] ← max(U)
7      foreach id ∈ U − delivered do
8          upd ← cloud.download(u, id)
9          deliver(upd)
10         delivered ← delivered ∪ {id}  P2P.disseminate(⟨UPDATE, u, last[u], id, upd⟩)
11 else
12     P2P.disseminate(⟨QUENCH, u, last[u]⟩)
13 target[u] ← ψ + uniform(0, α)
```

cause access quenching at w – and stops disseminating any QUENCH messages for which the timestamp is smaller than that of m (lines 10–12), since the knowledge contained in m is more recent.

Finally a node w joining the network could face the situation in which it logs in already timed out w.r.t. $\mathit{target}[u]$. The next step w would take, in this case, would be to execute Algorithm 6. Yet, it might be that there is some neighbor of w in the P2P network that has recent updates on u and is ready to send them, which means that the access would be unnecessary. To avoid that this situation, we impose an ordering on the simulation in which nodes *first* process incoming neighbor messages, if any, and then honour any timeouts, if these still stand. In a practical implementation, a joining node could achieve the same effect by allowing some short grace period on login, thus giving neighbors enough time to send any new information.

6.4 Evaluation

In this section we evaluate our protocol towards two goals:

1. provide supporting evidence that the protocol performs significantly better than a pure P2P approach, and can be competitive with centralized approaches;
2. provide a rough estimate of what kind of monetary and network costs one should expect from running it.

Algorithm 7: OnReceive

Trigger: Message is received from the P2P layer.

Input: Message m.

```
1  switch m.type do
2      case UPDATE
3          if m.id ∉ delivered then
4              deliver(m.upd)
5      case QUENCH
6          if m.t ≤ last[u] or clock() − m.t > δ then
7              P2P.remove(m)
8  if m.t > last[u] then
9      last[u] ← m.t
10     M ← P2P.query(⟨QUENCH, u, ·⟩)
11     forall the m′ ∈ M : m′.t < m.t do P2P.remove(m′)
12
13     target[u] ← ψ + uniform(0, α)
```

6.4.1 Baselines

We compare our protocol against three baselines. The first one, PurePoll, is the naïve protocol we described in Section 6.3. The second one, PureP2P, consists of disseminating updates exclusively over the social overlay, thus providing a baseline reference as to how much improvement we can obtain by using the cloud as an auxiliary mechanism to P2P dissemination. Finally, the third baseline, Server, emulates a centralized approach akin to Facebook. This is achieved by adding a special server node which is connected to everybody else, manages all profile pages, and can relay updates from sources to destinations with zero delay. This represents the best performance reference for our system. Clearly, we would like to perform as close as possible to Server, while incurring only modest monetary costs.

6.4.2 Experimental Setting

Protocols are evaluated over a sample of 700 ego networks picked uniformly at random from the Orkut crawl of Mislove et al. [69]. The original graph contains 3 million vertices (i.e., 3 million ego networks), 223 million (undirected) edges, and has an average clustering coefficient of 0.171. Our sample includes around 5% of the vertices in the graph, and its average clustering coefficient is slightly smaller. A summary of statistics can be found in

Metric	**Value**
Number of Vertices	137892
Number of Edges	1460043
Average Clustering Coefficient	0.112
Average Egonet Size	197.5
Maximum Egonet Size	2181

Table 6.1: Statistics for the ego network sample used in our experiments.

Table 6.1. This is the same dataset we used in the small-scale evaluation of Section 4.3.

For each ego network G_u in our sample, we single out *one* source node $v \in f^*(u)$ uniformly at random. This leads to a set of 700 (*source*, *ego network*) pairs, over which we run simulations. Let (v, G_u) be one such pair. In a nutshell, what we do is to simulate a number of updates originating at v, and measure their delays towards the receivers in $f^*(u)/\{v\}$, as well as a number of other metrics, as discussed in Section 6.4.3. We then compute population estimators for these metrics such as averages and empirical cumulative distribution functions, which are the basis for discussing our results.

The fact that we rely on estimators is the main driving force behind our choice to sparsely cover many ego networks (i.e., to pick 700 ego networks, and only one source per ego network) instead of covering fewer ego networks but more densely (i.e., to pick one big ego network, and 700 different sources). Since unbiased estimation relies on independent sampling, we want to insert as much variability as possible in the structure of the ego networks we study, to ensure that the values of the metrics are not correlated.

We then repeat, for each (*source*, *ego network*) pair, the following experiment 100 times. First, we set all nodes in G_u to offline and, in the case of PurePoll and Hybrid simulations, we also set all the value of $last[u]$ (the last time instant at which a node heard from u) to zero. Since this initial state is not representative of the steady-state regime of our system, we run the simulation for a *burn-in period* γ over which we do not perform any measurements. During burn-in, we simulate the churn model and, in the case of PurePoll and Hybrid, we also simulate the polling of the cloud, and the propagation of quench messages – which, at this point, is all we are going to get, given that we have not yet generated any update.

The goal is to have a representative online/offline configuration for the network before we start injecting updates, but also a representative number of quench messages going around and a representative "phase shift" for the last seen timestamps $last[u]$ at the various nodes. The nodes are initially all synchronized at $last[u] = 0$ but, as the burn-in progresses, they access the cloud at different rates due to their different inter-

session lengths, causing their values of $last[u]$ to become different, particularly for nodes in different delay groups. We have empirically established, with heuristics similar to those described in [3], that setting γ to 48 hours is enough to produce unbiased results.

Once the burn-in period is over, the experiment progresses by waiting for the first login of the source v, at which point we cause v to post an update to $\mathbf{pp}(u)$. Note that the time of login is the earliest time instant at which a user can post an update. To avoid having to deal with the complexities of building a user model – which would not buy us much in any case for the kind of performance and cost measurements we want to do – we choose to be pessimistic and assume that nodes always post their updates at beginning of one of their sessions.

The posting of the update by the source marks the beginning of our *measurement session*, which proceeds until the update reaches all destinations in $f^*(u)/\{v\}$. At that point, the experiment ends.

To enable a more precise discussion of the metrics we compute, described next, we represent the set of $n = 100$ independent experiments we run for an ego network G_u with source node v as $S(G_u) = \{e_1, \cdots, e_n\}$. The measurement session of experiment e_i starts at time instant s_i and ends at time instant f_i, and has *duration* $\mathbf{d}(e_i) = f_i - s_i$.

6.4.3 Metrics

Delay. As in Chapter 4, the two main indicators of performance we are interested in are the average end-to-end delay, which we refer to as **aed**, and the average receiver delay, **ard**. Each experiment $e \in S(G_u)$ generates exactly one **ed** and one **rd** delay sample per source/destination pair (v, w) in G_u. If we denote the **ed** sample for pair (v, w) generated by experiment e as $\mathbf{ed}(v, w, e)$, we can compute **aed** as the sample average:

$$\mathbf{aed}(v, w) = \frac{1}{|S(G_u)|} \sum_{e \in S(G_u)} \mathbf{ed}(v, w, e)$$

and **ard** can be computed with a similar formula.

Monetary cost. We measure the yearly costs of running the system by counting, for each node $w \in f^*(u)/\{v\}$ and each experiment $e \in S(G_u)$, the total number of times $\mathbf{cs}(w, e)$ that w accesses the cloud over the measurement session of e. We define the *average access rate* of w over G_u as:

$$\mathbf{acs}(w, G_u) = \frac{\sum_{e \in S(G_u)} \mathbf{cs}(w, e)}{\sum_{e \in S(G_u)} \mathbf{d}(e)} \frac{\text{access}}{\text{hour}}$$

Since most of these accesses are due to QUENCH messages, and updates are themselves small, we simplify the computation of costs as mentioned in Section 6.1 by assuming that

the dominant cost component is given by the price of GET requests. Let ρ be the cost of a GET request. The yearly cost incurred on node w for keeping up-to-date with the updates of one profile page $\mathbf{pp}(u)$ can be therefore approximated by:

$$\mathbf{ycs}(w,u) \sim \rho \times \mathbf{acs}(w, G_u) \frac{\text{access}}{\text{hour}} \times \left(24 \frac{\text{hour}}{\text{day}} \times 365 \text{ day}\right)$$

Clearly, to get an unbiased estimate of the overall yearly spendings $\widehat{\mathbf{cs}}(w)$ of w, we would need to estimate and sum the yearly costs incurred on w for keeping up-to-date with each of her friends in $f(w)$, i.e., we would need to compute $\sum_{m \in f(w)} \mathbf{ycs}(w,m)$. This in practice means we would have to compute estimates for 137 892 ego networks, which would be intractable given the high costs of these simulations.

We therefore choose to use two distinct approximation models for costs. These are less accurate, but also much less expensive to compute. Both models can be expressed as the product of $\mathbf{ycs}(w,u)$ by an approximation constant $\tau(w)$:

$$\widehat{\mathbf{cs}}(w) \sim \mathbf{ycs}(w,u) \times \tau(w)$$

the difference between the models being how we compute $\tau(w)$. In the simplest model, which we call the *flat approximation*, $\tau(w)$ is simply the average size of the ego networks in our sample, i.e., we multiply $\mathbf{ycs}(w)$ by $\tau(w) = 197.5$, for all w. This model applies a fixed "penalty" to all nodes, regardless of the conditions of the surrounding ego networks, or their number. In our second model, the *degree approximation*, $\tau(w)$ is instead the degree of w in the social network. This approximation assumes that costs increase linearly w.r.t. node degree. In particular, it makes the assumption that all friends of w incur the same **ycs** costs on w, and that the total cost can be computed as their sum. Intuitively, these approximations yield estimates that are similar for nodes whose degree is close to the average, but differ significantly for nodes with very low and very high degrees.

Given our fixed computational budget, these cost models represent a tradeoff we had to impose between obtaining unbiased estimates for delay (which requires many uncorrelated ego networks to be covered), and accurate estimates for cost (which requires that we densely cover neighboring, and therefore correlated, ego networks). Since our focus is on delays, we choose to be precise with the former, while settling for less accurate estimates for the latter.

Network cost. To assess the usage of network resources, we measure the number of messages a node processes per time unit, on average, to keep up with updates from its friends. Similarly as we did with requests to the cloud, let $\mathbf{msg}(w,e)$ represent the number of messages processed (sent/received) by w during the measurement session of experiment e. The average hourly rate $\mathbf{amsg}(w, G_u)$ at which w processed messages,

therefore, is given by:

$$\mathbf{amsg}(w, G_u) = \frac{\sum_{e \in S(G_u)} \mathbf{msg}(e, w)}{\sum_{e \in S(G_u)} \mathbf{d}(e)} \frac{\text{message}}{\text{hour}} \tag{6.3}$$

Again, this gives us the costs incurred on w while keeping up-to-date with a single friend. We adopt a similar approximation as we did with $\widehat{\mathbf{cs}}$ when computing the total message processing rate, and multiply **amsg** by $\tau(w)$:

$$\widehat{\mathbf{msg}}(w) \sim \mathbf{amsg}(w, G_u) \times \tau(w) \tag{6.4}$$

We again use both the flat and the degree approximations when computing τ, observing the same caveats as before.

6.4.4 Results

We use the shorthand notation Hybrid/ψ/α to refer to the variant of Hybrid with parameters ψ and α, with unit given in minutes. We use a similar notation, PurePoll/δ, to refer to the PurePoll variant with target delay bound δ.

We simulate two versions of Hybrid, Hybrid/30/14, and Hybrid/15/14. These parameter settings, as we later show, provide good performance in terms of delay and cost, and we use them for comparison against the baselines. However, as explained in Section 6.3, the target delay bound of Hybrid is randomized, and varies in $[\psi, \psi + \alpha]$ with average $\psi + \alpha/2$. Since PurePoll is not randomized, we compare each setting of Hybrid to three settings of PurePoll: *i) "fast"*, PurePoll/ψ; *ii) "intermediate"*, PurePoll/$(\psi + \alpha/2)$; *iii) "slow"*, PurePoll/$(\psi + \alpha)$.

This yields six PurePoll variants: three ($\delta \in \{30, 37, 44\}$) to compare against Hybrid/30/14 and three ($\delta \in \{15, 22, 29\}$) to compare against Hybrid/15/14. Since PurePoll/30 and PurePoll/29 behave essentially the same, we do not show the former and use the latter in both cases.

Finally, to understand at which point PurePoll can overtake Hybrid, we add a seventh setting for PurePoll in which we set $\delta = 5$.

6.4.4-1 Delay Figure 6.5 shows the empirical cumulative distributions functions (CDFs) for end-to-end and receiver delays of Hybrid and all baselines. Since **rd** is always zero for Server, we omit it from the plot. Complementary statistics are provided in Table 6.2.

The data confirms our claim that PureP2P suffers from significant performance issues even under exponential session inter-session lengths, with the **ard** distribution having a long tail, reaching values as high as 2.9 hours, and remaining nevertheless above 1.4h at the 99^{th} percentile. Further, the data shows that our cloud-based alternatives – Hybrid

	ard		
	avg.	99^{th}	**max.**
PureP2P	5.76m	1.5h	2.9h
Hybrid/30/14	48s	9.8m	15.2m
Hybrid/15/14	35s	7.2m	13.5m
PurePoll/44	9.94m	22.9m	27.1m
PurePoll/37	8.9m	21.6m	26.4m
PurePoll/29	6.7m	14.5m	16.6m
PurePoll/22	5.2m	11.9m	14.3m
PurePoll/15	4m	10.7m	12.3m
PurePoll/5	1.2m	2.8m	4.5m
Server	0	0	0

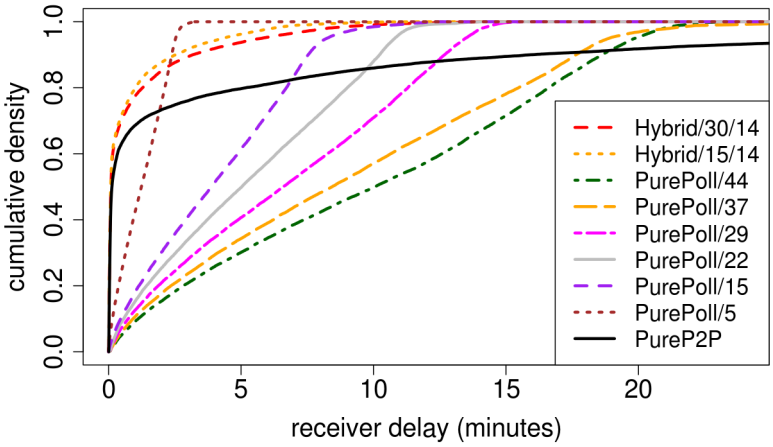
Table 6.2: Complementary statistics (m = minute, s = second, d = day).

and PurePoll– effectively *solve the problem of the long delay tail* by putting a bound on **rd**, which can be seen from the much smaller maximum and 99^{th} percentile and maximum values in comparison.

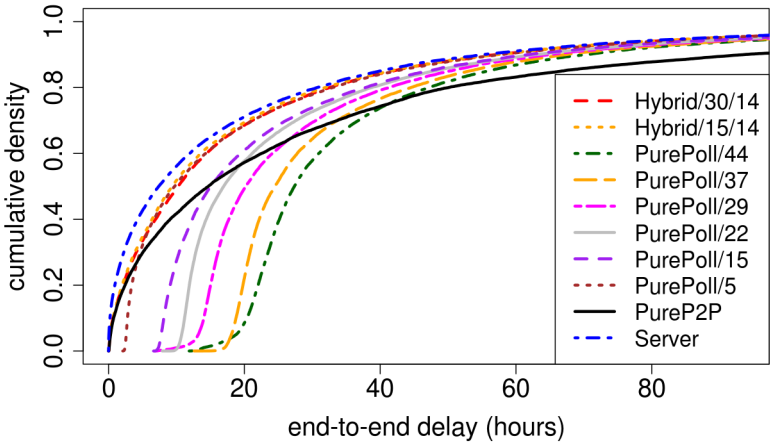
Finally, we can see that Hybrid consistently outperforms its associated PurePoll variants (including the fast one), while providing a better experience for a significant fraction of the users across all parameter settings, even as we compare Hybrid/30/14 to PurePoll/5. By combining both approaches, Hybrid effectively reconciles the best of both worlds: the fast performance of PureP2P for the regions of the network that exhibit low delay – which can be seen in Figure 6.5a as the nearly vertical shape of the CDF up until the 60^{th} percentile – with the ability of mitigating the long delay tails of PurePoll.

Table 6.2 shows that Hybrid has maximum and 99^{th} percentile **ard** values which are comparable to those of their *fast* PurePoll counterparts (i.e., Hybrid/ψ/α achieves performance similar to that of PurePoll/ψ), with Hybrid being nevertheless faster. Indeed, Hybrid/30/14 and Hybrid/15/14 perform around 837% and 685% percent faster, on average, than PurePoll/29 and PurePoll/15, respectively.

As for the end-to-end delay, Figure 6.5b shows that Hybrid promotes significant reductions in **ed** as well, particularly at the lower percentiles. Again, these reductions are for nodes which are connected by low-delay paths in the overlay. Since the polling period is never smaller than δ, this represents a barrier to PurePoll which does not exist for Hybrid. Further, we note that the Hybrid **ed** curves are much closer to Server than PureP2P.

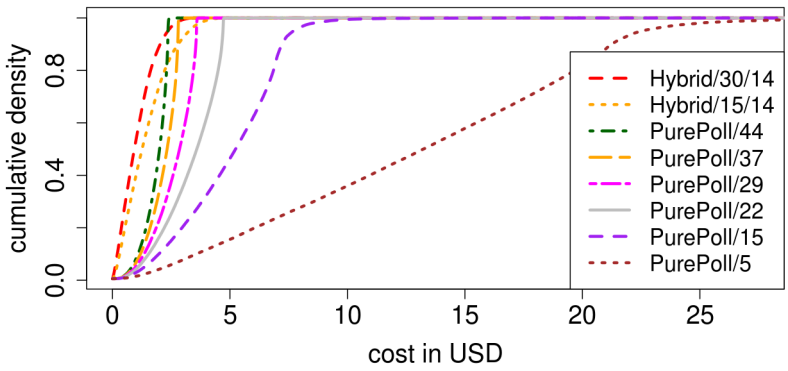


(a) **ard**

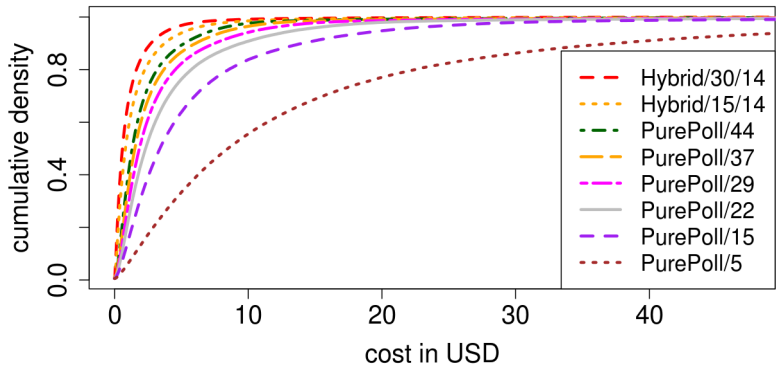


(b) **aed**

Figure 6.5: CDFs for **aed** and **ard** for all source/destination pairs.



(a) Flat approximation.



(b) Degree approximation.

Figure 6.6: Yearly monetary cost estimates for HYBRID.

The higher percentile range, instead, is dominated by the behavior of low availability nodes – i.e., nodes that stay offline for extended periods of time – and this causes the **aed** distributions to slowly converge. This can be seen from the curves getting closer together in Figure 6.5b as we move towards the higher percentiles. Since values are quite similar for all protocols, we omit them from Table 6.2. The maximum **aed** is, in any case, around 2 days, while the 99^{th} percentile is at about 3 hours, for all approaches – including SERVER.

6.4.4-2 Monetary cost Yearly cost figures are shown in Figures 6.6a and 6.6b. For these calculations, we use the currently published Amazon S3 pricing [5] according to the cost model in SectionSection 6.1. Additional statistics are provided in Table 6.3.

Again, the costs for HYBRID are generally lower than their associated PUREPOLL variants. Costs are extremely attractive under the flat model, with HYBRID/15/14 presenting a very good cost/latency tradeoff. The other side of the coin is given by the degree model, which reaches high maximum values. To understand what is going on, however, we need

	flat (USD)			degree (USD)		
	avg.	99^{th}	**max.**	**avg.**	99^{th}	**max.**
HYBRID/30/14	1	2.9	3.4	1.25	9	230
HYBRID/15/14	1.42	3.9	4.66	1.72	12.4	304
PUREPOLL/44	1.84	2.38	2.9	2.5	17.4	282
PUREPOLL/37	2.13	2.8	3.47	2.9	20.3	324
PUREPOLL/29	2.63	3.59	3.81	3.57	25	390
PUREPOLL/22	3.32	4.72	4.88	4.51	31.6	483
PUREPOLL/15	5	9.5	20	9.5	48	737
PUREPOLL/5	13	27	64	17	128	1909

Table 6.3: Complementary statistics for yearly costs, in US dollars.

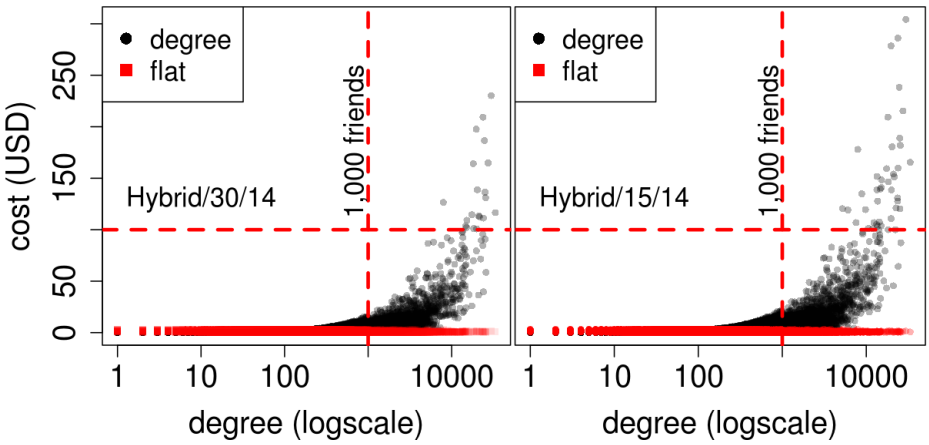


Figure 6.7: Degree vs. cost scatter plots for HYBRID, for both flat and degree approximations.

to take a closer look. Figure 6.7 presents scatter plots of estimated yearly costs versus node degrees for both HYBRID variants. We can see that high values all originate from a small set of extremely connected nodes, the most connected having 33 313 friends. For nodes with less than 1 000 friends, however, costs for HYBRID/30/7 and HYBRID/15/7 are no larger than 15 and 21 USD a year, respectively, which are still significantly below the $127 price tag of low-cost hosting solutions [75], or the minimum of $105 required to keep an Amazon EC2 micro instance (the absolute cheapest available) running for a year [4].

Even if some users do get to such large number of friends, we still do not expect to see such high costs in a real-world setting. Users with ego networks this big are likely to trim out uninteresting friends with which they will not want to keep in touch so often,

therefore bringing down costs considerably. And that is precisely the strong point of our solution: the user can decide whether and how much to pay, as well as which latency bounds to keep with which friends. The impact such choices might have on the latencies remains to be seen, though these should never, in any case, be higher than in PurePoll.

6.4.4-3 Network cost The last set of metrics we present regard the usage of P2P network resources. We analyse the average number of messages processed per second at each node, using the same flat and degree approximations as before to produce the overall estimates. We focus on quench messages, since our main interest is on the base cost of the protocol. Indeed, the cost incurred by updates depends on user posting frequency and habits, which are variables that are out of our control. We argue, however, that evaluating only quench overhead is a reasonable approach, since the bandwidth available for updates will ultimately be given by whatever is available, minus the overhead incurred by our protocol, which we measure here.

The CDF for message costs is presented in Figure 6.8, with additional statistics given in Table 6.4. The first thing to notice is that the two protocols present nearly-identical performance. This is a reflection of the fact that the quench mechanism causes the underlying gossip protocol to work continuously in both cases, i.e., maxcomp never goes quiescent, and keeps pumping a message a second for the entire duration of the experiment. The network overhead, therefore, is very similar for both variants, and should remain so for any variant with smaller values of ψ and α.

If we consider that the size of a quench message should be around 48 bytes (a 40-byte TCP/IP header plus 64 bits for an id and a timestamp), the overhead of running Hybrid is reasonable for a significant percentage of the nodes. Indeed, even under the pessimistic degree approximation, 90% of the nodes have to process less than 40 messages/second, which translates into running costs of around 2kB/sec, without any optimizations (e.g., aggregation of ids and timestamps under a single message). For the flat model, these costs are even lower, with 99% of the nodes having to process less than 70 messages a second, or around 3kB/sec.

Maximum values are still reasonable under the flat approximation, reaching at most 178 messages per second, or around 8.5kB/sec. Under the degree approximation, on the other hand, these clearly become unreasonably large. The high numbers are again mostly due to nodes with high connectivity. This phenomenon can be observed both in Table 6.4, where we take the degree of the node at the n^{th} percentile for both protocols, average them, and display them on the top row; and in the scatter plots of Figure 6.9, where we can see that very few nodes with degree of less than 1 000 would have to process more than 100 messages per second (the equivalent of 4.8kB/sec by our previous estimates). Indeed,

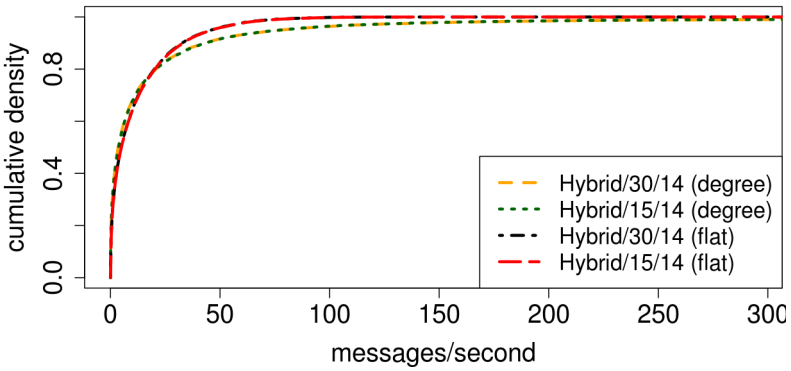


Figure 6.8: Message-per-second cost estimates for HYBRID.

	avg.	90^{th}	95^{th}	99^{th}	**max.**
node degree	197.5	409	658	2 897	33 313
HYBRID/30/14 (degree)	25.98	43.17	77.9	298.16	9 498
HYBRID/15/14 (degree)	26.02	43.17	77.6	294.73	9 472
HYBRID/30/14 (flat)	11.64	32.60	46	73.2	178
HYBRID/15/14 (flat)	11.8	33	56	74.8	168

Table 6.4: Complementary statistics for network costs, in QUENCH messages per second.

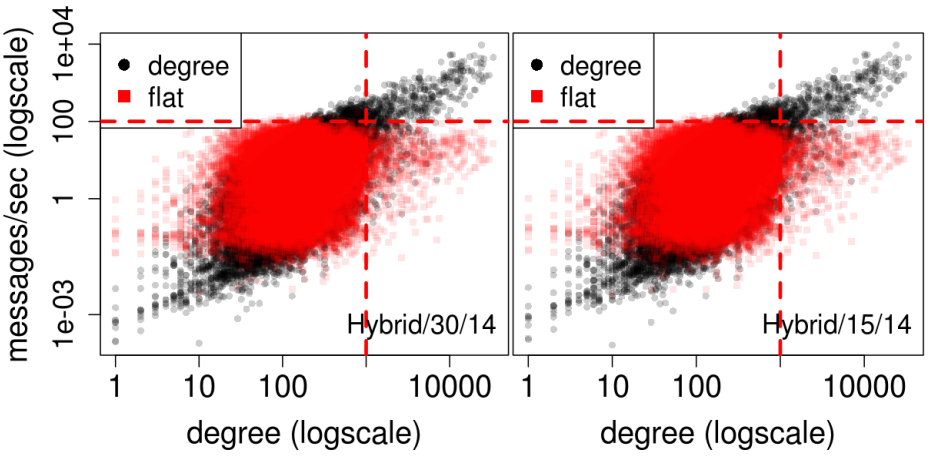


Figure 6.9: Degree vs. cost scatter plots for HYBRID.

under the flat approximation, only 0.02% of the nodes would go above 100 messages a second, while this number raises to 3% under the degree approximation.

We further note that the degree approximation is pessimistic under another important aspect, namely, it assumes a worst-case scenario in which a node w is concurrently participating in the dissemination of QUENCH messages for all of its friends, i.e., all the profile pages $\mathbf{pp}(u)$, $u \in V(G_w)$. Using the availability theorem of [128], however, we can predict that this is a rare situation in practice. For the availability assignment we have generated for these experiments (according to the model of Section 6.1), the theorem predicts that, on average, only around 39% of the nodes would be online at any given point in time, meaning that these values should be much lower.

Finally, network costs still have great margins of optimization, due to our choice of a pure push approach, which is known to tradeoff latency for overhead. An alternative would be to use this approach in a push-pull, anti-entropy protocol [28]. We believe this would significantly reduce network costs, while maintaining the good performance w.r.t. the other metrics.

6.5 Related Work

Cloud-assisted Decentralized OSNs. There are currently few proposals in the literature for hybridizing cloud and P2P architectures in the context of Decentralized OSNs. Confidant [60], which we already discussed in Section 2.1.2-3, is one of them. Confidant is rather different from our approach in that data is kept at the P2P layer and, therefore, is not guaranteed available. Monetary costs are potentially smaller than our approach since their simple "name servers" (the cloud components used for coordination) can be

run for free in platforms such as Google AppEngine. If such free service were to no longer be offered, however, that would change. In terms of privacy, Confidant offers a compromise similar to ours: although providers have no access to cyphertext, they are still able to guess the social network and analyse both data request and posting frequencies by analysing requests to individual name servers.

Vis-à-Vis [103] essentially proposes the use of cloud aliases that are full replicas of the corresponding peers – which they call "primaries". Aliases should come into play only when a primary is not available, effectively filling their availability gap. Aliases and primaries are kept in-sync by mechanisms similar to the ones we propose here: every update made to a primary should be replicated into a shared storage – e.g. S3 – which is visible by the alias. One unresolved issue is how to perform the "switchover" from primary to alias in case the primary crashes. Authors propose that either an inexpensive "ping" service could be made available by some utility provider, or that other nodes could activate aliases themselves upon failing to contact the corresponding primary. As we mentioned in 6.3, the main drawbacks to this approach are costs, and the amount of exposition to the cloud provider.

Kryczka et al. [54] propose a rather different idea in which peers are leveraged purely as a means to reduce the costs of a centralized OSN provider by offloading its servers. Sensitive data is still kept at the centralized provider, enabling its business model. Apart from the completely different goal, the presence of a centralized component that acts as a global coordinator for replications means that problems are much simpler.

P2P Decentralized OSNs. As with the work we presented in Chapter 3, the ideas in this chapter could offer added performance and reliability guarantees for asynchronous multicast messaging in pre-existing systems (e.g. PeerSoN [18] or Safebook [25]), while arguably providing reasonable privacy guarantees by the use of a known, authorized third-party to store updates, as opposed to randomly assigned DHT nodes.

Social overlays. As we have discussed in Section 2.2.1, there has been a growing interest in the use of social networks as communication networks, with social overlays having made their way into a number of system proposals over the past few years. Given that these networks are inherently susceptible to partitioning under churn, the work we present here represents an important step in rendering such systems practical.

Cloud-assisted P2P. The idea of using a high-availability cloud infrastructure to boost the performance of P2P systems is not new. Among the recent literature, the work closest to ours is by Gracia-Tinedo et al. [116], who propose the use of cloud helpers to increase data availability in a friend-to-friend (F2F) backup system. This is similar to our work in that F2F systems are also based on social overlays, and that data is also confined to ego networks. The problems are, however, of very a different nature. F2F backup is

concerned with providing high availability to the user backing up her data, which however must be made available only to herself. There is no notion of updates, or the need to disseminate them to friends beyond that of ensuring enough replication. Finally, backup data is in general different from our updates in that objects are large, shifting concerns to throughput instead of latency.

6.6 Discussion and Outlook

In this chapter, we have presented the final piece of our work: a way to mitigate the delay tail of social overlays under churn, by means of a hybrid protocol that leverages a highly available cloud infrastructure to adaptively support the overlay when and where needed, without sacrificing the fundamental property of allowing communication only among friends.

Our results show that dramatic improvements in dissemination delay can be achieved, while keeping costs low for users that have less than one thousand friends. A number of issues, however, still remain. Apart from network cost figures, which need to be improved, we need more reliable models for estimating network and monetary costs. Further, while we argue that the approach described in this chapter provides better privacy guarantees than relying on a centralized provider, it still reveals some significant amount of information to providers, which could be misused.

The protocol we present here is rather simple in that it sets a one-size-fits-all polling period to all nodes in the network. We are currently working on characterizing optimal polling intervals as a function of the surrounding P2P network, so as to understand how much the current approach can be improved.

Finally, the fact that we opted for running all dissemination with a push protocol led to significant network costs. Our immediate plans for future work will focus on improving that scenario, by investigating a combination of the current push approach with anti-entropy [28].

Chapter 7

Conclusions

> Follow the evidence to where it leads, even if the conclusion is uncomfortable.
>
> – *Steven James*, The Knight

Across the last decade, OSNs have entered people's lives, establishing themselves as ubiquitous communication tools. Yet, the current centralized solutions have been plagued by privacy and data ownership issues from the very start, an ailment that decentralized OSNs have the potential to cure. In this thesis, we have advocated in favour of an SO-based P2P approach as the way to achieve decentralization.

Social overlays appear as an interesting alternative, as not only can they help mitigate the trust-related issues of P2P, but they also exhibit a natural match towards OSNs. We propose to reap these potential benefits by means of a simple approach that provides one key functionality of modern OSNs: 1-hop, profile-based communication. Although the potential for synergy between SOs and P2P OSNs has been recognized in different ways in the past (e.g. [38]), no concrete proposals for an OSN based on social overlays had been made to this date.

In this thesis, we have pushed the idea forward and, along the way, have discovered and tackled a number of important problems. Starting from a dissemination protocol that had to deal with uneven clustering, we uncovered communication delays as a difficult-to-assess and fundamental problem of social overlays, with implications far beyond our proposal for a decentralized OSN, and into any system that is to be based on them. To render the assessment of the phenomenon tractable on large graphs, we worked on analytical models that can estimate communication delays at a more practical cost.

Finally, based on a better understanding of social overlays under churn, we proposed a new cloud-assisted dissemination protocol which built on the ideas of the remainder

of our work to promote efficient dissemination in a dynamic setting, while keeping costs relatively low. In trailing this path, we have learnt two important, and related, lessons.

1. *Structure matters.* Social overlays are different from other overlays in that their structure is defined a priori, by complex processes that are not necessarily friendly towards one's system-level goals (e.g. some nodes will simply have large degrees, and some network regions will simply be very sparse). Although certain structural properties of social networks have been understood for years (e.g. [52, 123]), the existing literature fails to provide good guidance when one is faced with a real graph, and has to implement a system on top of it. A good example of that is given by the counterintuitive anticentrality heuristic, which improves gossip protocol performance by disfavouring hubs. In that sense, we believe the work in this thesis can serve as a good starting point to researchers and practitioners who choose to adopt SOs in their work, by providing valuable insight on the general caveats of their application, in particular to decentralized online social networks.

2. *Evaluation of SO-based systems is hard.* Evaluating any system that runs over a social overlay when node-specific properties such as availability are involved requires one to somehow map those into the underlying graph. Since structure matters, the absence of data to guide such mapping means researchers have to come up with a number of hypotheses on how such properties might correlate or not to specific structural properties (e.g. node degree), and evaluate the network under a large number of them. This is compounded by the fact that claiming an understanding of local and global properties of a phenomenon over social overlays requires, inevitably, the examination of large datasets (again, a result of large variations in structure), and evaluation costs multiply. The development of efficient evaluation strategies is, therefore, of fundamental importance. From that perspective, our work on the analytical models of Chapters 4 and 5 brings important contributions by offering a path to new cost-effective tools in the assessment of one particularly critical aspect of SOs: communication delays.

As for future work, we have a broad horizon. With respect to update dissemination, most of our evaluation has been conducted in single-post scenarios, where we analyse the dissemination of an update at a time. One of the main reasons for that is because any evaluation involving multiple posts would require credible data (or a model) on posting frequencies and their correlations to graph-structural properties, which are largely missing in the literature. Yet, this leaves us with a hole in our knowledge of what could be the actual bandwidth requirements of a working system. A fruitful direction, in this sense, would be that of studying the "capacity" of temporal paths, i.e., studying how long they

remain stable in time and, therefore, how much data can we potentially pump through them. Such an understanding would provide us with a better idea of how much data can be communicated across two nodes in an SO over time, in a way that does not require posting frequencies to be modelled, much like our analysis of communication delays.

As for communication delays, although we believe to have made headway by identifying them as a problem and providing a useful model, a study of larger-scale datasets involving different availability assumptions is still missing. In this front, we have both short-term goals involving the application of the techniques presented in 5.4, and longer-term goals involving further development of our Markov model, as well as the exploration of semi-Markov models for the cases of heavy-tailed distributions.

While the cloud-based solution we have presented in Chapter 6 can effectively mitigate the delay tails, it is unoptimized in two important ways: the underlying pure-push protocol, which may lead to unnecessarily high network costs, and the one-size-fits-all value for the polling interval, which does not adjust to the surrounding delay structures and may lead to more accesses than required. We are currently working on both an evaluation of the solution which combines anti-entropy to increase network efficiency, and on a characterization of the optimal case for the protocol to understand what are the achievable latency/cost tradeoffs, and how these change as we set the polling intervals on individual nodes to different values.

Finally, although proposals for P2P OSNs have multiplied over the last years, most of these were relegated to academic papers or prototypes that barely made it into the wild. The more pragmatic open source community seems to favour decentralized server approaches instead, in which issues regarding availability and trust are much less pronounced. Yet, these solutions still require users to pay substantial sums of money for hosting, and we believe that, together with technology maturity, this constitutes one of the leading reasons why these solutions have not become more popular. Balance, however, could well be in the middle: the better cost/performance tradeoffs offered by hybrid solutions have, we believe, the potential to change the game for decentralized OSNs.

It is also worth noting that the emergence of low-cost, credit-card-sized PCs such as the Raspberry Pi [113] might as well represent a game-changing development to decentralized servers, since running a server at home might become a lot simpler and cheaper, e.g., picture an organization or individual releasing a "decentralized OSN image" that can just be flashed on a Raspberry Pi and used as a domestic appliance. At that point, the only barrier to wider scale and cost-effective adoption could be ISPs, which still routinely place restrictions on regular "home" contracts, such as blocking standard ports (e.g. HTTP) or imposing restrictions on the use of server software.

Finally, although we intend to eventually prototype our system, the answer as to

whether social overlays can indeed lead to better P2P by fostering cooperation among users will only be answered once these systems get deployed at large. Until that time comes, however, we remain optimistic, as friendship remains one of the tenets of altruistic behavior in human societies.

Bibliography

[1] L. M. Aiello, M. Milanesio, G. Ruffo, and R. Schifanella. Tempering Kademlia with a Robust Identity Based System. In *Proc. Intl. Conf. P2P Computing (P2P'08)*, pages 30–39, 2008.

[2] L. M. Aiello and G. Ruffo. Secure and Flexible Framework for Decentralized Social Network Services. In *Pervasive Computing and Communication Workshops*, pages 594–599, 2010.

[3] C. Alexopoulos and A. F. Seila. Implementing the Batch Means Method in Simulation Experiments. In *Proc. Winter Simulation Conference*, pages 214–221, 1996.

[4] Amazon.com, Inc. Amazon Elastic Compute Cloud (EC2). `https://aws.amazon.com/ec2/`. Visited February 2013.

[5] Amazon.com, Inc. Amazon Simple Storage Service (S3). `http://aws.amazon.com/s3/`. Visited March 2013.

[6] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin. Persona: an Online Social Network with User-Defined Privacy. In *Proc. ACM SIGCOMM*, pages 135–146, 2009.

[7] A. L. Barabási and R. Albert. Emergence of Scaling in Random Networks. *Science*, 286(5439):509–512, 1999.

[8] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida. Characterizing User Behavior in Online Social Networks. In *Proc. SIGCOMM Internet Measurement Conf. (IMC'09)*, pages 49–62, 2009.

[9] K. A. Berman. Vulnerability of scheduled networks and a generalization of Menger's Theorem. *Networks*, 48(1):125–134, 1996.

[10] C. Blake and R. Rodrigues. High Availability, Scalable Storage, Dynamic Peer Networks: Pick Two. In *Proc. Workthop on Hot Topics in Operating Systems (HotOS'03)*, pages 1–6, 2003.

[11] A. Boutet, D. Frey, R. Guerraoui, and A.-M. Kermarrec. WhatsUp: news from, for, through everyone. In *Proc. Conf. P2P Computing (P2P'10)*, pages 1–2, 2010.

[12] D. Boyd. Why Youth (Heart) Social Network Sites: The Role of Networked Publics in Teenage Social Life. In D. Buckingham, editor, *MacArthur Foundation Series on Digital Learning – Youth, Identity and Digital Media Volume*. MIT Press, 2007.

[13] D. Boyd. Facebook is a Utility, Utilities get Regulated. `http://www.zephoria.org/thoughts/archives/2010/05/15/facebook-is-a-utility-utilities-get-regulated.html`, 2010. Visited February 2013.

[14] D. Boyd and N. Ellison. Social Network Sites: Definition, History, and Scholarship. *Journal of Computer-Mediated Communication*, 13(1), 2007.

[15] D. Boyd and E. Hargittai. Facebook privacy settings: Who cares? *First Monday*, 15(8), 2010.

[16] D. Bricklin. Friend-to-Friend Networks. `http://www.bricklin.com/f2f.htm`, 2000. Visited Februrary 2013.

[17] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1:636—646, 2002.

[18] S. Buchegger, D. Schiberg, L. H. Vu, and A. Datta. PeerSoN: P2P social networking: early experiences and insights. In *Proc. ACM EuroSys Workshop on Social Network Systems (SNS'09)*, pages 46–52, 2009.

[19] B. Carpenter and S. Brim. Middleboxes: Taxonomy and issues. RFC 3234 (Informational), 2002.

[20] A. Chaintreau, A. Mtibaa, L. Massoulié, and C. Diot. The Diameter of Opportunistic Mobile Networks. In *Proc. ACM Intl. Conf. on emerging Networking EXperiments and Technologies (CoNEXT'07)*, 2007.

[21] D. Cho. E-mail Study Corroborates Six Degrees of Separation. *Scientific American*, August 2003.

[22] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg. SpiderCast: a scalable interest-aware overlay for topic-based pub/sub communication. In *Proc. Intl. Conf. on Distributed Event-Based Systems (DEBS'07)*, pages 12–25, 2007.

[23] A. E. F. Clementi, C. Macci, A. Monti, F. Pasquale, and R. Silvestri. Flooding Time in edge-Markovian Dynamic Graphs. In *Proc. ACM Symposium on Principles of Distributed Computing (PODC'08)*, pages 213–222, 2008.

[24] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.

[25] L. A. Cutillo, R. Molva, and M. nen. Safebook: A privacy-preserving online social network leveraging on real-life trust. *IEEE Communications Magazine*, 47(12):94–101, 2009.

[26] A. Datta and R. Sharma. GoDisco: Selective Gossip Based Dissemination of Information in Social Community Based Overlays. In *Proc. Intl. Conf. Distributed Computing and Networking (ICDCN'11)*, pages 227–238, 2011.

[27] David Easley and John Kleinberg. *Networks, Crowds, and Markets: Reasoning about a Highly Connected World*. Cambridge University Press, 1st edition, July 2010.

[28] A. Demers, D. Greene, C. Houser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proc. ACM Symposium on Principles of Distributed Computing (PODC'87)*, pages 1–12, 1987.

[29] Diaspora, Inc. Diaspora website. `https://joindiaspora.com/`, 2012. Visited February 2013.

[30] J. Douceur. The Sybil Attack. In *Proc. Intl. Workshop on Peer-to-Peer Systems (IPTPS'01)*, pages 251–260, 2001.

[31] D. A. Dunn, W. D. Grover, and M. H. MacGregor. Comparison of k-shortest paths and maximum flow routing for network facility restoration. *IEEE Journal on Selected Areas in Communication (JSAC)*, 12(1):88–99, 1994.

[32] Dyn Inc. Dynamic DNS. `http://dyn.com/dns/dyndns-pro/`. Visited March 2013.

[33] G. S. Fishman and I. J. B. F. Adan. How heavy-tailed distributions affect simulation-generated time averages. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 2(16):152–173, 2006.

[34] B. Fitzpatrick, B. Slatkin, and M. Atkins. PubSubHubbub Core 0.3 – Working Draft. `http://pubsubhubbub.googlecode.com/svn/trunk/pubsubhubbub-core-0.3.html`, 2010. Visited February 2013.

[35] Free Software Foundation. GNU Social website. `http://www.gnu.org/software/social/`. Visited February 2013.

[36] Friendica website. `http://friendica.com`, 2013. Visited February 2013.

[37] Friendica's User Directory. `http://dir.friendica.com/`. Visited February 2013.

[38] W. Galuba. Friend-to-friend computing: Building the social web at the internet edges. Technical Report LSIR-REPORT-2009-003, EPFL, 2009.

[39] B. Gerschback, G. Kreitz, and S. Buchegger. The Devil is in the Metadata – New Privacy Challenges in Decentralised Online Social Networks. In *Proc. Intl. Workshop on SEcurity and SOCial Networking (SESOC'12)*, pages 333–339, 2012.

[40] S. Girdzijauskas, G. Chockler, Y. Vigfusson, Y. Tock, and R. Melamed. Magnet: practical subscription clustering for Internet-scale publish/subscribe. In *Proc. Conf. Distributed Event-Based Systems (DEBS'10)*, pages 172–183, 2010.

[41] M. Gojka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in Facebook: A Case Study of Unbiased Sampling of OSNs. In *Proc. IEEE INFOCOM*, pages 1–9, 2010.

[42] Google Inc. Google App Engine. `https://developers.google.com/appengine/`. Visited February 2013.

[43] K. Graffi, C. Gross, D. Stingl, D. Hartung, A. Kovacevic, and R. Steinmetz. LifeSocial.KOM: A secure and P2P-based solution for online social networks. In *Proc. IEEE Consumer Communications and Networking Conf. (CCNC'11)*, pages 554–558, 2011.

[44] S. M. Hedetniemi, S. T. Hedetniemi, and A. L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, 1988.

[45] S. Jahid, S. Nilizadeh, P. Mittal, N. Borisov, and A. Kapadia. DECENT: A Decentralized Architecture for Enforcing Privacy in Online Social Networks. In *IEEE Intl. Workshop on SEcurity and SOCial Networking (SESOC'13)*, pages 326–332, 2012.

[46] J. Jiang, C. Wilson, X. Wang, P. Huang, W. Sha, Y. Dai, and B. Y. Zhao. Understanding Latent Interactions in Online Social Networks. In *Proc. SIGCOMM Internet Measurement Conf. (IMC'10)*, pages 369–382, 2010.

[47] C. Jones. Twitter says 250 000 accounts have been hacked in security breach. `http://www.guardian.co.uk/technology/2013/feb/02/twitter-hacked-accounts-reset-security`. Visited April 2013.

[48] P. S. Juste, D. Wolinsky, P. O. Boykin, M. J. Covington, and R. J. Figueiredo. SocialVPN: Enabling wide-area collaboration with integrated social and overlay networks. *Journal of Computer Networks*, 54(12):1926–1938, 2010.

[49] J. A. Kelner and P. Maymounkov. Electric Routing and Concurrent Flow Cutting. *Theoretical Computer Science*, 412(32):4123–4135, 2011.

[50] D. Kempe, J. Kleinberg, and A. Kumar. Connectivity and Inference Problems for Temporal Networks. *Journal of Computer and System Sciences*, 64(4):820–842, 2002.

[51] A.-M. Kermarrec, L. Massouli, and A. J. Ganesh. Probabilistic Reliable Dissemination in Large-Scale Systems. *IEEE Transactions Parallel Distributed Computing*, 14(3):248–258, 2003.

[52] J. Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proc. ACM Symposium on Principles of Distributed Computing (PODC'00)*, pages 163 – 170, 2000.

[53] J. Kriege and P. Buchholz. Equivalence Transformations for Acyclic Phase Type Distributions. Technical Report 827, Technische Universität Dortmund Fakultät für Informatik, 2009.

[54] M. Kryczka, R. Cuevas, C. Guerrero, E. Yoneki, and A. Azcorra. A first step towards user assisted online social networks. In *Proc. Workshop on Social Network Systems (SNS'10)*, 2010.

[55] A. Lancichinetti and S. Fortunato. Community detection algorithms: A comparative analysis. *Physical Review E*, 80(5), 2009.

[56] S. H. Lee, P. Kim, and H. Jeong. Statistical Properties of Sampled Networks. *Physical Review E*, 73(1):1–7, 2006.

[57] S. Legtchenko, S. Monnet, P. Sens, and G. Muller. RelaxDHT: a churn-resilient replication strategy for peer-to-peer. *ACM Transactions on Autonomous and Adaptive Systems*, 7(2), 2012.

[58] J. Li and F. Dabek. F2F: Reliable Storage in Open Networks. In *Proc. Intl. Workshop on Peer-to-Peer Systems (IPTPS'06)*, 2006.

[59] M.-J. Lin and K. Marzullo. Directional gossip: Gossip in a wide area network. In *Proc. European Dependable Computing Conference (EDCC-3)*, pages 364–379, 1999.

[60] D. Liu, A. Shakimov, R. Cáceres, A. Varshavsky, and L. P. Cox. Confidant: Protecting OSN Data without Locking it Up. In *Proc. Intl. Conf. on Middleware (Middleware'11)*, pages 61–80, 2011.

[61] A. Loupasakis, N. Ntarmos, and P. Triantafillou. eXO: Decentralized Autonomous Scalable Social Networking. In *Proc. Conf. on Innovative Data Systems Research (CIDR'11)*, pages 85–95, 2011.

[62] M. Macgirvin. DFRN Protocol Specification, Version 2.2. `https://macgirvin.com/spec/dfrn2.pdf`, 2010. Visited February 2013.

[63] P. Maymounkonv. Tonika: social routing with organic security. `http://pdos.csail.mit.edu/~petar/5ttt.org/`, 2013. Visited February 2013.

[64] M. McPherson, L. Smith-Lovin, and J. M. Cook. Birds of a Feather: Homophily in Social Networks. *Annual Review of Sociology*, 27(1):415–444, 2001.

[65] G. Mega, A. Montresor, and G. P. Picco. Can Gossip Lead to Privacy? In *Proc. Workshop on Social Networks and Distributed Systems (SNDS'10)*, 2010.

[66] G. Mega, A. Montresor, and G. P. Picco. Efficient Dissemination in Decentralized Social Networks. In *Proc. Intl. Conf. P2P Computing (P2P'11)*, pages 338–347, 2011.

[67] G. Mega, A. Montresor, and G. P. Picco. On Churn and Communication Delays in Social Overlays. In *Proc. Intl. Conf. P2P Computing (P2P'12)*, pages 214–224, 2012.

[68] R. Miller. Facebook: $50 Million A Year on Data Centers. `http://www.datacenterknowledge.com/archives/2010/09/16/facebook-50-million-a-year-on-data-centers/`, 2010. Visited February 2013.

[69] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and Analysis of Online Social Networks. In *Proc. ACM SIGCOMM Internet Measurement Conf. (IMC'07)*, pages 29–42, 2007.

[70] A. Montresor and M. Jelasity. Peersim: A scalable p2p simulator. In *Proc. Conf. P2P Computing (P2P'09)*, pages 99–100, 2009.

[71] S. Nagaraja. Anonymity in the wild: Mixes of unstructured networks. In *Proc. Workshop on Privacy Enhancing Technologies (PET'07)*, pages 254–272, 2007.

[72] A. Narayanan and V. Shmatikov. De-anonymizing Social Networks. In *Proc. IEEE Symposium on Security and Privacy*, pages 173–187, 2009.

[73] M. F. Neuts and M. E. Pagano. Generating random variates from a distribution of phase type. In *Proc. Winter Simulation Conf.*, volume 2, pages 381–387, 1981.

[74] M. M. Neuts. *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach*. Dover Publications, January 1995.

[75] New Dream Network, LLC. Dreamhost website. `http://dreamhost.com/`, 2013. Visted February 2013.

[76] S. Nilizadeh, S. Jahid, P. Mittal, N. Borisov, and A. Kapadia. Cachet: A Decentralized Architecture for Privacy Preserving Social Networking with Caching. In *Proc. ACM Intl. Conf. on emerging Networking EXperiments and Technologies (CoNEXT'12)*, pages 337–348, 2012.

[77] E. Nygren, R. K. Sitaraman, and J. Sun. The Akamai Network: A Platform for High-Performance Internet Applications. *ACM SIGOPS Operating Systems Review*, 44(3), 2010.

[78] A. Olteanu and G. Pierre. Towards Robust and Scalable Peer-to-Peer Social Networks. In *Proc. Workshop on Social Network Systems (SNS'12)*, 2012.

[79] OpenID Foundation. OpenID Authentication 2.0 – Final. `http://openid.net/specs/openid-authentication-2_0.html`, 2007. Visited February 2013.

[80] C. H. Papadimitrou and D. Ratajczak. On a Conjecture Related to Geometric Routing. *Theoretical Computer Science*, 344(1):3–14, 2005.

[81] T. Paul, B. Greschbach, S. Buchegger, and T. Strufe. Exploring Decentralization Dimensions of Social Networking Services: Adversaries and Availability. In *Proc. ACM Intl. Workshop on Hot Topics in Interdisciplinary Social Networks Research*, pages 49–56, 2012.

[82] J. F. Pérez and G. Riaño. jPhase: an Object-Oriented Tool for Modeling Phase-Type Distributions. In *Proc. Workshop on Tools for solving Structured Markov Chains (SMCtools'06)*, 2006.

[83] A. Pfitzmann and M. Hansen. Anonymity, Unlikability, Unobservability, Pseudonymity, and Identity Management – A Consolidated Proposal for Terminology (v0.23). `http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.23.pdf`, 2005.

[84] B. Pittel. On Spreading a Rumor. *SIAM Journal on Applied Mathematics*, 47(1):213–223, 1987.

[85] B. C. Popescu, B. Crispo, and A. S. Tanenbaum. Safe and Private Data Sharing with Turtle: Friends Team-Up and Beat the System. In *Proc. Cambridge Intl. Workshop on Security Protocols*, pages 213–220, 2006.

[86] M. Poppendieck and T. Poppendieck. *Lean Software Development: An Agile Toolkit.* Addison-Wesley, 2003.

[87] J. M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez. The Little Engine(s) That Could: Scaling Online Social Networks. In *Proc. ACM SIGCOMM*, pages 375–386, 2010.

[88] F. Rahimian, S. Girdzijauskas, A. Hossein Payberah, and S. Haridi. Vitis: A Gossip-based Hybrid Overlay for internet-scale Publish/Subscribe Enabling Rendezvous Routing in Unstructured Overlay Networks. In *Proc. Intl. Parallel & Distributed Processing Symposium (IPDPS'11)*, pages 746–757, 2011.

[89] A. Rapoport. Spread of information through a population with sociostructural bias: I. assumption of transitivity. *Bulletin of Mathematical Biophysics 15*, pages 523–533, 1953.

[90] S. I. Resnick. *Adventures in Stochastic Processes.* Birkäuser, 1st edition, September 1992.

[91] J. M. Rogers. *Private and Censorship-Resistant Communication over Public Networks.* PhD thesis, University College London, 2011.

[92] S. M. Ross. *Simulation.* Academic Press, 3rd edition, January 2002.

[93] S. M. Ross. *A First Course in Probability.* Prentice Hall, 7th edition, May 2005.

[94] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Proc. Intl. Conf. on Middleware (Middleware'01)*, pages 329–350, 2001.

[95] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proc. ACM Symposium on Operating Systems Principles (SOSP'01)*, pages 188–201, 2001.

[96] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 6120 (Proposed Standard), March 2011.

[97] A. Sala, H. Zheng, B. Y. Zhao, S. Gaito, and G. P. Rossi. Brief Announcement: Revisiting The Power-law Degree Distribution for Social Graph Analysis. In *Proc.*

ACM Symposium on Principles of Distributed Computing (PODC'10), pages 400–401, 2010.

[98] O. Sandberg. Distributed Routing in Small-World Networks. In *Proc. SIAM Workshop on ALgorithm ENgineering and EXperiments (ALENEX'06)*, pages 144–155, 2006.

[99] S. Saroiu, K. P. Gummadi, and S. D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proc. Multimedia Computing and Networking (MMCN)*, 2002.

[100] S. Scellato, I. Leontiadis, C. Mascolo, P. Basu, and M. Zafer. Evaluating Temporal Robustness of Mobile Networks. *IEEE Transactions on Mobile Computing*, 12(1):105–117, 2013.

[101] F. Schneider, A. Feldmann, B. Krishnamurthy, and W. Willinger. Understanding Online Social Network Usage from a Network Perspective. In *Proc. SIGCOMM Internet Measurement Conf. (IMC'09)*, pages 35–48, 2009.

[102] S.-W. Seong, J. Seo, M. Nasielski, D. Sengupta, S. Hangal, S. K. Teh, R. Chu, B. Dodson, and M. S. Lam. PrPl: A Decentralized Social Networking Infrastructure. In *Proc. ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, 2010.

[103] A. Shakimov, L. P. Cox, A. Varshavsky, and R. Cáceres. Privacy, Cost and Availability Tradeoffs in Decentralized OSNs. In *Proc. Workshop on Online Social Networks (WOSN'09)*, pages 13–18, 2009.

[104] A. Shakimov, H. Lim, R. Cáceres, L. P. Cox, K. Li, D. Liu, and A. Varshavsky. Vis-á-Vis: Privacy-preserving online social networking via Virtual Individual Servers. In *Proc. Intl. Conf. Communication Systems and Networks (COMSNETS'11)*, pages 1–10, 2011.

[105] R. Sharma and A. Datta. SuperNova: Super-peers Based Architecture for Decentralized Online Social Networks. In *Proc. Intl. Conf. Communication Systems and Networks (COMSNETS'12)*, pages 1–10, 2012.

[106] R. Sharma, A. Datta, M. Dell'Amico, and P. Michiardi. An empirical study of availability in friend-to-friend storage systems. In *Proc. Intl. Conf. P2P Computing (P2P'11)*, pages 348–351, 2011.

[107] A. Singh, G. Urdaneta, M. van Steen, and R. Vitenberg. Robust Overlays for Privacy-preserving Data Dissemination Over a Social Graph. In *Proc. Intl. Conf. on Distributed Computing Systems (ICDCS'12)*, pages 234–244, 2012.

[108] M. Steiner, T. En-Jajjary, and E. W. Biersack. A Global View of KAD. In *Proc. SIGCOMM Internet Measurement Conf. (IMC'07)*, pages 117–122, 2007.

[109] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Transactions on Networking (TON)*, 11:17–32, 2003.

[110] A. S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2nd edition, October 2006.

[111] J. Tang, M. Musolesi, C. Mascolo, and V. Latora. Characterizing temporal distance and reachability in mobile and online social networks. *SIGCOMM Computer Communication Review*, 40, 2010.

[112] The Freenet Project website. `https://freenetproject.org/`. Visited March 2013.

[113] The Raspberry Pi Foundation. Raspberry Pi: An ARM GNU/Linux box for $25. Take a byte! `http://www.raspberrypi.org/`. Visited April 2013.

[114] The Telegraph. Facebook hits 500m: social media by numbers. `http://www.telegraph.co.uk/technology/facebook/7903071/Facebook-hits-500m-social-media-by-numbers.html`. Visited March 2013.

[115] H. C. Tijms. *A First Course in Stochastic Models*. Wiley, 2nd edition, April 2003.

[116] R. G. Tinedo and M. S.-A. P. G. López. Analysis of data availability in F2F storage systems: When correlations matter. In *Proc. Intl. Conf. P2P Computing (P2P'12)*, pages 225–236, 2012.

[117] L. Toka, M. Dell'Amico, and P. Michiardi. Data transfer scheduling for P2P storage. In *Proc. Conf. P2P Computing (P2P'11)*, pages 132–141, 2011.

[118] D. N. Tran, F. Chiang, and J. Li. Friendstore: Cooperative Online Backup Using Trusted Nodes. In *Proc. Workshop on Social Network Systems (SocialNets'08)*, pages 37–42, 2008.

[119] Twitaholic website. `http://www.twitaholic.com`. Visited March 2013.

[120] G. Urdaneta, G. Pierre, and M. van Steen. A survey of DHT security techniques. *ACM Computing Surveys (CSUR)*, 43(2), 2011.

[121] Vodafone Group. OneSocialWeb website. `http://onesocialweb.org`. Visited February 2013.

[122] W3C Community and Business Groups. OStatus Wiki. `http://www.w3.org/community/ostatus/wiki/Main_Page`, 2012. Visited February 2013.

[123] D. J. Watts and S. H. Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393:440–442, 1998.

[124] N. J. Welton and A. E. Ades. Estimation of markov chain transition probabilities and rates from fully and partially observed data: uncertainty propagation, evidence synthesis, and model calibration. *Medical Decision Making*, 25(6):633–645, 2005.

[125] C. Wilson, B. Boe, A. Sala, K. P. N. Puttaswamy, and B. Y. Zhao. User interactions in social networks and their implications. In *Proc. ACM European Conf. on Computer Systems (EuroSys'09)*, pages 205–218, 2009.

[126] Wolfram Research, Inc. *Mathematica 9.0*. 2012.

[127] B. Wong and S. Guha. Quasar: A Probabilistic Publish-Subscribe System for Social Networks. In *Proc. Intl. Workshop on Peer-to-Peer Systems (IPTPS'08)*, 2008.

[128] Z. Yao, D. Leonard, X. Wang, and D. Longuinov. Modeling Heterogeneous User Churn and Local Resilience of Unstructured P2P Networks. In *Proc. Intl. Conf. Network Protocols (ICNP'06)*, pages 32–41, 2006.

[129] J. Y. Yen. Finding the K Shortest Loopless Paths in a Network. *Management Science*, 17(11):712–716, 1971.

[130] C. A. Yeung, I. Liccardi, K. Lu, O. Seneviratne, and T. Berners-Lee. Decentralization: The future of online social networking. In *In Proc. W3C Workshop on the Future of Social Networking*, 2009.

Appendix A

Kolmogorov's Forward Equations

Let $\{\mathbf{X}(t), t > 0\}$ be a stochastic process defined by a finite, continuous-time Markov chain. As in Section 5.2.2, let $P_{ij}^{\mathbf{X}} = \mathbb{P}[X(t+s) = i \mid X(s) = j]$. Further, let $P(t)$ represent the matrix $\{P_{ij}^{\mathbf{X}}(t)\}$. If A is the generator matrix of $\mathbf{X}$, then, the following equations hold:

$$P(t) = P'(t)A \tag{A.1}$$

If chain $\mathbf{X}$ is regular – which is true of all finite chains – it is a known result [90] that Equation (A.1) admits the following solution:

$$P(t) = e^{At} = \sum_{n=0}^{\infty} \frac{t^n A^n}{n!} \tag{A.2}$$

But that is an open-form solution. With modern solvers, the system can sometimes be solved analytically and yield a closed-form expression. Recall we want to solve Equation (A.1) for our absorbing chain of Section 5.2.2. We have already computed its generator matrix. With those in hand, we can write the system of 16 forward equations for $\mathbf{Y}(t)$ (using $P(t)$ to simplify the notation) as:

$$\begin{aligned}
P'_{11}(t) &= \mu_v P_{12}(t) + \mu_u P_{13}(t) - (\gamma_u + \gamma_v) P_{11}(t) \\
P'_{21}(t) &= \mu_v P_{22}(t) + \mu_u P_{23}(t) - (\gamma_u + \gamma_v) P_{21}(t) \\
P'_{31}(t) &= \mu_v P_{32}(t) + \mu_u P_{33}(t) - (\gamma_u + \gamma_v) P_{31}(t) \\
P'_{41}(t) &= \mu_v P_{42}(t) + \mu_u P_{43}(t) - (\gamma_u + \gamma_v) P_{41}(t) \\
P'_{12}(t) &= \gamma_v P_{11}(t) - (\gamma_u + \mu_v) P_{12}(t) \\
P'_{22}(t) &= \gamma_v P_{21}(t) - (\gamma_u + \mu_v) P_{22}(t) \\
P'_{32}(t) &= \gamma_v P_{31}(t) - (\gamma_u + \mu_v) P_{32}(t) \\
P'_{42}(t) &= \gamma_v P_{41}(t) - (\gamma_u + \mu_v) P_{42}(t)
\end{aligned}$$

$$
\begin{aligned}
P'_{13}(t) &= \gamma_u P_{11}(t) - (\gamma_v + \mu_u) P_{13}(t) \\
P'_{23}(t) &= \gamma_u P_{21}(t) - (\gamma_v + \mu_u) P_{23}(t) \\
P'_{33}(t) &= \gamma_u P_{31}(t) - (\gamma_v + \mu_u) P_{33}(t) \\
P'_{43}(t) &= \gamma_u P_{41}(t) - (\gamma_v + \mu_u) P_{43}(t) \\
P'_{14}(t) &= \gamma_u P_{12}(t) + \gamma_v P_{13}(t) \\
P'_{24}(t) &= \gamma_u P_{22}(t) + \gamma_v P_{23}(t) \\
P'_{34}(t) &= \gamma_u P_{32}(t) + \gamma_v P_{33}(t) \\
P'_{44}(t) &= \gamma_u P_{42}(t) + \gamma_v P_{43}(t)
\end{aligned}
$$

Subject to the boundary conditions:

$$
P_{ij}(0) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}
$$

For which we have not, unfortunately, been able to find a solution.

Appendix B

Delays Over Top-k Graphs

In this chapter, we present our partial results on the problem of how to provide "closed form[1]" expressions to end-to-end delays over the $G_{k,u,v}$ graphs formed by putting together the top-k shortest paths connecting a pair of nodes u and v (referred loosely to as "top-k graphs", for convenience).

B.1 Modelling Issues

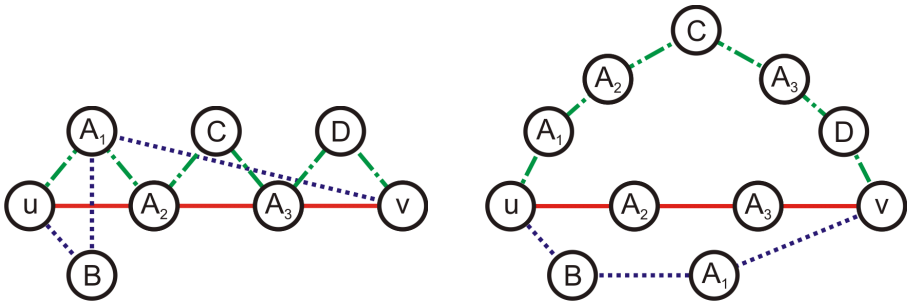
We would like our model to produce numbers that match the approximations obtained by simulations over these same top-k graphs in Section 4.4.4. This is not, however, a simple task, due to two main reasons.

B.1.1 Extra paths

Although the top-k graphs we were building in Section 4.4.4 were generated by putting together k paths (where k was a small number), the resulting graphs might end up allowing a lot more than k paths between source and destination. This is shown in Figure B.1a: although $G_{k,u,v}$ is composed by three edge-disjoint paths, it is clear that a message flowing from u to v through this graph would be allowed to follow other paths as well, e.g., $\{u, A_2, A_3, D, v\}$.

Since the full simulations we ran in Section 4.4.4 over the $G_{k,u,v}$ allow such paths to be followed, to accurately model delay over $G_{k,u,v}$ we would need to take them into account as well. The problem is that $G_{k,u,v}$ has no simplifying properties – it is, by all accounts, an arbitrary graph and, as such, can be as difficult to analyse as the original graph. The only thing it has to its advantage is being small. This leads to the question: *how much*

[1] We use "closed form" under quotes because obtaining closed form expressions is in general not possible when talking about arbitrary graphs. Whatever expression we come up with will have to encode the graph structure somehow in it and will, therefore, be different from one graph to the other.



(a) A top-k graph. (b) An "unfolded" top-k graph.

Figure B.1: Top-k graphs.

are these extra paths really contributing to delay? If contribution is small, then maybe we do not have to model them.

To answer this question, we can run a simple experiment in which we constrain our messages to follow exactly one of the top-k paths when going from u towards v. To that end, we produce an "unfolded" version of $G_{k,u,v}$ which contains only the top-k paths, put together in parallel. This effectively causes the message to be tunnelled through a path, precluding it from changing paths once it has "chosen" one of them. Vertices shared among paths are replicated. Unfolding the graph in Figure B.1a would result in the graph in Figure B.1b. Answering our question then amounts to comparing the results of running full simulations over unfolded graphs with those of running full simulations over $G_{k,u,v}$.

This raises, however, a question that leads us to our second and most important issue. *How do we simulate replicated vertices?*

B.1.2 Independence

If we want to emulate exactly what would happen if a message were allowed to follow one of the top-k paths in $G_{k,u,v}$, then the up/down behavior of replicated vertices must be "coupled": in other words, the copies of vertices A_1, A_2, and A_3 in Figure B.1b must behave as if they were one, going up and down together. And this is where things get complicated: all of the probabilistic machinery we have built thus far assumes that edge delays are independent. But, if this assumption holds, this means that A_1, A_2 and A_3 should be *decoupled* – i.e. it should be possible to have a system state in which A_1 is up, but one of its copies is down. This is clearly a problem, and building a model on top of such assumption would introduce an imprecision. The question, then, is *how imprecise would this assumption of independence be?* Again, to answer this question, we run experiments in which replicated vertices are both coupled and decoupled, and compare these with the full simulations on $G_{k,u,v}$.

B.1.3 Evaluation of Issues

For a given graph G, let us establish that:

1. **aed** is the "ground truth" for delay; i.e., it is the delay number we get from running full simulations over G;
2. $\mathbf{aed_k}$ is the delay estimate produced by running full simulations over $G_{k,u,v}$;
3. $\mathbf{aed_k^c}$ is the delay estimate produced by running full simulations over the unfolded graph, with coupled replicated vertices. This allows us to gauge the impact of removing extra paths from the top-k estimate. If the impact is low, it means that less complex models might suffice;
4. $\mathbf{aed_k^d}$ is the delay estimate produced by running full simulations over the unfolded graph, with decoupled replicated vertices. This allows us to gauge the impact of adding the independence assumption to edge delays.

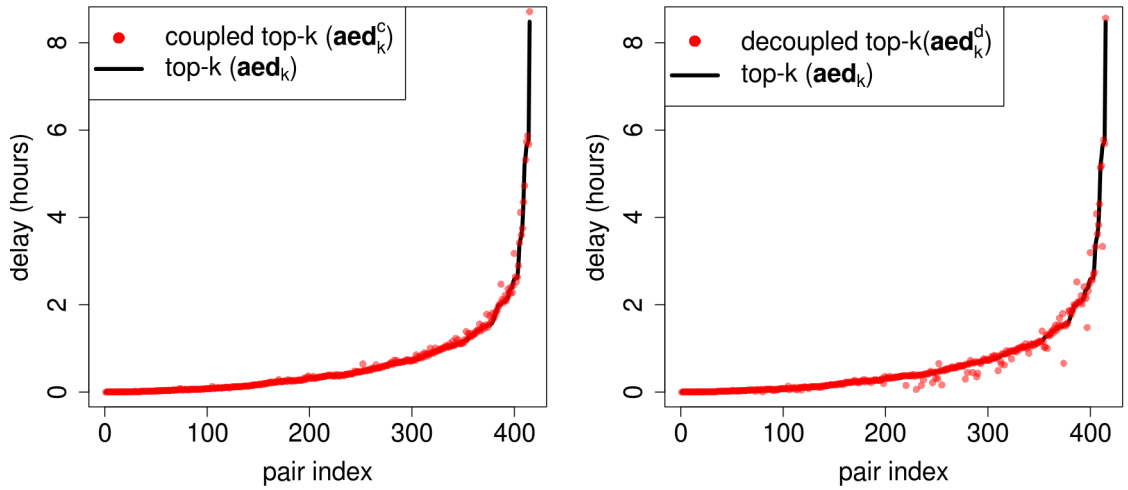
We run the experiments described in Section B.1.1 and Section B.1.2 to evaluate the contributions of extra paths and measure the imprecision introduced by assuming that edge delays are independent. For that, we use a small set of 600 source/destination pairs, taken at random from the 137 260 pairs of the sample we used in Chapter 4. Simulations are repeated 100 000 times.

Impact of extra paths. Figure B.2a shows a scatterplot comparing $\mathbf{aed_k}$ (shown as a black line) and $\mathbf{aed_k^c}$ (shown as dots). The fact that dots do not land far from the black line means delay estimates are not significantly impacted by them, suggesting that, in the sample we consider, extra paths are not our worst problem.

Impact of independence. Figure B.2b, on the other hand, shows a comparison between $\mathbf{aed_k}$ (black line) and $\mathbf{aed_k^d}$ (dots). Points fall farther away from the black line, indicating that the error is more severe.

Converting these impressions on numbers, Figure B.3 shows cumulative distribution functions of how far each of the estimates falls from the ground truth in percentage terms i.e., it shows $r_k = \left|\frac{\mathbf{aed_k}}{\mathbf{aed}} - 1\right|$, $r_k^c = \left|\frac{\mathbf{aed_k^c}}{\mathbf{aed}} - 1\right|$, and $r_k^d = \left|\frac{\mathbf{aed_k^d}}{\mathbf{aed}} - 1\right|$. Additional statistics are provided in Table B.1. While r_k^d would seem to be more accurate, what we actually see as we filter out the noisy area – delays smaller than 0.5h, which in the scatterplots look almost identical from Figure B.2a to Figure B.2b – is that independence adds to imprecision more than the removal of the extra paths.

Indeed, not only accuracy is worse, but by looking at Figure B.2b we see that there is also a loss of the upper bound property: by assuming independence of edge delays, we in some cases effectively "do better" r_k (and even **aed**). To see why, note that an edge is allowed, by independence, to be concurrently active and inactive for different paths, even if it is shared among them. The same goes for vertices: in Figure B.1b, say, A_1 allowed



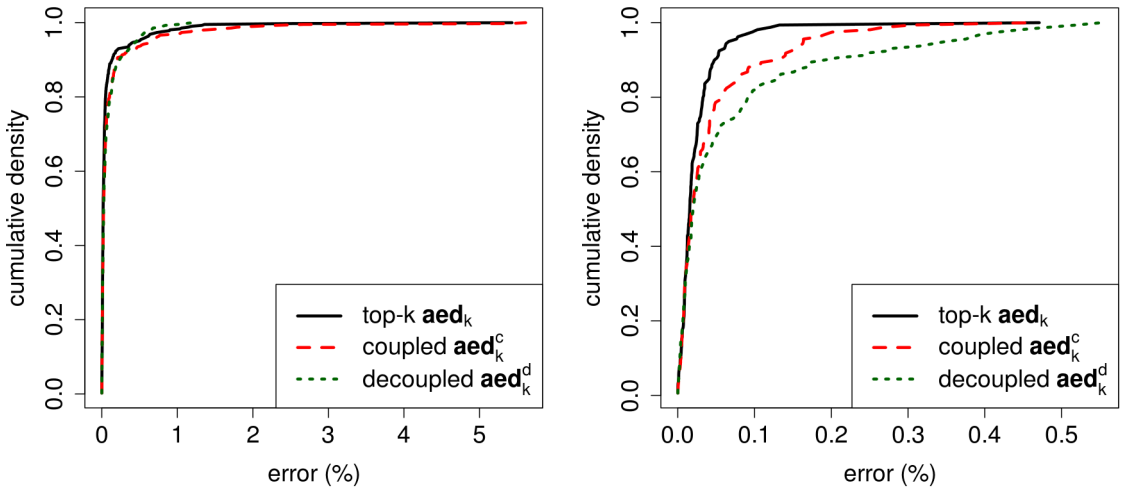
(a) Contributions of extra paths.

(b) Imprecision of independence.

Figure B.2: Unfolded top-k graphs versus top-k full sims.

	avg.	95^{th}	99^{th}	**max.**
r_k	9%	48%	118%	543%
r_k^c	13.7%	64%	201%	561%
r_k^d	8.7%	45%	79%	120%
r_k (*)	2.5%	7%	12%	47%
r_k^c (*)	4.3%	16%	28%	46%
r_k^d (*)	6.8%	35%	49%	55%

Table B.1: Complementary statistics for approximation errors. (*) are only for pairs in which **aed** $>$ 0.5h.



(a) All pairs.

(b) Pairs for which **aed** $>$ 0.5h.

Figure B.3: Normalized error CDFs.

to be down for one path, and up for another. This gives A_1 has two "chances" of being up, one for each of its copies, effectively increasing its availability and reducing delay.

B.2 A Multipath Delay Model

There are two main options for dealing with the problems we have outlined. We can either:

1. model extra paths and account for dependence, or;
2. mitigate both problems by constructing top-k graphs from least-cost, *vertex-disjoint paths* instead. Vertex-disjoint paths are much easier to deal with, since they *i)* inherently do not give raise to extra paths, and *ii)* do not give raise to paths that share vertices beyond source and destination and are, therefore, less amenable to the inherent imprecision of models that assume edge delay independence.

Figure B.4 illustrates the idea by depicting a top-k graph made from vertex-disjoint paths. Since paths are disjoint, the only sets of edges for which delays are not independent are sets A and B: if either source or destination are down, they are all deactivated together. This is, however, in stark contrast with the graph in Figure B.1a, made from edge-disjoint paths, in which all vertices are shared among at least two different paths. This means that every single edge is dependent on some other edge that belongs to another

path, and they all are, therefore, potential sources of imprecision.

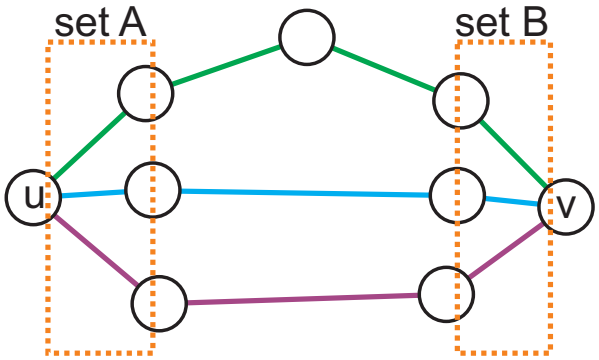


Figure B.4: A top-k graph made of vertex-disjoint paths. Sets A and B mark edges for which delays are not independently distributed.

Yet, as a first step, we need to understand whether top-k graphs made from vertex-disjoint paths even provide an acceptable approximation to end-to-end delay. We investigate that next.

B.2.1 Feasibility of Vertex-Disjoint Paths

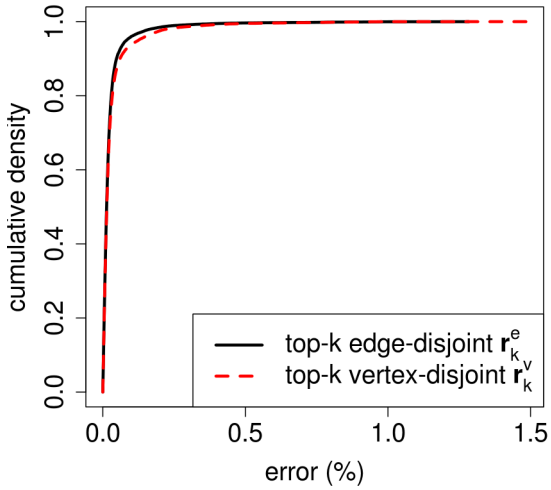
We measure the delay approximation quality of graphs constructed by selecting the top-k, least-cost, vertex-disjoint paths connecting a pair of vertices u and v by comparing its quality with that of the edge-disjoint approach we had adopted in Section 4.4.3.

To make a clear distinction, we refer to source/destination delays measured over graphs with top-k edge-disjoint and vertex-disjoint paths as $\mathbf{aed_k^e}$ and $\mathbf{aed_k^v}$, respectively. As with the experiments we conducted in Section 4.4.3, we use $k = 10$. Further, as before, we refer to the "ground truth" for the end-to-end delay between a pair of nodes u and v as **aed**. **aed** is obtained by running full simulations over the original graph G.

For these experiments, we evaluate all of the 137 260 source/destination pairs from the dataset of Section 4.3.2; i.e, for each source/destination pair u and v we:

1. construct two top-k graphs: one made from the least-cost edge-disjoint, and the other made from the least-cost vertex-disjoint paths connecting u and v;
2. for each graph, we run n full simulations (more later);
3. each simulation yields a delay point (the delay between u and v), which we then use to compute the averages $\mathbf{aed_k^e}$ and $\mathbf{aed_k^v}$.

Unlike in Section 4.3.2, the number n of full simulations we run for each source/destination pair here is not fixed. Instead, we leverage on the fact that averages under Markov processes are well-behaved (i.e. they follow a normal law [33]), and implement a batched



(a) CDFs of r_e^k and r_v^k.

Statistic	r_k^e	r_k^v
Average	2.6%	3.1%
50^{th} percentile	1.3%	1.3%
75^{th} percentile	2.5%	2.7%
90^{th} percentile	4.7%	6%
95^{th} percentile	8.3%	12.5%
99^{th} percentile	26.3%	35.3%
Maximum	128.3%	148.6%

(b) Complementary statistics.

Figure B.5: Comparison of prediction accuracy of edge and vertex-disjoint top-k graphs.

means estimator [3] for variance. Simulations are then repeated until we get a confidence interval that has a width of less than 5% of the sample average at 95% confidence. Estimates satisfying our quality criterion emerge after around 10 000 repetitions, which is a 10-fold improvement over the 100 000 repetitions we were using before.

As before, we look at the normalized error metrics $r_k^e = \left|\frac{\mathbf{aed_k^e}}{\mathbf{aed}} - 1\right|$ and $r_k^v = \left|\frac{\mathbf{aed_k^v}}{\mathbf{aed}} - 1\right|$ which tell us how far, in percentage terms, approximations fall from the ground truth **aed**. Figure B.5a shows the cumulative distributions for the two error metrics over all source/destination pairs, and Table B.3 provides complementary statistics. To avoid the usual anomalies with normalized metrics in the low-delay region, errors are only shown for pairs for which $\mathbf{aed} > 0.5$.

Data shows that the top-k, least-cost edge-disjoint paths provide a better estimate for delay than the vertex-disjoint paths. Further, the graphs generated by the vertex-disjoint heuristic are around 30% larger than the ones generated by the edge-disjoint heuristic. The decrease in accuracy, however, is not dramatic, and the decrease in analytical complexity more than compensates for the increase in size. Vertex-disjoint paths are, therefore, a viable alternative.

B.2.2 Delays Over Vertex-Disjoint Paths

Let $\{P_1, \cdots, P_k\}$ be a set of vertex-disjoint paths sharing a common source u and destination v, such that $G_{k,u,v} = \bigcup_{i=1}^{k} P_k$. Since vertex-disjoint paths do not share vertices

beyond source and destination, we can attempt to model delay between u and v over $G_{k,u,v}$ by constructing a simpler model which assumes edge delays to be independent.

The intuition is that every message m can follow exactly one of k paths. The minimum possible delay incurred on m is, therefore, the minimum among the delays of all k paths. Formally, let $\mathbf{D}_{k,u,v}$ be the random variable representing end-to-end delay between u and v over $G_{k,u,v}$. Recalling that $\mathbf{D}_P$ denotes the random variable representing the delay of path P, then:

$$\mathbf{D}_{k,u,v} \sim \min\{\mathbf{D}_{P_1}, \cdots, \mathbf{D}_{P_k}\} \tag{B.1}$$

We refer to the approximation of $\mathbf{D}_{k,u,v}$ provided by $\min\{\mathbf{D}_{P_1}, \cdots, \mathbf{D}_{P_k}\}$ as $\widehat{\mathbf{D}}_{k,u,v}$. Once again, the reason why the relationship between the $\widehat{\mathbf{D}}_{k,u,v}$ and $\mathbf{D}_{k,u,v}$ is approximate is because the former assumes edge delay independence, whereas the latter does not. Since, as we reasoned in Section B.1.3, independence is an optimistic assumption from the point of view of delay, we have that:

$$\mathbb{E}(\widehat{\mathbf{D}}_{k,u,v}) \leq \mathbb{E}(\mathbf{D}_{k,u,v}) \tag{B.2}$$

In other words, $\mathbb{E}(\widehat{\mathbf{D}}_{k,u,v})$ provides a lower bound to $\mathbb{E}(\widehat{\mathbf{D}}_{k,u,v})$. We will not provide a formal proof of Equation (B.2) – rather, we provide numerical evidence that it holds in Section B.4.

B.3 Expected Delay Over Vertex-Disjoint Paths

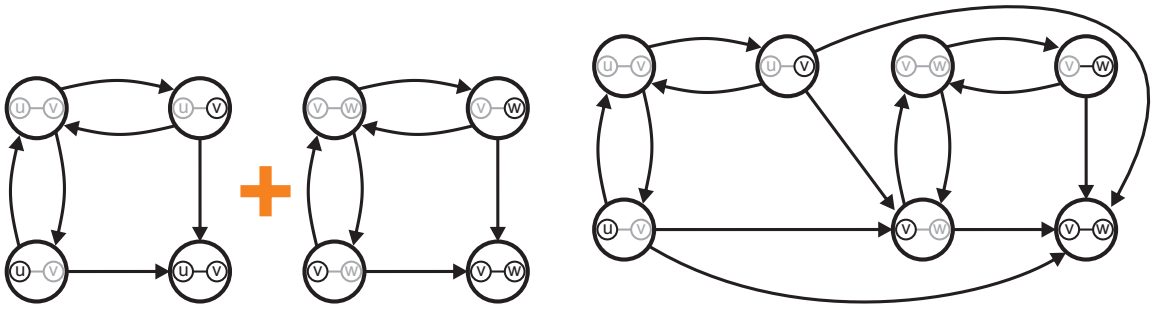
Understanding the distribution of $\widehat{\mathbf{D}}_{k,u,v}$ requires us to examine what are the distributions of path delays. From Theorem 3 we know that path delays are sums of edge delays. Therefore, we start by taking a closer look at the distributions of the latter.

B.3.1 Distributions of Edge and Path Delays

Let $\text{PH}(\boldsymbol{\tau}, \mathbf{S})$ denote a phase-type distribution with parameters $\boldsymbol{\tau}$ and $\mathbf{S}$. As before, let $\mathbf{X}_e$ be the random variable representing the delay incurred by an edge $e = (u, v)$, and let π_1 denote the probability that v is online when the message reaches u. Then, for $t \leq 0$, $\mathbf{X}_e$ exhibits a *mixed distribution* with cumulative density:

$$\mathbb{P}[\mathbf{X}_e \leq t] = \pi_0 + (1 - \pi_0)\left(1 - \boldsymbol{\tau} e^{\mathbf{S}t}\right) \tag{B.3}$$

Such mixed distributions are, in fact, phase-type distributions with *atoms at zero*, which can, fortunately, be dealt with by existing theory. Let the initial probability vector of a phase-type distribution $\text{PH}(\boldsymbol{\tau}, \mathbf{S})$ with an atom at zero be denoted as $(\boldsymbol{\tau}, \tau_m)$, where τ_m represents the density of the atom at zero – i.e. it represents the probability that the corresponding CTMC starts at the absorbing state m. Then:



(a) Chains for two separate edges.

(b) Chain resulting from convolution.

Figure B.6: Markov chain representation of the convolution of phase-type distributions.

Theorem 4 (Convolution of Phase-type Distributions [74])**.** *Let* $\mathbf{X} \sim PH(\boldsymbol{\tau}, \mathbf{S})$ *and* $\mathbf{Y} \sim PH(\boldsymbol{\alpha}, \mathbf{T})$ *be two phase-type distributed variables with initial probability vectors* $(\boldsymbol{\tau}, \tau_m)$ *and* $(\boldsymbol{\alpha}, \alpha_k)$*. Then* $\mathbf{Z} = \mathbf{X} + \mathbf{Y} \sim PH(\boldsymbol{\gamma}, \mathbf{L})$ *with* $\boldsymbol{\gamma} = (\boldsymbol{\tau}, \tau_m \boldsymbol{\alpha})$*, and:*

$$\begin{bmatrix} \mathbf{L} & \mathbf{l} \\ \mathbf{0} & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{S} & \mathbf{s}\boldsymbol{\gamma} & \alpha_k \mathbf{s} \\ \mathbf{0} & \mathbf{T} & \mathbf{t} \\ \mathbf{0} & \mathbf{0} & 0 \end{bmatrix} \tag{B.4}$$

The interpretation for this theorem is straightforward: it builds a new Markov chain in which the absorbing state of the first chain ($\mathbf{X}$) is replaced by edges towards the initial states of the second chain ($\mathbf{Y}$), in a way that that respects the initial probability distribution $\boldsymbol{\alpha}$ for the transient states of Y. The resulting chain is absorbing, so that the absorption time is phase-type distributed as well. Figure B.6 shows the Markov chain that results from convoluting two edge delay distributions as prescribed by the theorem (transition rates omitted).

With those in hand, the density of the atom at zero for the new distribution (i.e. the probability for the new chain to start at the absorbing state) can be computed as:

$$\gamma_{m+k+1} = 1 - \boldsymbol{\gamma} \cdot \mathbf{1} \tag{B.5}$$

B.3.2 Validation

To provide evidence that our path delay model is correct, we run simulations over 10 line graphs of size 10, repeating simulations 3 000 000 times. We then perform two comparisons:

1. we compute the analytical expectation from the predicted phase-type distribution directly from Equation (5.13) (i.e. without using Equation (5.3.2)), and compare it

against the average end-to-end delay towards the rightmost vertex in the line graph. Results are shown in Figure B.7a.

2. We compare simulation output against an equivalent number of samples drawn the predicted phase-type distribution by performing a two-sample, two-sided Kolmogorov-Smirnov (K-S) test [92]. p-values for the 10 tests are shown in Figure B.7b.

As a last piece of evidence, we plot the histogram of one of the simulation runs against the PDF for the predicted phase-type distribution, computed using jPhase [82]. This is shown in Figure B.7c.

Given that averages seem to match quite well, and that the p-values for the K-S tests are all below the 95% threshold, we conclude that there is enough evidence to say that our model is correct.

B.3.3 Distributions of Delays Over Multiple Paths

Now that we know that path delays are phase-type distributed, we can apply a final result [74] to obtain the distribution of $\widehat{\mathbf{D}}_{k,u,v}$. This result requires a special matrix operation called *Kronecker product*, which we describe next.

Definition 2 (Kronecker Product)**.** *Let* $\mathbf{A}$ *be an* $m \times n$ *matrix and* $\mathbf{B}$ *a* $p \times q$ *matrix. The Kronecker product* $\mathbf{A} \otimes \mathbf{B}$ *is the* $mp \times nq$ *block matrix:*

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix} \tag{B.6}$$

And the final theorem we need:

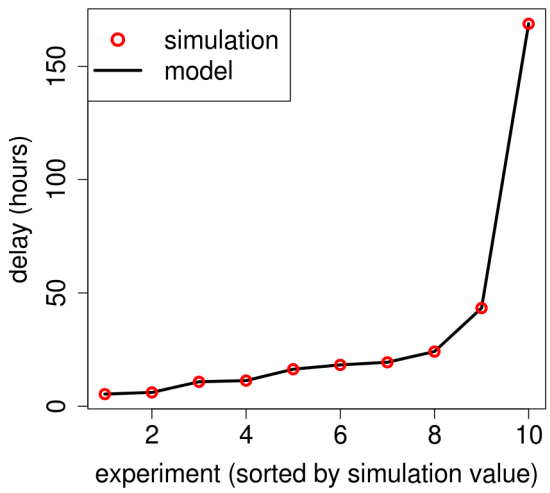
Theorem 5. *Let* $\mathbf{X} \sim PH(\boldsymbol{\tau}, \mathbf{S})$ *and* $\mathbf{Y} \sim PH(\boldsymbol{\alpha}, \mathbf{T})$ *be two phase-type distributed variables. Further, let* $\mathbf{I}_\mathbf{A}$ *represent the identity matrix with the same dimensions as the square matrix A, and let* $\otimes$ *represent the Kronecker product. Then* $\mathbf{Z} = \min\{\mathbf{X}, \mathbf{Y}\} \sim PH(\boldsymbol{\gamma}, \mathbf{L})$*, and:*

$$\mathbf{L} = \mathbf{S} \otimes \mathbf{I}_\mathbf{T} + \mathbf{I}_\mathbf{S} \otimes \mathbf{T} \tag{B.7}$$

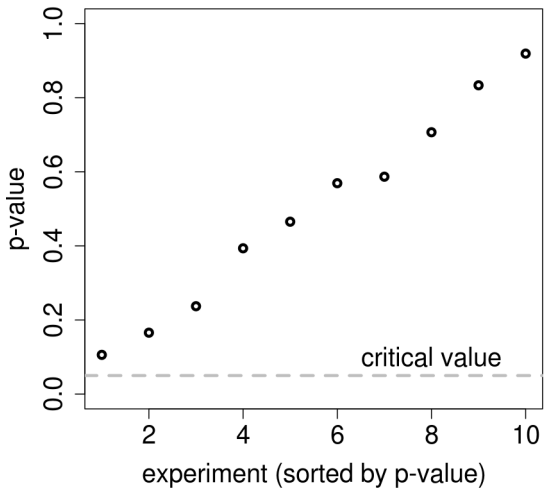
$$\boldsymbol{\gamma} = \boldsymbol{\tau} \otimes \boldsymbol{\alpha} \tag{B.8}$$

Which essentially allows us to compute the distribution for $\widehat{\mathbf{D}}_{k,u,v}$. This simple model has, however, two significant limitations.

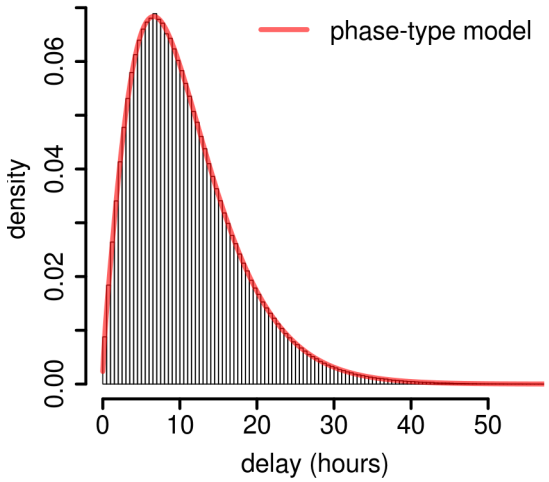
Inherent imprecision. As already discussed, assuming edge delay independence introduces an inherent imprecision into the model. The upper bound property is lost, with the approximation becoming *lower bound* for top-k delays instead. The less available the



(a) Delay averages.



(b) p-values for K-S tests.



(c) Delay distribution on simulation and model.

Figure B.7: Validation of path delay.

sender or the receiver, the worst the imprecision. The fact that we have a lower bound ($\mathbb{E}(\widehat{\mathbf{D}}_{k,u,v})$) for an upper bound on delay ($\mathbb{E}(\mathbf{D}_{k,u,v})$) can make the numbers we produced by the model difficult to interpret. We evaluate model precision in Section B.4.

Cost. The fact we need to perform a Kronecker product for each of the k paths means that the number of rows and columns in the matrices grows exponentially with k. A path of length $n-1$ generates a matrix of size $3n \times 3n$. Assuming all paths have size at least $n-1$, the minimum of k paths, as for Theorem 5, generates a matrix of size:

$$l^2 \geq (3n)^k \times (3n)^k = (3n)^{2k} \tag{B.9}$$

Assuming 64 bits per element and paths of length 1, a k of 10 – which is what we have been using – would require or 29.2 *petabytes* of RAM. In practice, however, our paths are much longer than that: the average length from our sample 3.43, but some of them have sizes as large as 17.

Computational costs are high, too: even if we settle for expected delay, Equation (5.13) says we need invert the matrix that comes out of the Kronecker product. Inverting matrices with billions of elements is not cheap, and can quickly become more costly than performing simulations. Fortunately, at least for costs, there is a way to mitigate the issue.

B.3.4 A Refined Hybrid Analytical/Simulation Model

The cost perspective can be mitigated if, instead of attempting to obtain the distribution of $\widehat{\mathbf{D}}_{k,u,v}$ directly, we try to indirectly sample from it. Population parameters such as mean delays can then be computed numerically by the use of point estimators.

This can be achieved as follows. Recall that $\widehat{\mathbf{D}}_{k,u,v} = \min\{\mathbf{D}_{P_1}, \cdots, \mathbf{D}_{P_k}\}$. From B.3.1, we know how to compute the parameters for the phase-type distributions for the path delays $\mathbf{D}_{P_i}$, $(1 \leq i \leq k)$. To generate a sample (variate) from $\widehat{\mathbf{D}}_{k,u,v}$, therefore, all we have to do is generate a sample from each of the $\mathbf{D}_{P_i}$, and then take the minimum. This is illustrated in Algorithm 8: every iteration of the loop in lines 2–7 generates exactly one sample for $\widehat{\mathbf{D}}_{k,u,v}$, which gets accumulated into variable s (line 7). The procedure is repeated n times and, at the end, the algorithm returns the sample average (line 8).

The key to the complexity of Algorithm 8 is function *sampleFrom*. Neuts and Pagano [73] provide us with a way to generate samples from $\mathbf{D}_{P_i}$: simply simulate the underlying continuous-time Markov chain, and measure the time-to-absorption. The algorithm, therefore, is effectively a *simulation approach*, except that simulating the underlying CTMC is *much cheaper* than running full simulations over $G_{k,u,v}$, for two reasons. First, with a CTMC, there is no need to simulate up/down behavior of individual nodes. Second,

Algorithm 8: EstimateMultipathDelay

Input: Number of samples n; phase-type distributions $f_1, \cdots, f_k$ for path delays.

```
1 s ← 0                                              % Sum of all samples
2 for i ← 1 to n do
3     m ← +∞
4     for j ← 1 to k do
5         r ← sampleFrom(f_j)
6         m ← min{m, r}
7     s ← s + m
8 return s/n
```

and most importantly, there is no need to perform any burn-in since starting probabilities for each state are computed analytically beforehand.

B.4 Validation

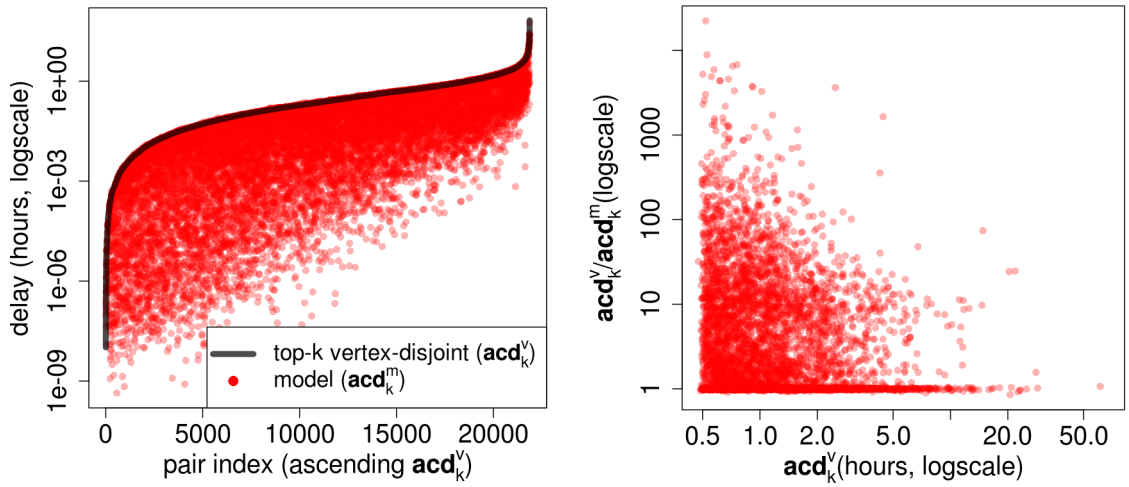
In this section, we evaluate our top-k vertex-disjoint model by: *i)* numerically quantifying the magnitude of its inherent imprecision; *ii)* measuring its usefulness by comparing it against the remaining estimation techniques we have developed thus far, namely, the single-path upper bounds of Section 4.4.2, and the full simulations over top-k graphs, constructed both with edge and vertex-disjoint paths.

Since we will reference a large number of delay estimates, it is useful to recall the notation. As before, we let $\mathbf{aed}$, $\mathbf{aed_k^v}$, and $\mathbf{aed_k^e}$ denote the average delay estimates obtained by running full simulations over: *i)* the original graph (i.e. our "ground truth" for end-to-end delay), *ii)* a top-k graph built with vertex-disjoint paths (i.e. approximates $\mathbb{E}(\mathbf{D}_{k,u,v})$), and *iii)* a top-$k$ graph built with edge-disjoint paths.

Further, we define:

1. $\mathbf{aed_k^m}$ to be the delay estimate produced by Algorithm 8 (i.e. approximates $\mathbb{E}(\widehat{\mathbf{D}}_{k,u,v})$);
2. $\mathbf{aed}_1$ to be the delay estimate produced by the single-path upper bound of Section 4.4.2, computed with Equation (5.3.2).

We evaluate delays over the same set of 137 260 source-destination pairs we have used across this chapter. Simulations, when run, are repeated at most 100 000 times, or until we get a confidence interval that is wide by less than 5% of the sample average at 95% confidence.



(a) (b)

Figure B.8: Multipath model plots.

B.4.1 Model Imprecision

The inherent imprecision of our model is quantified by comparing its estimate ($\mathbf{aed_k^m}$) against full simulations ran over the top-k graphs made from vertex disjoint paths ($\mathbf{aed_k^v}$). First, we show in Figure B.8a a scatterplot of both $\mathbf{aed_k^m}$ (dots) and $\mathbf{aed_k^v}$ (black line), ordered by their value of $\mathbf{aed_k^v}$. We can see that our claim made in Equation (B.2) – namely that the model consistently underestimates $\mathbf{aed_k^v}$, bounding it from below – is correct.

The magnitude of this imprecision is better quantified, instead, in Figure B.8b, where we take the ratio $\frac{\mathbf{aed_k^v}}{\mathbf{aed_k^m}}$ and plot it against the corresponding values for $\mathbf{aed_k^v}$. Model predictions are rather erratic, with values that are can be off by factors of up to 6 774. Predictions tend to fare slightly better in the higher delay range, but not by much. Complementary statistics for the error are provided in Table B.2: up until the 50% percentile, predictions are rather reasonable, but quickly deteriorate as we move to higher percentiles.

B.4.2 Comparison with Other Estimation Approaches

Finally, we compare $\mathbf{aed_k^m}$ with the remaining approaches. For each source-destination pair and each delay estimate i, we compute a normalized error metric as follows. For $\mathbf{aed_1}$, $\mathbf{aed_k^v}$, $\mathbf{aed_k^e}$ – which are always larger than $\mathbf{aed}$ – we compute the metrics r_1, r_k^v, r_k^e we have already been using, i.e., for an estimate i, we take the error to be $\left|\frac{i}{\mathbf{aed}} - 1\right| = \frac{i}{\mathbf{aed}} - 1$.

	Underestimation
Average	26%
50^{th} percentile	16%
75^{th} percentile	571%
90^{th} percentile	2576%
95^{th} percentile	6384%
Maximum	6774%

Table B.2: Error statistics for the multipath model.

Statistic	r_k^e	r_k^v	r_1	r_k^m	o_k^m
Average	2.6%	3.1%	8.4%	$+\infty$	1654%
50^{th} percentile	1.3%	1.7%	1.9%	23%	25%
75^{th} percentile	2.5%	3.9%	7.9%	85%	571%
90^{th} percentile	4.7%	11%	23%	96%	2397%
95^{th} percentile	8.3%	19.5%	39%	98%	6210%
99^{th} percentile	26.3%	44.3%	89%	99%	45989%
Maximum	128.3%	158.6%	191%	$+\infty$	1.2×10^8%

Table B.3: Comparison among estimation approaches.

For $\mathbf{aed_k^m}$, things are more complicated. Since it might *underestimate* **aed**, simply taking the normalized distance $r_k^m = \left|\frac{\mathbf{aed_k^m}}{\mathbf{aed}} - 1\right|$ would not be fair: since delay can be at most zero, there is less distance to "cover" when underestimating. To see why, imagine we had a trivial estimation method that always returns zero. The maximum error incurred by this estimation method would be 100%, even though it is not even trying to estimate anything.

What we want, one could argue, is a measure of how $\mathbf{aed_k^m}$ and **aed** are *equal* and, to that end, the metric $o_k^m = \frac{\max\{\mathbf{aed_k^m}, \mathbf{aed}\}}{\min\{\mathbf{aed_k^m}, \mathbf{aed}\}}$ could be better. The problem is that it penalizes underestimation more than it penalizes overestimation: an estimate that is 100% larger than **aed** will yield an error of 1, whereas an estimate 100% smaller will tend to infinity. We therefore analyse both metrics.

Figure B.9 shows the cumulative distributions for all error metrics, with additional statistics provided in Table B.3. The curves for r_1, r_k^v, r_k^e are rather similar, reflecting the fact that we get progressively better approximations as we add more paths, with r_k^e being the best performer. The multipath model, on the other hand, exhibits a somewhat erratic behavior, with much larger error values even under r_k^m. Under o_k^m, the error skyrockets, also as a reflex of heavy underestimation.

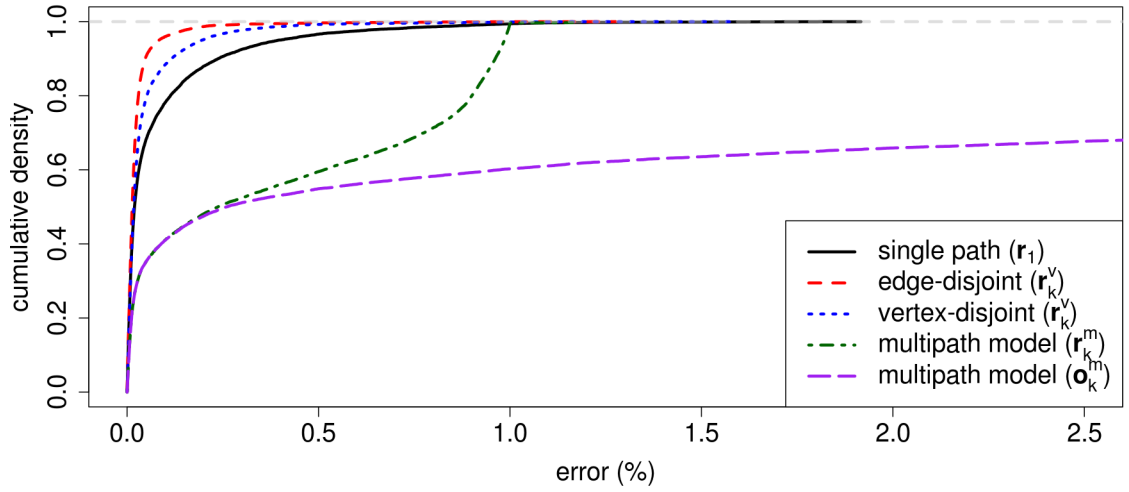


Figure B.9: Error metric distribution for all approaches.

B.5 Discussion and Outlook

In this chapter, we have shown the challenges of delay modelling over top-k graphs, and presented our results on a partial model. The model has one major and one minor shortcoming. Respectively, it is inherently inaccurate due to the assumption of edge delay independence, and, in its purest form, it has unreasonably large memory requirements. The latter issue can be remedied by the adoption of the hybrid technique of Section B.3.4, though we intend to explore, as future work, the fact that *acyclic* phase-type distributions [53] like ours can be reduced, possibly leading to savings in memory consumption and enabling instances combining more paths to be built.

The issue of model imprecision is more pressing, however, and the fact that its accuracy remains low even with vertex-disjoint paths means that edge delay dependence has to be accounted for somehow. Clearly, there is a Markov chain representation for the top-k graph that takes dependence into account, and such chain will still be absorbing since the process still "ends" as the message reaches the destination. As our next step, we intend to explore how to explicitly build these chains, and analyse the resulting phase-type distributions to assess the feasibility of the approach.