E-squads : a novel paradigm to build privacy-preserving ubiquitous applications

Adrien Luxey

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Thèse de doctorat de

L'UNIVERSITÉ DE RENNES 1
Comue Université Bretagne Loire

École Doctorale N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Informatique

Par

Adrien LUXEY

Les e-squads : Un nouveau paradigme pour la conception d'applications ubiquitaires respectant le droit à la vie privée

Thèse présentée et soutenue à Rennes (France), le 29 Novembre 2019
Unité de recherche : Irisa (UMR 6074)

Rapporteurs avant soutenance :

Romain ROUVOY	Professeur des Universités	Université de Lille
Vivien QUÉMA	Professeur des Universités	Grenoble INP

Composition du Jury :

Présidente :	Anne-Marie KERMARREC	Directrice de recherche	Univ Rennes, CNRS, Inria, IRISA
Rapporteurs :	Romain ROUVOY	Professeur des Universités	Université de Lille
	Vivien QUÉMA	Professeur des Universités	Grenoble INP
Examinatrice :	Sonia BEN MOKHTAR	Directrice de recherche	CNRS Lyon
Dir. de thèse :	Yérom-David BROMBERG	Professeur des Universités	Univ Rennes, CNRS, Inria, IRISA

Notre héritage n'est précédé d'aucun testament.

René Char

Technology is neither good nor bad; nor is it neutral.

Melvin Kranzberg

A tenured scientist

Table of Contents

Chapter 1

Introduction

Humanity is facing an ongoing crisis, caused by yet another revolution of its communication means [1]. The information era—empowered by the Internet and the multiplication of connected appliances at our disposal—is not living up to our hopes: what was once advocated like a decentralised revolution empowering people is turning into a deceitful, exploitative mass surveillance program. In this thesis, we explore new perspectives to put the power back into the people's hands—literally, by letting our devices partake in our digital lives, instead of being disguised espionage weapons turned against the gullible population.

The tension between centralisation and decentralisation Two trends have always competed in organising societies at large. Centralisation, through a pyramidal power structure, allows fast decision-making and high degrees of control, sometimes at the expense of individual liberties. In reaction, decentralisation advocates smaller decision bodies, aggregated by communication and consensus.

In informatics, this antagonism has always been evident. From the 1950's era of huge mainframes, shared among multiple users, to the personal computer advent in the 70's. From the centralised traditional communication channels (phone, radio, television) to the distributed packet switching that birthed the Internet. Computer science trends have always switched back and forth towards centralised or decentralised architectures.

Today, the inherently distributed Internet is the foundation of a very centralised, integrated applicative infrastructure, where a few monopolies—the major cloud companies—capture most of the traffic.

The centralised cloud model and its drawbacks At the time of writing, the winning communication paradigm is the following: 'dumb devices empowered by the limitless cloud'. In other words, people own several devices, that keep a constant connection to the 'cloud' (a very concrete horde of data-centres) to achieve their tasks.

This paradigm allows low-end devices to perform complex operations (e.g. multimedia editing or vocal interaction) by outsourcing all processing to remote data-centres. Resolving to cloud services also eases the life of application designers, by taking serious burdens out of their shoulders: ensuring the users' data availability, integrity and security, for instance, becomes the cloud provider's job. However, these advantages are balanced by serious drawbacks.

Firstly, the environmental impact of the cloud model is tremendous. Information & Communication Technologies (ICT) as a whole consumed 7% of the global electricity offer in 2012 [2, 3], more than 10% as of today, and a 21% share in 2030 according to estimates [4, 5]. The interesting fact is that, over the lifetime of a mobile appliance (including manufacturing, charging, operation and disposal), 90% of its electricity and carbon footprints are imputable to network communication and data-centres [5, 6]. The music industry also has a bigger carbon footprint in the streaming era than it ever had when it manufactured vinyls, cassettes and CD-ROMs [7].

Secondly, the cloud model poses the problem of data governance: when Alice talks with Bob online, the data they produce is theirs. Still, most cloud services use this data to display targeted advertising to their users, which allows them to provide free service in exchange. Although users agree to this when signing the terms & conditions, it is a bad deal: it trades the people's right to privacy [8] without their free and informed consent [9, 10]. In addition, their personal information might end up publicly available, or could be pirated.

The sole fact that seemingly harmless personal information is so easily available online represents great threats for the individuals, free speech, and democratic societies. For instance, some private health insurance companies cherry-pick customers using unwarranted datasets [11], State surveillance agencies cannot resist the urge to 'collect it all' [12–14], and the Cambridge Analytica company even biased the US' 2016 presidential elections and the UK's Brexit referendum by spreading highly targeted disinformation using just stolen Facebook 'like' datasets [15–18].

Although the cloud paradigm is convenient, and will always remain useful, most personal data needs to escape its databases and find a more privacy-preserving place to live. We have hope, though, as we witness the multiplication of connected devices per capita.

The rise of the Internet of Things (IoT) The concept of *ubiquitous computing*, proposed by Mark Weiser as early as 1991 [19], advocates that computer technologies,

as they get assimilated by societies, will proliferate in our environment, while fading in the background (and be less obtrusive). Ubiquitous computing, under the new flag name 'IoT', has gained a lot of leverage in recent years, as technical challenges get solved to accomplish it.

Its promises are appealing: to 'bring information technology beyond the big problems like corporate finance and school homework, to the little annoyances like Where are the car-keys, Can I get a parking place, and Is that shirt I saw last week at Macy's still on the rack?' [20]; or to distribute the production of renewable electricity in every household, build a 'smart' energy grid and finally accomplish energy-efficient societies [1].

However, the IoT's threats to the civil liberties are just as big: the variety and granularity of personal data generated by the newer connected appliances aretremendous, and would allow astounding degrees of mass surveillance. The data output by a smart electricity meter exposes the inhabitants' private lives in unprecedented ways [21]; connected cars are not safe from remote attacks [22]; and fitness wearables happily share their users' health records [23, 24] when they are not simply publicly available [25]. Other polemics tackle the lack of interoperability between devices [26–28], or the environmental impact of multiplying semiconductor-based productions (that contain highly toxic chemicals and non-renewable rare-earth metals) [29].

Given the massive adoption of the IoT by the public (see the makers communities [30] or the planetary success of the Raspberry Pi micro-computers [31]) and by the industry, we have to agree with Cisco's prediction [32] that the number of devices per person is bound to keep rising (from 2.4 per capita in 2017, to 3.6 per capita by 2022). The IoT trend is here to stay, but it still has to prove its worth.

It is our responsibility to pave a righteous path for the IoT, one that empowers the people and facilitates sustainable societies, before it is irrevocably jeopardized by the surveillance economy. We propose to make people's devices actors of the information decentralisation, by introducing the novel concept of *e-squads*:

Definition. *An **e-squad** is a set of connected ('smart') appliances* owned *by a single individual, and rendered intelligent through device-to-device communication among the e-squad. By aggregating information on its user, the fleet of devices learns its own dynamics, and can predict its own future state. By ensuring that the collected data remains on the individual's peripherals, the e-squad's knowledge benefits its owner in complete privacy.*

In this thesis, we explore the potential of e-squads, as enablers of services that respect

the human rights, in addition to being (at least!) as convenient and delightful [33] as their cloud counterparts.

In the next chapter, we present the state of the art, and show that enabling e-squads is yet an open research problem. Chapters 3 to 5 present our contributions. Finally, we conclude our dissertation in chapter 6.

Chapter 2

State of the Art

We now review the current state of the research in the domains we deem important and closely related to the problem at hand: to take advantage of the proliferation of personal connected devices to organise them as e-squads, and use them to build quality privacy-preserving services. Our presentation will lead us to the open research questions that we ought to resolve to enable the e-squad paradigm.

Some definitions To start off, let us clarify our prevalent concepts. A system is *centralised* when a subset of it represents an authority that needs to be trusted by all for the system to work [34]. A *decentralised* system, by opposition, has no such authority, and every peer is deemed potentially adversarial. A *distributed* system arises when distributed entities have to solve a problem, and that 'each entity has only a partial knowledge of the many parameters involved in the problem that has to be solved' [35]. As such, Michel Raynal states that 'distributed computing can be characterised by the term *uncertainty*' [35]. *Cloud computing* is 'a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services)' [36]. A cloud service provider, owning computing resources, must be able to rapidly provision and release the latter at the client's demand with minimal management operation.

Note that all cloud computing platforms are nowadays massively distributed: huge clusters of computers orchestrate their clients' provisioning demands and varying workloads. Although, they are not decentralised: it's the same entity (the company) that manages the whole cluster and is trusted by its clients to provide quality service.

These concepts are often confused, and we make no exception to this rule, as we tend to use 'distributed' and 'decentralised' interchangeably throughout this document, meaning 'decentralised' in both cases.

We now proceed to presenting works related to our problem. We start by demonstrating the intractable privacy issues with the cloud, before turning our attention to alternatives

to centralised, cloud networks. Finally, we present studies in multi-device environments.

2.1 'Privacy is dead' ... in the cloud

Today's Internet is built atop a small set of protocols (notably TCP/IP and HTTP) that are fundamentally broken privacy-wise: they do not support encryption by default, and leak metadata. The problem gets worse when all communications follow a *client-server model* backed by the same cloud companies: all leaked information ends up in the hands of a few omniscient players.

The encryption problem is now tackled by a majority of the web through Transport Layer Security (TLS) [37], or the newer QUIC protocol [38] (that supersedes TCP altogether). However nice, these protocols only provide encryption on the wire, not on the servers' databases. Thanks to recent advances in fully-homomorphic encryption [39] and private information retrieval [40, 41], it is now possible to store encrypted data in the database and compute on it. Still, these approaches' complexity [42], their freshness, and the lack of interest of service providers, make them unlikely to gain mainstream adoption.

Furthermore, encrypting data is not enough. Our protocols always leak the sender and receiver's addresses, along with time, volume, and frequency of transmitted information. In certain cases, like regular browsing, the amount of metadata rises to fabulous proportions: the 'fingerprint' that we leave behind us (comprising browser and OS types and versions, screen size, installed fonts...) has been shown to be unique in most cases [43, 44]. Each user having a specific hardware & software configuration, they have a unique fingerprint. In other words, trackers that follow us around as we browse (e.g. social media sharing buttons and advertising banners) can easily recognise us from one website to another, and build a full history of our online activity. This is enough to build actionable intelligence—as General Michael Hayden, former director of the NSA and the CIA, once told: 'we kill people based on metadata' [45].

This metadata haemorrhage is not going away any time soon, as (i) it is needed by services to operate properly [46, 47] and (ii) the business model of most service providers is precisely to collect and monetize such information [48].

We claim that this data governance problem can only be solved by discarding the client-server model altogether. It is responsible for the 'stupid client against the almighty server' dichotomy. To put power back into our hands, our devices need to become actors of the services we use.

2.2 Today's alternatives

We first present a new trend, that poses as an alternative to the cloud model, namely the edge cloud model, before diving into decentralised networks.

2.2.1 The edge cloud: a poor diversion

The edge cloud computing model leverages small data-centres ('cloudlets') located close to the end-users (at the *edge* of the network) to provide online services. The prime argument is that nascent technologies (e.g. virtual reality, augmented reality, self-driving cars) require both high bandwidth communication and real-time network latencies (below tens of milliseconds) [49, 50]. This feat is hard to achieve with the traditional cloud model, where the data-centres are located close to the Internet backbone, far from the users—which motivates the introduction of cloudlets.

Although edge cloud proponents plead its better privacy properties compared to the cloud model [50–52], we are very doubtful. The privacy question, as always, boils down to 'who controls the data?'. As shown by the high participation of Internet Service Providers (ISPs) in edge computing associations [50], the trend is most appealing to them: ISPs would play a key role in the installation of on-demand computing capacities at the edge. In the end, the mass data harvesting would go on, with only more agents involved in the feast.

Unless the proposal is rephrased, such that the 'edge' refers to the people's devices, collaborating with equipment in close proximity in a decentralised fashion [53], we deem the edge cloud a dead-end for privacy.

2.2.2 Decentralised networks

The privacy problem is certainly not new, and we are not the first to question the client-server model in favour of a decentralised design. Contributions in decentralised systems can be split per topology, from the least decentralised to the most, and this is the approach we take to dig into this field.

Federated Federated networks are communities of peers, each grouped around a server—following the classical client-server model, such that each server can communicate with one another, forming a global decentralised system. Each community's data only resides

on its server, while interconnections between instances are governed by permission rules. Members of a single community only have to trust their server administrator with their data, and nobody owns it all, thus avoiding mass data extortion.

Although not the purest realisation of a decentralised system [1], federated networks have shown their applicability for a long time. Some major protocols follow this design, such as the SMTP mail transfer protocol [54] (first introduced in 1981) or the XMPP communication protocol [55].

Furthermore, federated systems have been getting a lot of attention for social activities lately, notably thanks to the W3C standard ActivityPub [56]. Services like Mastodon (microblogging) [57], PeerTube (video publishing) [58] and NextCloud (file synchronisation and sharing) [59] embraced the standard, birthing an new ecosystem of inter-compatible decentralised applications coined the *fediverse* [60].

Almost decentralised Further down the decentralised rabbit hole, we find proposals that are mostly decentralised, with some centralisation points. Indeed, distributed systems often require trust, such as for authenticating peers (which does not contradict anonymity, but is important to avoid malicious peers posing as other ones) [34], or keeping a registry of online nodes. Such centralisation points simplify the implementation of crucial modules, but constitute a single point of failure: they are easy targets to disrupt the entirety of the network, or to trick its users.

Traditional Public Key Infrastructures (PKI) are a notorious example, notably the ubiquitous TLS protocol [37]. To authenticate peers, TLS employs digital certificates delivered by trusted third-party Certificate Authorities (CA), that vouch for the authenticity of a peer's identity. The TLS CAs are prone to man-in-the-middle attacks: given that the attacker has a trusted certificate of their own (or stole one of the CAs' certificate) and can intercept traffic [62], they can impersonate the correspondent and read all client traffic. State agencies have been caught doing just that, at the Internet scale [63]; enterprises can buy dedicated boxes to spy on their employees in the same way [62, 64]. PGP's web of trust is another example of PKI, although less centralised: people accumulate keys of people they trust, and vouch for them, thus creating chains of trust resembling a social graph, or a web.

Other examples of semi-decentralised networks include mix networks [65] and onion

1. We shall remember the words of William of Baskerville, however, and appreciate even the impure solutions. In Umberto Eco's *The name of the rose* [61], he said to be terrified of the purists, and above all, of their haste.

routing [66], two prevalent anonymity system architectures. Tor [67], to this day the most used onion routing network, trusts 10 directory authorities (DAs) to provide an hourly consensus containing the list of online onion relays. These relays are then used by clients to build anonymizing onion routes passing through three of them. The DAs are well known, surveyed, and do not trust each other; still, the entirety of the network relies on their consensus.

Finally, the BitTorrent peer-to-peer (P2P) file sharing network [68] outsources the bookkeeping of online peers sharing a particular file to a centralised tracker, which leaks all of the users' addresses publicly.

Fully decentralised Purely decentralised networks eschew any central point of synchrony, making them more resilient to attacks or failures. On the other hand, such systems still need trust, e.g. for peer discovery and authentication, but it needs to be built without a trusted authority.

A compelling building block for storing prime importance data is to resort to Distributed Hash Tables (DHT) like Kademlia [69]. A DHT provides a decentralised way of redundantly storing simple <key, value> items among a set of peers, such that each node only stores a small fraction of the whole data. Peers can add new <key value> pairs, query for content given a key, and, when they do not know of a certain key, they can forward requests until the querier finds their content. Alas, DHTs do not anonymise their peers' queries, and are susceptible to attacks [70, 71]. Solutions have been proposed [70, 72], although they mostly tackle individual security problems. In the end, DHTs are a usable tool, but its security properties depend on their implementations. Still, they have proved to be useful in practice: BitTorrent uses them to eschew trackers and better resist censorship [73]; Tor employs a DHT for hidden services discovery [67]; the IPFS [74] and Dat [75], two content sharing P2P networks, also employ them for peer and content discovery.

Another solution is to resort to gossip messaging [76, 77], where information is shared among peers like a rumour, until everyone knows about it. Gossip protocols ensure, with an overwhelming probability, that a piece of information will reach every online peer at a bounded network cost. Unlike DHTs, they are more adapted to fast propagation than storage and queries. An interesting development for node discovery is Random Peer Sampling (RPS) [78, 79], where each peer maintains a constantly changing random pool of online neighbours' addresses. The result is a *dynamic*, *complete* and *random* graph

of neighbouring relationships, without the need for maintaining a DHT. As with the DHTs, gossip algorithms are sensitive to attacks, although good contributions tackle these issues for the RPS case [80–82]. Interesting services leveraging gossip protocols include the Scuttlebutt P2P social networking platform [83–85], and GNUnet, a new protocol stack for building 'secure, distributed, and privacy-preserving applications' [48].

The Bitcoin P2P cash system [86], after few years of disbelief, caused a revolution in societies' conceptions of currency and in the distributed system community. Its foundation, the blockchain, is a decentralised append-only ledger, offering users with two operations: to read any block, and to append a block to the chain. The originality lies in the fact that an *unbound* number of peers agree upon the content and order of the blocks despite the presence of malicious nodes. Blocks being sequentially linked, the next block depends on all previous ones, ensuring that the full history can not be tampered with. This resilience constitutes a major improvement from the previous state of the art. Many proposals used a blockchain as the base of their trust model [87, 88]. Ethereum [89] stores versatile smart contracts, i.e. arbitrary code, in its blockchain, which shows the applicability of the blockchain beyond cash systems.

Lastly, an interesting avenue for building trust in a fully decentralised network is to leverage the Trusted Execution Environment (TEE) that comes with most recent processors (e.g. Intel's Software Guard eXtensions (SGX) [90] or ARM's TrustZone). They are co-processors that work only on encrypted memory, and provide verifiable operation, although at an expensive computing cost. Kim et al [91] envisioned to put every Tor relay inside SGX, which would ensure their validity and hide the relays registry inside the TEE's encrypted memory. Broader contributions tackle e.g. the problem of encrypting content for a group of peers [92]. A caveat is that each TEE needs to be authenticated to be trusted, using a PKI infrastructure, which is centralised. To take the example of Intel SGX, one can eschew trust in their distributed system by using SGX enclaves; instead, everyone would need to trust Intel's authentication services.

We see that literature and implementations abound on distributed systems. Only one question remains: *why aren't they mainstream yet?* Researchers do tackle the question [34, 93], and dig out many open research questions. However, it seems that the main reasons for the lack of adoption of distributed systems remain social and political. The recent shift of George Danezis—from publishing relevant research on anonymity systems [34, 40, 62, 94, 95] to building Facebook's blockchain, Libra [96]—might be a clue [97].

Nevertheless, the *fediverse* trendiness highlights the growing public interest for privacy

and decentralisation. As we witness the rise of the IoT, its inter-connectivity and empowerment promises, but also the greater threats it might pose the the people's privacy, we are convinced about the necessity to build decentralised, privacy-preserving applications using these very devices.

2.3 The multi-device paradigm

Despite the growing trend of IoT products and research, to the best of our knowledge we are the first to propose to network one's devices in order to build new services. Most contributions nowadays take interest in sensor networks [98], data processing, trading energy [99]...; we are most interested in leveraging user-facing devices for new applications. We will thus focus our bibliographic report on studying the people's usage of their devices.

As early as 2009, Karlson et al. [100] showed that the phone was already seen by Microsoft researchers as their always-on primary computer, notably used to access and triage professional communications. They studied both professional computer and smartphone usage patterns, demonstrating various user preferences. An interesting fact is the omnipresence of the phone (most often used before and after the PC), while the latter would be offline most of the day when not intensively used. Most users did not take interest in doing cross-device tasks at the time; they were happy with the data synchrony between their appliances.

In 2012, Google led a study specifically targeted at understanding 'multi-screen' interactions, encompassing smartphone, tablet, PC and TV [101]. They were the first to observe simultaneous usage of these devices, for multi-tasking or complementary usage. In 2013, Kawsar et al. [102] acknowledge the shift from desktop -centric networking to an 'activity first, device next' trend. Newer studies, from 2015, have compared smartphone and tablet usage [103] or the multi-device paradigm specifically [104]. They showed that multi-device usage was still mostly sequential (maybe due to impractical software design for complementary use), and sometimes problematic due to lack of data synchrony or incompatible formats. Jokela et al. [104] report that users would appreciate more interaction between their devices, notably by using direct communication in spite of the cloud.

Wagner et al. developed Device Analyzer [105], a data collection app to investigate mobile usage and context. In 2013, they presented their findings: a wide disparity of network connectivity (10% of the participants having no signal 40% of the time, while half the users have no signal only 5% of the time), and an impressive median uptime of

92.4% (when out of battery, the phone is plugged to a charger within 12 hours in 90% of the cases). In other words, the mobile is certainly our most reliably connected device as of today.

We should also acknowledge that tablets have been loosing market shares continuously since 2015 [106], despite previous claims that it would replace PCs, and maybe in part due to the increase of smartphones' sizes, that nowadays reach 6 inches [107].

The literature on the multi-device paradigm is still scarce, and mostly takes interest in the way users interact with their devices. It shows that our device usage is transitioning to task-driven interactions that can span several devices (e.g. browsing for goods on a smartphone before buying on a computer). Still, using devices in collaboration remains an ad-hoc, niche process. New proposals need to be made to put collaboration at the core of our multi-device interactions.

2.4 The path toward e-squads

To enable the e-squad paradigm, we must answer a number of research questions that are still opened to debate.

From lonely devices to the e-squad We defined the e-squad as a set of appliances rendered *intelligent* through device-to-device communication. By learning information on itself and the user's behaviour, an e-squad can understand its dynamics and make inferences: What task is the user trying to accomplish? Which devices are the most likely to remain online in the following hours? Will they turn on their laptop anytime soon? This knowledge paves the way to smart services, that will, e.g., transfer a computation from a disconnecting device to another, or forward back-ups to the most connected device to keep them at the user's disposal.

Each device knows a little bit about their user's behaviour: a user's workstation activity is directly tied to their working hours, the smart-TV to their leisure time, and smartphones can give insights on their user's location and activity. Our first challenge is to aggregate this local information into global knowledge in a decentralised way. Then, the e-squad needs to turn this knowledge into actionable intelligence, for services to make smart decisions.

Knowledge is dependability As shown in the previous section, personal devices are intermittently available. Most appliances remain active only for the duration of their usage, while mobile appliances undergo frequent connectivity variations or interruptions. This state of affairs contrasts drastically with the cloud paradigm, which goal is to provide always-on computing resources for service providers. We won't ask our users to keep their devices permanently online, but the services hosted by the e-squads still need to provide an impeccable quality of service at a reasonable resource cost, for the user's experience to remain delightful.

Our second challenge is to use the e-squad's knowledge to build dependable services despite the underlying infrastructure's transient connectivity.

Privacy-preserving e-squad federations The Internet is notably made for connecting people, and e-squads would not go far if they only enabled single-user services. We want to propose multi-user applications by federating each person's e-squad into a broader distributed network. However, the sole online presence of one's devices already leaks information about them.

Our last challenge is to build robust multi-user services that preserve each person's privacy. On one hand, the more information published by each e-squad about, e.g., their availability, the more dependable the service can be; on the other hand, the published data could enable by-standers to learn private information about an e-squad owner. We need to assert the privacy implications of publishing device-related information, and find solutions to keep the security risk as low as possible.

We have seen that the centralisation of the Internet is questioned by many different actors of the informatics community. Proposals in distributed systems notably abound, solving one hard problem at a time. The emergence of the IoT, and the consequent multiplication of devices per users, have been duly noted. Yet, literature is scarce when it comes to allowing one's connected appliances to collaborate into a personal, smart network. Our proposal, the e-squad paradigm, could help bridging the gap between the IoT and distributed network communities. We are not there yet, as several research questions need to be addressed before the e-squad model becomes a reality. In the remainder of this thesis, we propose several solutions to these problems, mere delineations of the work yet to come. Still, we hope that our contributions will appeal to the reader's mind, and make them long for a more decentralised, humane, and shiny connected future.

Chapter 3

Fluid user interactions inside the e-squad

For starters, we will present Sprinkler. It is an example of a service handled by a set of a user's devices, transformed into an intelligent e-squad thanks to gossip communication. The problem solved by this service is fairly ubiquitous: to enable continuous interactions across one's devices.

At this point, we do not tackle the reliability of delivered services and make the unrealistic assumption that one's devices are always online. Dependability despite devices' disconnections is the main topic of the next chapter.

3.1 Introducing Sprinkler

Our modern societies have us surrounded by an ecosystem of connected devices, and each of them still behaves as if it were alone. Michal Levin, a UX designer at Google, stated three design principles that can guide application developers to build applications targeting the ecosystem, not single devices. The principles are coined the *3Cs*, for Consistency, Continuity, and Complementarity [108]. *Consistency* is most fundamental to propose a delightful experience: a consistent application has the same design, aesthetics, and core functionality whatever device it runs on. Specific features may be added according to each device's capabilities. Follows *continuous* design: the user's activity should be able to continue or progress as they switch devices, without any noticeable interruption. The Grail is the *complementary* design, when devices complement each other into helping the user achieve their goals, as a connected group.

The consistency problem is being solved, thanks to web technologies and cross-platform building frameworks like Cordova, NativeScript and many more [109–115]. Continuity is still a day-to-day concern (who does not type tens of passwords a day?), although many applications do propose *session handoff* solutions (the fact of transferring an applica-

tion's state from one device to another). All major web browsers feature a 'sync' option; online note-taking applications are legion (Google Keep, Evernote, Microsoft OneNote, Simplenote, you name it); and file synchronization services like DropBox are life-savers as stopgaps for continuous interaction. Nevertheless, these solutions are harmful snake oils, as they all resort to third-party cloud providers to store highly sensitive personal data: the entire state of one's online activity.

The multi-device ecosystem must free their users from cloud silos. To this end, we need solutions for applicative session handoff among the e-squad. Some academic proposals tackle P2P application state sharing [116–118], while a few industrial applications propose P2P file synchronisation [119–121]. Alas, all these proposals boil down to flooding each e-squad device with each new session, to guarantee that it will be accessible where needed. Such a brute-force broadcast mechanism, despite its simplicity, leads to very poor performances as it implies: (i) redundant messages, (ii) overconsumption of network bandwidth, (iii) a higher latency that inherently impacts the fluidity of user interaction, and (iv) faster energy depletion. Further, it does not scale well with the number of devices.

In this chapter, we introduce Sprinkler, a novel approach to perform predictive session handoff by learning in a *decentralized* manner how the user behaves. Sprinkler combines a probabilistic dissemination protocol and a proactive session handoff mechanism in order to provide seamlessly fluid user interactions among one's e-squad. Our solution: (i) does not rely on any centralization point, (ii) has a bounded and known network resource consumption, (iii) is able to predict which device is the most likely to be used next, enabling the transfer of the ongoing interaction session without blind flooding, (iv) respects the user's right to privacy, and (v) scales to an arbitrary number of devices.

Our contributions are as follows:

— We have designed Sprinkler, a protocol based on two algorithms: the Sprinkler *Gossiper*, which allows devices to gain knowledge of the user's behaviour in a distributed manner; and the Sprinkler *Session Handoff* mechanism, which uses this knowledge to proactively send chunks of the current session to devices that will most probably be used next;
— We have evaluated our approach with 8 different discrete time Markov models to emulate user behaviours;
— We have demonstrated that there is no ideal dissemination protocol. It is all about the trade-off between prediction accuracy, latency, and network consumption. We show in particular that the performance of Sprinkler greatly depends on the

user's behaviour. The more predictable a user is, the better SPRINKLER performs;
— SPRINKLER allows designers to efficiently trade off network costs for fluidity, and is for instance able to reduce network costs by up to 80% against a flooding strategy while maintaining a fluid user experience.

In the rest of this chapter, we present SPRINKLER's concepts and approach in section 3.2, and evaluate its performance in section 3.3.

3.2 Our approach

Our goal is to provide a communication protocol such that every time the user (that we call Alice) opens one of her devices, she finds her Web applications as she left them, regardless of the device she has used previously. Because the day-to-day usage information of one's appliances is a private asset, we wish to avoid relying on an external storage system, which would threaten the user's right to privacy [8]. To this end, our protocol should only involve Alice's devices, by leveraging distributed communication strategies. Secondly, we want our protocol to be lightweight, to avoid draining power from the mobile appliances running it.

The state of the user's applications is stored in a blob, arbitrarily heavy in size, thereafter called a *session*. Our protocol, SPRINKLER, *proactively* shares portions (*chunks*) of the user's session whenever she leaves a device, by *predicting* the device she will use next. This prediction is achieved by letting devices learn the user's behaviour by gossiping information among themselves. We wish to minimize, on one hand, the time Alice has to wait for her previous session to be fetched when she opens a new device, and on the other hand, the network traffic induced by session exchanges.

Towards this aim, our protocol is constituted of two algorithms: the SPRINKLER *Gossiper* will allow devices to gain knowledge of the user's behaviour; the SPRINKLER *Session Handoff* algorithm will use this information to proactively send chunks of the current session to devices that will most probably be used next.

3.2.1 Solution outline

We consider a user, Alice, who uses a number of N devices, $\mathcal{D} = \{d_1, ..., d_N\}$ (her e-squad). We can model Alice's use of her devices as a sequence of *interactions*: $S = \{r_1, r_2, ..., r_i, ...\}$. Each interaction r_i is characterized by a pair $(d_{r_i}, t_{r_i}) \in \mathcal{D} \times \mathbb{R}$, which

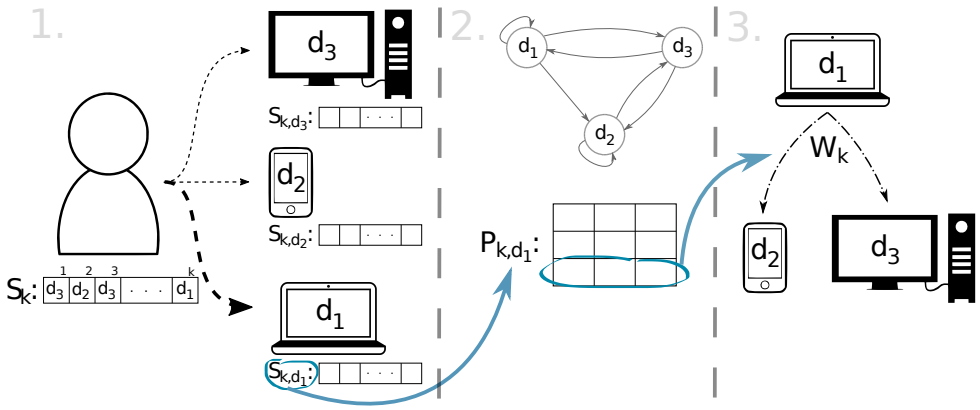


Figure 3.1 – Pipeline of the Session Handoff: from user behaviour to model inference to sharing of the session data. 1. The user uses d_1; 2. d_1 uses its local sequence S_{k,d_1} to compute a Markov chain governed by probabilities P_{k,d_1}; 3. when the user leaves d_1, it sends chunks of their session to other devices by computing W_k, the amount of data to send to each device, based on the Markov chain.

means that Alice started using the device d_{r_i} at time t_{r_i} (and stopped using it before $t_{r_{i+1}}$). We assume that Alice only uses one device at a time, such that no two interactions share the same timestamp.

For the purpose of our experiments, we consider that Alice's devices have access to a synchronized physical clock, creating a total order on the sequence. The same total order could be obtained with logical clocks, e.g. Lamport timestamps [122], since interactions are never concurrent.

Our Session Handoff procedure is illustrated in fig. 3.1.

Propagating the user's behaviour From a global point of view, the sequence S_k contains all the k interactions performed by the user since the beginning of the program's execution. Locally, however, each device d initially only knows about the sequence $S_k|_d$ of interactions that took place on it, that is:

$$S_k|_d = \{r \in S_k,\ r{\downarrow}\mathcal{D} = d\} \implies S_k = \bigcup_{d \in \mathcal{D}} S_k|_d,$$

where $r{\downarrow}\mathcal{D}$ represents the projection of interaction r on the set of devices, i.e., the device on which r took place.

To gain knowledge on the user's behaviour, the devices gossip information about interactions among themselves. This way, at step k, every device d knows a (possibly in-

complete) local sequence $S_{k,d}$ of the user's actions, such that:

$$S_k|_d \subseteq S_{k,d} \subseteq S_k.$$

Inferring a probabilistic model An ordered sequence S of interactions can be used to compute a discrete time Markov chain, representing the probability that the user swaps from a device to another, for each pair of devices in $\mathcal{D}$. To do so, we firstly need to compute the matrix of transition counts $C = (c_{d_i,d_j})_{(d_i,d_j)\in\mathcal{D}^2}$ between devices:

$$c_{d_i,d_j} = \left|\{r_t, r_{t+1} \in S,\ (r_t{\downarrow}\mathcal{D}) = d_i \wedge (r_{t+1}{\downarrow}\mathcal{D}) = d_j\}\right|.$$

C captures the number of times Alice switches between each pair of devices (d_i, d_j) in S. From C, we can derive the matrix of transition probabilities $P = (p_{d_i,d_j})_{(d_i,d_j)\in\mathcal{D}^2}$, (i.e. the weights of the Markov chain's edges):

$$p_{d_i,d_j} = \frac{c_{d_i,d_j}}{\sum_{d_k\in\mathcal{D}} c_{d_i,d_k}} = \mathrm{P}\left[d_i \to d_j\right],$$

where $d_i \to d_j$ means "Alice uses the device d_j right after d_i". p_{d_i,d_j} thus represents the probability that Alice switches from d_i to d_j, according to the sequence of the user's interactions, S. We call Ψ the operation of generating a Markov transition matrix P from a sequence S: $P = \Psi(S)$.

A device d_{curr} that is currently being used by Alice at step k (i.e. d_1 in fig. 3.1) uses its local sequence $S_{k,d_{\text{curr}}}$ to compute the transition matrix $P_{k,d_{\text{curr}}} = \Psi\left(S_{k,d_{\text{curr}}}\right)$. This provides d_{curr} with the transition vector $\mathbf{p}_k$ that contains, for each $d \in \mathcal{D}$, the probability that the user will switch from d_{curr} to d:

$$\mathbf{p}_k = P_{k,d_{\text{curr}}}\left(d_{\text{curr}}, *\right) = \left(\mathrm{P}\left[d_{\text{curr}} \to d \mid S_{k,d_{\text{curr}}}\right]\right)_{d\in\mathcal{D}}. \tag{3.1}$$

Note that $\mathbf{p}_k\left(d_{\text{curr}}\right)$ is usually not null: the user sometimes switches back to the same device.

Performing Session Handoff After Alice closes her current device d_{curr}, we want to proactively send the blob containing her application state—her *session*—to the next device she will use: d_{next}.

Throughout the rest of this chapter, we assume that every session weighs w_{sess} bytes. Because d_{curr} cannot be sure of which device will be used next, it sends portions of its

session to several selected peers. We call these portions *session chunks*, and assume that they can weigh any size from 0 to w_{sess}. Finally, we define the set $\mathcal{D}'$ of devices to which d_{curr} can potentially send session chunks: $\mathcal{D}' = \mathcal{D} \setminus \{d_{\text{curr}}\}$.

d_{curr} sends $w_{k,d} \in [0, w_{\text{sess}}]$ bytes of the session to each device $d \in \mathcal{D}'$, resulting in the vector W_k of all data chunks sent by d_{curr} at step k:

$$W_k = (w_{k,d} \in [0, w_{\text{sess}}])_{d \in \mathcal{D}'} = f(\mathbf{p}_k). \tag{3.2}$$

The performance of the Session Handoff depends on the accuracy of $\mathbf{p}_k$, which depends on the local sequence of d_{curr}, $S_{k,d_{\text{curr}}}$. In the following, we first present Sprinkler Gossiper, which reliably propagates a user's sequence of interactions to all devices, before discussing Sprinkler Session Handoff, which handles the proactive migration of a user's session.

3.2.2 Decentralized knowledge aggregation

Initially, a device can only observe interactions taking place locally. In order to predict a user's future behaviour, the e-squad's devices must, however, gain a global overview of the user's past behaviour, and thus aggregate their local knowledge into a global interaction sequence. We propose to perform this aggregation with a probabilistic dissemination protocol [76, 123, 124] that we have called Sprinkler Gossiper.

Intuition

The Gossiper implements a *reactive* and *incremental* aggregation that involves all of a user's devices. The protocol is invoked every time the user (say Alice) leaves a device to move to another one, signaling the end of an interaction, and the start of a new one. Sprinkler Gossiper is *gossip-based*, i.e., it uses randomized message exchanges between devices to propagate the sequence of interactions performed by Alice. This randomized approach makes our protocol both *lightweight* and *robust*, two properties that are central to decentralized session handoff. We use a *push-pull* strategy [79] to propagate information, i.e., when a device p contacts a device q, p sends new information to q (*push*), but also requests any new information q might have (*pull*).

In order to avoid redundant communication rounds, the Gossiper further keeps track of each device's local perception of other device's knowledge using a mechanism inspired from vector clocks [125] combined with incremental diffs.

Algorithm

Table 3.1 – Variables & parameters of Sprinkler

	Variables maintained by the device p belonging to Alice
S_p	p's knowledge of Alice's interaction sequence, i.e. its *local sequence*. S_p is initialized with a small number of core devices (possibly only one) that Alice uses regularly.
$RV_p[\cdot]$	$RV_p[q]$ contains p's idea of what is known to device q. Initially $RV_p[q] = \varnothing$ for all $q \in \mathcal{D} \setminus \{p\}$.
	Parameters of the algorithm
f	The *fanout* of the probabilistic broadcast, that is the number of devices that each device communicates new information with.

The algorithm of Sprinkler Gossiper is shown in figures 3.2 and 3.3. To ease our explanation, the *request* part of the push/pull exchange is shown in fig. 3.2 from the point of view of p, while the *reply* part is shown in fig. 3.3 from the point of view of q. (All devices execute both parts in practice.) We assume that p and q are owned by Alice, and that she is currently using device p. p's current knowledge of Alice's sequence of interactions is stored in variable S_p, while the array $RV_p[\cdot]$ stores p's *remote view* of other devices' knowledge of Alice's sequence (Table 3.1).

The algorithm starts when Alice begins a new interaction by opening device p (line 1). The algorithm inserts a new interaction record $\langle p, \text{timestamp} \rangle$ into p's local interaction view S_p (lines 3-4), and launches the probabilistic dissemination, implemented in gossipUpdate().

gossipUpdate() first selects a small random set of f other devices from p's local sequence S_p (lines 6-8). As a consequence, the only devices that participate in Sprinkler are the ones that Alice already used at least once.

These random devices are selected from S_p, i.e., the sequence of interactions already learned by p, from which we exclude p and the most recent device found in S_p (which is likely to be up to date). p initiates a push/pull exchange with each selected peer q which is not known to know at least as much as p (lines 9-13). This *knowledge check* is performed at lines 10-11, using $RV_p[q]$, p's idea of the interactions that are known to q. By construction, and in the absence of communication faults, $RV_p[q]$ underestimates q's

1: **on event** Alice opens device p
2: $r \leftarrow \langle p, \text{timestamp} \rangle$ ▷ New interaction r
3: $S_p \leftarrow S_p \cup \{r\}$ ▷ Updating p's local view
4: GOSSIPUPDATE() ▷ Triggering the dissemination

5: **function** GOSSIPUPDATE($exclude[= \varnothing]$)
6: $last\_device \leftarrow$ most recent device in S_p
7: $exclude \leftarrow exclude \cup \{p, last\_device\}$
8: $peers \leftarrow f$ devices from {devices from S_p} $\setminus$ $exclude$
9: **for** $q \in peers$ **do** ▷ Looping through random peers
10: $S_{\text{diffpush}} \leftarrow S_p \setminus RV_p[q]$
11: **if** $|S_{\text{diffpush}}| > 0$ **then** ▷ Only sending new data
12: **send** $\langle \text{REQ} : S_{\text{diffpush}} \rangle$ to q ▷ Push/pull to q
13: $RV_p[q] \leftarrow RV_p[q] \cup S_{\text{diffpush}}$ ▷ Tracking q

14: **on receive** $\langle \text{ANS} : S_{\text{diffpull}} \rangle$ from q: ▷ Pull reply
15: $S_p \leftarrow S_p \cup S_{\text{diffpull}}$
16: $RV_p[q] \leftarrow RV_p[q] \cup S_{\text{diffpull}}$

Figure 3.2 – SPRINKLER's push/pull request (on device p)

17: **on receive** $\langle \text{REQ} : S_{\text{diffpush}} \rangle$ from p: ▷ Push/pull request
18: $RV_q[p] \leftarrow RV_q[p] \cup S_{\text{diffpush}}$ ▷ Tracking p
19: **if** $S_{\text{diffpush}} \nsubseteq S_q$ **then** ▷ Is S_{diffpush} new?
20: $S_q \leftarrow S_q \cup S_{\text{diffpush}}$
21: GOSSIPUPDATE($\{p\}$) ▷ Propagating new data
22: $S_{\text{diffpull}} \leftarrow S_q \setminus RV_q[p]$
23: **if** $|S_{\text{diffpull}}| > 0$ **then** ▷ Anything new for p?
24: **send** $\langle \text{ANS} : S_{\text{diffpull}} \rangle$ to p ▷ Answering pull
25: $RV_q[p] \leftarrow RV_q[p] \cup S_{\text{diffpull}}$

Figure 3.3 – SPRINKLER's push/pull reply (on device q)

actual knowledge (i.e., $RV_p[q] \subseteq S_q$) [1], which means no new information is missed.

The **send** operation at line 12 starts the actual push/pull exchange with q: the incremental update S_{diffpush} is sent to q. On receiving S_{diffpush} (line 17, fig. 3.3), q first processes p's incremental update (lines 18-21), by *(i)* adding it to q's idea of p's view (line 18), *(ii)* updating its own view if needed (line 20), and *(iii)* launching a cascading dissemination in case the diff contains information new to q. The condition at line 19 ensures the dissemination eventually stops, as increments are never gossiped twice by the same device.

1. This is because any interactions added to $RV_p[q]$ by p have either been sent to q (lines 13 and 25) or received from q (line 16).

$\{p\}$ is passed as the *exclude* parameter to GOSSIPUPDATE() to increase the probability of hitting uninformed devices.

Lines 22-25 implement q's reply to p's pull request. Again, q only replies to p if it might possess new information (test on line 23), in order to reduce communication. The (possible) reply from q to p is processed by p at lines 14-16 (fig. 3.2).

Bootstrap and reliability of Sprinkler Gossiper

Because a device p only gossips with other devices found in its sequence view S_p, p's view needs to be initialized to a default value of a few core devices regularly used by Alice e.g. $\{\langle a, 0\rangle, \langle b, 0\rangle\}$, where 0 is an arbitrary bootstrapping timestamp. The use of S_p as a source of gossiping candidates prevents devices not used by Alice to be involved in the protocol. When a new device is used by Alice, on the other hand, it propagates updates containing at least itself (lines 2-3), and automatically becomes known to the rest of the system.

The overall reliability of the gossip diffusion is governed by its fanout coefficient f (which appears at line 8). In a reliable network, a fanout f slightly over $\log(N)$ ensures that all devices will be reached with a very high probability [126]. In practice we therefore use $f = \lceil \log(N) \rceil$.

Although we assume a reliable network with always-on devices, transient communication failures might occur. When this happens, the protocol might temporarily fail to reach all nodes, but full propagation will resume when the network recovers. While the network is degraded, $RV_p[q]$ might diverge, and might contain information not included in S_q. This is because we do not insure that the message $\langle \text{REQ} : S_{\text{diffpush}} \rangle$ sent at line 12 is successfully received by q before modifying p's remote view $RV_p[q]$. In most situations, however, other nodes will provide q with the missed information when the network recovers. This choice favors communication lightness over reliability, but turns out to work well in practice (as shown in Section 3.3.2).

3.2.3 The Session Handoff algorithm

Fig. 3.4 shows the different steps undertaken by SPRINKLER Session Handoff when Alice moves from device d_{curr} (her mobile phone here) to another device d_{next} (her laptop) between interactions r_k (on device d_{curr}) and r_{k+1} (on device d_{next}).

When Alice leaves her mobile phone (d_{curr}, label ❶) at the end of interaction r_k,

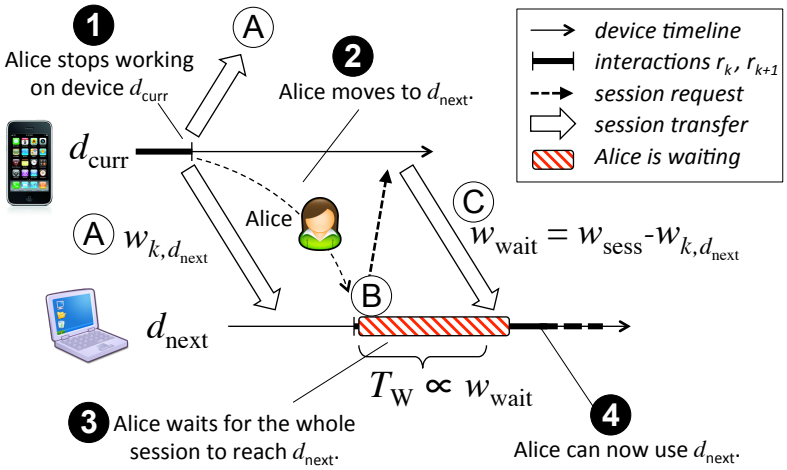


Figure 3.4 – Timeline of the session handoff from device d_{curr} to d_{next}.

Sprinkler proactively sends a *partial session state* to her other devices (label Ⓐ). (We assume here that we can detect the end of an interaction, e.g., using some activity recognition mechanism.) The amount of state each device receives, W_k, is decided using the user behavioural model constructed so far by Sprinkler Gossiper. (We detail this below.) We note $w_{k,d}$ the amount of session state proactively received by device d at the end of interaction r_k. In fig. 3.4, Alice's laptop proactively receives $w_{k,d_{\text{next}}}$ of Alice's session on her mobile phone.

When Alice reaches her laptop ❷, Sprinkler Session Handoff reactively requests the remainder of the session state that has not reached the laptop yet Ⓑ[1]. While the remaining session state $w_{\text{wait}} = w_{\text{sess}} - w_{k,d_{\text{next}}}$ is downloaded from d_{curr}, Alice must wait (❸, hashed bar on the laptop timeline) until she can finally use her device at ❹. Assuming latency is negligible compared to the download time of w_{wait}, the waiting time of Alice T_W is proportional to w_{wait}.

The perfect session handoff algorithm would always provide its user with the last state of her applications when she opens a device, whichever of her appliances she has previously used, and without any waiting time. In addition, given that at least some of her devices are mobile assets with limited resources, this algorithm should consume no more than the

1. In order to fetch this remaining state, d_{next} must know the address of d_{curr}. It can be achieved by several means: if d_{curr} sent chunks of the last session to d_{next} (which is not the case when $w_{k,d_{\text{next}}} = 0$) or by looking at the penultimate interaction in $S_{k+1,d_{\text{next}}}$ (though d_{next}'s local sequence can be incomplete). In addition, when d_{next} wrongly believes that a given device d is the device used at interaction r_k and asks it the last session, d uses its own knowledge to redirect d_{next} to the right device, d_{curr}. If d_{next} was still unable to locate d_{curr}, the Session Handoff would fail completely: Alice's previous session would never be loaded on d_{next}. Consequently, we evaluate a reactive handoff in Section 3.3.2.

bare minimum: it would have sent the application session once, from the device Alice just quit to the one she is about to use.

Such an algorithm is not feasible, since no one can predict the future. Instead, the current device d_{curr} infers which devices are most likely to be used next, to *proactively* send them chunks of the session's blob when the user quits d_{curr}. If d_{curr} sends more session chunks to the other devices, the waiting time T_W will lower at the cost of higher network traffic. On the other hand, if d_{curr} does not send any of the current's session state, the waiting cost will be maximal, as d_{next} will have to reactively fetch the entire session when she opens it. We call this *network cost* C_N. There is a clear trade-off between the waiting time T_W and the network traffic C_N.

Formulating the handoff cost

Thanks to the Gossiper, d_{curr} knows a subset of the user sequence of interactions $S_{k,d_{\text{curr}}}$. By computing the transition vector $\mathbf{p}_k$ (see Equation 3.1) out of $S_{k,d_{\text{curr}}}$, the Session Handoff algorithm outputs the data vector W_k (see Equation 3.2), that contains the amount of bytes of the session to send to each other device.

We formulate the waiting time T_W and the network cost C_N described earlier:

— $T_W \propto w_{\text{wait}} = w_{\text{sess}} - w_{k,d_{\text{next}}}$: The waiting time T_W is proportional to the quantity of the current session that the next device d_{next} still needs to download. In the particular case where $d_{\text{curr}} = d_{\text{next}}$, the session is already fully on the device, leading to a null waiting cost $T_W = 0$;

— $C_N = \sum_{d \in \mathcal{D}'} w_{k,d}$: the network cost C_N is the sum of all the session chunks proactively sent from d_{curr} to the other appliances in $\mathcal{D}'$.

Computing W_k

The goal of the Session Handoff algorithm is to figure out the best vector of sent data W_k based on the transition vector $\mathbf{p}_k$. d_{curr} wants to minimize T_W given that it will send a total of C_N bytes of the session to its peers. We introduce the parameter γ, that controls the amount of data the device d_{curr} will proactively send to the other appliances: it will try to send $\gamma * w_{\text{sess}}$ bytes.

We propose two different solutions for the calculus of W_k:

— **Uniform**: As a baseline, our first solution is to send the same quantity of the session to each device in $\mathcal{D}'$, regardless of $\mathbf{p}_k$:

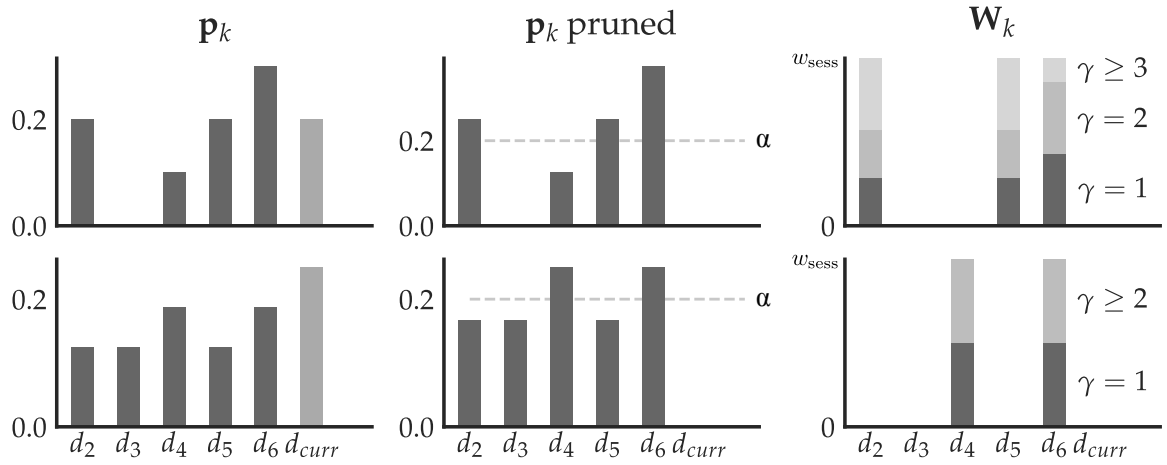


Figure 3.5 – Two examples (one per line) of *proportional* computations of W_k given different $\mathbf{p}_k$, with $N = 6$ devices. $\mathbf{p}_k$ is shown on the leftmost column. On the centre column, $\mathbf{p}_k$ is normalized after $p_{d_{\text{curr}},d_{\text{curr}}}$ is pruned from it; the threshold α is displayed as well. On the rightmost column, we see W_k, the amount of data that would proactively be sent to other devices, for different values of parameter γ (see equations 3.3 and 3.4).

$$\forall d \in \mathcal{D}', w_{k,d} = w_{\text{sess}} * \frac{\gamma}{N-1}.$$

— **Proportional**: Our second solution is to make W_k proportional to $\mathbf{p}_k$. This way, devices having a high probability of being used next will naturally receive a bigger portion of the user's session. The most simple computation would be the following:

$$\forall d \in \mathcal{D}', w_{k,d} = w_{\text{sess}} * \min(\gamma * \mathbf{p}_k(d), 1).$$

However, a device d_{low} may have a very low probability $\mathbf{p}_k(d_{\text{low}})$ of being chosen. The preceding calculus would result in a negligible $w_{k,d_{\text{low}}}$ compared to the cost of the network exchange. Therefore, we compute a new transition vector $\mathbf{p'}_k$, such that any probability inferior to a certain threshold α is null. In practice, we arbitrarily set $\alpha = 1/(N-1)$. We introduce a variable β to ensure that $\mathbf{p'}_k$ sums to one. It is computed as follows:

$$\beta = \frac{1}{\sum\limits_{\substack{d \in \mathcal{D}' \\ \mathbf{p}_k(d) > \alpha}} \mathbf{p}_k(d)};$$

$$\forall d \in \mathcal{D}', \quad \mathbf{p'}_k(d) = \begin{cases} 0 & \text{if } \mathbf{p}_k(d) \leq \alpha \\ \beta * \mathbf{p}_k(d) & \text{else} \end{cases}. \tag{3.3}$$

The vector of sent data is then simply proportional to this new pruned vector of probabilities:

$$\forall d \in \mathcal{D}', w_{k,d} = w_{\text{sess}} * \min(\gamma * \mathbf{p'}_k(d), 1). \tag{3.4}$$

This way, we send session chunks only to devices that have a high enough probability of being used. See figure 3.5 for a graphical depiction of the proportional computation of W_k.

We have proposed two solutions to dispatch session chunks. The first (uniform) will give us a comparison point to observe the influence of the behavioural knowledge inferred with Sprinkler Gossiper (in the proportional approach). The parameter γ will allow us to tune the overall quantity of data that d_{curr} will send to its peers: from $\gamma = 0$, that solely relies on reactive handoff, to a maximal γ, that consists of sending the entirety of the session to each device having a non-null probability of being used next.

3.3 Evaluation

In the following section, we make a thorough evaluation of the Sprinkler protocol. We start with presenting our evaluation process in section 3.3.1, before evaluating Sprinkler in detail in section 3.3.2.

3.3.1 Testbed

In our futuristic scenario, we imagine a user controlling a dozen of devices sequentially. To the best of our knowledge, there exists no real-world dataset describing a user's behaviour among that many devices. We thus evaluate Sprinkler contributions by launching several virtual devices (implemented in Go) used by an emulated user (in Python). The devices are packed into Docker containers, and all run inside the same IP network. In

the following, we first propose several behavioural models that emulate a fictitious user's activity, before presenting our experimental setup.

Proposed user behaviour models

In Section 3.2.1, we have represented a user's behaviour as a growing sequence S of interactions with her appliances. To emulate these interactions, we propose to use a number of discrete-time Markov models $\mathcal{M}$, from which we then generate the sequences of interactions driving the evaluation of our contributions. Given the set $\mathcal{D} = \{d_1, \ldots, d_N\}$ of a user's devices, a discrete-time Markov user model $\mathcal{P}_\mathcal{M}$ takes the form:

$$\mathcal{P}_\mathcal{M} = \left(p_{d_i,d_j}\right)_{(d_i,d_j)\in\mathcal{D}^2} \quad \text{s.t.} \quad p_{d_i,d_j} = \mathrm{P}\left[d_i \rightarrow d_j\right].$$

We propose to use 5 different strategies to generate these user models, and for 3 of these, we create two variants, leading to a total of 8 user models. These models differ in terms of *density* (representing how many potential next devices might be picked after the current one) and *uniformity* (representing the extent to which selection of a next device is biased or not).

Figure 3.6 shows examples of the transition matrices $\mathcal{P}_\mathcal{M}$ generated by the 8 models, represented as heatmaps (using $N = 12$ devices). The creation strategies are explicated below:

1. **uniform:** The worst case scenario for our framework is a completely uniform model, where Alice chooses her next device with an even probability of $1/N$. In this situation, the currently used device cannot guess what appliance will be used next, making the session handoff as good as random;
2. **cyclic:** The best usage pattern is when Alice uses her devices in a cyclic order (making a circular Markov chain), because the devices always succeed in the prediction of the appliance she will use next. In this model, every transition vector $\mathcal{P}_\mathcal{M}(d, *)$ is constant;
3. **sequence:** This model is computed from a random model sequence $S_\mathcal{M}$ containing l interactions. $S_\mathcal{M}$ is populated by devices randomly selected with an uneven probability $P_{\text{devices}} \in [0,1]^N$. P_{devices} thus favors the use of certain devices (e.g. Alice uses her smart-phone more often than her mother's laptop). We compute the transition matrix $\mathcal{P}_\mathcal{M} = \Psi(S_\mathcal{M})$ as was shown in Section 3.2.1. The longer the

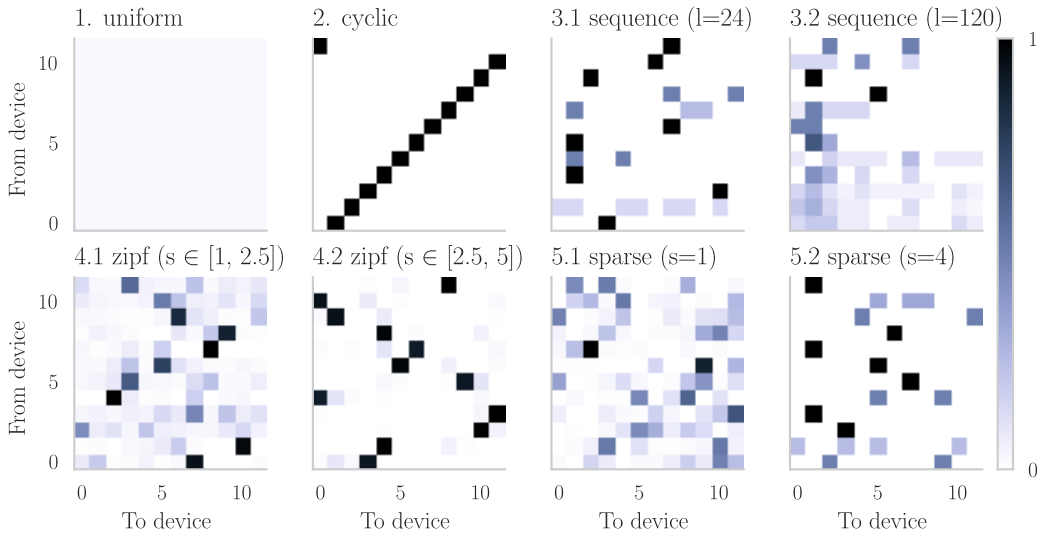


Figure 3.6 – Heatmaps of the transition matrix $\mathcal{P}_\mathcal{M}$ of each proposed model, with $N = 12$ devices. For each model, the i^{th} row of the heatmap represents the vector of transition probabilities from d_i: $\mathcal{P}_\mathcal{M}(d_i, *)$. Model *1. uniform* has a constant probability of $1/N$.

model sequence $S_\mathcal{M}$, the denser the output matrix. We thus generated two *sequence* models: **3.1**, with $l = 2 * N$, and **3.2**, with $l = 10 * N$;

4. **zipf:** Many processes in real life follow a Zipf law [127]: word occurrences, citations of scientific articles, wealth per inhabitant... Zipf's discrete law of probability $z(k; n, s)$ is defined by a constant population size $n \in \mathbb{N}$, a rank $k \in \mathbb{N}^+$, and a constant exponent $s \in [1, +\infty[$. It states that: $z(k; n, s) = z(1; n, s) * k^{-s}$, i.e., the probability of occurrence of the k^{th} most frequent element $z(k; n, s)$ equals the probability of occurrence of the most frequent element $z(1; n, s)$ times k^{-s}. The bigger the exponent s, the faster the function approaches zero, and the more $z(1; n, s)$ dominates the other probabilities.

   In our futuristic scenario where a user would own and frequently use a dozen of devices, the assumption that the transition probability between her devices follows a Zipf law is intuitively sound.

   We propose a model where, to each device d's transition vector $\mathcal{P}_\mathcal{M}(d, *)$, we assign a random permutation of Zipf's law's PMF using $n = N$ and a random exponent s. We propose two variants: in model **4.1**, we pick s from $[1, 2.5]$, which keeps the biggest probability $z(1; n, s) \in [0.32, 0.75]$. In model **4.2**, we draw s from $[2.5, 5]$, such that $z(1; n, s) \in [0.75, 0.96]$. Note that $\mathcal{P}_\mathcal{M}$ is always dense using the zipf model, but the probabilities' heterogeneity grows with the exponent s;

5. **sparse:** A sparse transition matrix contains null probabilities: there are certain devices d_i and d_j such that d_j will never be used after d_i. This is realistic, e.g., two desktop computers from two faraway locations will never be accessed in a row. For Sprinkler's handoff, this sparsity prevents the used device to have to choose between too many appliances to send session chunks to. For each $d \in \mathcal{D}$, we compute the transition vector $\mathcal{P}_{\mathcal{M}}(d, *)$ by drawing samples from a Zipf law $Z(n, s)$ with a "big" n (e.g. 1000) and a fixed s:

$$\mathcal{P}_{\mathcal{M}}(d, *) = \frac{X}{\sum\limits_{x \in X} x} \text{ s.t. } \begin{cases} X = \{Z(n, s) - 1\}^N \\ \sum\limits_{x \in X} x \neq 0 \end{cases}$$

The bigger the exponent s, the bigger the probability that $Z(n, s)$ yields one, i.e., that an outgoing transition equals zero. We proposed two models **5.1** and **5.2** with $s = 1$ and $s = 4$ respectively. We see that $\mathcal{P}_{\mathcal{M}}$'s sparsity is proportional to the exponent s.

While creating these models, we always ensure that the Markov graph is strongly connected, in order to effectively see the user switch between the N devices, instead of looping through a small subset of $\mathcal{D}$.

To generate sequences of the user's activity for each model, we randomly walk on the Markov graph derived from $\mathcal{P}_{\mathcal{M}}$, starting from a random device. The resulting sequence S_{tot} is then used to drive the evaluation of the session handoff.

Methodology

To evaluate our system, we deploy an e-squad of N virtual devices implementing the Sprinkler algorithms. We perform one evaluation per behavioural model presented above. From each of them, we obtain an interaction sequence $S_{\text{tot}} = \{r_1, \ldots, r_L\}$ of size L. It is split in two: the first subsequence $S_{\text{init}} = \{r_1, \ldots, r_{L_{\text{init}}}\}$ of size $L_{\text{init}} < L$ is fed to the devices on bootstrap (cf. Section 3.2.2) to let them know their respective addresses, and to give appliances an initial knowledge of the user's behaviour. The second subsequence $S_{\text{exp}} = \{r_{L_{\text{init}}+1}, \ldots, r_L\}$ (of size $L_{\text{exp}} = L - L_{\text{init}}$) provides the interactions performed during the experiment.

In our experiments, a user interaction is atomic: opening and closing a device is instantaneous, and generates a new session. While the Sprinkler Gossiper algorithm has been effectively implemented by the devices, the Session Handoff is only simulated: based

on the (genuine) current device's local sequence, we determine the amount of session data it sends to its peers, and finally compute the network cost C_N and the waiting time T_W for this interaction (cf. Section 3.2.3).

Parameters We set the number of devices to $N = 12$. We argue that this number of devices is already three times above current usage behaviours [101]. The initial sequence length L_{init} is set to 30. We consider that a tech-hungry user, owning and regularly switching among twelve devices, would easily achieve 30 interactions per day. Such a sequence length is big enough to contain most devices', yet small enough to provide only a coarse estimation of the user's real behaviour. The second subsequence is set to a length of $L_{\text{exp}} = 70$.

The Sprinkler protocol has three parameters: the Gossiper's fanout f, the Session Handoff parameter γ that controls the overall amount of session data sent, and α that controls the probability of usage below which we do not send any session data to a device. As already stated, we fix $f = \lceil \log(N) \rceil$, because this value has been proven sufficient for a probabilistic broadcast to reach all of its participants with a very high probability [126]. α has been arbitrarily set to $1/(N-1)$. γ is set to 1 when comparing the different usage behaviours in figure 3.7 (thus bounding the network cost C_N to the session's size w_{sess}); it varies from 0 to $N-1$ in figure 3.8.

According to [128], a web application session can weigh between 10kB to an unbounded value (in the mega-bytes) depending on the state-collection method (e.g. snapshots or event logging), and obviously on the application. However, desktop applications such as photo edition or Computer-Assisted Design software create save files of hundreds of MB. Considering that e.g. Adobe Photoshop now ships a mobile version, syncing such saves using Sprinkler is a compelling use-case. Hence, we consider that a session weighs between 10kB and 100MB.

3.3.2 Conducted experiments

We first evaluate the Sprinkler Gossiper. Then, we discuss the performance of the proactive session handoff, and of the reactive fallback.

Performance of the Sprinkler Gossiper

The goal of the Gossiper algorithm (Section 3.2.2) is to successfully propagate a user's overall interaction sequence S_t to all devices. To assess the Gossiper's efficiency, we thus compare the size of the real sequence S_t with the local version of the sequence $S_{t,d}$ maintained by each device at that time.

We aggregate the traces from all our experiments (one per user model), thus leading to 6391 studied local sequences. 577 (9.0%) of them are incomplete. Among incomplete local sequences, the median difference from the real sequence ($|S_t| - |S_{t,d}|$) is 1, while the maximal difference is 7 (one tenth of L_{exp}).

We conclude that our algorithm is able to perfectly propagate the sequence of the user's activity most of the time. When not the case, the drift of the local sequence is controlled: the devices eventually get the missing information from other peers, and end up perfectly knowing the user's behaviour again. Overall, we believe the local sequences are generally able to generate a fairly unbiased $\mathbf{p}_k$ for the Session Handoff to share session chunks.

Additionally in terms of network cost, each device receives a median amount of activity-related data of 3.5kB per user interaction, leading, for $N = 12$ devices, to a global Gossiper network traffic of $12 * 3.5kB = 42kB$ for each interaction. Increasing the number of devices that the user owns has a direct impact on the overall traffic, but the traffic per node is fixed on average (thanks to the probabilistic broadcast properties).

The session handoff

We want to understand the kind of user behaviour for which our algorithm is best suited. To do so, we compare the waiting times obtained with the different user models for a fixed γ. Thus, we set Sprinkler's parameter γ to 1, i.e., in total, the currently used device can only proactively share as much as the session's size w_{sess}, distributed among the possible next devices.

Remember that a centralized session handoff solution only functions reactively: the normalized user's waiting time T_W is always one in this situation. However, it always scores a proactive network cost C_N of zero.

Figure 3.7 shows the boxplots of the normalized waiting times (s.t. $T_W = w_{\text{wait}}/w_{\text{sess}}$) for each user model (see figure 3.6). The boxes' central line represents their median, the edges show the first and last quartile, while the whiskers represent 3/2 of the interquartile

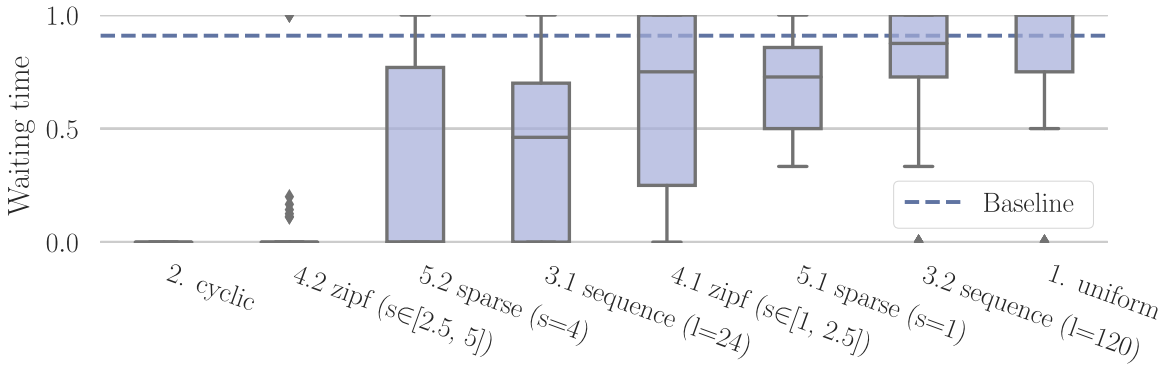


Figure 3.7 – Box plots of the user's waiting time (normalized) after the session handoff, grouped by behavioural model. Here, $\gamma = 1$: each device shares no more than the session's size to all the other devices. *Lower is better*: a null waiting time means that the session was sent proactively in its totality; a waiting time of one means that the current device needs to download the whole session reactively. We see that the handoff's performance highly depends on the user model.

range (they are Tukey plots [129]). The models are sorted by increasing lower quartile and median. A waiting time of zero means that the entire session was sent proactively; $T_W = 1$ means that none of it was sent, and that the entire session had to be downloaded reactively when the user opened her device. The dotted line represents the median waiting time of the baseline (Section 3.2.3). It is constant, because the baseline does not take transition probabilities into account.

As expected, the Session Handoff algorithm performs best with the *2. cyclic* model, where the next used device d_{next} is always known. As a result, the cyclic model scores a constant waiting time $T_W = 0$, meaning that the session is always entirely sent proactively to the right device. The algorithm performs worst with the *1. uniform* model, where d_{next} is unpredictable. It leads to a wait time $T_W > 0.75$ in 75% of the cases. The second best model for the Session Handoff is the *4.2 zipf* with $s \in [2.5, 5]$, where one transition probability greatly overpowers the others in each row (as can be observed in figure 3.6). This model is very similar to the cyclic one, with some added noise: the proactive handoff is mostly perfect.

The next two models, *5.2 sparse* ($s = 4$) and *3.1 sequence* ($l = 24$), show fairly similar results. Both models' upper whiskers reach one: the waiting time is highly variable. Figure 3.6 shows nearly identical transition matrices for these two models: mostly deterministic, apart from some equiprobable transitions. We argue that the difference of median waiting time (0.46 for model *3.1* against 0 for *5.2*) is caused by the uniform probability

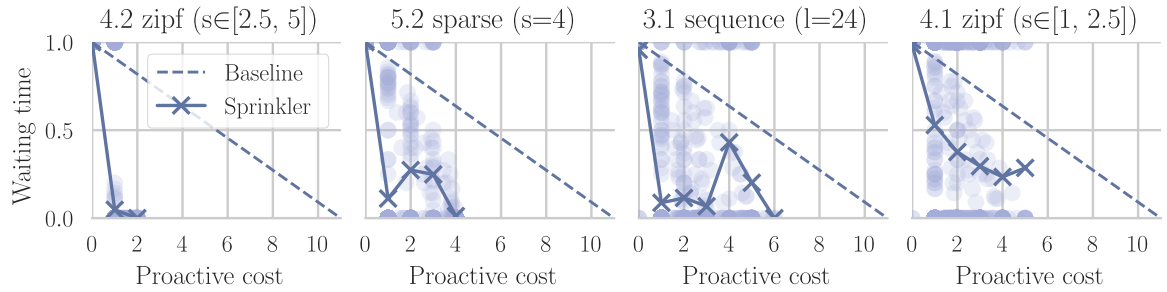


Figure 3.8 – The user's waiting time T_W and proactive network cost C_N (both normalized by w_{sess}) as a function of the parameter $\gamma \in [\![0, N-1]\!]$, for several behavioural models. Each point represents the (T_W, C_N) couple for a single file exchange using the proportional W_k computation. The full line is the moving average of T_W as a function of C_N. The dotted line is the baseline's T_W as a function of C_N. *Lower-left is better*: the handoff is more efficient when a small consumption increase yields big waiting time gains.

of switching among 7 devices when using device #1 in model *3.1*. For these two models, we observe that a little loss in predictability of the user's behaviour causes more unsteady results, even though the session handoff scores are still good most of the time.

The last three models, *4.1 zipf* ($s \in [1, 2.5]$), *5.1 sparse* ($s = 1$) and *3.2 sequence* ($l = 120$) all show a median waiting time above 0.5 (resp. 0.75, 0.73 and 0.88). The three of them display very dense transition matrices (apart from some constant vectors in *3.2*): we consider them as different types of unpredictable behaviour. Indeed, in the *4.1 zipf* model, one transition probability continues to dominate the others, while the two others show many uniform probabilities of transition. This leads to d_{curr} always sending small chunks of the session to several devices, which hampers the overall results.

This experiment shows that the performance of the proactive session handoff greatly depends on the user's behaviour. Given our simple inference model, we get better results when the user has predictable habits, despite some variability (e.g. *4.2 zipf*, *5.2 sparse*, *3.1 sequence*).

We performed a second study (figure 3.8), this time by varying the γ parameter for a fixed user model, allowing us to observe the influence of the normalized proactive network cost C_N on the waiting time T_W. The figure shows the normalized waiting time $T_W = w_{\text{wait}}/w_{\text{sess}}$ as a function of the network cost C_N, for four different user models, and for every value of $\gamma \in [\![0, N-1]\!]$.

The baseline, represented as a dotted line, shows a linearly decreasing waiting time as the network cost increases. Each dot represents the outcome of a single handoff. The full line represents the average of the waiting time over a sliding window of the network

cost. Note that, in our Session Handoff algorithm, the proactive network cost C_N is often lower than γ: indeed, the current device will only send its session to devices that have a non-null probability of being used next according to $\mathbf{p'}_k$ (cf. Section 3.2.3). In this graphic, lower is better: the *4.2 zipf* ($s \in [2.5, 5]$) model shows a close-to-perfect handoff. Then, we displayed results for *5.2 sparse* ($s = 4$), *3.1 sequence* ($l = 24$) and *4.1 zipf* ($s \in [1, 2.5]$), in the same order that they appear in figure 3.7.

Very good session handoff traces will look like *4.2 zipf*: a small increase in the allowed network cost leads to a dramatic decrease in waiting time. Furthermore, *4.2 zipf* is very short tailed: devices never send more than twice the session size. On the other hand, *4.1 zipf* shows poor results: at $C_N = 1$, the average waiting time is above 0.5. The function monotonically decreases until $C_N = 5$: only very unpredictable handoffs cause that much data exchange, thus scoring more than for $C_N = 4$. Finally, *5.2* and *3.1* show a combination of the first and last plots: they attain a very low waiting time at $C_N = 1$ (i.e. deterministic handoffs), and a longer T_W as the network cost increases (i.e., unpredictable handoffs).

We also observe many points at $T_W = 1$: this situation only arises when a device is being used for the first time ever. As it is not in the sequence of interactions, the previous device could not have sent it any session chunks, resulting in a failure of the proactive handoff. Fortunately, once this device joins the gossip, it will be able to proactively receive the session.

Looking at this experiment, we observe two possible use-cases for Sprinkler: it could either be preferable to keep a bounded network cost, leading to variable waiting times (as in figure 3.7); or it could be preferable to have mostly perfect proactive handoffs, while keeping network costs at a reasonable level. Indeed, combining all our models except the uniform one, we see that the proactive network cost C_N never exceeds 7 times the session size, which is far for flooding.

Overall, we find our predictive approach for distributed session handoff promising: for a bounded cost, it can send most of a user's session to her next device in the majority of cases. Future works on the behavioural model (e.g., using timestamps or locations) and on the session handoff decision algorithm hold further potential to drastically lower the user's waiting time.

Reactive handoff

Since our proactive session handoff might be imperfect, we ought to propose a functioning reactive fallback. In a situation where the previous session was only partly downloaded, d_{curr} needs to locate d_{prev} to be able to retrieve the remainder of the last session from it. We assume that d_{prev} is always connected when d_{curr} requests the session.

There are four ways d_{curr} can find d_{prev}'s address:

1. If d_{curr} has an up-to-date local sequence $S_{k,d_{\text{curr}}}$ containing d_{prev} in its last interaction (i.e. d_{prev} is in $S_{k,d_{\text{curr}}}$);
2. If d_{prev} has sent chunks of its session to d_{curr} (that is when $w_{k-1,d_{\text{curr}}} \neq 0$);
3. If the previous and current devices are the same ($d_{\text{curr}} = d_{\text{prev}}$);
4. If d_{curr} wrongly believed that the previous device was d, but d (which knew the right d_{prev}) *redirected* d_{curr} to the right device.

It is only when none of these solutions work that we fail to provide Alice with her last session.

d_{prev} in $S_{k,d_{\text{curr}}}$	$w_{k-1,d_{\text{curr}}} \neq 0$	$d_{\text{curr}} = d_{\text{prev}}$	Redirected	Failed
523 (94.7%)	407 (73.7%)	15 (2.71%)	16 (2.90%)	2 (0.36%)

Table 3.2 – How does d_{curr} know the address of d_{prev}?

Table 3.2 shows how d_{curr} identifies d_{prev} in all handoff occurrences of our experiment (using the 8 models and $\gamma = 1$). In total, there are $8 * 69 = 552$ data points. Note that the first and second categories are not exclusive. We see that, even though 5% of the used devices have an erroneous sequence during handoff, the current device quasi always finds the address of its ancestor. As a result, the reactive handoff succeeds in 99.64% of the cases.

Still, a complete failure of the handoff is unacceptable; we target a success rate of 100%. It appears one of the limitations of Sprinkler is the dependency to d_{prev}, that is the only device knowing the previous session.

3.4 Conclusion

Sprinkler demonstrates that, through probabilistic information dissemination, a user's devices can harness enough information to provide a distributed proactive ses-

sion handoff service that is cleverer than flooding. We see that, in any case, SPRINKLER reduces the user's waiting time, at the cost of additional network communications (compared to the cloud model). Despite our simplistic user behavioural model , we showed that the predictive session handoff performs better, at a lower network cost, when the user acts in a predictable way.

Still, SPRINKLER makes a very unrealistic hypothesis: all devices are always considered online. The previous device being the only holder of the previous session, its sudden disconnection can result in a complete failure of the session handoff. This assumptions would hamper SPRINKLER's performance in a real setting.

In the coming chapter, we remove this hypothesis: we study devices that frequently turn on and off, and modify SPRINKLER to create a *dependable* session handoff protocol using e-squads. This protocol will prove the practical applicability of e-squads for such services.

Chapter 4

Dependable e-squad services despite the devices' poor connectivity

In this chapter, we present our follow-up on the Sprinkler project, so-called Cascade. It aims at making distributed session handoff using e-squad dependable in a realistic environment. In the wild, users frequently turn their devices on and off (in distributed systems, this frequent join/leave pattern is called *churn*). An e-squad has to account for this churn to provide a reliable service.

4.1 Introducting Cascade

All major cloud service companies (e.g. Microsoft Azure, Google Compute Engine, Amazon Web Services, OVH) have a Service Level Agreement (SLA) that promises a minimum availability above 99.9% for rented machines; reimbursement policies in case of failure follow. It goes without saying that we cannot expect citizens to keep their devices as available; it is not even desirable.

We are stepping into a major distinction between the cloud model and the e-squad: with the latter, we have to take downtime for granted. How can we offer any dependable service in such conditions? By doing the same as cloud companies do to offer such high availability: redundancy.

We present Cascade, that continues the work undertaken in Sprinkler to propose a distributed proactive session handoff solution using a user's e-squad. To make our proposal *dependable*, we adapted the world-renowned P2P file sharing protocol BitTorrent [68] to the problem at hand: we eschewed centralised trackers, and limited session sharing to the devices most likely to be used next. This way, each session is intelligently backed up among the e-squad, and the user does not have to fear the loss of their session as they happily switch devices.

The specific Cascade contributions are the following:

— We propose a *decentralised session handoff* protocol adapting ideas of the BitTorrent protocol to ensure the handoff's dependability. It is backed by a *communication automaton*, a rigorous formalism capturing the formalism of communicating entities.
— We propose to determine the *cohort of seeders* (the devices/peers offering a particular file in BitTorrent) of each session based on the knowledge gathered by the e-squad on their user.
— We present an *in-depth evaluation* of our proposal based on a number of synthetic user models and a challenging churn model, in order to exercise our solution across a wide range of usage scenarios.

The remainder of this chapter will consist of a thorough presentation of our protocol in section 4.2, followed by an evaluation of its performance in section 4.3. We conclude this chapter in section 4.4.

4.2 Our approach

Our protocol, Cascade, aims at efficiently sharing the state of a user's applications (thereafter called a *session*) among her devices. Its main objective is to be *proactive*: ideally, the user's previous session would already be waiting for her on the device she is about to use. To do so, Cascade gathers information on the user's behaviour, and tries to predict her next location. If the proactive handoff fails, the newly opened device must still be able to download the previous session *reactively* at the moment the user opens it. Alas, mobile appliances are frequently disconnected from the network or turned off. Knowing so, the reactive session handoff must still function reliably, even if the device on which the previous interaction took place is offline when the user opens the next one. Due to the limited resources inherent to mobile appliances, a secondary objective of Cascade is to be *lightweight*. Applicative sessions are considered arbitrary heavy: the protocol must minimize the network traffic by avoiding sessions exchanges to irrelevant devices.

We will first introduce our session handoff protocol in Section 4.2.1. Then, Section 4.2.2 will cover how Cascade gathers knowledge on the user to proactively exchange the user's session between devices.

4.2.1 Peer-to-peer session handoff

CASCADE is loosely based on the BitTorrent file-sharing protocol [68]. Similarly to BitTorrent, a newly added file (i.e. the latest session) is split into n chunks of fixed-size. Chunks are then distributed among nodes (i.e. the devices) in a decentralized fashion. Furthermore, any peer having some session chunks can share them again, thus increasing the number of sources for this chunk beyond the original uploader. However, unlike BitTorrent CASCADE does not locate peers through a central server, or using the BitTorrent DHT protocol [73], but instead manages its own list of peers.

To provide the transfer of sessions among peers, CASCADE uses a small set of messages $\mathcal{M} = \{Sess, Get_X, Have_X, Chunk_X, Bootstrap\}$ according to a communicating automaton $\mathcal{CA}$ defined as the following:

Exchanging session metadata To advertise a new session s to its peers, a device d sends a message $Sess$ containing the metadata $Sess = \left(\textcircled{\scriptsize 🕒}, W_s, \mathcal{H}, h, bf\right)$. This metadata information describes a specific session s of size W_s that is split into N_s chunks of size W_p [1]. A 20 byte `SHA1` hash is calculated for each chunk, and is stored in a contiguous array h: $\forall j \in [\![0, N_s[\![,\ h[j]$ contains the 20 byte hash of the j_{th} piece of the session. Finally a `SHA1` hash $\mathcal{H}$ is computed from the hash-array h. Each session has a unique timestamp, noted $\textcircled{\scriptsize 🕒}$, that enables us to have a total order on a sequence of sessions across time. Additionally, the bit field bf is an array of N_s booleans, such that, $\forall j \in [\![0, N_s[\![,\ bf[j] = 1$ means that the sender owns the j^{th} piece, and can share it with the receiver ($bf[j] = 0$ otherwise). In follow-up communications, devices only use $\mathcal{H}$ to reference a session. Indeed, SHA1 being a cryptographic hash function, it guarantees (with very high probability) that no two sessions can share the same hash, making $\mathcal{H}$ a valid identifier.

Exchanging chunks The message Get_X (resp. $Have_X$) enables a node to request (resp. acknowledge) a specific chunk having the hash $h[X]$. Furthermore, the message $Chunk_X$ carries the raw data of the chunk being previously requested. *Bootstrap* enables a new device to request the latest ongoing session to a remote peer. All exchanged messages have mandatory fields such as the hash $\mathcal{H}$ of the session it refers to.

Finally, to get metadata fields from a message, we use the operator $\triangleright$. For instance, getting the $\mathcal{H}$ field from a message S is noted $S \triangleright \mathcal{H}$.

1. apart from the last piece, that can be shorter.

Definition 1. *A **communicating automaton** $\mathcal{CA}$ is a tuple $(Q, \mathcal{M}, q_0, \mathcal{A}, Evt, \mathcal{C}, Act, \rightarrow)$, where Q is a finite set of states, $\mathcal{M}$ is a finite set of messages as defined previously, $q_0 \in Q$ is the starting state and $\mathcal{A} \subset Q$ is a set of accepting states. Evt is a set of event types such that $Evt = \{\Rightarrow, \Leftarrow, \#, !, \gamma, \epsilon\}$ where $\Rightarrow$ (resp. $\Leftarrow$) denotes a received (resp. sent) message from the underlying network. Further we have 3 different way to propagate messages to a set of peers: $\#$, $!$ and γ. $!$ is a traditional propagation of multicast messages to a set of peers, and γ is a basic broadcast. Finally, $\#$ is a predictive multicast: it dynamically selects peers by inferring the user's future behaviour, and is used to propagate the session to relevant peers. The predictive multicast is one of the key contributions of the chapter, and will be discussed in details in Section 4.2.2. γ is used when a peer needs to do a traditional broadcast. Moreover, ϵ event enables to trigger a transition without the occurrence of an external event. For instance, when Alice is opening or closing her device, the latter may trigger by itself an internal event. Act is the set of actions performed when a transition is taken. Additionally, $\mathcal{B}(\mathcal{M})$ is the set of constraint conjunctions on $\mathcal{M}$. Finally, $\rightarrow \subseteq Q \times Evt \times \mathcal{M} \times \mathcal{B}(\mathcal{M}) \times Act$ is the set of transitions.*

A transition has the following form $s_1 \xrightarrow{\mathcal{L}} s_2$ and changes the state of the communicating automaton from s_1 to s_2 once the label $\mathcal{L}$ is evaluated to true. The transition label $\mathcal{L}$ is defined such as $\mathcal{L} \subseteq Evt \times \mathcal{B}(M) \times Act$, and has the following format:

$$\mathcal{L} = Event \ ; \ Constraints \ ; \ Actions$$

According to the aforementioned definitions, figure 4.1 and figure 4.2 describe the control flow that enables CASCADE to exchange sessions, and more particularly the behaviour of a device when it acts as a client (i.e as a *leecher*) or as a server (i.e as a *seeder*). We call S_l the local session stored on a device, and S_r the session received from a remote peer. (As long as a device only owns some chunks of the latest session, it participates in the network as a seeder and a leecher concurrently.)

The user activity is modelled through an ϵ transition. For instance, as soon as a user uses a device that does not have any session stored locally, the device must proceed to a bootstrap to get the latest ongoing session (see figure 4.1 ❶, transition ❍→ ⑥). In a similar manner, a new session is created every time the user leaves a device d (see figure 4.2 ❶). After chunking the new session into pieces, d advertises it through predictive multicast $\#$ (see figure 4.2 ❷). The recipients perform another predictive multicast to advertise this new session only if they have never been notified of it yet (see figure 4.1 ❶, transition ⑤

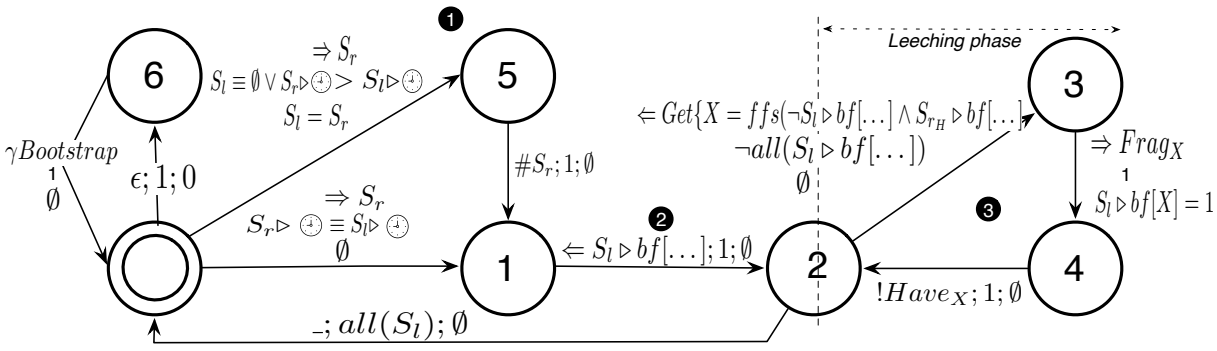


Figure 4.1 – Client behaviour of CASCADE, i.e leecher

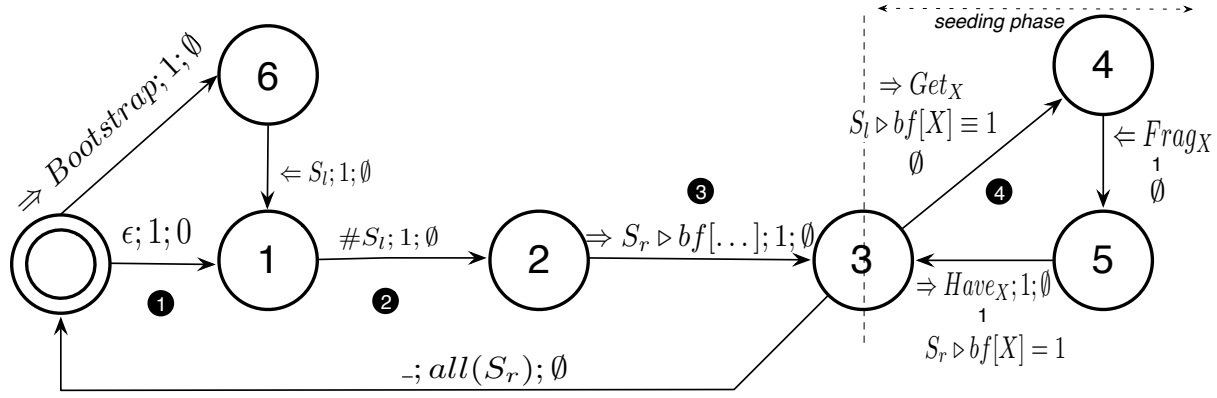


Figure 4.2 – Server behaviour of CASCADE, i.e seeder

$\rightarrow$ ①). A new session is detected when its related metadata message S_r is received from a remote peer with a timestamp $\textcircled{\scriptsize 4}$ higher than the one of the local session's metadata noted S_l. When this case occurs, *leechers* flush both metadata information of the local ongoing session S_l and its related raw data chunks (see figure 4.1 ❶, transition ❍ $\rightarrow$ ⑤). Thereafter, S_l is initialized from S_r with a bit field bf containing only zeros as they do not yet have any data chunks of the new detected session. In any case, when *leechers* receive a session, either new or not, they answer with the bit field of the ongoing session to notify their missing chunks (see figure 4.1 ❷). *Leechers* then enter in a *leeching phase* (see figure 4.1 ❸): they first calculate the *Find First Set* (ffs) of the inverted bit field of their ongoing session (i.e. $ffs(\neg S_l \triangleright bf[\ldots] \wedge S_r \triangleright bf[\ldots])$), to find the index of the first bit that is set to 1 and thus identify the next chunk to request . Every time a peer d' finished downloading a chunk, it performs a traditional multicast to propagate the news to any

device that was communicating with d'. This way, peers will be able to request chunks to d', and not only to d, the device on which the interaction took place. Meanwhile, *seeders* send data chunks to *leechers* as long as they possess data chunks unknown to the *leechers* (see figure 4.2 ❸, ❹).

As a corollary, the transmission of the session can continue even if its original source shuts down, as long as the latter had time to send its entire session once.

4.2.2 Predictive and reliable peers selection

As CASCADE does not rely on any trackers, one of the key features of our approach is to be able to find the most adequate peers to share the chunks of the current session in a reliable manner. To achieve this aim, we need to overcome three issues: (i) representing the user's behaviour across time in a compact manner, (ii) propagating the user's behaviour among devices, (iii) knowing which peers are online and finally (iv) selecting adequate peers to share the session.

ISSUE 1: representing the user's behaviour As in SPRINKLER, we represent Alice's use of her devices as an ever-growing sequence containing k *interactions*: $S_k = \{r_1, ..., r_i, ..., r_k\}$. Knowing that the user owns a set of N devices $\mathcal{D} = \{d_1, .., d_N\}$, each interaction r_i is characterized by a couple $r_i = (d, t) \in \mathcal{D} \times \mathbb{R}$, which means that Alice started using the device $r_i.d$ at time $r_i.t$ (and stopped using it before $r_{i+1}.t$).

From a global point of view, the sequence S_k contains every of the k interactions performed by the user since the beginning of the program execution. Locally, however, each device d only knows about the sequence $S_k|_d$ of interactions that took place on it, that is:

$$S_k|_d = \{r \in S_k,\ r.d = d\} \implies S_k = \bigcup_{d \in \mathcal{D}} S_k|_d,$$

ISSUE 2: propagating the user's behaviour In order to predict the user's future behaviour, all devices must gain a global understanding of the user's past behaviour, by aggregating their respective local knowledge into a global interaction sequence. Each device d thus holds a local sequence $S_{k,d}$ that is an aggregate of the others appliances' interactions, such that:

$$S_k|_d \subseteq S_{k,d} \subseteq S_k.$$

Our goal is to keep the devices' local sequence $S_{k,d}$ as close to S_k as possible.

To achieve this aggregation, we adapt the SPRINKLER Gossiper algorithm presented in section 3.2.2. To cope with the devices connection and disconnection, we simply add acknowledgements to each network communication. A round only completes (with views updates) when both devices have been online the whole time. The updated dissemination algorithm is called STORYTELLER.

ISSUE 3: knowing which peers are online Before selecting peers to exchange the session with, the currently used device must sort out the ones that are offline. This problems knows many solutions, including mature industrial ones [130], as it is a classical need in distributed systems. We suppose the presence of a peer discovery mechanism in our system.

ISSUE 4: selecting adequate peers A device must have some insight on which device will most probably be used next to intelligently choose peers to share its new session with.

We consider the end of interaction k, that took place on the device d_{curr}, thus generating the session s_k. At interaction $k+1$, Alice will grab the device d_{next}. CASCADE uses the global sequence S_k of past interactions (or more precisely its estimation provided by STORYTELLER) to compute a list of the devices that Alice will most likely use after d_{curr}.

This takes the form of a *score array* $sc = [sc_{d_{\text{curr}}\to d}]_{d\in\mathcal{D}}$ such that $sc_{d_{\text{curr}}\to d}$ is the number of times Alice switched from d_{curr} to d in the past. It is proportional to the observed experimental probability that Alice switches from d_{curr} to d in S_k:

$$sc_{d_{\text{curr}}\to d} = \left|\{(r_t, r_{t+1}) \subseteq S_k \mid r_t.d = d_{\text{curr}} \wedge r_{t+1}.d = d\}\right|.$$

An intuitive approach to peers selection would be to preferably target the devices that have the highest scores, i.e. the highest probability of being used. However, Alice might use an unexpected device, and d_{next} might not be connected when devices share s_k, which would trigger a reactive download when Alice grabs d_{next}. For these reasons, the session handoff must also reach some other devices with a low probability.

To achieve this, we borrow the well known *roulette wheel selection* function [131] from the genetic algorithms literature: when selecting individuals to reproduce from a generation to another, the roulette wheel selection picks members of the population with a probability proportional to their individual fitness score (that is $sc_{d_{\text{curr}}\to d}$). The name comes from the analogy with a roulette wheel in a casino, where each individual is given a

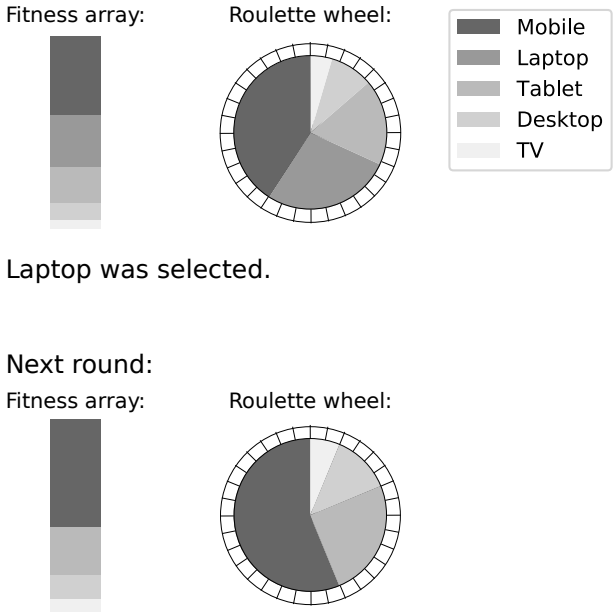


Figure 4.3 – Illustration of a roulette-wheel selection.

number of holes on the wheel proportional to its score. The ball has an equal probability of landing in every hole, thus giving fitter individuals a better chance of being picked (while still allowing less fit individuals a chance of being chosen).

In our case, a device d performs a roulette wheel selection, using sc, to pick up to f peers to share a new session s_k. f is StoryTeller's *fanout*, a configuration parameter. Every time d chooses a device, it is removed from sc. In figure 4.3, we see a graphical representation of sc depicted as the "fitness array": Alice's mobile has the highest likelihood of being selected after d_{curr}, and her TV has the least non-null one. *Devices that are either disconnected or have a probability of 0 are never selected*: if they were to be chosen next by Alice, they would have to reactively download the session from online devices. In figure 4.3, d selected Alice's laptop. After recreating a new roulette wheel with the laptop left out, d can select another device. The algorithm stops when d selected f online peers, or when d could not find more online devices with a non-null probability of being used next.

If d could not find any *online* peer with a non-null score, it falls back to picking up to f random connected peers in its local sequence. This fallback is particularly useful while facing a lot of remote disconnections, where this situation can arise fairly often.

This peers selection function is responsible for the predictive multicast, depicted by the symbol # in the previous figures 4.1 and 4.2. By directing the flow of sessions exchanges,

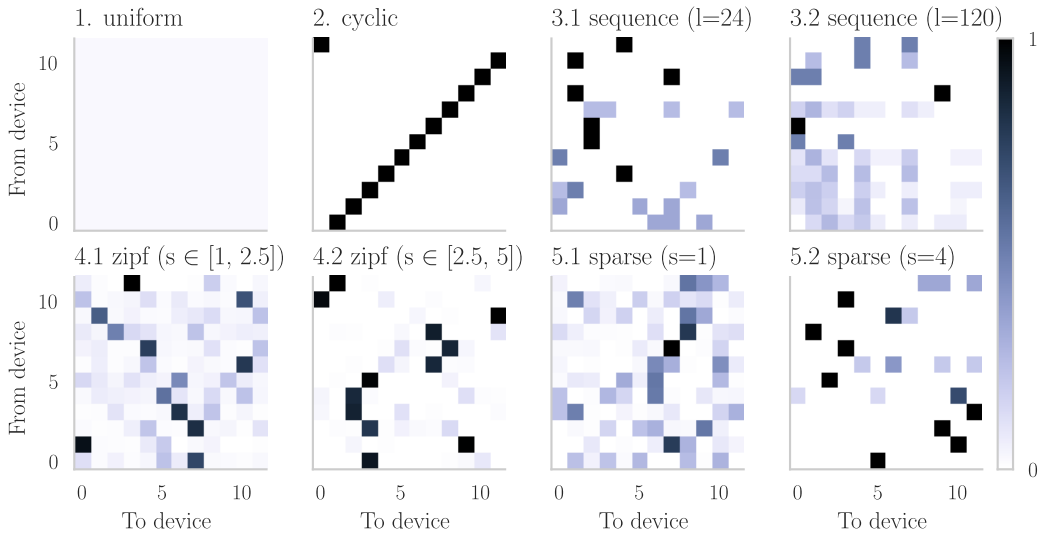


Figure 4.4 – Equivalent to figure 3.6 with different sampling: heatmaps of the transition matrix $\mathcal{P}_{\mathcal{M}}$ of each proposed model, with $N = 12$ devices. For each model, the i^{th} row of the heatmap represents the vector of transition from d_i: $\mathcal{P}_{\mathcal{M}}(d_i, *)$. Model *1. uniform* has a constant probability of 1/12.

it drives the performance of the proactive session handoff, and guarantees the possibility of the reactive fallback.

We now proceed to the evaluation, where we analyse Cascade's efficiency at performing proactive and reactive session handoff among an e-squad, and its network consumption.

4.3 Evaluation

We will first present our evaluation testbed in sectio 4.3.1 before diving into the evaluation of Cascade in 4.3.2.

4.3.1 Testbed

As with Sprinkler, the absence of real-world data caused us to evaluate Cascade using virtual devices (with the protocol implemented in Go) controlled by a simulated user (driven by Python code).

In the same way as Sprinkler (cf section 3.3.1), we simulated the client using Markov models $\mathcal{M}$ represented by their transition matrix $\mathcal{P}_{\mathcal{M}}$. Figure 4.4 shows the different model heatmaps. They are sampled from the same distributions as presented in section 3.3.1, although new samples generate new behaviours.

Churn model

We do expect one's appliances to frequently loose Internet connection, depending on the user's actions or the devices' system. One of CASCADE's goals is precisely to remain functional despite devices' churn. As long as two devices are online, they should try their best to bring the user's session to the device she will use next.

To assess CASCADE's resilience, we have designed a difficult churn model: devices gain and loose connectivity at random time intervals. Offline devices are awakened by the user when she decides to use them, but this is the only correlation between the device usage and the churning model. Such an autonomous churning model is more general than when the user's behaviour drives the churn: if our algorithm survives the former, it should handle the latter.

We randomly assign a target *average uptime rate* $\upsilon \in [0, 1]$ to each device d on bootstrap. Initially, d is connected with a probability υ. Stutzbach and Rejaie [132] showed that the Weibull distribution fits P2P churning behaviours quite well, therefore we draw connection and disconnection time intervals from a Weibull distribution $\mathcal{W}(\lambda, \sigma)$. With a small shape parameter $\sigma < 2$, Weibull shows a long tail, generating long intervals with a small probability; we choose $\sigma = 1.2$. λ is the scale parameter: it is used to change the distribution's expected value without changing its shape.

For a given device d having an average uptime rate of υ, we call T_{off} the random variable representing the time during which the device d remains disconnected, and T_{on} the random variable representing the time that d stays up. They are computed as follows:

$$T_{\text{off}} \sim t_{\min} + \mathcal{W}(1 - \upsilon, 1.2)$$
$$T_{\text{on}} \sim t_{\min} + \mathcal{W}(\upsilon, 1.2)$$

Weibull's support is $\mathbb{R}^+$, which makes $t_{\min}$ (a constant parameter) the lower bound of the intervals.

Figure 4.5 (a) shows two histograms representing the distribution of the values taken by T_{off} and T_{on} for $t_{\min} = 0.5$, and $\upsilon = 0.7$. We see that T_{on} (having $\lambda = 0.7$) shows a much longer tail than T_{off} (having $\lambda = 0.3$). Using the same parameters, figure 4.5 (b) shows the timeline of a device execution. We observe that our model provides uneven connection and disconnection durations. In this case, the output uptime rate is of 62.6%, which is close to the target 70% provided as parameter.

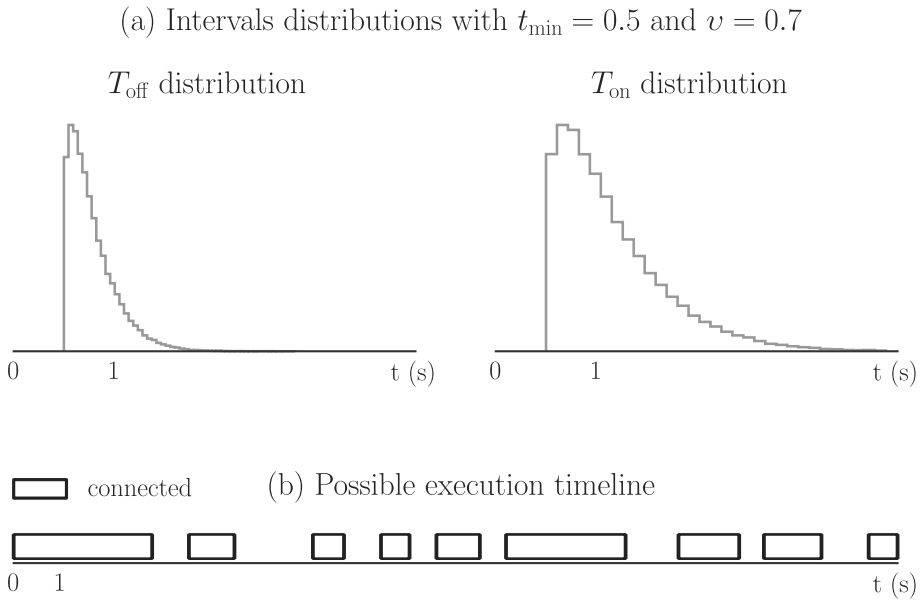


Figure 4.5 – (a) Probability density function of the random variables driving the disconnection and connection intervals (resp. T_{off} and T_{on}) using $\sigma = 1.2$, $t_{\min} = 0.5$, and $\upsilon = 0.7$; (b) sample timeline of a device's execution using these parameters.

In practice, our churn model makes the behaviour of devices fairly unpredictable from the point of view of their peers. Because the devices still need to know whether their peers are online or not when passing sessions (to avoid completely failing a handoff—cf section 4.2.2, ISSUE 3), we crafted a simple oracle, thus avoiding the need to bloat our protocol with acknowledgements. When selecting peers, a device can query a shared object to know whether the queried peer is online at this precise instant (which does not guarantee that it will remain online long enough for a successful communication). A better solution would be to leverage a distributed failure detector such as Consul [130], although this elementary oracle was enough to suit our needs.

We have designed two behavioural models (one for the user's activity, and one for the devices state) to provide our protocol with a challenging testbed. Although certainly not realistic, due to our lack of field data, we believe that our user models show a wide diversity of behaviours, and that our churning model is uncompromising enough to assess the efficiency of Cascade.

Methodology

An experiment is made of different steps. As in SPRINKLER's evaluation, we first compute a behavioural model, used to create the sequence of the user's actions S_{tot} of size L. Then, this sequence is split between the bootstrap part S_{init} (of size L_{init}), given to each device on boot, and S_{exp} (of size L_{exp}) that provides the order of the devices' usage. During the experiment, devices will connect and disconnect following the churning model described above; unless we deactivate the churning, in which case the devices are always online. An *experiment set* comprises one experiment per user behavioural model. We conducted 10 experiment sets with the same parameters twice: once enabling the churn, and another time with the churn deactivated for comparison.

Parameters Every session weighs the same size $W_s = 1\text{MB}$, while the session pieces weight $W_p = 16\text{KB}$: each session is chunked into 64 pieces. CASCADE's fanout f, that drives the session handoff's spread, is set to 4: it is just superior to $\log(N)$, as suggested by Kermarrec et al [126]. Further, we set the number of devices to $N = 12$, three times above current device ownership habits [101]. The initial sequence size is set to $L_{\text{init}} = 30$. With such a small initial knowledge, peers selection will make very coarse estimations of the real user's behaviour, but will still get the address of most devices. We can imagine that a tech-hungry user would switch 30 times between her devices in a day or two: until this time, our protocol would only gather data without attempting proactive handoffs. The experiment sequence size is empirically set to $L_{\text{exp}} = 70$.

The user performs a new interaction every 2 seconds. Churn-wise, devices stay connected for a minimum time of $t_{\text{min}} = 0.5s$, we randomly pick the average uptime rate υ between 70% and 100%. It is unrealistic to imagine all of a user's devices connected at least 70% of the time, but the churn rate is at the same time very fast, enough to assess the protocol's resiliency to disconnected devices.

Unless otherwise noted, the following results stem from this aggregate of experiments sets, using the parameters described.

4.3.2 Conducted experiments

We now display the experiment's results. We first evaluate the performance of STORYTELLER, before examining the cost and performance of the whole CASCADE protocol with and without churn.

Evaluation of StoryTeller

As already mentioned, StoryTeller is an upgrade of Sprinkler Gossiper, with acknowledgements after each message. We need to ensure that acknowledgements allow a decent propagation of the user's sequence of interaction to all devices, despite the devices churn. This is why we now reproduce the same experiment that was made in the previous chapter at section 3.3.2.

We compare the length of each device d's sequence $S_{t,d}$ with that of the full sequence S_t, after each new interaction (i.e. $\forall t \in [L_{\text{init}}, L_{\text{init}} + L_{\text{exp}}]$). We compiled the results in Table 4.1, after 10 experiment sets (described in section 4.3.2) for each case.

Churn	# sequences	# incomplete	median diff.	max. diff.
Disabled	61133	93 (0.15%)	1	1
Enabled	44949	1718 (3.8%)	1	7

Table 4.1 – Performance of StoryTeller

The amount of recorded sequences is 30% bigger without churn: devices being always online, they share information more often. The amount of incomplete sessions is 25x higher with churn, and incomplete sessions have up to 7 interactions missing, against only one in the other case. Not surprisingly, churning impairs StoryTeller's efficiency. The median number of missing interactions (among incomplete sequences) is the same in both situations, though, showing a controlled shift between the ground-truth and the local sequences in both situations.

The discrepancy results with churn are the same as the ones of Sprinkler Gossiper (median difference of 1, and maximum of 7). Without churn, though, our maximum discrepancy is only 1. In any case, Sprinkler Gossiper showed 9.0% of incomplete sequences, while StoryTeller only climbs up to 3.8% with churn. We conclude that the acknowledgements have greatly improved the dependability of this sub-protocol, and we trust that the small drifts between local and full sequences should not hinder the predictive broadcast used for session handoff.

Evaluation of Cascade

To evaluate Cascade, our primary metric is the success rate of the session handoff. To have an accurate overview of the quality of this metric, we need to get 3 different underlying indicators. First, we must count the amount of successful *proactive* handoffs

(when the previous session is lying on the user's next device before she opens it). Second, we must also count the successful amount of *reactive* handoffs (when the previous session is downloaded from remote peers at the time the user opens her next device). Thirdly, we must get the number of *miss*, i.e. how often the next device completely failed to retrieve the user's session.

The main objective of Cascade is to achieve the biggest rate of successful *proactive* handoffs, and to gracefully fall back to *reactive* handoff. However, a distributed alternative to cloud-based session handoff services must first and foremost be dependable to convince its userbase. Hence, it is crucial that we reach a *miss* rate of nearly 0%.

As an additional metric, we also measure, during each interaction, the number of session exchanges that were completed. This provides a metric of Cascade's cost: the lower the better. Without churn, the ideal proactive session handoff algorithm would only share the session once: from the currently used to the next one. Unfortunately, in real life, one cannot know the future.

Devices will thus have to share their session with *some* peers to have a reasonable probability to reach the correct device. Moreover, because devices can disconnect at unpredictable times, it is better to send several copies of the current session as backups.

In figure 4.6, we show two plots for each case: (a) without and (b) with churn. In each sub-figure, the leftmost plot shows a bar per user behavioural model described in figure 4.4, plus an additional bar combining the results from all user models (denoted 'all'), that read Cascade's success rate. The darkest portion represents the rate of successful proactive session handoffs, while the lightest part shows the rate of reactive handoffs. The remaining unfilled area thus represents the rate of missed handoffs. The rightmost plot shows a box and whiskers plot per user model, plus an additional one combining all user models ('all'), reading the distribution of the number of completed session exchanges per interaction. Again, the boxes are Tukey plots [129].

A world without churn When devices are always on, the peers selection function presented in section 4.2.2 can show its full potential: the session exchange decision is always made according to the sequence of the user's behaviour (cf section 4.3.1), and is not impeded by the remote devices' state. These results are shown in figure 4.6 (a).

We will first take interest in the success rate bar plots. We foremost see that each bar reaches 100%: the session handoff never misses. We will thus focus on the proactive success rate, that does depend on the user's behaviour.

As planned, the worst case scenario is the *1. uniform* model, with only 20% success rate. Two reasons explain this bad result: (i) the user's behaviour is unpredictable, (ii) the peers selection is as good as random implying a wide spread of the session. The best situation arises with model *2. cyclic* that reaches 100% success rate. Since the user's behaviour is deterministic, the currently used device always knows to which (unique) device it should send the current session.

The next best scores, lying around 95%, are reached with the models *3.1 sequence* $l = 24$, *4.2 zipf* $s \in [2.5, 5]$ and *5.2 sparse* $s = 4$. These three models are the ones with the most heterogeneous probabilities: a couple of devices always have a much bigger probability of being chosen than the others. We conclude that the peers selection function works best when the user behaves without ambiguity.

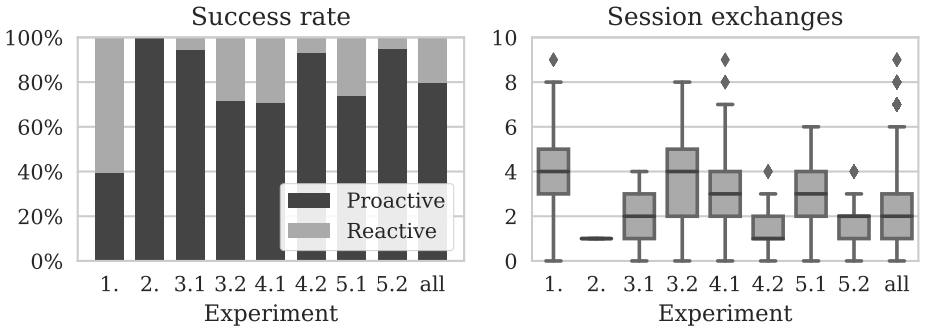
The last three models, that score from 70 to 75%, propose different user behaviours: in *4.1 zipf* $s \in [1, 2.5]$, one device always has more than one chance out of three of being chosen; in *3.2 sequence* $l = 120$ and *5.1 sparse* $s = 1$, several devices have similar odds of being chosen, while the rest have zero or close. We see that fairly unpredictable behaviours still provide efficient proactive handoff: devices fall back to reactive download in less than one third of the cases.

Globally, the proactive handoff succeeds four out of five times when devices are always up.

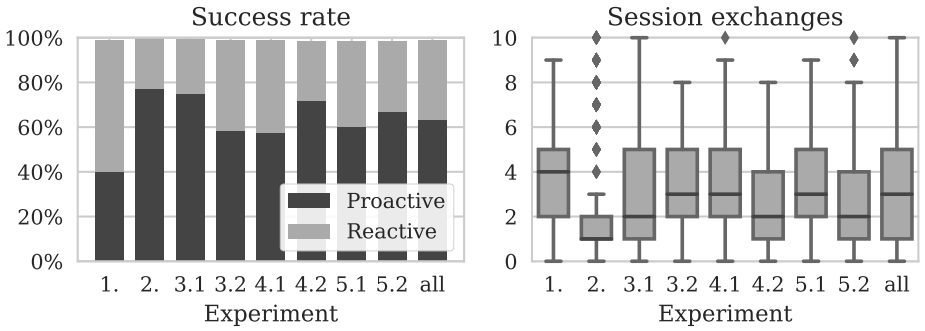
Turning our attention to the session exchanges box plots, we see that the spread of the session handoff is minimal when the user's behaviour is the most predictable. Note that zero session exchanges most certainly that a device anticipated that it would be its own successor. Indeed, Cascade selects peers proportionally to their probability of being used next: when a device's probability (its score) dominates the others, it will always receive the session. It is when the algorithm cannot choose between devices that it will pick the peers more randomly, thus increasing the total number of session exchanges.

We observe that, for the sparsest models (*2.*, *3.1*, *4.2* and *5.2*), the number of session exchanges is always inferior or equal to $f = 4$, the fanout of the peers selection. This means that the number of devices having a non-zero probability of being used next is never higher than f in these cases. However, in the model having the second densest transition matrix, i.e. *3.2*, the median reaches $f = 4$. The *1. uniform* model does not have a bigger median (even though it exchanges the session more than 6 times in 25% of the situations).

We understand that the *minimum* number of session exchanges will asymptotically



(a) Without churn



(b) With churn

Models: **1.** uniform; **2.** cyclic; **3.1** sequence $l = 24$; **3.2** sequence $l = 120$; **4.1** zipf $s \in [1, 2.5]$; **4.2** zipf $s \in [2.5, 5]$; **5.1** sparse $s = 1$; **5.2** sparse $s = 4$.

Figure 4.6 – Success rate of the session and amount of exchanged sessions per interaction. (a) shows the results for the experiments sets without churn, while (b) shows the results in presence of churn.

reach f as the score array gets denser: if our selection function can find f devices having a non-null probability of being chosen, it *will* send the session to f devices, whatever their scores. And the score array is bound to get denser with time, as the user makes more and more unprecedented device choices. A remedy would be to allow the selection algorithm to forget: by keeping only the last interactions in the sequence, the score array would forget rare devices switches with time.

The *maximum* number of session exchanges, on its part, *does* depend on the probabilities of the devices usage. When a few devices dominate the probability of being used, every device wants to share the new session with them, thus keeping the session exchanges directed at these few.

Responsible users shutting down their devices at random Now that we have thoroughly studied CASCADE's behaviour in an ideal (yet power-wasting) world where our user never turns off their devices, we can see how its performances hold when they do. These results are displayed in figure 4.6 (b).

Again, we will start by commenting the success rate plot. We firstly observe that the handoff still succeeds in 98.9% of the cases overall. The complete miss most often arises when the devices that were informed of the previous session (i) are not chosen next and (ii) are disconnected when the user picks the next device: this could happen with any user model.

Apart from these few misses, this plot is very close to one without churn, except that the proactive success rate dropped from 80.0% to 63.3%. The reason is simply that the peers selection now *can* fail at finding online nodes having a non-zero probability of being picked next, in which case it falls back on choosing random online peers to share the session with (cf section 4.2.2). We also see that, if the rank of each user model has remained the same, the approaches that used to be the most successful have suffered a heavier drop in their proactive success rate (*2. cyclic* has lost 23%, while *1. uniform* has not changed a bit).

Now detailing the rightmost plot, we observe a global increase of the amount of sessions exchanged. Overall, the median number of session exchanges has risen from two to three, and shows a much higher dispersion. This remark holds true to most user behaviour models, apart from the *3.2* (which median session exchanges when down from 4 to 3): one of the densest transition matrices. This state of fact fits with our previous assumptions: given that model *3.2* sends close to $f = 4$ sessions per interactions, it is less sensitive to churn. Among these numerous peers, the selection function has more odds of finding one online peer than with other sparser models. As a consequence, it more rarely resolves to random selection (as opposed to e.g. *4.2*, that went from a median session exchanges of one with very little dispersion to a median of two, with four or more session exchanges in 25% of the cases).

Wee see that the determinism of the user's behaviour in some models, that yielded close to perfect proactive results without churn, impedes the results drastically in presence of churn.

If we were to deploy CASCADE in a real-world system, we could expect proactive results and network costs lying between the uniform worst-case scenario (40% successful proactive handoffs for a median of 4 session exchanges per interaction) and a fairly unpre-

dictable model like *3.2* (whose proactive success rate is 60%, with a median of 3 session exchanges per interactions). More importantly, whatever the user's behaviour, we believe that Cascade would remain as dependable as it proved to be in the above experiments.

4.4 Conclusion

With Cascade, we have demonstrated that a ubiquitous service like session handoff could be effectively handled by an e-squad.

Such a service handles very sensitive data, as a user's applicative sessions contain most of their computing activity. Providing such service aside from the cloud drastically diminishes the potential attack surface on people's privacy. Considering the behavioural models we build by just knowing when a user connects to their device, one can only imagine the insights gathered by cloud providers on their daily routines, by just operating a cloud session handoff service.

Furthermore, we see that our solution provides very convincing performance, despite the challenging churn model we put it through: 99% success rate, for a median network cost of three session exchanges per handoff. Cascade proved to be a dependable solution regardless of the connectivity constraints of e-squads.

Finally, our *proactive* solution beats any state of the art solution by providing true *continuous* interaction 60% of the time.

We could improve Cascade's prediction model to increase the proactive success rate (e.g. using the user's location or time/date). Although, this would require field studies to be interesting (one of our limits being the complete simulation of the user's behaviour), which are notoriously time consuming. We leave that problem for future work, and argue that our simplistic prediction model sufficed to show the applicability of the e-squad approach to session handoff.

Another interesting future question would be to address the local connectivity possibilities of an e-squad's devices. Until now, we have bluntly considered that all devices could communicate with one another on the same basis, essentially considering that they shared a same IP network. At the wireless era, though, nothing is less true! Each device has different protocol capabilities (WiFi, cellular network, Bluetooth, LoRa...), and every pair of devices has different alternatives to exchange information. One could leverage contributions from the Device-to-Device communication field [133, 134] to build network-aware e-squads.

Now that the interest of e-squads has been demonstrated for single-user applications, we turn our heads to the multi-user paradigm: can we federate e-squads to provide privacy-preserving services?

The next chapter addresses the universal problem of file exchange between users, while putting a major emphasis on the participating users' privacy. We have shown that one's e-squad could infer their user's behaviour—such information should unquestionably remain private.

Is a throng of low-end, churnful e-squads capable of providing competitive file exchange performance? How can we leverage the power of the people's e-squads while retaining their privacy? These are the questions we now address.

Chapter 5

Federating e-squads to build privacy-preserving applications

A classical use-case for multi-user interactions is file transfer between two persons. In this chapter, we will see how e-squads can be used to propose an anonymous file transfer solution. We will present a novel onion routing scheme for exchanging files that is both *stateless*, to allow any device joining the system to participate without prior setup, and *predictive*: it uses the e-squads' information to craft reliable onion routes, and enable private file exchange.

5.1 Introducing Spores

In this new era of wireless communication, we have come to expect high degrees of synchrony and interactivity with our devices. The growing importance of e-squads is profoundly modifying how we live [135–138], and has opened new areas of research such as *proxemics*, the study of proximity-based interactions in a world of ubiquitous computing [139]. As we meet people, we engage in multi-device interactions spanning across our respective e-squads: we co-edit projects, exchange digital content, play online and listen to collaborative playlists. Except that what looks like a multi-e-squad ballet is most often just a bunch of device-to-cloud duos.

As of now, the ubiquitous computing feat is mostly enabled by corporate cloud services. As was already told, this dependency comes at the price of privacy. Our interpersonal activities are thoroughly studied for dubious purposes, and data encryption, which is often advanced as a seal of trust, is insufficient [140, 141]. Metadata information (such as what a person shared with whom and when) has been shown to suffice to create detailed profiles about people [16]. Society is in a crucial need to limit the accessibility of such data to private companies and governments alike, in order to have the privacy needed to develop our own intimate lives, associations and political ideas.

Digital content being at the centre of our numeric lives, their creation and exchange should happen freely. In the previous chapters, we facilitated the secure usage of applications among one's e-squad, enabling content creation aside from eavesdroppers. In this chapter, we tackle the private exchange of files between two persons.

Fully anonymous file exchange is a notoriously hard challenge. The last two decades have witnessed countless attempts at building privacy-preserving and anonymous P2P data-sharing networks [142–146]. These approaches typically leverage *onion routing* [147], with some improvements to perform node discovery in a distributed manner, a step that is otherwise centralized in traditional onion routing schemes. However, because of the high churn typically experienced by P2P networks [132], the performance of these systems has been discouraging [148]: churn causes routes to disappear, they must be reconstructed frequently, and this is particularly costly.

By contrast, Tor [67], one of the most popular anonymity networks, trades lower anonymity guarantees for a much better performance. Tor's lower protection arises from two key characteristics: (i) a partially centralized design where a set of ten *directory servers* keep the registry of online routers to pave onion routes and (ii) a small amount of highly available onion routers (around six thousands) compared to the number of connected users (between two and three millions).

In this chapter, we demonstrate how good performance can be reconciled with strong privacy protection in file transfer systems. We present *Stateless Predictive Onion Routing for E-Squads* (SPORES), a novel fully decentralized, autonomous, and self-organizing file transfer service between the e-squads of different users that preserves privacy in terms of both data *and* metadata. Our proposal hinges on three key ideas: *Firstly*, the emergence of e-squads makes it possible to apply P2P technology to one's own devices, thereby reducing one's exposure to security and privacy threats from third parties. *Secondly*, we consider *proxemics relationships* as a key enabler to initiate file transfers. Physical proximity allow users to share routing information out-of-band. In contrast to state-of-the-art P2P systems, users of SPORES agree out-of-band to share a file, but do not keep it available for anyone else in the future. *Thirdly*, we exploit distributed machine learning techniques to predict the future availability of user devices, and use this ability to implement a *probabilistic predictive onion routing* protocol. Our protocol exploits a new mix-header format that uses onion layers made of sets of nodes rather single nodes. The use of multiple candidate nodes in individual layers, together with our estimation of future device availability, exploited in a stateless way, allows us to overcome the inherent fragility of pure P2P file

transfer schemes while providing strong privacy properties.

Our contributions are the following:

— We propose a novel mix-header format for set-based onion routing which allows each layer of an onion route to contain several alternative next nodes.

— Building upon it, we introduce Probabilistic Onion Routing (POR), a protocol that allows stateless onion routing of messages. The several candidate relays per hop, along with its statelessness, make POR tailored for networks with high churn: a route remains available as long as a single node is online per layer, and nodes need no bootstrap phase to start forwarding messages.

— We provide a protocol for an e-squad to predict the availability of its devices, using Markov models and e-squad overlays (based on SPRINKLER).

— We use POR to realize SPORES, a privacy-preserving file transfer service among users. SPORES uses the e-squads' availability predictions to craft reliable probabilistic onion routes despite the low-availability relays, and eschews any third-party storage or centralized bootstrapping service.

The remainder of this chapter is organised as follows: we first make an overview of our file exchange protocol, highlighting the security properties we wish to ensure. Then, section 5.3 presents SPORES and its sub-systems in details. We carry on with an evaluation of SPORES' performance in section 5.4. We come back to the security properties and limitations in section 5.5 before concluding this chapter in section 5.6.

5.2 System overview & desired security properties

SPORES is a decentralised system made of users' e-squads. Each device can communicate with any other device that it knows the address of (e.g. using the Internet). A user's devices communicate through their e-squad overlay, so they know the identity of their fellow e-squad members. Although, they do not know other devices' owners a priori, and learning that information should be hard. We also assume that users communicate some limited information out-of-band, i.e. through some other channel than the one described.

In the cloud setting, when Alice wants to send a file to Bob, she uploads it from one of her devices to the cloud, and provides Bob with a link to the file online, using another channel. Thereafter, Bob can retrieve the file from the cloud to one of his devices. This setting provides two main privacy properties:

— Except for the cloud provider, Alice and Bob are the only persons aware of their file exchange (including its nature, size, etc.);
— Alice and Bob learn nothing about each others' devices. The cloud provider knows the address of both the sender and receiver, though.

In the Spores setting, we want to preserve the aforementioned properties, while getting rid of the cloud third-party that could act as a spy. We want Alice and Bob to learn nothing about each other's devices. While helping in the file transit, no Spores participant should gain knowledge about the exchanged file, nor the identity of its sender and receiver (we want sender-receiver anonymity). In other words, Spores tackles the problem of friendly surveillance, where participants are honest-but-curious about what is being done on the network.

Spores does not intend to overcome the Global Passive Adversary (GPA), where an attacker can listen on all the communication pipes. This attack setting is notoriously hard to overcome [67]; its circumvention almost always involves generation of cover traffic [145, 149–151], which is incompatible with the constrained resources of mobile end-devices.

We do take for granted that subsets of nodes (at the very least, e-squads) are colluding into breaking the participants' anonymity, making Spores subject to *traffic confirmation attacks* [152], where an attacker observing both ends of a circuit can link the sender to the receiver. To circumvent the latter, we argue that drastically increasing the number of onion relays effectively drowns the attacker in an ocean of rightful peers, thus making the attack impracticable. Tor, for instance, only has about six thousands active relays, for two to three million online users. By letting each user's e-squad take part in the routing process, the number of relays jumps by three orders of magnitude.

5.3 Our approach

We now detail the operation of our anonymous file-sharing service for e-squads. As shown on figure 5.1, from bottom to top, Spores is constituted of two network overlays, the Probabilistic Onion Routing (Por) protocol, and the file exchange service. The latter creates probailistic onion routes (Pors) to exchange files in a private way. The devices' addresses constituting the onion routes are collected using the global overlay. Routes are only created *once*, during the file exchange bootstrap. The e-squad overlays compute each device's probability of remaining online in the near future, which allows the route creation

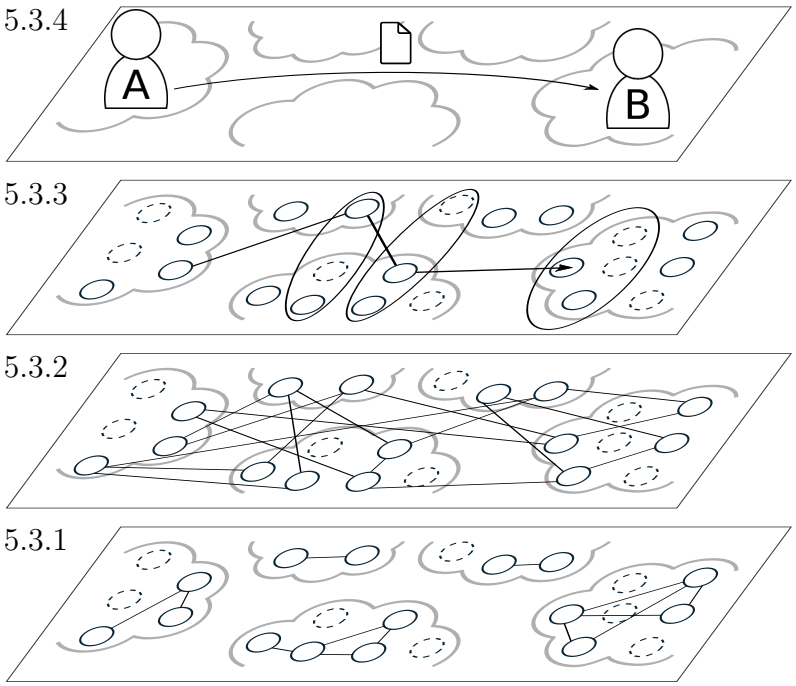


Figure 5.1 – The subsystems constituting SPORES: the e-squad overlay enabling communication inside each e-squad, presented in section 5.3.1; the global overlay allowing all devices to reach each other, described in section 5.3.2; the Stateless Probabilistic Onion Routes (PORs) protocol, that will be covered in section 5.3.3; finally, we will present SPORES, the anonymous e-squad file exchange protocol in section 5.3.4.

process to try to maximize the future availability of the routes.

We now detail each sub-system from the ground up: we first present the e-squad overlay and its several purposes, before covering the global overlay. We then specify the POR protocol, before finally diving into our anonymous file exchange service.

5.3.1 The e-squad overlay

At the root of SPORES is predictive routing. Devices must predict and publish their own probability of remaining online, which depends on their user's behaviour. This information is then used to create dependable stateless onion routes (PORs).

As was seen in the previous chapters, devices need to collect information on their user to make such predictions. For that reason, we build yet another variation of the SPRINKLER Gossiper protocol presented in section 3.2.2, that lets each e-squad device learn the global sequence S of their user's behaviour by gossiping every new update among them.

In the e-squad overlay, devices inform each other every time they turn on (which will allow them to model their user's behaviour), but also share every file exchange information their user participates in (as an uploader or receiver). Although file exchanges only have one sender and one receiver device, the latter information allows any e-squad member to receive chunks and acknowledgements while the end-device is offline (more on the topic in part 5.3.4).

Sharing the user's behaviour

The e-squad overlay protocol exactly mirrors the Sprinkler Gossiper (section 3.2.2) with acknowledgements (added in Cascade 4.2.2 to ensure good performance despite devices' churn). The user's activity is seen as an ever-growing sequence S of timestamped interactions r_i. Since interactions are timed, and no two interactions happen at once, there is a total order of the interactions in the sequence.

Contrary to previous implementations, interactions are not only about device usage anymore. They comprise both device usage events, and file exchange events. A file exchange is tied to a single device (sender or receiver), and all the e-squad needs to know what device is involved in which file exchange. To this end, an interaction r is constituted of the following fields:

$$r = (ts, d, \texttt{typ}, f) \in \mathbb{R} \times \mathcal{D} \times \mathcal{T} \times \mathcal{F}$$

such that: $ts \in \mathbb{R}$ is the interaction timestamp, $d \in \mathcal{D}$ is the descriptor (see section 5.3.2) for the device where the interaction r took place ($\mathcal{D}$ being the set of the user's devices), $\mathcal{T} = \{\text{USE}, \text{DL}, \text{UL}\}$ is the set of interaction *types* (resp. device usage, new file download, and new file download). When $\texttt{typ} = \text{DL}$ (resp. UL), $f \in \mathcal{F}$ contains the unique ID of the file that just started downloading (resp. uploading) on d. When $\texttt{type} = \text{USE}$, it means that device d was connected at time ts. Devices issue a USE message when they are grabbed, and every T seconds when they remain connected.

Modelling the user's behaviour

Given the sequence of devices' usage $S^U = \{r \in S, r.\texttt{typ} = \text{USE}\}$, each device needs to compute its own probability P_d of staying online in the near future, that they will publish to the overall network. We propose a simplistic model, that could be much improved using more user data and a more complex prediction. Still, we demonstrate in our evaluation

that a simple prediction like the following suffices at providing convincing improvements for the PORs availability.

First of all, using only S^U, each device builds an availability sequence $X = X_1, \ldots, X_i, \ldots$, where X_i contains the set of online devices during the interval $[t_i, t_{i+1}[$ (see equation 5.1). The observation sequence has a period of T: $\forall i, t_{i+1} = t_i + T$. The sequence X can be represented as a 2D sparse matrix of booleans, as shown in the table 5.1, that represents the devices usage of a user owning a laptop, a phone, and a smart-TV.

Table 5.1 – Sample of an e-squad availability sequence X, as a 2D boolean matrix.

	$\cdots$	X_{i-1}	X_i	X_{i+1}	$\cdots$
Laptop	$\cdots$	1	0	0	$\cdots$
Phone	$\cdots$	1	1	1	$\cdots$
TV	$\cdots$	0	1	0	$\cdots$

$$X_i(d) = 1 \iff \exists r \in S^U, r.d = d \wedge t_i \leq r.ts < t_{i+1}. \tag{5.1}$$

Now, to predict P_d, we consider that the stochastic process X follows the Markov property: 'the future only depends on the present, not on the past'. We use the hypothesis in equation 5.2.

This allows us to state that the probability for d to be online in the near future only depends on its probability to stay online after the current round $X_i = x$. To estimate this probability, we simply count [1]the number of times the current situation x led to a situation where d was also online (equation 5.3):

$$P_d = P\left[X_{i+1}(d) = 1 \mid X_i = x, \ldots, X_0 = x_0\right] = P\left[X_{i+1}(d) = 1 \mid X_i = x\right] \tag{5.2}$$

$$= \frac{\left|\{X_j \in X, X_j = x \wedge X_{j+1}(d) = 1\}_{0 \leq j < i}\right|}{\left|\{X_j \in X, X_j = x\}_{0 \leq j < i}\right|} \tag{5.3}$$

Alas, given the high dimensionality of the state space, it is very likely that x was never seen in X until now, leading to an undefined P_d. In such a case, we fall back on P'_d, an estimate of the probability that d stays online two turns in a row:

1. Because we work with low-probability events observed with small amounts of data, there is a possibility that an event never occurs in X. To counter that, we apply *add-one smoothing* [153] while computing probabilities. We left this engineering optimization out of the demonstration for clarity.

$$P'_d = \frac{\left|\{X_j \in X, X_j(d) = 1 \wedge X_{j+1}(d) = 1\}_{0 \leq j < i}\right|}{\left|\{X_j \in X, X_j(d) = 1\}_{0 \leq j < i}\right|}$$

When x was never observed, P_d takes the value of P'_d.

5.3.2 The global overlay

To create PORs, devices need to know the address, encryption key, and probability of remaining available of some nodes in the system. Given the decentralised nature of SPORES, we cannot rely on a central registry of online peers as e.g. Tor does. Instead, we deploy a global Random Peer Sampling (RPS) service [79, 154].

Essentially, each node maintains a view $\mathcal{V}_{\text{RPS}}$ containing $l_{\mathcal{V}}$ other devices' descriptors. Every T_{RPS} seconds, the view is updated as follows: a device d pops the oldest descriptor d' from its view, then swaps a predefined number of l_{gossip} elements from $\mathcal{V}_{\text{RPS}}$ with d'. Both devices add a fresh descriptor of themselves to the view exchange. If d' was offline, its descriptor is simply removed from d's view, with no further modification to $\mathcal{V}_{\text{RPS}}$.

This allows for two things: firstly, each device's view contains a constantly changing random sample of participating devices; secondly, stale descriptors get removed from one's view after a bounded time, such that $\mathcal{V}_{\text{RPS}}$ mostly contains online devices' descriptors.

Given their epidemic nature, RPS services are very sensitive to Byzantine attacks, where malicious nodes gossip bad views in order to disrupt the randomness of the neighbourhood graph. Several proposals overcome this limitation, sometimes by relying on a trusted third-party [80], sometimes by asserting the validity of received messages [81, 82]. We leverage on the latter, so as to remain entirely decentralised.

In SPORES, each device d computes its own asymmetric key pair (pk_d, sk_d), and publishes its descriptor to remote peers. It contains:

— the node's address $@_d$;
— its public key pk_d;
— its estimated probability of staying online P_d.

5.3.3 Pors: Probabilistic Onion Routes

Let us first describe the principles of onion routing as seen in Tor [67], as of today the most mature anonymity network for anonymous communications.

Onion routing makes connections between a client (say Alice) and their correspondent (Bob) go through two or more servers (or relays) before reaching their destination. With Tor, to create a route, Alice randomly picks three relays to constitute the path, and incrementally establishes TLS connections to each of them through the route. Once the route is established, it constitutes a persistent two-way TCP stream, although the traffic is internally chunked into fixed-size messages (or cells). Cells contain a header and a payload, that are encrypted altogether by the client several times: once per relay. Upon reception of a cell from the sender to its destination, each relay deciphers it using the encryption keys negotiated during the TLS connection bootstrap. Bob finally receives the message originally written by Alice, and can answer back on the same pipe. Messages on this direction are incrementally encrypted by the relays, such that Alice receives Bob's message hidden under three layers of encryption. She decrypts it using the keys that were negotiated with the relays during the connection establishment.

The anonymizing property stems from the fact that each hop $\mathcal{L}_i$ only knows the address of the previous relay $\mathcal{L}_{i-1}$ (that sent the message) and the address of the next $\mathcal{L}_{i+1}$ (determined at the connection's establishment). Given that routes contain two or more hops, no intermediary knows both the sender and the receiver of a message, thus making the communication anonymous.

Several relays per layer The basic idea of Probabilistic Onion Routing (POR) is depicted in figure 5.2: each message sent from Alice to Bob can pass through several candidate nodes at each hop, instead of only one in traditional onion routing.

In onion routing, when any of the relays becomes unavailable, the route is broken and a new one needs to be created. The prime interest of PORs is that they are resilient to intermediaries churn: we only need one online relay per layer for the route to function. In practice, when a node from layer $\mathcal{L}_i$ has a message to transmit, it tries sending it to each device in layer $\mathcal{L}_{i+1}$ in random order, until it succeeds or all attempts fail. In the latter case, the message is dropped.

In contrast to Tor, PORs do not create TLS connections, which would be inapplicable with several nodes per hop. Instead, routes are stateless: all routing information is contained inside the header, as we will now see.

Por header Figure 5.3 shows the format of POR headers (along with an example of a full header as sent by Alice in figure 5.2). A message's header $\mathcal{H}_i$, as received by any

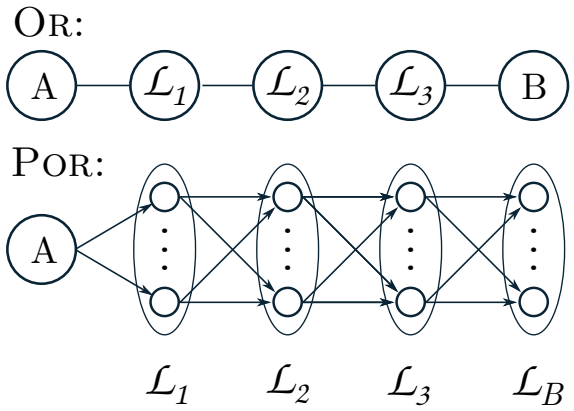


Figure 5.2 – Two different onion ways for Alice (A) to anonymously communicate with Bob (B). In Onion Routing (Or), each layer $\mathcal{L}$ is constituted of only one node. In Probabilistic Onion Routing (Por), there are several *candidate* nodes that each message can go through at each layer.

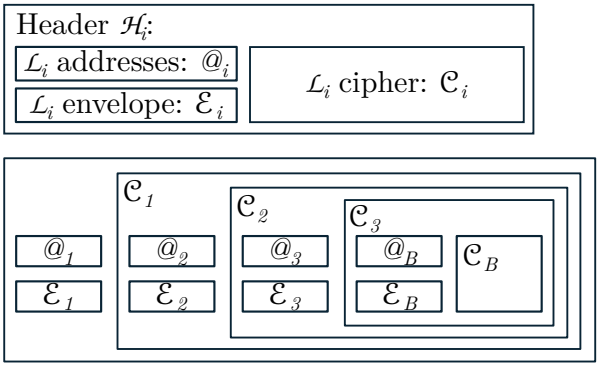


Figure 5.3 – Format of a Por header (on top), and the full header as sent by Alice in figure 5.2. Each cipher $\mathcal{C}_i$ is encrypted using a symmetric key hidden in the envelope $\mathcal{E}_i$, only readable by members of layer $\mathcal{L}_i$. $\mathcal{C}_B$ can contain arbitrary data, including nil.

member of $\mathcal{L}_i$, is constituted of three parts:

$$\mathcal{H}_i = (@_i, \mathcal{E}_i, \mathcal{C}_i)$$

— $@_i$: The addresses of all members of the current layer $\mathcal{L}_i$, used by nodes of $\mathcal{L}_{i-1}$ to forward the present message;
— $\mathcal{E}_i$: An envelope, destined to $\mathcal{L}_i$, that will allow them to decrypt the cipher $\mathcal{C}_i$.
— $\mathcal{C}_i$: A cipher, that can be deciphered by any member of $\mathcal{L}_i$ using $\mathcal{E}_i$. It contains the next hop's header $\mathcal{H}_{i+1}$, notably the addresses of $\mathcal{L}_{i+1}$'s nodes ($@_{i+1}$) to which the current node should forward the decrypted header and its payload.

When the message reaches Bob, the cipher will not unravel into a header, which will tell him that this message is for him.

By getting rid of TLS connections in favour of header-based routes, Por enables *stateless* routing: no prior communication is needed with relays to establish onion routes, they simply decipher any received message's header and forward them to the next layer, as indicated by the header. This is particularly interesting for short-lived nodes such as seldom connected personal devices as we target: they can participate in the system as soon as they join, without any bootstrap phase. Their disconnection does not mandate a new route construction.

On the other hand, PORs are connectionless one-way channels (UDP-like), and the header is not fixed in size due to the lack of re-encryption between each hop. In particular, POR does not guarantee messages integrity nor order (each cell potentially travelling through a different path). It is the role of the upper abstraction layer (e.g. our file exchange protocol SPORES) to guarantee reliable & ordered transmission.

Cryptographic primitives The encrypted header $\mathcal{C}_i$ containing the addresses of the next layer needs to be decipherable by any of the current layer $\mathcal{L}_i$'s members, and only by them. This cryptographic scheme is coined Broadcast Encryption (BE) [155]. We derive our encryption process from Hybrid Encryption [156] (as used in PGP), where a message $\mathcal{M}$ is encrypted into a cipher $\mathcal{C}$ using a unique symmetric key k (e.g. using AES). Each member of the group $\mathcal{L}_i$ must be given this key, which is the purpose of the envelope $\mathcal{E}$. It contains the concatenation of k encrypted with each member's public key (using e.g. RSA). Upon reception of a ciphered message $(\mathcal{E}, \mathcal{C})$, a peer attempts to decrypt each portion of the envelope with its private key, until it succeeds (and gets k to decrypt $\mathcal{C}$) or fails.

We write down our broadcast encryption/decryption algorithms in algorithm 5.1, and its application to header cryptography in algorithm 5.2. Below, we present our cryptographic primitives, and explain our algorithms:

Let $\mathcal{C} \leftarrow SE(\mathcal{M}, k)$ and $\mathcal{M} \leftarrow SD(\mathcal{C}, k)$ be symmetric primitives for encrypting/decrypting with key k, such that SD returns $\perp$ on decryption failure. The cipher size $|\mathcal{C}|$ is the smallest multiple of the symmetric block size S_{sym} superior or equal to the message size $|\mathcal{M}|$. We write that $|\mathcal{C}| = \lceil |\mathcal{M}| \rceil_{S_{\text{sym}}}$

Let $\mathcal{C} \leftarrow AE(\mathcal{M}, pk)$ and $\mathcal{M} \leftarrow AD(\mathcal{C}, sk)$ be asymmetric ones for encrypting/decrypting with the public key pk (resp. secret key sk), such that AD returns $\perp$ on failure. The cipher size $|\mathcal{C}|$ is always equal to the asymmetric block size S_{asym}, such that the message should always be shorter or as long as the block size.

$(\mathcal{E}, \mathcal{C}) \leftarrow BE(\mathcal{M}, pk_{\mathcal{L}})$ — Given a layer $\mathcal{L}$'s public keys $pk_{\mathcal{L}}$, and a message $\mathcal{M}$ to encrypt, *BE broadcast encrypts* $\mathcal{M}$ by outputting an envelope $\mathcal{E}$ containing a symmetric key k encrypted with each $pk \in pk_{\mathcal{L}}$, and a ciphertext $\mathcal{C}$ containing the message encrypted with k.

$\mathcal{M} \leftarrow BD(\mathcal{E}, \mathcal{C}, sk)$ — Given an envelope $\mathcal{E}$, a ciphertext $\mathcal{C}$ and a secret key sk, *BD broadcast decrypts* the message $\mathcal{M}$ containing the expected plaintext, or $\perp$ if the decryption failed.

Algorithm 5.1 The Broadcast Encrypt/Decrypt algorithms

```
1: function BE(M, pk_L)
2:     k ← random symmetric key
3:     C ← SE(M, k)
4:     E ← {AE(k, pk_i)}_{pk_i ∈ pk_L}
5:     return E, C
```

```
1: function BD(E, C, sk)
2:     for e ∈ E do
3:         k ← AD(e, sk)
4:         if k ≠ ⊥ then
5:             return SD(C, k)
6:     return ⊥
```

Algorithm 5.2 The Header Encrypt/Decrypt algorithms

```
1: function HE(H, L)
2:     for L ∈ reverse(L) do
3:         H.E, H.C ← BE(H, L.pk)
4:         H.@ ← L.@
5:     return H
```

```
1: function HD(H, sk)
2:     return BD(H.E, H.C, sk)
```

In algorithm 5.2, we write $\mathcal{L}.pk$ and $\mathcal{L}.@$ to refer to a layer's nodes' public keys and addresses. $\mathbf{L} = \{\mathcal{L}_i\}_i$ is an array of layers. `reverse`($\mathbf{L}$) means we iterate on $\mathbf{L}$ in reverse order (starting from the last element).

$\mathcal{H}' \leftarrow HE(\mathcal{H}, \mathbf{L})$ — Given an inner header $\mathcal{H}$ (that can be any message including nil), and an array of layers $\mathbf{L}$, HE recursively encrypts $\mathcal{H}$ for each $\mathcal{L} \in \mathbf{L}$ starting from the last layer.

$\mathcal{H}' \leftarrow HD(\mathcal{H}, sk)$ — HD attempts to decrypt a layer of the header $\mathcal{H}$ using the secret key sk. It returns $\perp$ on failure.

In the end, what does a header weigh? The envelope $\mathcal{E}_i$ is built of the concatenation of asymmetric ciphers for each of layer $\mathcal{L}_i$'s members: $|\mathcal{E}_i| = S_{\text{asym}} * |\mathcal{L}_i|$. Each header cipher $\mathcal{C}_i$ being symmetrically encrypted, its size depends on the inner header's size and the symmetric block size: $|\mathcal{C}_i| = \lceil |\mathcal{H}_{i+1}| \rceil_{S_{\text{sym}}}$. Let us not develop $\mathcal{H}_B$ as it can cary any data. Finally, assuming that each address weighs an identical size $|@| \ll S_{\text{asym}}$, and that the number of nodes per layer is equal for each layer, we see that the header size $S_{\mathcal{H}}$ grows linearly with the number of layers $\#_{\mathcal{L}} = |\mathbf{L}|$, the number of nodes per layer $S_{\mathcal{L}}$ and the asymmetric block size S_{asym}:

$$
\begin{aligned}
&|\mathcal{H}_i| = \lceil |\mathcal{H}_{i+1}| \rceil + S_{\mathcal{L}} \times (S_{\text{asym}} + |@|) < |\mathcal{H}_{i+1}| + S_{\text{sym}} + S_{\mathcal{L}} \times (S_{\text{asym}} + |@|) \\
&\implies \quad S_{\mathcal{H}} < \lceil |\mathcal{H}_B| \rceil + \#_{\mathcal{L}} \times S_{\text{sym}} + \#_{\mathcal{L}} \times S_{\mathcal{L}} \times (S_{\text{asym}} + |@|) \\
&\implies \quad S_{\mathcal{H}} = O\left(\#_{\mathcal{L}} \times S_{\mathcal{L}} \times S_{\text{asym}}\right) \qquad (5.4)
\end{aligned}
$$

We dismissed newer private broadcast encryption schemes [157, 158] that propose constant decryption time or smaller sizes by trading-off higher encryption cost, as our layers have a bounded configurable size that should never exceed tens of devices (as will be explained in section 5.3.4).

Algorithm 5.3 Receiving a POR message on d

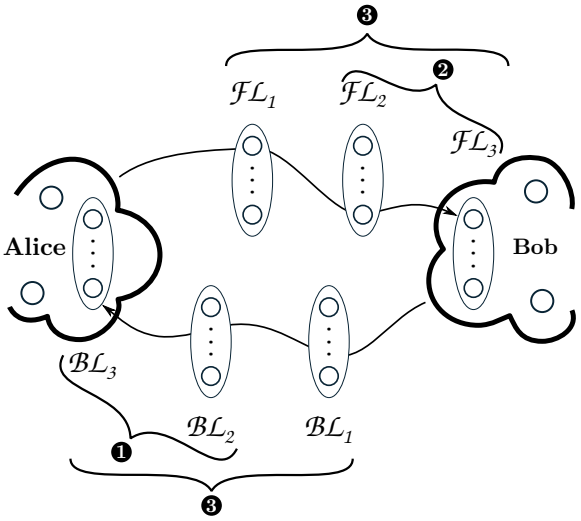
```
1: on receive M = ⟨H, P⟩
2:     H' ← HD(H, sk_d)
3:     if H' = ⊥ then
4:         return ⊥        ▷ Decryption failed
5:     else if H' ≠ header then
6:         process P       ▷ I am the recipient
7:     else
8:         Forward(H', P)
```

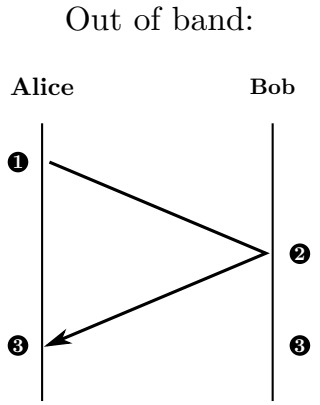
```
1: function FORWARD(H', P)
2:     for @ ∈ random(H'.@) do
3:         if send(@, ⟨H', P⟩, T_out) then
4:             return ⊤            ▷ Success
5:     return ⊥    ▷ All offline: drop mess.
```

Forwarding messages We finally display the message reception and forwarding procedure in algorithm 5.3, that runs on any node participating in SPORES (we consider a device d). Upon reception of a message $\mathcal{M} = \langle \mathcal{H}, \mathcal{P} \rangle$, d first attempts to decipher the header $\mathcal{H}$ using its secret key sk_d. Three cases are possible: either the decryption fails, which constitutes an error—the message is dropped; either the output is not a header, in which case the message is destined to d; either d obtains a header $\mathcal{H}'$, in which case it forwards the message to the next layer using `Forward` function. The latter iterates over each address @ in $\mathcal{H}'.@$ in random order, and attempts to send the message to @. The `send` function called at line 3 takes three parameters: the recipient's address, the message to send, and a timeout duration. T_{out} is a configuration parameter, usually below a second. If the `send` call succeeds, the message is duly forwarded. If the for loop returns without any successful attempt, all the next layer is considered offline, and the message is dropped. Note that it takes $\#_{\mathcal{L}} \times T_{\text{out}}$ seconds to drop a message when the next layer is offline.

5.3.4 Spores: File exchanges through Por

We now have all the building blocks to perform anonymous file transfers using e-squads. In this section, we first present how two users agree upon probabilistic onion routes for their exchange, including the intelligent selection to maximise the routes' availability; finally, we discuss the file exchange protocol built atop PORs.





Algorithm 5.4 Route initialisation by device d_A uploading file f

1: **function** InitUpload(f, θ)
2: $fd \leftarrow$ `BuildFileDescriptor`(f)
3: $\mathtt{sq} \leftarrow \{r.d, r.d \neq d_A\}_{r \in S}$
4: $\mathcal{BL}_3 \leftarrow$ `PickLayer`($\mathtt{sq}, \theta$) $\cup \{d_A\}$
5: $\mathcal{BL}_2 \leftarrow$ `PickLayer`($\mathcal{V}_{\text{RPS}}, \theta$)
6: $\mathcal{BH}_1 \leftarrow HE\,(fd.\mathtt{ID}, [\mathcal{BL}_2, \mathcal{BL}_3])$
7: send $\langle fd, \mathcal{BH}_1 \rangle$ to d_B $\longrightarrow\longrightarrow\longrightarrow$
8: **on receive** $\langle \mathcal{FH}_1 \rangle$ $\longleftarrow\longleftarrow\longleftarrow\longleftarrow$
9: $\mathcal{FL}_1 \leftarrow$ `PickLayer`($\mathcal{V}_{\text{RPS}}, \theta$)
10: $\mathcal{FH} \leftarrow HE\,(\mathcal{FH}_1, [\mathcal{FL}_1])$
11: $r \leftarrow (\mathtt{Now}(), d_A, \text{UL}, fd.\mathtt{ID})$
12: $S \leftarrow S \cup \{r\}$ ▷ Shared to e-squad
13: Start sending f

Algorithm 5.5 Route initialisation by device d_B downloading file f

1: **on receive** $\langle fd, \mathcal{BH}_1 \rangle$
2: $\mathtt{sq} \leftarrow \{r.d, r.d \neq d_B\}_{r \in S}$
3: $\mathcal{FL}_3 \leftarrow$ `PickLayer`($\mathtt{sq}, \theta$) $\cup \{d_B\}$
4: $\mathcal{FL}_2 \leftarrow$ `PickLayer`($\mathcal{V}_{\text{RPS}}, \theta$)
5: $\mathcal{FH}_1 \leftarrow HE\,(fd.\mathtt{ID}, [\mathcal{FL}_2, \mathcal{FL}_3])$
6: send $\langle \mathcal{FH}_1 \rangle$ to d_A
7: $\mathcal{BL}_1 \leftarrow$ `PickLayer`($\mathcal{V}_{\text{RPS}}, \theta$)
8: $\mathcal{BH} \leftarrow HE\,(\mathcal{BH}_1, [\mathcal{BL}_1])$
9: $r \leftarrow (\mathtt{Now}(), d_B, \text{DL}, fd.\mathtt{ID})$
10: $S \leftarrow S \cup \{r\}$ ▷ Shared to e-squad
11: Start receiving f

Figure 5.4 – Schematic and algorithms of the out-of-band routes creation process between Alice (sender of file f) and Bob (receiver). At ❶ (line 1 of algorithm 5.4), Alice's uploading device, d_A, computes the end of the route to reach Alice and the file descriptor fd, before sending them to Bob's receiving device d_B (line 7). Then, at ❷ (algorithm 5.5 line 1), d_B builds the end of the route to reach Bob, and sends it back to d_A (line 6). Finally, at ❸ (alg. 5.4 l. 9 and alg. 5.5 l. 7), both devices finish the routes to reach each other, share the file exchange information to their respective e-squad, and start exchanging f.

Routes creation Figure 5.4 depicts the creation process of a route between our beloved Alice (uploader) and Bob (the receiver). As already mentioned, we leverage on proxemics relationships, such that the handshaking process initialising a file exchange is performed out-of-band (on another communication channel such as Near Field Communication (NFC), LAN, Bluetooth, carrier pigeon, or else). The initialisation serves two purposes: to provide Bob with the exchanged file metadata (we come back to it in the next paragraphs), and to decide upon the PORs that will be used throughout the transfer.

Since PORs are one-way only, Alice and Bob need to agree upon two routes: a *forward* one, form Alice to Bob, that will carry file chunks, and a *backward* route, from Bob to Alice, that will transport Bob's acknowledgements of the chunks. Furthermore, as SPORES heralds sender-receiver anonymity, both parties build a portion of the route, so as to maintain their end of the route hidden inside the encrypted header.

Figure 5.4 details how this feat is done:

— At ❶, Alice's sending device d_A crafts the inner part of the header for the route to herself, $\mathcal{BH}_1$, on algorithm 5.4 at lines 3-6. The final layer $\mathcal{BL}_3$ is only constituted of Alice's devices: d_A picks candidates from her e-squad sequence S (line 3), and lastly adds its own descriptor to $\mathcal{BL}_3$ (l. 4). Their addresses are hidden inside a layer of encryption, and will only readable by nodes of $\mathcal{BL}_2$ (in particular, they will not be accessible to Bob). On line 7, d_A sends $\mathcal{BH}_1$ to Bob, along with the file metadata fd.

— At ❷, Bob's receiving device d_B builds its half of the forward route ($\mathcal{FH}_1$) just like d_A did at ❶, see algorithm 5.5 lines 2-5. Again, $\mathcal{FL}_3$ is only made of Bob's e-squad, but its addresses are hidden in a layer of encryption. d_B sends $\mathcal{FH}_1$ back to d_A on line 6.

— Finally, at ❸, both devices bootstrapping before starting the file exchange. They first finish the route they will use to reach the other end ($\mathcal{FH}$ on alg. 5.4 l. 9-10 & $\mathcal{BH}$ on alg. 5.5 l. 7-8), then inform their e-squad that they started sharing a file by adding an interaction r to their sequence S (resp. l. 11-12 and l. 9-10), and finally start exchanging f (line 13 and 11 respectively).

Relays selection We now detail the `PickLayer`$(\mathcal{V}, \theta)$ function, that takes care of intelligently selecting a layer's devices. It takes two parameters: an input set of candidate nodes $\mathcal{V}$, and the *availability threshold* $\theta \in]0, 1]$, a configuration parameter that represents the desired maximum probability that all of the layer's nodes fall offline at the same

time (i.e. the probability that the layer be unavailable).

`PickLayer` iteratively picks a *random* node from $\mathcal{V}$ without replacement, adds it to the output layer $\mathcal{L}$, and computes the *probability that all of the layer's nodes fall offline at once*, $P_{\mathcal{L}}^{\text{off}}$:

$$P_{\mathcal{L}}^{\text{off}} = \prod_{d \in \mathcal{L}} 1 - P_d \tag{5.5}$$

P_d being the probability that device d remains online (cf. equation 5.2). The function returns either when the offline probability $P_{\mathcal{L}}^{\text{off}}$ falls below the threshold θ, or when the input view $\mathcal{V}$ is emptied. As a baseline that will be used in the evaluation, `PickLayer` randomly chooses a predetermined number of nodes from the input view, without caring for the layer's probability of becoming unavailable.

Non-e-squad layers are built with $\mathcal{V}_{\text{RPS}}$ as input: it comprises a random pool of global SPORES participants that were online not long ago at least (cf. section 5.3.2). The RPS view size $l_{\mathcal{V}}$ should be chosen big enough for `PickLayer` to reach the configured θ, but small enough that the view's stale descriptors get evicted in a reasonable amount of time. E-squad layers, on the other hand (that is, $\mathcal{FL}_3$ and $\mathcal{BL}_3$), only have the less numerous e-squad members as input, such that `PickLayer` might not be able to reach the threshold before emptying the candidate list.

The smaller the threshold θ, the more nodes per layer, the better the route's availability, but also the bigger the header. There is a trade-off between the readiness of routes and the message transit overhead.

Finally note that randomly picking descriptor from one's view avoids biasing the relay selection in favour of supposedly highly connected nodes. Indeed, the devices' availability estimate is published by themselves, and should not be trusted. Our approach gives no interest for attackers to lie on this value, while it encourages everyone to provide good estimate, for the sake of the routes' reliability.

Exchanging a file As already told, POR enables anonymous UDP-like channels: order and integrity of the messages are not guaranteed by the protocol. These features must be supplied by the application running on top of POR—in our case: SPORES.

A file f exchanged through SPORES is chunked into fixed-size pieces, that are transmitted in order by the sender, along with their position (or ID). To ensure chunks integrity, we borrow from BitTorrent [68], as was done in section 4.2.1: the file descriptor fd that is computed with `BuildFileDescriptor`(f) and provided to the receiver on bootstrap

notably contains a `SHA1` hash per chunk. The receiver verifies that the expected and computed hashes match every time they receive a chunk. `BuildFileDescriptor`(f) creates the following descriptor:

$$fd = (\texttt{ID}, \texttt{size}, \texttt{chunkSize}, \texttt{\#Chunks}, \texttt{chunksHash}, \texttt{hash})$$

Each file is given a unique, random `ID`, picked by the uploader. The file descriptor also provides the file `size`, number of chunks and chunk size. The `chunksHash` is the concatenation of each chunk's SHA1 hash, used by the receiver to verify the integrity of each chunk. Finally, `hash` is the SHA1 hash of `chunksHash`, to verify the integrity of `chunksHash` upon file transfer initialisation. Using SHA1 hashes, we ensure the file integrity. The order is guaranteed by the following sliding-window protocol.

SPORES implements the Selective Repeat Automatic Repeat-reQuest (ARQ) [159–161] algorithm, that lets the sender send several chunks at once, and to allow the receiver to accept them out of order. The sender provides the chunk ID of each piece sent on the forward route, while the receiver sends back an acknowledgement (ACK) with the same ID for each received piece, using the backward route. When the sender does not receive an ACK after sending a chunk, it retries sending after a timeout of several seconds. The file exchange completes when the sender has received an ACK for each file chunk.

Finally, as can be seen in line 4 of algorithm 5.4 and line 3 of algorithm 5.5, any e-squad member can receive chunks/ACKs in spite of the proper message recipient. By comparing the innermost part of the receiver header (cf. alg. 5.4 line 6 and alg. 5.5 line 5) to the file IDs in their sequence S, each e-squad member knows to which device a message is destined, and can forward it to their recipient. If the receiver is currently offline, they forward the message to any online e-squad member, until the recipient comes back online and is able to finally receive the message. In essence, the whole e-squad acts as a cache for received messages while the actual recipient is offline.

By using several building blocks, mostly adapted from the literature, we have proposed an entirely decentralised anonymous file exchange service for e-squads. It is specifically tailored for networks with high churn, and, thanks to its gossip components, it can scale to a theoretically unbounded number of users. We now present our implementation of SPORES, and study its performance and efficiency.

5.4 Evaluation

To evaluate SPORES, we built a prototype in 6100 lines of Go, including all core functions except the cryptography. The users' behaviours, driving the devices churn, were simulated with 1600 lines of Python. Each device runs as a Docker container, participating in a unique virtual network. Due to the scale of the experiment, and to generate more realistic network traffic, each user's devices are scattered over a multi-host Docker Swarm.

We first describe our evaluation protocol, before presenting our results in section 5.4.2.

5.4.1 Testbed

We now present our user behavioural models, before going through our experimental setup.

User behavioural models

As for our previous experiments, the absence of real-world data fitting our needs—that is on daily personal usage of more than a few devices—incited us to produce fictitious models of user behaviour. Their goal is to exhibit patterns of devices connections and disconnections that are challenging, while being based on simple approximations.

We first consider that users switch location at fixed intervals of period T. Second, the users' device usage depends only on their location (one needs to have their device available to turn it on and off). Third, the Markovian hypothesis: the user's future location only depends on their last location. Finally, the usage probability is independent for each device.

In practice, each user switches between three different locations ('Home', 'Outside' and 'Work') according to the same Markov chain of order one (cf. top of figure 5.5 for its transition probabilities). Each user's device is modelled by a different Hidden Markov Model (HMM): given a hidden stochastic process (the user's location, shared by all the e-squad's HMMs), a device d has a given probability of being online at each location, represented by a observation matrix $B_d \in [0,1]^{3\times 2}$. The following represents the probability of the phone's availability across its user's location, using the same parameters as figure 5.5:

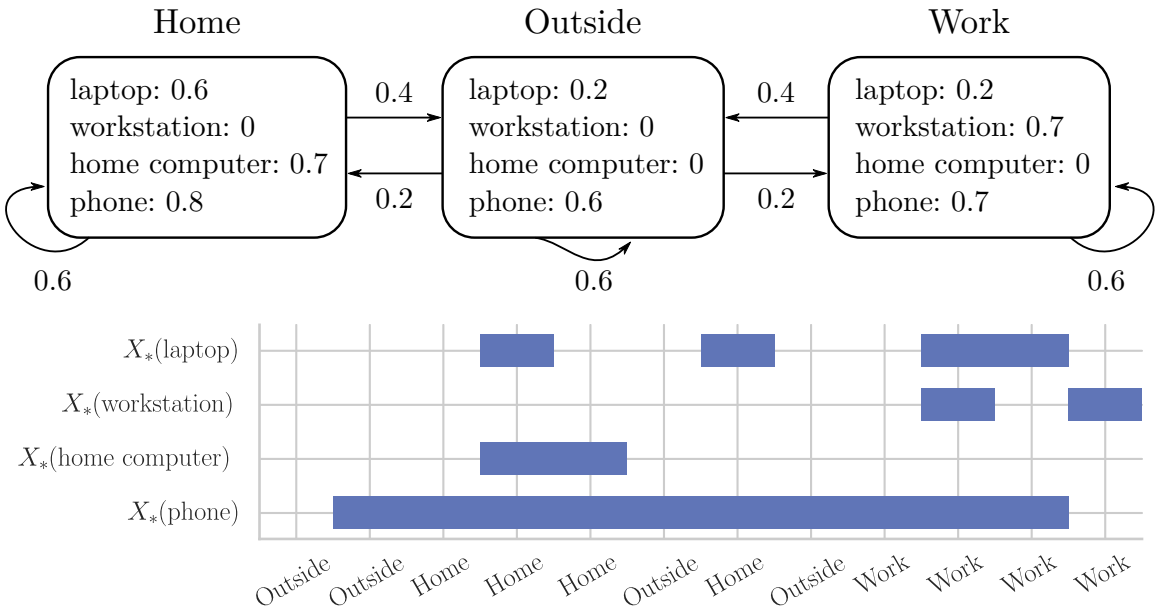


Figure 5.5 – Example model of a user's behaviour (on top) and sample devices connection timeline (bottom) with the location of the user on the x-axis. Each device has an independent probability of being online at each location: one Hidden Markov Model (HMM) per device share the same state (location) sequence.

$$B_{\text{phone}} = \begin{array}{c} \\ P(\text{phone}) \\ P(\overline{\text{phone}}) \end{array} \begin{array}{c} \begin{array}{ccc} \textit{Home} & \textit{Outside} & \textit{Work} \end{array} \\ \begin{bmatrix} 0.8 & 0.6 & 0.7 \\ 0.2 & 0.4 & 0.3 \end{bmatrix} \end{array}$$

We now describe the random generation process of a user's behaviour. Each user is given 3 to 10 devices of different types ('mobile', 'portable', 'fixed' or 'server'), including at least one mobile. Devices have 30% chances of being of each type, except for the server type, which has only 10% chances of being picked. Each device has randomly picked availability probabilities, depending on their device type:

— *Mobiles* can be accessed anywhere with a varying probability from 60 to 100%;
— *Portables* (laptops) can be accessed anywhere with a probability from 20 to 80%;
— *Fixed* appliances can only be accessed from a single location (and never outside), with a probability from 30 to 90%;
— *Servers* have a constant probability superior or equal to 90%.

The top part of figure 5.5 shows a possible output of this selection process. HMMs are generative probabilistic models, which allows the generation of observation sequences. The

bottom part of the figure displays a sample sequence of devices' availabilities, represented in the same way as the sequence X in table 5.1. In addition, the user's location is shown in the x-axis—this information is not given to the devices, though: it remains *hidden*.

This coarse approximation of an office worker using some to many devices provides us with unrealistic yet satisfactory availability sequences: they depict very unstable devices, that can turn on for brief periods of time, and sometimes leave the e-squad entirely disconnected. Future works could extend on having more realistic user behavioural models (through e.g. fields studies), and more complex, insightful availability predictions. We will show in section 5.4.2 that SPORES' reliability nevertheless benefits our simplistic availability inference.

Methodology

Let us describe our experimental tesbed (specific parameters are given afterwards). We deploy a bunch users—incarnated by their devices—on a Docker Swarm constituted of several hosts. The devices are Docker containers (running Go code) randomly spread among the machines, thus generating more realistic network latencies compared to a host-local deployment. When all containers are launched, we start each user's scheduler (in Python), that turns devices on and off according to their model. Once the bootstrap is over, we start exchanging files sequentially, according to a predefined planning, and leave consequent pauses between exchanges (to avoid saturating the network). Each transfer bootstrap faithfully matches the out-of-band initialisation explicated in section 5.3.4: we randomly pick two online devices from different users, and perform the out-of-band file exchange initialisation through REST calls to the devices. Devices then immediately start sharing the file.

In the following, we always compare duos of experiments, where the first serves as the baseline for the second one. To obtain comparable results between experiments, we fix the scheduler's random seed, thus creating identical users (with the same devices and the same activity sequences), and identical files exchanged at the same time using the same sender and receiver. However, the devices' behaviour is not the same across experiments: routes choice and message paths are different—there is still a fair amount of variability in the results.

The cryptographic functions from section 5.3.3 were not implemented in our prototype. This allowed us to gather more precise data about message exchanges, as relays could read each message's content; on the other hand, the header size is smaller without encryption

($O(\#_{\mathcal{L}} \times S_{\mathcal{L}} \times |@|)$), and the transfer times do not account for cryptographic computations.

Parameters The experiments were deployed on 3 Amazon AWS 'r5.large' VMs (with 2 Intel Xeon Platinum CPUs and 16GB or RAM), from one 't3.small' (with 2 vCPUs and 2GB of RAM) orchestrator, all on the same availability zone. The latter did not host any device, but ran the devices' scheduler and initiated every file exchange through REST calls to the participating devices. We deployed 12 users in each case, resulting in 77 devices per experiment on average (6.4 devices/user).

Users change location (and switch their devices' state) every $T = 10s$. The RPS period $T_{\text{RPS}} = 1s$, its view is bound to $l_{\mathcal{V}} = 30$, while peers exchange up to $l_{\text{gossip}} = 6$ descriptors per communication round. 10 RPS rounds take place before the connectivity of the devices change, which allow devices to recycle, on average, a third ($T/T_{\text{RPS}}/l_{\mathcal{V}}$) of their stale descriptors before the network state changes again. In other words, on average, $\mathcal{V}_{\text{RPS}}$ contains no descriptors older than 3 usage rounds. Between two file exchanges, we pause for 12 rounds, i.e. 2 minutes (to avoid having too many concurrent file exchanges on the network). We wait 60 rounds (10 minutes) after the last file exchange started before calling experiments to an end.

Routes always contain three layers (two over the whole network, plus one comprising only the recipient's e-squad members). Sprinkler Gossiper's parameters are the following: each new piece of information is spread up to $f = 3$ fellow e-squad members. As with previous chapters, the devices are fed with an initial knowledge of their user's behaviour of $L_{\text{init}} = 50$ previous rounds, allowing them to start off with acceptable predictions of their future availability (see section 5.3.1).

Unless otherwise noted, the availability threshold θ is 10^{-3}, and the maximum layer size $M_{\mathcal{L}} = 16$. We exchange 50 files of 100MiB per experiment, using a chunk size of 512KiB.

5.4.2 Conducted experiments

We now present experiments displaying several aspects of the Spores file exchange protocol. We first study the network dynamics of the Pors, before taking interest in Spores' resilience to the low availability of its peers.

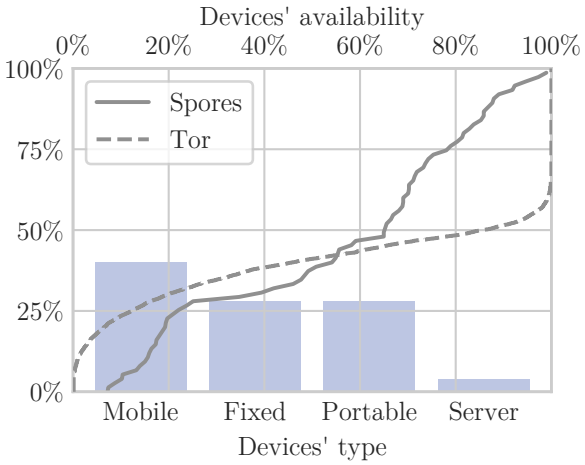


Figure 5.6 – Cumulative Distribution Function (CDF) of the devices' availability ratio over the whole time of the *predictive* experiment (full gray line), and proportion of each device type (pale blue bars). The availability of Tor's relays over the month of September 2018 is shown in dotted gray for comparison. Over 75 devices, 40% were mobiles, 28% were fixed or portables, and only 4% were servers. Half the devices are online at least 65.0% of the time, however 25% have an availability lower than 21.5%.

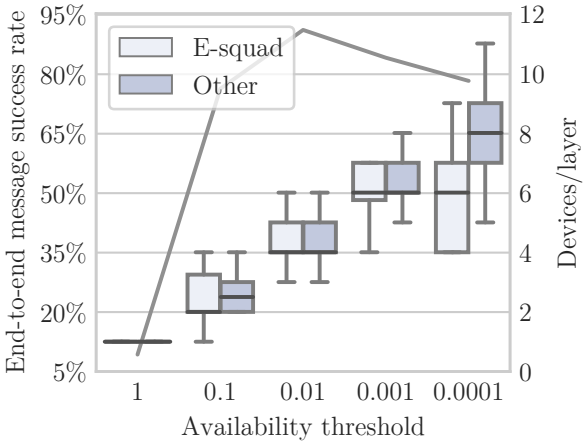


Figure 5.7 – Tukey boxplots [129] of the layer size per availability threshold θ (right axis, median in darker gray), and end-to-end messages exchange success rate per θ (grey line, left axis). We split the e-squad layers (that sometimes fail to reach θ) and the other layers (that always reach it). The exchange success rate is maximal for $\theta = 0.01$ (91.2%): more devices per layer adds overhead without benefits, while less devices per layer makes routes unreliable (at $\theta = 1$, the success rate is 9.28%).

Probabilistic onion routes

To observe the mechanics of PORs, we run two experiments with the same random seed (resulting in identical users, and identical file exchange times and participants). The first, *predictive* experiment assigns a different availability threshold θ to each file exchange, with $\theta \in \{1, 0.1, 0.01, 0.001, 0.0001\}$. The second experiment (coined *random*), does not make use of the devices availability estimate, and instead randomly picks a fixed number of devices per layer, with the layer size $S_{\mathcal{L}} \in \{1, 2, 4, 6, 8\}$. Since we exchange 50 files in total, 10 files are exchanged with each configuration.

Let us first observe the effects of the user sampling on the devices' types and availability, on figure 5.6. We remind that, because of the identical random seed, the *predictive* and *random* experiments have the same characteristics on that regard. 75 devices participated in each experiment (6.25 per user). Mobiles—which availability varies between 60 and 100%—were the most represented (40%), while servers only accounted for 4% of

the peers. However, less available fixed and portable devices still represent the majority of the peers (56%), leading to a quarter of the devices being online less than 21.5% of the time. The median availability of 65.0% seems complaisant to our experiment, until we compare ourselves with a deployed onion routing network like Tor: the dotted line shows the actual devices availability CDF of the 10'521 relays that constituted the Tor network in September 2018. We see that, although roughly 55% percent of Tor's relays are less connected than our simulated devices, an astonishing 33.7% of Tor's servers reach 100% availability, while none of our devices do. In Tor, the relay selection favouring reliable, powerful servers, the disconnection of a relay is bad luck. In Spores, churnful routes are the norm.

We now take interest in the routes selection and their reliability. We first study the *predictive* experiment only, by showing, in figure 5.7, the layer size and the routes reliability as a function of the availability threshold θ. We separated measurements of the e-squad layers (the last of each route, see figure 5.4) and non-e-squad layers, as e-squad layers fail to reach the availability threshold in 33.0% of the cases, while the other layers always reach it. This is caused by the lesser number of e-squad members compared to the candidates in $\mathcal{V}_{\text{RPS}}$. The dotted line represents the end-to-end message exchange success rate (from sender to receiver, thus accounting for the e-squad layer). First, we see that $\theta = 1$ resembles traditional onion routing network with 1 node/layer, causing terribly unreliable routes given our poor relays' availability. The 9.28% success rate is not helped by the frequent message retries attempted by file senders: 71.1% of the overall traffic stems from routes with $\theta = 1$. For $\theta < 1$, we observe that the number of devices per non-e-squad layer rises linearly (with steps of 2) as the availability threshold exponentially devices by steps of 10. The dispersion of the layer size increases too, which shows the variability of the relays' availability in each layer. Interestingly enough, the message success rate does not rise indefinitely, but peaks at $\theta = 0.01$, before decreasing when using more conservative availability thresholds (78.3% success rate at $\theta = 10^{-4}$). Visibly, the overhead induced by bigger layer sizes is not compensated by the better availability: by attempting to reach more devices at each hop, the message transmission is slowed down, and the forwarding relay itself risks disconnection; the header also gets bigger, increasing the network cost. Finally, we see that the maximum layer size $M_{\mathcal{L}} = 16$ is never reached.

Let us now compare the *predictive* and *random* route selection algorithms. We first study, in figure 5.8, the influence of the layer size on the probability that the layer be offline (that is $P_{\mathcal{L}}^{\text{off}}$ from equation 5.5), for both experiments. Data points are positioned

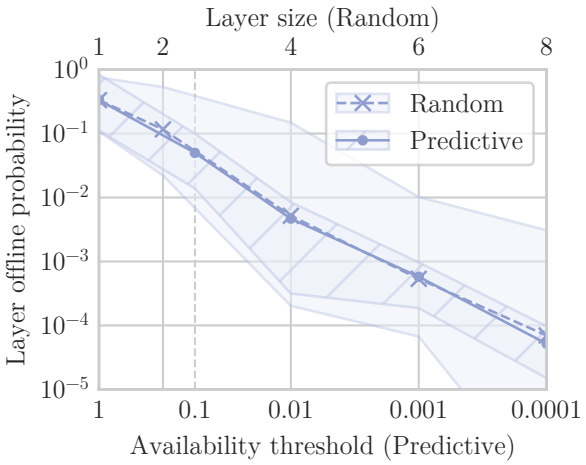


Figure 5.8 – Probability that a *non-e-squad* layer is entirely offline ($P_{\mathcal{L}}^{\text{off}}$) as a function of the layer size, split by experiment. The y-axis is logarithmic. Blue areas represent the range of the layer offline probabilities at each point; lines represent medians. The x positions reflect the median layer size for each studied parameter value. We see that the maximum probability is below the target availability threshold at each point, and that the predictive algorithm yields more consistent output than the random one.

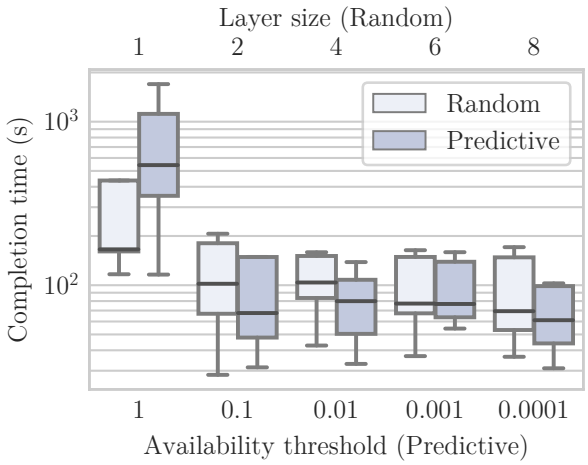


Figure 5.9 – Boxplots of the file exchange completion times as a function of the considered parameter for the *random* (lighter boxes, top x-axis) and *predictive* (darker boxes, bottom x-axis) experiments. The y-axis is logarithmic. The leftmost boxes show the great variability of using traditional onion routes with a churnful network. The other boxes show better completion times, with the *predictive* algorithm providing better median times and lower variance in general.

according to the median layer size for each value of the varied parameter (e.g. at $\theta = 0.1$, the median $S_{\mathcal{L}}$ is 2.5, see figure 5.7). The curves show the median $P_{\mathcal{L}}^{\text{off}}$ at each point, while the areas display the minimum and maximum $P_{\mathcal{L}}^{\text{off}}$. Both curves follow an $O(c^n)$ function, where $c \in]0, 1[$ is the probability that a device is offline. We see that the *predictive* experiment yields more consistent $P_{\mathcal{L}}^{\text{off}}$ than the *random*. Most importantly, the highest $P_{\mathcal{L}}^{\text{off}}$ is dominated by θ, whereas the random case provides no such guarantee. In figure 5.9, we lastly display the file exchange completion times for both experiments. What stands out the most is the order of magnitude gained by using two nodes per layer, compared to only one, which demonstrates the interest of multipath routing in churnful networks. Finally, except for the leftmost case (where the *predictive* and *random* algorithms are equivalent), we observe equivalent or better median completion times using the *predictive* routing algorithm, which highlights the interest of controlling the layers' probability of being offline.

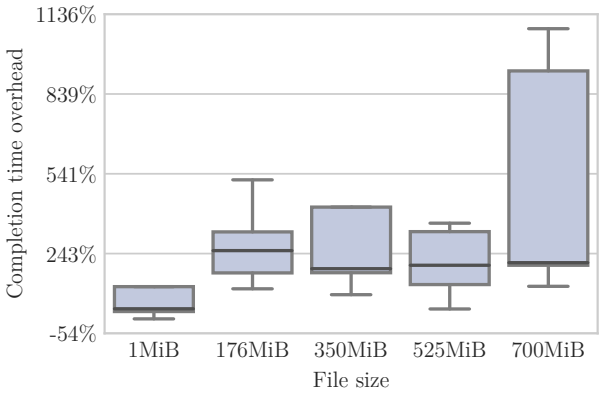


Figure 5.10 – Overhead of the file transmission time with churn, compared to the ideal case without churn, as Tukey boxplots [129] (median in darker grey), per file size. We see that, despite some variability, the median overhead lies around 200% of the ideal transmission time, regardless of the file size. The overhead dispersion, on the over hand, increases with the size.

Spores' resilience to churn

We now evaluate Spores' dependability given the low connectivity of its peers. To do so, we perform an experiment where we send files of varying size, and compare it against the same experiment without churn (all devices are always connected). The file sizes are 1MiB, 176MiB, 350MiB, 525MiB and 700MiB; we send 50 files, hence 10 per size. In the 'churnless' baseline, all messages go through, transfer times are minimal, and the number of exchanged messages is the theoretical minimum. To express Spores' reliability, we measure the overhead induced by the devices' churn in terms of file transfer time and network traffic.

Without churn, the file transfer's duration (fd in seconds) faithfully follows the file size (fs in bytes) according to the following linear equation: $fd = 1.97 \times 10^{-7} \times fs - 6.86 \times 10^{-2}$, with a determination coefficient of $r^2 = 0.9999$. The measured average bandwidth is 5.51MiB/s. With churn, the devices' availability closely resembles that of figure **??**. All file exchanges succeed, but 10.3% of the messages are lost, and are thus retransmitted. This generates an end-to-end network overhead of 1.60GiB, that is 11.1% of the 14.4GiB needed to exchange all files in the ideal case.

Figure 5.10 shows the rate of the transfer time overhead per file size, as box plots. The median transfer time overhead is of 200% (i.e. it takes 3 times longer to download a file through churnful Spores as it would without churn). To illustrate, the median transfer time for 700MiB files is 144s without churn, while it is 444s when users turn

their devices off. We see that the median overhead does not depend on the file size, although bigger files imply a much bigger variance of the completion time. The devices being offline a portion of the time—including the file sender and receiver—we consider this time overhead satisfactory.

In this evaluation, we have deployed Spores in a churnful network and studied how its features enabled reliable file exchange. It was shown that the high degree of peers churn did not disrupt the completion of file transfers, and only caused a manageable overhead in terms of transfer duration and network consumption. We also observed that multipath routing was of great benefit for onion routing in churnful environments.

Although the predictive route selection protocol permitted to bound each layer's risk of falling offline, the benefit compared to random selection was not up to the mark. We suspect that our simplistic availability prediction holds part of the blame. A peer selection favouring decently available nodes would also improve performance, but, for security reasons, it is inapplicable when devices publish their availability estimate themselves. Tor biases the selection towards relays that are deemed proficient by the (partly centralised) directory authorities. An avenue for improvement would be to crowd-source the estimation of relays' availability by their peers in a decentralised way, before biasing the route selection to prefer more dependable nodes.

In the end, Spores proves to be a robust anonymous P2P file exchange protocol. The e-squad component enables the availability predictions, and make devices collaborate into exchanging a file. The probabilistic routing proposal, providing reliable routes on the global network, showed its efficiency.

The e-squad enables a user to keep receiving messages while their receiving device is disconnected, which clearly advocates our claim: e-squads enable users to experience reliable privacy-preserving services despite the inherent unreliability of decentralised networks.

5.5 Security discussion

One limitation of Spores, is the variable size of its headers. Because onion headers are created once and decrypted without re-encryption at each hop of the route, headers shrink as they traverse layers. A curious relay can use this information to infer a message's position on its route. A makeshift solution to cover tracks would be to pad a random number of bytes in the innermost part of the header; but a satisfactory proposal to avoid

this information leakage would need to be investigated seriously. Since every route's last layer is only constituted of one user's nodes, the variable header size can help attackers determine which relays belong to whom. Still, an attacker can only observe the last layer of *random* messages, when they were chosen to be the penultimate hop on a route: they cannot target specific users.

Interesting paths for further study include the HORNET network-layer stateless onion-routing protocol [95]. Its header format, Sphinx [94], could be adapted for fixed-size POR headers.

Another of SPORES' flaw is that we trust e-squads to provide their devices' probability of being online. This is pretty naive, as a byzantine user that wants to observe many messages can just publish probabilities of 100% for all their devices and maximize their probability of being picked by the route creation algorithm. Thankfully, the roulette wheel selection algorithm weakens this threat: by leaving poorly connected devices a chance of being picked, it lessens the chances that highly connected nodes will always be chosen. In any case, we propose a better approach to the availability probability estimation in the next chapter, where devices estimate the probability of remote peers themselves, instead of trusting them.

Finally, we did not consider the problem of public key verification, making SPORES subject to attacks such as identity theft and man-in-the-middle. However, this problem of Decentralised Public Key Infrastructure (DPKI) is being tackled by numerous academics, notably using blockchain infrastructures [87, 162]. Blockchains can also help solve the incentive problem (to motivate users to keep contributing to the network even when they have no file to share): this is precisely proposed by the Nym project [163]

5.6 Conclusion

With SPORES, we have proposed an anonymous file exchange protocol enabled by the federation of people's e-squads. The proposal demonstrates the ability of a user' devices to guard their privacy, as e-squads collaborate to form a robust anonymity network. Furthermore, the coordination inside the e-squad compensates the appliances' unreliability; in the end, the provided service is dependable, and even powerful. Finally, our probabilistic onion routing proposal remedies the instability of the global network; it could be adapted to legacy onion networks in order to let users' devices participate in the network.

There are still open questions regarding the handling of P2P systems' unreliability. We

proposed a simplistic model to infer the future availability of network routes. The prediction itself could be improved, notably through better user modelling, to be more accurate and foreseeing. Instead, if the prediction was made by other peers (not self-published), it would enable non-uniform peer sampling during routes creation. But how could devices with a partial view of the system assess remote peers' connectedness? Aggregating the partial information could be a way to crowd-source the prediction. Another improvement path is to strengthen the probabilistic routes operation. Undoubtedly, PORs headers could be lighter, using more advanced broadcast encryption schemes. A more advanced question is whether it is possible to create two-way, connected routes (TCP-like) using our multipath routing approach.

We hope that our proposal makes the case for e-squads, their benefits for individuals, and their power as a federated network.

Chapter 6

Conclusion

Conviviale est la société où l'homme contrôle l'outil.

Ivan Illich

It is wise to have decisions of great moment monitored by generalists. Experts and specialists lead you quickly into chaos.

Frank Herbert

In this dissertation, we have proposed a new paradigm to put data back in the users' hands in the context of the rising IoT. This proposition leverages aspects of distributed systems and machine learning so as to make one's devices an intelligent *e-squad*, able to provide dependable services despite the inherent poor connectivity of end-user appliances.

We first showed how learning the user's behaviour among the e-squad enables enlightened decision-making in the context of applicative session handoff. Secondly, we applied redundancy and smart dissemination techniques in order to make the session handoff reliable in the face of devices churn. Finally, we showed that e-squads could be federated between users to create a global anonymous file exchange network.

This is just the beginning of the journey however, as open questions still remain to be addressed before e-squads can become a tangible reality? Firstly, distributed machine learning is still a nascent research field. We lacked the field data to propose fine-grained and powerful inference techniques to better understand our user. Future work needs to be done to collect datetime and geolocations, and to map them to user activities. Another promising path would be to aggregate the computing capabilities of the user devices in order to perform high-end machine learning despite individually constrained resources. Secondly, more work needs to be done to overcome the connection intermittency, high failure rate and variable performance of user devices as opposed to the high quality of cloud-backed infrastructure. Better modelling of the user's behaviour would notably help

strengthen e-squads reliability. This leads to the third challenge: in the multi-user context, the tension between reliability and users' privacy needs to be thoroughly studied. By exposing more information about each user, a federated e-squad network would better understand its churn dynamics, and ensure a good performance; although, the network should not allow to disclose private user information such as their location, habits and lifestyle.

There is another daunting issue that was not addressed in this thesis: the interoperability between IoT devices is still a pipe dream at the time of writing. Although relevant research is being made, e.g. to propose middlewares, compatible protocols or light and secure virtualization techniques, it is salient that the biggest problem here is not technical: it is economical. Why would competing companies allow inter-connection of their products, at the risk of reducing their performance, without any regulatory or monetary incentives? One can only hope that public communities, building ad-hoc inter-compatible IoT solutions, will demonstrate to the corporate world the benefits of device collaboration.

The over-centralisation of the Internet is self-explanatory, given that it was invented and developed within western countries deeply rooted in a second revolution infrastructure (where, for instance, vertically-integrated for-profit corporations prevail). The flaws of centralisation (e.g. failing to acknowledge local demands) and of for-profit economies (e.g. failing to tackle the environmental crisis) are equally challenging the Internet and their birthing societies. For this very reason, we cannot expect alternative communication paradigms to emerge from the same 'tech elites' that currently dominate the Internet. Revolutionary paradigms can only emerge from the population at large, and from their particular needs and usage. Towards this goal, our biggest challenge is not a research problem, but yet again societal. Informatics needs to be demystified and taught to the general public; not to seek new experts, but to make of everyone a *generalist*.

Appendix A

Résumé en français

Nos sociétés sont au milieu d'une crise causée par une nouvelle révolution de nos moyens de communication [1]. L'ère de l'information ne tient pas ses promesses : la nouvelle facilité d'accès aux connaissances était supposée permettre aux peuples de se rendre maîtres de leur destin, or la surveillance de masse que nous subissons ressemble plutôt à un nouveau mode d'esclavage. Dans cette thèse, nous étudions de nouvelles façons de rendre le pouvoir aux masses, en faisant de nos périphériques connectés les gardiens de notre vie digitale, et non les mouchards déguisés à quoi ils ressemblent actuellement.

Il y a toujours eu une tension entre centralisation et décentralisation dans les sociétés humaines. La centralisation, avec sa structure de pouvoir pyramidale et son contrôle de la base, permet à la tête de prendre rapidement des décisions qui s'appliqueront à tous—parfois au détriment des volontés individuelles. En réaction, la décentralisation encourage des structures de décision plus petites, agrégées ensemble par voie de communication.

Cet antagonisme a toujours été particulièrement remarquable en informatique. Nous passâmes des unités centrales géantes des années 50 aux ordinateurs personnels, et de media centralisés tels que la télé ou la radio à la commutation de paquets décentralisée qui donna naissance à l'Internet.

Mais aujourd'hui, l'Internet—décentralisé par construction—est le théâtre d'une nouvelle centralisation : quelques monopoles, vastes silos d'information, capturent la majorité du trafic.

Le modèle centralisé et ses limites Aujourd'hui, le paradigme de communication majoritaire sur le net est le suivant : nos périphériques jugés ineptes sont rendus intelligents par le cloud tout puissant. En d'autres termes, chacun d'entre nous possède plusieurs appareils qui communiquent continuellement avec le « cloud » (une horde bien réelle de centres de données) pour accomplir leurs tâches.

Ce paradigme permet indéniablement à des périphériques aux ressources limitées d'accomplir des opérations complexes, telles que l'édition multimédia ou l'interaction par commande vocale. En outre, en prenant la charge des pires besognes des concepteurs d'applications—comme assurer la disponibilité des données—les fournisseurs de cloud facilitent grandement le développement logiciel. Mais c'est à un prix d'un autre ordre que se paye cette facilité.

La facture est en premier lieu adressée à la planète: l'ensemble des Technologies de l'Information et de la Communication (TIC) consommait 7% de l'électricité mondiale en 2012 [2, 3]. plus de 10% en 2019, et on estime atteindre 21% d'ici 2030 [4, 5]. Là où le bât blesse, c'est que 90% de l'empreinte carbone et de énergie consommée par un téléphone moderne est à imputer aux communications réseau et aux centres de données qu'il utilise; ceci incluant la production de l'appareil, son chargement, son fonctionnement et son retraitement [5, 6].

Le second coût est notre vie privée. Quand Alice et Bob discutent en ligne, il semble raisonnable d'affirmer que les données qu'ils génèrent ce faisant leur appartiennent. Pourtant, la plupart des hébergeurs de service utilisent ces données pour proposer des publicités ciblées à leurs clients, leur permettant ainsi de rendre leur service gratuit. Malgré le fait que les utilisateurs acceptent les termes et conditions du service, cet accord est contestable : il bafoue notre droit (inaliénable) à la vie privée [8] sans notre consentement libre et informé [9, 10]. Ces données, une fois hors de notre contrôle, ne sont pas à l'abri de piratage, et peuvent se voir vendues ou piratées.

Le simple fait que des données personnelles—aussi insignifiantes soient-elles—se retrouvent si facilement en accès libre constitue une menace d'envergure envers les individus, la liberté d'expression, et par extension envers le fonctionnement démocratique des sociétés. Dans l'ordre, on peut citer le fait que les sociétés d'assurance usurpent ces données pour sélectionnent leurs clients [11] ; que les agences de renseignement étatiques ne puissent résister à la tentation de « tout collecter » [12–14] ; ou enfin que la société Cambridge Analytica se soit servie de bases de données de « j'aime » Facebook volées pour biaiser les élections présidentielles américaines de 2016, et le referendum du Brexit, en propageant de la désinformation hautement ciblée aux électeurs [15–18].

Il est urgent que nos informations personnelles échappent au contrôle du cloud pour s'établir ailleurs, dans des espaces plus respectueux de la vie privée. C'est donc emplis de crainte, mais aussi d'espoir, que nous assistons par ailleurs à la multiplication de périphériques connectés par habitant.

L'essor de l'Internet des Objets Il y a bientôt 30 ans, Mark Weiser prophétisait l'avènement de l'*informatique ubiquitaire* [19] : plus l'informatique sera assimilée par la société, plus elle proliférera dans notre environnement, se fondant parallèlement dans le décor. On peut comparer cette prédiction avec l'adoption d'une autre technologie : l'écriture. Lire et écrire ne nous demandent pas un gros effort conscient ; pourtant, vous pouvez sans doute compter plusieurs centaines de supports d'écriture dans votre environnement immédiat. Récemment, l'informatique ubiquitaire, sous sa nouvelle appellation « Internet of Things» (IoT), a attiré beaucoup d'attention au fur et à mesure que les enjeux techniques se résolvent pour la réaliser.

Les promesses de l'IoT sont attrayantes: « mener l'informatique au-delà des gros problèmes tels que la haute finance et les devoirs scolaire, vers les petits soucies tels que "Où sont les clés de la voiture ?", "Où puis-je me garer ?", et "Est-ce que ce T-shirt que j'ai vu la semaine dernière à Macy's est toujours en rayon ?".» [20]. Des auteurs plus récents comme Jeremy Rifkin voient l'IoT comme moteur de la transition énergétique : il permettrait de distribuer la production d'énergie renouvelable dans chaque foyer, et de créer une grille où production et consommation seraient intelligemment balancées [1].

Hélas, l'IoT représente tout autant une menace pour les libertés civiles: la variété et la granularité des informations personnelles générées par les nouveaux périphériques connectés ont de quoi faire tourner la tête, et permettraient un degré de surveillance des populations jusqu'ici inconnu. Pour citer quelques craintes : les données générées par les compteurs électriques intelligents permettent de déterminer précisément la présence et l'activité des citoyens chez eux [21] ; les voitures connectées ne sont pas à l'abri du piratage [22] ; les bracelets de fitness n'hésitent pas à partager publiquement les bulletins de santé de leurs utilisateurs [23, 24], quand ces données ne sont pas simplement en accès libre [25]. D'autres polémiques sur le sujet ont trait au manque d'interopérabilité entre périphériques [26–28], ou à l'impact environnemental de la production de semi-conducteurs (contentant des produits hautement toxiques et des terres rares non-renouvelables) [29].

Constatant l'indéniable adoption de l'IoT par le public (cf. les communautés de *makers* [30] ou le succès planétaire des micro-ordinateurs Raspberry Pi [31]) et par l'industrie, nous ne pouvons qu'admettre la prédiction de Cisco selon laquelle le nombre de périphériques par personne continuera à augmenter (de 2.4 par habitant en 2017 à 3.6 en 2022) [32].

Devant cet état de fait, il est de la responsabilité de la communauté informatique de proposer une voie saine pour l'IoT. Une voie qui bénéficie aux peuples et aide à la

construction de sociétés plus pérennes, avant que l'économie de la surveillance ne le compromette inéluctablement. Nous proposons ici un nouveau concept, celui d'*e-squad*, afin de rendre les utilisateurs acteurs de la décentralisation de l'information :

Définition. *Une* ***e-squad*** *est un ensemble de périphériques connectés* appartenant *à un seul individu, rendue intelligente par voie de communication d'appareil à appareil au sein de l'e-squad. En agrégeant leurs informations sur l'utilisateur, la flotte de périphériques apprend sa propre dynamique, ce qui lui permet de prédire son état à venir. En s'assurant que les données collectées ne quittent pas les appareils de l'utilisateur, la connaissance dont dispose l'e-squad bénéficie à ce dernier tout en respectant sa vie privée.*

Dans cette thèse, nous explorons le potentiel des e-squads comme fournisseurs de services respectueux des droits humains, en plus d'être (au moins !) aussi pratiques et agréables que leurs équivalents dans le cloud [33].

Ce résumé se compose d'une revue de l'état de l'art dans la section suivante, avant de présenter brièvement nos contributions des sections A.3 à A.5. Nous conclurons notre exposé à la section A.6.

A.1 Contexte

Avant de nous intéresser à la littérature existante sur le sujet qui nous intéresse et de proposer un quelconque axe de recherche, il semble important de définir précisément quelques concepts.

Quelques définitions Un système est *centralisé* quand il existe en son sein un sous-ensemble représentant une autorité en laquelle tout le système a foi [34]. Un système *décentralisé*, par opposition, ne contient pas de telle autorité, si bien que chaque pair est considéré comme potentiellement adverse. Un système est *distribué* quand des entités distantes sont face à un problème, et que chacune d'entre elles n'a qu'une connaissance partielle des paramètres dudit problème. D'après Michel Raynal, la caractéristique principale des systèmes distribués est leur inhérente *incertitude* [35]. Le *cloud computing* est un modèle permettant de réserver sur demande des ressources informatiques (réseau, serveurs, stockage, applications ou services), de façon simple et configurable. Le fournisseur du service doit pouvoir fournir et libérer facilement les ressources avec un minimum d'intervention humaine [36].

On note que la plupart des plate-formes de cloud sont massivement distribuées : d'énormes grappes d'ordinateurs orchestrent les ressources de leurs clients afin d'assumer la variation de leurs besoins. Néanmoins, ces plate-formes ne sont pas décentralisées : chacun des clients accorde sa confiance à la société d'hébergement.

Nous continuons l'exposé en présentant certains problèmes liés au modèle centralisé cloud, avant de nous tourner vers ses alternatives. Enfin, on présentera des études sur l'évolution de l'usage de périphériques multiples.

A.1.1 Des limites du cloud

L'Internet d'aujourd'hui repose sur une poignée de protocoles (notamment TCP/IP et HTTP) qui ne fournissent aucune protection de la vie privée : ils ne supportent pas le chiffrement des données, et exposent des méta-données sensibles. Ce problème cause une gigantesque capitalisation de l'information par les quelques fournisseurs de cloud majoritaires, chez qui la majorité des services du net sont hébergés selon le modèle client-serveur.

Le chiffrement des communications est désormais résolu grâce à des protocoles comme Transport Layer Security (TLS) [37] ou encore le plus récent QUIC (qui remplace entièrement TCP) [38]. Pour ce qui est la confidentialité des données stockées sur le serveur, on voit mûrir des modèles tels que le chiffrement homomorphique [39] ou la récupération privée d'information (Private Information Retrieval) [40, 41]. Hélas, leur complexité et leur coût freinent leur adoption industrielle [42].

Quoi qu'il en soit, chiffrer les données ne suffit pas ; en effet, les méta-données inhérentes à la communication (telles que les adresses d'expédition et de réception, l'horodatage, le volume et la fréquence des messages) suffisent largement à extraire une connaissance exploitable par le renseignement, par exemple. Comme le disait le Général Michael Hayden, ancien directeur de la CIA et de la NSA : « On tue des gens en se basant sur les méta-données » [45]. Dans le cas de la navigation web, il a été démontré que les nombreuses informations fournies par les navigateurs constituent une *empreinte* qui suffit à différencier chaque utilisateur dans la plupart des cas [43, 44]. Les bannières de publicité et autres boutons de partage, présents sur la quasi-totalité des pages web, peuvent ainsi pister les individus d'un site à l'autre.

Dans l'état actuel des choses, cette hémorragie de méta-données n'est pas prête de s'arrêter. Les méta-donnée sont en effet souvent nécessaires au bon fonctionnement des services [46, 47] ; c'est aussi le modèle économique de nombre d'entreprises du web que

de collecter et vendre toutes ces informations [48].

C'est pourquoi nous estimons qu'une meilleure gouvernance des données ne peut être atteinte qu'en remettant en cause le modèle client-serveur. Pour rendre du pouvoir aux utilisateurs, il faut que nos appareils connectés fournissent eux-mêmes des services, plutôt que d'agir en simples interfaces de services hébergés sur des serveurs distants.

A.1.2 Des alternatives au cloud

Commençons par traiter d'un paradigme hybride qui se pose en alternative au cloud, à savoir le *edge cloud*, ou cloud en bordure de réseau.

Le edge cloud

Ce modèle propose de déployer de petits centres de données (appelés « servlets ») au plus près des utilisateurs, dans chaque groupe résidentiel par exemple. Le meilleur argument en la faveur de cette évolution est que certaines nouvelles technologies (comme la réalité virtuelle, la réalité augmentée ou les voitures autonomes) ont besoin et d'une volumineuse bande passante, et d'une très faible latence, pour opérer correctement [49, 50]. Rapprocher les serveurs des clients résoudrait ce problème.

Les partisans du edge plaident qu'il fournit de meilleurs garanties de vie privée que le cloud actuel, ce dont nous doutons [50–52]. En effet, les cloudlets sont hors du contrôle des utilisateurs au même titre que les autres centres de données.

À moins de reformuler la définition pour qu'edge se rapporte à l'utilisation des périphériques des utilisateurs eux-mêmes, nous ne voyons pas dans ce modèle un intérêt en terme de vie privée [53].

Les réseaux décentralisés

Nous ne sommes bien entendu pas les premiers à remettre en question la centralisation des réseaux. Les propositions abondent, et pour les présenter, nous étudierons les topologies des moins, au plus décentralisées.

La fédération Un réseau fédéré est constitué de communautés de pairs, chacune centrée autour d'un serveur (qui lui, suit le modèle client-serveur), de sorte que les communautés peuvent communiquer entre elles, formant ainsi un vaste réseau décentralisé. Les données de chaque communauté sont stockées sur un seul serveur, tandis que les inter-connexions

sont soumises à des règles paramétrables. D'anciens protocoles suivent cette topologie, à la fois moins sujette aux monopoles et simple à mettre en place (comparé à un réseau entièrement décentralisé), tels que SMTP [54], le protocole d'échanges de mails, ou encore XMPP, un protocole de communication instantanée [55].

Ce type de réseaux connaît un regain de popularité pour les applications sociales, notamment grâce au nouveau standard ActivityPub [56] publié par le consortium W3C. Des services comme Mastodon (micro-blogage) [57], PeerTube (publication vidéo) [58] et NextCloud (synchronisation et partage de fichiers) [59] sont par exemple inter-compatibles grâce à ActivityPub, formant un nouvel écosystème d'applications décentralisées dénommé *fédiverse* [60].

Quasi-décentralisé Certains réseaux décentralisés font le choix de conserver une partie de centralisation, pour les briques critiques du système telles que l'authentification ou la tenue d'un registre de clients connectés. Ceci simplifie l'implémentation de tels systèmes, même si ces briques centralisées constituent un point critique, dont la défaillance ou la compromission mettrait en danger tout le système.

On peut citer les Infrastructures de Gestion de Clés (IGC, ou Public Key Infrastructures, PKI, en anglais) comme exemple notoire, notamment utilisées dans le protocole TLS [37]. Pour authentifier les correspondants, TLS emploie des certificats d'identité délivrés par des Autorités de Certification (AC) tierces, qui garantissent l'authenticité de l'identité des acteurs. Ces dernières AC sont cruciales, et leur malfonctionnement compromet tout le système. On peut citer les attaques par interception du trafic entre clients et AC, exécutées par la NSA pour déchiffrer le contenu des messages [62].

On peut aussi considérer le célèbre réseau d'anonymat Tor [67], dont le protocole de routage en oignon [66] permet de ne pas divulguer l'identité du client sur le net. Sur ce réseau, dix serveurs, les *directory authorities* (DAs) publient toutes les heures un consensus contenant la liste des serveurs actuellement connectés, ainsi que toutes les informations nécessaires pour les utiliser. Bien que ces dix serveurs ne se fassent pas confiance entre eux, tout le réseau repose sur leur consensus et leur fait donc entièrement confiance.

Entièrement décentralisé Les réseaux purement décentralisés se débarrassent de tout point de défaillance unique en distribuant toutes les briques constituant le système. Cela apporte une grande résilience au système, ainsi qu'une plus grande sécurité, au prix d'une conception plus ardue (du fait de la fameuse incertitude inhérente aux systèmes dis-

tribués).

Afin de stocker de l'information de manière décentralisée, citons les algorithmes de Table de Hachage Distribuée (DHT en anglais), qui permet de stocker des paires de <clé, valeur> de façon redondante entre plusieurs pairs, de sorte que chaque pair stocke une sous-partie de l'ensemble des données. Chaque utilisateur peut demander la valeur d'une clé ou ajouter une paire, à la manière d'une base de données. Les DHT sont a priori susceptibles à des attaques, pour désanonymiser les clients ou corrompre les données ; de bonnes contributions s'attaquent néanmoins à ces problèmes [70–72]. BitTorrent emploie par exemple la DHT Kademlia [69] pour assurer la découverte de pairs en évitant les trackers centralisés. Les réseaux d'échange de fichiers Dat [75] et IPFS [74] recourent aussi à une DHT pour router les utilisateurs au contenu disponible.

Une autre solution pour échanger de l'information de façon décentralisée est la famille des algorithmes de rumeur, ou épidémiques [76, 77]. En propageant une information comme une rumeur, l'information atteint tous les nœuds connectés avec une probabilité écrasante et à un coût réseau borné. Au contraire des DHT, on s'intéresse plus ici à une propagation rapide des messages, sans offrir la possibilité de faire des requêtes. Parmi le algorithmes épidémiques, la famille de la sélection de pairs aléatoire (Random Peer Sampling, RPS, en anglais) [78, 79], assure la découverte de pairs : chaque nœud du RPS possède une vue dynamique contenant un sous-ensemble aléatoire des participants connectés au réseau. Ici aussi, la susceptibilité aux attaques est non-négligeable, malgré de bonnes contributions pour l'amoindrir [80–82]. Citons le réseau social Scuttlebutt [83–85] ou encore la pile de protocoles GNUNet [48] comme exemples d'utilisateurs d'algorithmes épidémiques.

Le système de paiement pair-à-pair BitCoin [86], après quelques années de scepticisme, a révolutionné notre conception de la monnaie, et a fourni un nouvel outil aux réseaux distribués. Son fondement, la *blockchain*, est une liste chaînée de blocs d'information, que chacun peut lire intégralement, et à laquelle on peut ajouter des blocs en fin de chaîne. La puissance de la blockchain réside dans le fait qu'un nombre non-borné de pairs s'entendent sur la valeur des blocs malgré la présence d'attaquants ; c'est une avancée non-négligeable de l'état de l'art. Elle permet de créer de la confiance (dans la véracité de la chaîne) malgré des pairs qui ne se font pas confiance entre eux. Des contributions académiques font aujourd'hui de la blockchain un outil [87, 88], et on peut citer Ethereum [89], un réseau qui emploie la blockchain pour stocker des « contrats intelligents » (des bouts de code arbitraires), ce qui montre l'intérêt de la chaîne au-delà des systèmes de paiement.

La littérature ne manque donc pas en réseaux distribués. À la question « Pourquoi ne sont-ils pas massivement employés ? », nous pouvons répondre qu'il reste nombre de défis les rendant difficiles à appliquer malgré de bonnes contributions académiques [34, 93]. Néanmoins, les principales raisons semblent économiques et sociales. La transition de carrière de George Danezis—quittant la communauté « vie privée » où il publia de nombreux articles d'importance [34, 40, 62, 94, 95] pour contribuer à la création de la blockchain de Facebook, Libra [96]—peut être une piste [97].

A.1.3 Le paradigme multi-machines

Malgré la grande popularité de l'IoT, il n'existe pas, que l'on sache, d'articles proposant de réunir les périphériques connectés d'un utilisateur en réseau. La plupart des contributions existantes touchent aux réseaux de capteurs [98], au traitement de données, à l'échange d'énergie [99]... Notre intérêt se porte sur la mise en collaboration des machines proches de leur utilisateur, si bien qu'on traite ici de l'évolution de leur usage.

Dès 2009, une enquête de Microsoft montre que l'on utilise son téléphone, toujours allumé, comme sa première source d'information [100]. Il est consulté avant et après l'ordinateur dans la journée, tandis que ce dernier est éteint à moins d'être intensément utilisé. La plupart des participants à l'enquête n'attendaient pas particulièrement de collaboration entre leurs périphériques, et s'estimaient satisfaits de la synchronie des données déjà présente.

En 2012, Google étudia spécifiquement les interactions « multi-écran » entre téléphone, PC, tablette et smart-TV [101]. Ils furent les premiers à observer un usage simultané (parallèle ou complémentaire) de plusieurs machines. En 2013, Kawsar et al. [102] remarquent la transition d'un usage de l'informatique centré sur l'ordinateur, à un usage dirigé par la tâche à accomplir (la sélection du périphérique étant faite dans un deuxième temps). Depuis 2015, on a comparé les usages de la tablette et du téléphone [103], ainsi que le paradigme multi-machines spécifiquement [104]. On voit que l'usage est principalement séquentiel, et peut tout de même se trouver compliqué par le manque d'interopérabilité (formats de données incompatibles par exemple). Jokela et al. [104] rapportent que les utilisateurs apprécieraient plus de collaboration entre leurs périphériques, par communication directe plutôt que par le cloud.

Notons que les tablettes, qui furent présentées comme une alternative aux ordinateurs, perdent des parts de marché depuis 2015 [106]. Sans doute l'augmentation de la taille des téléphones rend-elle leur usage obsolète [107].

On voit bien la transition à une interaction motivée par des tâches, qui s'éloigne du PC comme source première d'interaction. La collaboration entre périphériques reste un procédé peu normalisé, en attendant des contributions qui changeront la donne.

A.2 Vers les e-squads

Pour réaliser le paradigme des e-squads, des questions de recherche doivent être posées.

Des périphériques isolés à le-squad On a dit qu'une e-squad était un ensemble d'appareils rendu intelligent grâce à la communication pair-à-pair. Chaque périphérique connaît en effet en partie les habitudes de son utilisateur (le téléphone sait où il se situe, la station de travail n'est allumée que quand il travaille, et la télé quand il est oisif). En mettant en commun ces informations, les périphériques peuvent collectivement prédire le comportement utilisateur, et répondre par exemple à ces questions : Quelle tâche l'utilisateur essaye-t-il d'accomplir ? Quels appareils vont rester en ligne dans les prochaines heures ? Va-t-il allumer son ordinateur portable ?

En répondant à ces questions, les périphériques pourront assurer une qualité de service et ainsi être considérés intelligents. La question est donc premièrement d'agréger les informations disparates des périphériques en un savoir global.

La connaissance, c'est la fiabilité La différence la plus importante entre le modèle cloud et l'e-squad, c'est que le cloud fournit des garanties de disponibilité de l'ordre de 99.9%, tandis que les appareils des utilisateurs sont souvent hors-ligne. Comment donc assurer des services fiables dans un monde si intermittent ?

La question ici est d'employer la connaissance de l'e-squad afin de rendre les services fiables et transparents.

Fédération d'e-squads et vie privée Le rôle d'Internet est principalement de mettre en contact des gens ; les e-squads n'iraient pas loin si elles se cantonnaient à un seul utilisateur. On veut proposer des applications multi-utilisateurs en fédérant les e-squads, mais sans pour autant mettre en danger leur vie privée. Pourtant, la simple présence des périphériques d'un utilisateur sur le réseau fournit des informations sur son comportement.

D'un côté, plus une e-squad fournit d'information sur sa dynamique, plus le service sera fiable. De l'autre, on souhaite limiter la fuite d'information pour protéger chaque

utilisateur. Le juste milieu à trouver constitue le troisième défi, afin d'avoir d'une part un service fiable et de l'autre, de ne pas compromettre les utilisateurs du système.

Ces questions sont traitées dans les versions anglaises des sections suivantes. Dans ce résumé, nous ne montrons qu'un aperçu de nos contributions. Nous espérons néanmoins que notre argumentaire aura séduit l'imaginaire du lecteur, et leur aura fait rêver d'un futur connecté plus décentralisé et humain.

A.3 Des interactions fluides au sein de l'e-squad

Dans la version anglaise de cette section (chapitre 3), nous présentons en détail le protocole Sprinkler, qui emploie l'e-squad d'un utilisateur afin de faire transiter la session applicative des utilisateurs d'un périphérique à l'autre. Ce problème assez universel—où l'utilisateur aimerait retrouver l'état de son application (les onglets de son navigateur ou son projet d'édition vidéo en cours) alors qu'il passe d'un appareil à l'autre—connaît pour le moment uniquement des solutions qui utilisent le cloud. Pour les raisons citées en introduction, nous considérons cette approche nocive, et souhaitons lui trouver de meilleures alternatives.

Pour réaliser Sprinkler, nous proposons d'abord un protocole épidémique d'échange de l'activité de l'utilisateur entre périphériques, ce qui permet de construire une base de savoir au sein de l'e-squad. Ensuite, nous utilisons ce savoir pour inférer les appareils que l'utilisateur a le plus de chances d'utiliser à chaque transition, via des approches d'apprentissage statistique. Ceci nous permet de partager sa session au prochain appareil avant qu'il ne s'y connecte, réduisant dans certains cas le temps d'attente de l'utilisateur à zéro—ce qui constitue une avancée par rapport au modèle cloud, où l'utilisateur a toujours à attendre que sa session soit téléchargée avant de pouvoir utiliser son périphérique.

Nous faisons ici l'hypothèse que tous les périphériques sont toujours connectés ; hypothèse irréaliste que nous attaquons dans la prochaine section.

A.4 Fiabilité de service malgré la piètre connectivité des périphériques de l'e-squad

Avec le protocole Cascade (cf. chapitre 4), on poursuit le travail entrepris avec Sprinkler, mais cette fois-ci en admettant que les périphériques d'un utilisateur sont majori-

tairement déconnectés. Notre but est de réussir à transmettre sa session à l'utilisateur dans tous les cas, quels que soient les périphériques connectés, en maximisant les échanges de session *proactifs* (où la session se retrouve sur le bon périphérique avant que l'utilisateur ne s'y connecte, réduisant son temps d'attente à zéro). On cherche d'autre part à minimiser le trafic réseau engendré par l'échange de session.

Pour assurer une qualité du service d'échange de session malgré les fréquentes déconnexions, nous stockons la session de façon redondante, sur plusieurs périphériques. Pour ce faire, nous adaptons le protocole pair-à-pair BitTorrent, qui sert à diffuser des fichiers entre personnes. Grâce à une sélection intelligente des pairs à qui transmettre chaque session, nous trouvons un juste milieu entre la qualité de service et sa consommation réseau. On constate en outre que le comportement de l'utilisateur influe sur notre performance : on réussit plus souvent à envoyer proactivement la session si l'utilisateur se comporte de façon prévisible.

A.5 Fédérer les e-squads pour créer des applications respectueuses de la vie privée

Dans ce chapitre 5 du document en anglais, nous joignons les e-squads de plusieurs utilisateurs afin de créer un réseau d'échange de fichiers anonyme, nommé Spores. Le nerf de la guerre est ici de trouver le bon compromis entre performance de l'échange de fichier (taux de succès à maximiser et temps de transfert à minimiser) tout en s'assurant de l'anonymat des utilisateurs.

On propose pour ce faire une adaptation du routage en oignon à une infrastructure peu connectée, en employant l'intelligence des e-squads et des protocoles distribués comme la sélection de pairs aléatoire. Notamment, notre adaptation du routage en oignon permet à chaque paquet de prendre une route différente, choisissant sa route parmi des alternatives pré-établies en fonction de la disponibilité des nœuds sur la route. On constate notamment que les performances sont proportionnelles à la prévisibilité des utilisateurs.

A.6 Conclusion

Dans cette dissertation, nous avons donc proposé un nouveau paradigme permettant de rendre aux utilisateurs le contrôle de leurs données grâce au nombre croissant de pé-

riphériques connectés par habitant. En employant des contributions existantes en réseaux distribués et apprentissage statistique, nous avons transformé les périphériques d'un utilisateur en une e-squad intelligente, capable de surmonter ses limites (notamment sa piètre connectivité) pour fournir des services performants et fiables. Enfin, nous avons entrepris de fédérer les e-squads de plusieurs utilisateurs en un réseau plus large, respectueux de la vie privée de chacun.

Ce n'est qu'un début, car de nombreuses questions restent ouvertes. Premièrement, en terme d'apprentissage, il faudra des études de cas pour glaner plus d'information sur l'usage des utilisateurs. Il faudra ensuite proposer des modèles d'apprentissage plus performants que nos preuves de concept (en utilisant notamment géolocalisation, date et heure) pour mieux modéliser le comportement le l'utilisateur. Dans une autre veine, de nouveaux algorithmes d'apprentissage distribué pourraient être proposés afin de surmonter les performances médiocres de chaque périphérique et bâtir des modèles complexes du comportement. Enfin, le contexte multi-utilisateurs comporte son lot de défis, notamment celui d'assurer la sécurité des données des individus tout en conservant une bonne qualité de service.

D'autres questions ont été laissées pour compte, et notamment le problème primordial de l'interopérabilité des appareils de l'IoT. Il n'est actuellement pas dans l'intérêt des fabriquants de permettre à du code ne leur appartenant pas de s'exécuter sur leurs produits (cela ne ferait que nuire à leur performance). Il faudra inciter les industriels à cette interopérabilité en leur démontrant qu'ils ont tout à gagner à laisser les périphériques collaborer ; ceci passera sans doute par des régulations, et par l'accroissement des réussites des solutions ad-hoc, notamment entreprises par les communautés de makers.

Pour en revenir à la centralisation, l'état des réseaux actuel n'est pas étonnant quand on voit quelles sociétés popularisèrent Internet. Les modèles de seconde révolution industrielle, verticalement intégrés et à intérêt commercial, mènent naturellement à la centralisation. Et les crises qui menacent Internet sont les mêmes que celles qui menacent les sociétés qui le virent naître : le manque de considération pour les demandes locales (lié à la centralisation) ou l'incapacité à sérieusement envisager la crise environnementale (liée à l'intérêt commercial). Pour cette raison, on ne peut s'attendre à voir naître des alternatives sérieuses au modèle centralisé chez les mêmes élites technologiques qui exploitent actuellement Internet. C'est parmi la masse que se trouvent les besoins de décentralisation. Notre plus grand défi pour réussir à décentraliser les réseaux n'est donc pas un problème de recherche, mais social. L'informatique doit être démystifiée et enseignée au

plus large public ; pas dans le but d'en faire de nouveaux experts, mais afin que chacun puisse au moins devenir un *généraliste*.

Bibliography

[1] Jeremy Rifkin, *The Third Industrial Revolution: How Lateral Power is Transforming Energy, the Economy, and the World*, St. Martins Griffin, 2013, ISBN: 9780230341975.

[2] Peter Corcoran and Anders Andrae, « Emerging Trends in Electricity Consumption for Consumer ICT », *in*: (July 2013).

[3] Gary Cook, Jude Lee, Tamina Tsai, Ada Kong, John Deans, Brian Johnson, and Elizabeth Jardim, *Clicking Clean: Who Is Winning the Race to Build a Green Internet?*, Jan. 2017.

[4] Anders S. G. Andrae and Tomas Edler, « On Global Electricity Usage of Communication Technology: Trends to 2030 », *in*: *Challenges* 6.*1* (June 2015), pp. 117–157, DOI: 10.3390/challe6010117.

[5] Janine Morley, Kelly Widdicks, and Mike Hazas, « Digitalisation, Energy and Data Demand: The Impact of Internet Traffic on Overall and Peak Electricity Consumption », *in*: *Energy Research & Social Science* 38 (Apr. 2018), pp. 128–137, ISSN: 22146296, DOI: 10.1016/j.erss.2018.01.018.

[6] Roland Hischier, Vlad C. Coroama, Daniel Schien, and Mohammad Ahmadi Achachlouei, « Grey Energy and Environmental Impacts of ICT Hardware », *in*: *ICT Innovations for Sustainability*, ed. by Lorenz M. Hilty and Bernard Aebischer, Advances in Intelligent Systems and Computing, Springer International Publishing, 2015, pp. 171–189, ISBN: 978-3-319-09228-7.

[7] Kyle Devine and Matt Brennan, « Music Streaming Has a Far Worse Carbon Footprint than the Heyday of Records and CDs – New Findings », *in*: *The Conversation* (Apr. 2019).

[8] *Universal Declaration of Human Rights*, Art. 12, United Nations, Dec. 1948.

[9] The Europeran Parliament and the Council of the European Union, *General Data Protection Regulation (GDPR), Council regulation (EU) no 2016/679*, Article 7, 2016, URL: https://gdpr-info.eu/art-7-gdpr/.

[10] La Quadrature du Net, *First Sanction against Google Following Our Collective Complaints*, Jan. 2019, URL: https://www.laquadrature.net/en/2019/01/21/first-sanction-against-google-following-our-collective-complaints/.

[11] Yeslam Al-Saggaf, « The Use of Data Mining by Private Health Insurance Companies and Customers' Privacy: An Ethical Analysis », *in*: *Cambridge Quarterly of Healthcare Ethics* 24.*3* (July 2015), pp. 281–292, ISSN: 0963-1801, 1469-2147, DOI: 10.1017/S0963180114000607.

[12] Ellen Nakashima and Joby Warrick, « For NSA Chief, Terrorist Threat Drives Passion to 'Collect It All' », *in*: *Washington Post* (July 2013), ISSN: 0190-8286.

[13] David Greene and Katitza Rodriguez, *NSA Mass Surveillance Programs: Unnecessary and Disproportionate*, tech. rep., Electronic Frontier Foundation (EFF), May 2014, p. 23.

[14] *NSA Utah Data Center - Serving Our Nation's Intelligence Community*, URL: https://nsa.gov1.info/utah-data-center/.

[15] Carole Cadwalladr, « 'I Made Steve Bannon's Psychological Warfare Tool': Meet the Data War Whistleblower », *in*: *The Guardian* (Mar. 2018), URL: https://www.theguardian.com/news/2018/mar/17/data-war-whistleblower-christopher-wylie-faceook-nix-bannon-trump.

[16] Michal Kosinski, David Stillwell, and Thore Graepel, « Private Traits and Attributes Are Predictable from Digital Records of Human Behavior », *in*: *Proceedings of the National Academy of Sciences* 110.*15* (Apr. 2013), pp. 5802–5805, ISSN: 0027-8424, 1091-6490, DOI: 10.1073/pnas.1218772110.

[17] Wu Youyou, Michal Kosinski, and David Stillwell, « Computer-Based Personality Judgments Are More Accurate than Those Made by Humans », *in*: *Proceedings of the National Academy of Sciences* 112.*4* (Jan. 2015), pp. 1036–1040, ISSN: 0027-8424, 1091-6490, DOI: 10.1073/pnas.1418680112.

[18] S. C. Matz, M. Kosinski, G. Nave, and D. J. Stillwell, « Psychological Targeting as an Effective Approach to Digital Mass Persuasion », *in*: *Proceedings of the National Academy of Sciences* 114.*48* (Nov. 2017), pp. 12714–12719, ISSN: 0027-8424, 1091-6490, DOI: 10.1073/pnas.1710966114.

[19] Mark Weiser, « The Computer for the 21st Century », *in*: *Scientific American* 265.*3* (1991), pp. 94–105, ISSN: 0036-8733.

[20] Mark Weiser and John Seely Brown, « Beyond Calculation », *in*: ed. by Peter J. Denning and Robert M. Metcalfe, 1997, chap. The Coming Age of Calm Technolgy, pp. 75–85, ISBN: 0-38794932-1.

[21] Giulio Giaconi, Deniz Gunduz, and H. Vincent Poor, « Privacy-Aware Smart Metering: Progress and Challenges », *in*: *IEEE Signal Processing Magazine* 35.*6* (Nov. 2018), pp. 59–78, ISSN: 1053-5888, 1558-0792, DOI: 10.1109/MSP.2018.2841410, arXiv: 1802.01166.

[22] Andy Greenberg, « Hackers Remotely Kill a Jeep on the Highway—With Me in It », *in*: *Wired* (July 2015), ISSN: 1059-1028.

[23] Greig Paul and James Irvine, « Privacy Implications of Wearable Health Devices », *in*: *Proceedings of the 7th International Conference on Security of Information and Networks*, SIN '14, New York, NY, USA: ACM, 2014, 117:117–117:121, ISBN: 978-1-4503-3033-6, DOI: 10.1145/2659651.2659683.

[24] « Healthcare Spend in Wearables to Reach $60bn by 2023 », *in*: *Juniper Research* (Jan. 2019), URL: https://www.juniperresearch.com/press/press-releases/healthcare-spend-in-wearables-reach-60-bn-2023.

[25] Leena Roa, *Sexual Activity Tracked By Fitbit Shows Up In Google Search Results*, July 2011, URL: http://tcrn.ch/lYiNqp.

[26] Yérom-David Bromberg, Paul Grace, Laurent Réveillère, and Gordon S. Blair, « Bridging the Interoperability Gap: Overcoming Combined Application and Middleware Heterogeneity », *in*: *Middleware 2011*, ed. by David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Fabio Kon, and Anne-Marie Kermarrec, vol. 7049, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 390–409, ISBN: 978-3-642-25820-6 978-3-642-25821-3, DOI: 10.1007/978-3-642-25821-3_20.

[27] Gordon Blair, Yérom-David Bromberg, Geoff Coulson, Yehia Elkhatib, Laurent Réveillère, Heverson B. Ribeiro, Etienne Rivière, and François Taïani, « Holons: Towards a Systematic Approach to Composing Systems of Systems », *in*: *Proceedings of the 14th International Workshop on Adaptive and Reflective Middleware - ARM 2015*, Vancouver, BC, Canada: ACM Press, 2015, pp. 1–6, ISBN: 978-1-4503-3733-5, DOI: 10.1145/2834965.2834970.

[28] Alexis Huf and Frank Siqueira, « Composition of Heterogeneous Web Services: A Systematic Review », *in*: *Journal of Network and Computer Applications* 143 (Oct. 2019), pp. 89–110, ISSN: 1084-8045, DOI: 10.1016/j.jnca.2019.06.008.

[29] C. A. de Souza, J. N. Correa, M. M. Oliveira, A. Aagaard, and M. Presser, « IoT Driven Business Model Innovation and Sustainability: A Literature Review and a Case Study in Brazil », *in*: *2019 Global IoT Summit (GIoTS)*, June 2019, pp. 1–6, DOI: 10.1109/GIOTS.2019.8766371.

[30] Cory Doctorow, *Makers*, Dec. 2010.

[31] Phillip Torrone, *25 Million + Raspberry Pi Computers Sold*, Mar. 2019, URL: https://blog.adafruit.com/2019/03/15/25-million-raspberry-pi-computers-sold-raspberry_pi-raspberrypi/.

[32] *Cisco Visual Networking Index: Forecast and Trends, 2017–2022*, tech. rep., Cisco, Feb. 2019, URL: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html.

[33] *Ethical design manifesto*, URL: https://ind.ie/ethical-design.

[34] Carmela Troncoso, Marios Isaakidis, George Danezis, and Harry Halpin, « Systematizing Decentralization and Privacy: Lessons from 15 Years of Research and Deployments », *in*: *Proceedings on Privacy Enhancing Technologies* 2017.*4* (Oct. 2017), pp. 404–426, ISSN: 2299-0984, DOI: 10.1515/popets-2017-0056.

[35] Michel Raynal, *Distributed Algorithms for Message-Passing Systems*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, ISBN: 978-3-642-38122-5 978-3-642-38123-2, DOI: 10.1007/978-3-642-38123-2.

[36] Peter Mell, Tim Grance, et al., « The NIST Definition of Cloud Computing », *in*: (2011).

[37] E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.3*, RFC 8446, RFC Editor, Aug. 2018.

[38] Jana Iyengar and Martin Thomson, *QUIC: A UDP-Based Multiplexed and Secure Transport*, Internet-Draft draft-ietf-quic-transport-22, Work in Progress, Internet Engineering Task Force, July 2019, 148 pp., URL: https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-22.

[39] Craig Gentry, « Fully Homomorphic Encryption Using Ideal Lattices », *in*: *Proceedings of the 41st Annual ACM Symposium on Symposium on Theory of Computing - STOC '09*, Bethesda, MD, USA: ACM Press, 2009, p. 169, ISBN: 978-1-60558-506-2, DOI: 10.1145/1536414.1536440.

[40] Nikita Borisov, George Danezis, and Ian Goldberg, « DP5: A Private Presence Service », *in*: *Proceedings on Privacy Enhancing Technologies* 2015.*2* (2015), pp. 4–24, ISSN: 2299-0984, DOI: 10.1515/popets-2015-0008.

[41] K. Banawan and S. Ulukus, « The Capacity of Private Information Retrieval From Coded Databases », *in*: *IEEE Transactions on Information Theory* 64.*3* (Mar. 2018), pp. 1945–1956, ISSN: 0018-9448, DOI: 10.1109/TIT.2018.2791994.

[42] Diego Chialva and Ann Dooms, « Conditionals in Homomorphic Encryption and Machine Learning Applications », *in*: *arXiv:1810.12380 [cs]* (Oct. 2018), arXiv: 1810.12380 [cs].

[43] Peter Eckersley, « How Unique Is Your Web Browser? », *in*: *Privacy Enhancing Technologies*, vol. 6205, Springer, 2010, pp. 1–18.

[44] Alejandro Gómez-Boix, Pierre Laperdrix, and Benoit Baudry, « Hiding in the Crowd: An Analysis of the Effectiveness of Browser Fingerprinting at Large Scale », *in*: *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*, Lyon, France: ACM Press, 2018, pp. 309–318, ISBN: 978-1-4503-5639-8, DOI: 10.1145/3178876.3186097.

[45] David Cole, *'We Kill People Based on Metadata'*, May 2014.

[46] Steve Baker and Benoit Jacob, *[Public WebGL] about the VENDOR, RENDERER, and VERSION Strings*, Nov. 2010, URL: https://www.khronos.org/webgl/public-mailing-list/public_webgl/1011/msg00221.php.

[47] Keaton Mowery and Hovav Shacham, « Pixel Perfect: Fingerprinting Canvas in HTML5 », *in*: *Proceedings of W2SP 2012*, ed. by Matt Fredrikson, IEEE Computer Society, May 2012.

[48] Christian Grothoff, « The GNUnet System », PhD Thesis, Université de Rennes 1, 2017.

[49] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, « The Case for VM-Based Cloudlets in Mobile Computing », *in*: *IEEE Pervasive Computing* 8.*4* (Oct. 2009), pp. 14–23, ISSN: 1536-1268, DOI: 10.1109/MPRV.2009.82.

[50] Mahadev Satyanarayanan, « The Emergence of Edge Computing », *in*: *Computer* 50.*1* (2017), pp. 30–39.

[51] N. Li, C. Tsigkanos, Z. Jin, S. Dustdar, Z. Hu, and C. Ghezzi, « POET: Privacy on the Edge with Bidirectional Data Transformations », *in*: *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom*, Mar. 2019, pp. 1–10, DOI: 10.1109/PERCOM.2019.8767395.

[52] Melody Moh and Robinson Raju, « Using Machine Learning for Protecting the Security and Privacy of Internet of Things (IoT) Systems », *in*: *Fog and Edge Computing*, John Wiley & Sons, Ltd, 2019, pp. 223–257, ISBN: 978-1-119-52508-0, DOI: 10.1002/9781119525080.ch10.

[53] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere, « Edge-Centric Computing: Vision and Challenges », *in*: *ACM SIGCOMM Computer Communication Review* 45.*5* (2015), pp. 37–42.

[54] J. Klensin, *Simple Mail Transfer Protocol*, RFC 2821, http://www.rfc-editor.org/rfc/rfc2821.txt, RFC Editor, Apr. 2001, URL: http://www.rfc-editor.org/rfc/rfc2821.txt.

[55] P. Saint-Andre, *Extensible Messaging and Presence Protocol (XMPP): Core*, RFC 6120, http://www.rfc-editor.org/rfc/rfc6120.txt, RFC Editor, Mar. 2011, URL: http://www.rfc-editor.org/rfc/rfc6120.txt.

[56] Christopher Webber and Jessica Tallon, *ActivityPub*, W3C Recommendation, W3C, Jan. 2018, URL: https://www.w3.org/TR/2018/REC-activitypub-20180123/.

[57] *Mastodon*, URL: https://joinmastodon.org/.

[58] *PeerTube*, URL: https://joinpeertube.org/en/.

[59] *Nextcloud*, URL: https://nextcloud.org/.

[60] *THe Fediverse Network*, URL: https://fediverse.network/.

[61] Umberto Eco, *Il nome della rosa*, Sonzogno, Etas S.p.A: Gruppo Editoriale Fabbri-Bompiani, 1980, ISBN: 978-0-15-144647-6.

[62] Christopher Soghoian and Sid Stamm, « Certified Lies: Detecting and Defeating Government Interception Attacks against SSL (Short Paper) », *in*: *Financial Cryptography and Data Security*, ed. by George Danezis, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, pp. 250–259, ISBN: 978-3-642-27576-0.

[63] Bruce Schneier, *New NSA Leak Shows MITM Attacks Against Major Internet Services - Schneier on Security*, Sept. 2013, URL: `https://www.schneier.com/blog/archives/2013/09/new_nsa_leak_sh.html`.

[64] Ryan Singel, « Law Enforcement Appliance Subverts SSL », *in*: *Wired* (Mar. 2010), ISSN: 1059-1028, URL: `https://www.wired.com/2010/03/packet-forensics/`.

[65] David L Chaum, « Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms », *in*: 24.*2* (1981), p. 5.

[66] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson, « Hiding Routing Information », *in*: *Information Hiding*, ed. by Gerhard Goos, Juris Hartmanis, Jan Leeuwen, and Ross Anderson, vol. 1174, Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 137–150, ISBN: 978-3-540-61996-3 978-3-540-49589-5, DOI: `10.1007/3-540-61996-8_37`.

[67] Roger Dingledine, Nick Mathewson, and Paul Syverson, *Tor: The Second-Generation Onion Router*, tech. rep., Naval Research Lab Washington DC, 2004.

[68] Bram Cohen, *The BitTorrent Protocol Specification*, tech. rep., BitTorrent Inc., 2008, URL: `http://www.bittorrent.org/beps/bep_0003.html`.

[69] Petar Maymounkov and David Mazieres, « Kademlia: A Peer-to-Peer Information System Based on the Xor Metric », *in*: *International Workshop on Peer-to-Peer Systems*, Springer, 2002, pp. 53–65.

[70] I. Baumgart and S. Mies, « S/Kademlia: A Practicable Approach towards Secure Key-Based Routing », *in*: *2007 International Conference on Parallel and Distributed Systems*, Dec. 2007, pp. 1–8, DOI: `10.1109/ICPADS.2007.4447808`.

[71] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen, « A Survey of DHT Security Techniques », *in*: *ACM Comput. Surv.* 43.*2* (Feb. 2011), 8:1–8:49, ISSN: 0360-0300, DOI: `10.1145/1883612.1883615`.

[72] Qiyan Wang and Nikita Borisov, « Octopus: A Secure and Anonymous DHT Lookup », *in*: *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference On*, IEEE, 2012, pp. 325–334.

[73] Andrew Loewenstern and Arvid Norberg, *The BitTorrent DHT Protocol*, tech. rep., BitTorrent Inc., 2008, URL: http://www.bittorrent.org/beps/bep_0005.html.

[74] Juan Benet, « IPFS - Content Addressed, Versioned, P2P File System », *in*: (July 2014), p. 11.

[75] Maxwell Ogden, « Dat - Distributed Dataset Synchronization And Versioning », *in*: (Jan. 2018), DOI: 10.31219/osf.io/nsv2c.

[76] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry, « Epidemic Algorithms for Replicated Database Maintenance », *in*: *ACM* (1987).

[77] Michel Raynal, *Distributed Algorithms for Message-Passing Systems*, Springer Publishing Company, Incorporated, 2013, ISBN: 978-3-642-38122-5.

[78] Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen, « The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations », *in*: *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '04, New York, NY, USA: Springer-Verlag New York, Inc., 2004, pp. 79–98, ISBN: 978-3-540-23428-9.

[79] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten Van Steen, « Gossip-based peer sampling », *in*: *ACM ToCS* 25.*3* (2007), p. 8.

[80] A. Bakker and M. v Steen, « PuppetCast: A Secure Peer Sampling Protocol », *in*: *2008 European Conference on Computer Network Defense*, Dec. 2008, pp. 3–10, DOI: 10.1109/EC2ND.2008.7.

[81] Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer, « Brahms: Byzantine Resilient Random Membership Sampling », *in*: *Computer Networks* 53.*13* (Aug. 2009), pp. 2340–2359, ISSN: 13891286, DOI: 10.1016/j.comnet.2009.03.008.

[82] Gian Paolo Jesi, Alberto Montresor, and Maarten van Steen, « Secure Peer Sampling », *in*: *Computer Networks* 54.*12* (Aug. 2010), pp. 2086–2098, ISSN: 13891286, DOI: 10.1016/j.comnet.2010.03.020.

[83] *Secure Scuttlebutt - a Decent(Ralised) Secure Gossip Platform*, 2018, URL: https://www.scuttlebutt.nz/.

[84] Christian Tschudin, « A Broadcast-Only Communication Model Based on Replicated Append-Only Logs », *in*: *ACM SIGCOMM Computer Communication Review* 49.*2* (May 2019), pp. 37–43, ISSN: 01464833, DOI: 10.1145/3336937.3336943.

[85] Dominic Tarr, *Scalable Secure Scuttlebutt*, Aug. 2019, URL: https://github.com/dominictarr/scalable-secure-scuttlebutt/blob/master/paper.md.

[86] Satoshi Nakamoto, « Bitcoin: A Peer-to-Peer Electronic Cash System », *in*: (Nov. 2008), p. 9.

[87] Harry Halpin, « NEXTLEAP: Decentralizing Identity with Privacy for Secure Messaging », *in*: *Proceedings of the 12th International Conference on Availability, Reliability and Security - ARES '17*, Reggio Calabria, Italy: ACM Press, 2017, pp. 1–10, ISBN: 978-1-4503-5257-4, DOI: 10.1145/3098954.3104056.

[88] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov, « Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol », *in*: *Advances in Cryptology – CRYPTO 2017*, ed. by Jonathan Katz and Hovav Shacham, vol. 10401, Cham: Springer International Publishing, 2017, pp. 357–388, ISBN: 978-3-319-63687-0 978-3-319-63688-7, DOI: 10.1007/978-3-319-63688-7_12.

[89] *Ethereum*, URL: https://ethereum.org.

[90] Victor Costan and Srinivas Devadas, « Intel SGX Explained », *in*: *IACR Cryptology ePrint Archive* 2016 (2016), p. 86.

[91] Seongmin Kim, Juhyeng Han, Jaehyeong Ha, Taesoo Kim, and Dongsu Han, « SGX-Tor: A Secure and Practical Tor Anonymity Network With SGX Enclaves », *in*: *IEEE/ACM Transactions on Networking* 26.*5* (Oct. 2018), pp. 2174–2187, ISSN: 1063-6692, 1558-2566, DOI: 10.1109/TNET.2018.2868054.

[92] S. Contiu, R. Pires, S. Vaucher, M. Pasin, P. Felber, and L. Réveillère, « IBBE-SGX: Cryptographic Group Access Control Using Trusted Execution Environments », *in*: *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2018, pp. 207–218, DOI: 10.1109/DSN.2018.00032.

[93] Tai Liu, Zain Tariq, Jay Chen, and Barath Raghavan, « The Barriers to Overthrowing Internet Feudalism », *in*: *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, HotNets-XVI, New York, NY, USA: ACM, 2017, pp. 72–79, ISBN: 978-1-4503-5569-8, DOI: 10.1145/3152434.3152454.

[94] G. Danezis and I. Goldberg, « Sphinx: A Compact and Provably Secure Mix Format », *in*: *2009 30th IEEE Symposium on Security and Privacy*, May 2009, pp. 269–282, DOI: 10.1109/SP.2009.15.

[95] Chen Chen, Daniele Enrico Asoni, David Barrera, George Danezis, and Adrian Perrig, « HORNET: High-Speed Onion Routing at the Network Layer », *in*: *arXiv:1507.05724 [cs]* (July 2015), arXiv: 1507.05724 [cs].

[96] Zachary Amsden et al., « The Libra Blockchain », *in*: *Calibra corp.* (2019), p. 29.

[97] Nym, *Libra Is Blockchain Virtue-Signalling at Its Worst — Centralised, Not Private, and Good for Facebook*, https://medium.com/@nymtech/libra-is-blockchain-virtue-signalling-at-its-worst-centralised-not-private-and-good-for-5591ff7cc524, June 2019.

[98] H. Debnath, N. Gehani, X. Ding, R. Curtmola, and C. Borcea, « Sentio: Distributed Sensor Virtualization for Mobile Apps », *in*: *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, Mar. 2018, pp. 1–9, DOI: 10.1109/PERCOM.2018.8444605.

[99] Joon Park, Ruzanna Chitchyan, Anastasia Angelopoulou, and Jordan Murkin, « A Block-Free Distributed Ledger for P2P Energy Trading: Case with IOTA? », *in*: *Advanced Information Systems Engineering*, ed. by Paolo Giorgini and Barbara Weber, Lecture Notes in Computer Science, Springer International Publishing, 2019, pp. 111–125, ISBN: 978-3-030-21290-2.

[100] Amy K. Karlson, Brian R. Meyers, Andy Jacobs, Paul Johns, and Shaun K. Kane, « Working Overtime: Patterns of Smartphone and PC Usage in the Day of an Information Worker », *in*: *International Conference on Pervasive Computing*, Springer, 2009, pp. 398–405.

[101] Google, *The new multi-screen world: Understanding cross-platform consumer behavior*. Tech. rep., 2012, URL: https://www.google.com/think/research-studies/the-new-multi-screen-world-study.html.

[102] Fahim Kawsar and A.J. Bernheim Brush, « Home Computing Unplugged: Why, Where and when People Use Different Connected Devices at Home », *in*: *UBICOMP'13*, Zurich, Switzerland, pp. 627–636, ISBN: 978-1-4503-1770-2, DOI: 10.1145/2493432.2493494.

[103] Hendrik Müller, Jennifer L. Gove, John S. Webb, and Aaron Cheang, « Understanding and Comparing Smartphone and Tablet Use: Insights from a Large-Scale Diary Study », *in*: *Proceedings of the Annual Meeting of the Australian Special Interest Group for Computer Human Interaction*, OzCHI '15, New York, NY, USA: ACM, 2015, pp. 427–436, ISBN: 978-1-4503-3673-4, DOI: 10.1145/2838739.2838748.

[104] Tero Jokela, Jarno Ojala, and Thomas Olsson, « A Diary Study on Combining Multiple Information Devices in Everyday Activities and Tasks », *in*: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, New York, NY, USA: ACM, 2015, pp. 3903–3912, ISBN: 978-1-4503-3145-6, DOI: 10.1145/2702123.2702211.

[105] Daniel T. Wagner, Andrew Rice, and Alastair R. Beresford, « Device Analyzer: Understanding Smartphone Usage », *in*: *International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, Springer, 2013, pp. 195–208.

[106] • *Chart: The Tablet Boom Is Long Over | Statista*, https://www.statista.com/chart/16081/tab market-growth/, URL: https://www.statista.com/chart/16081/tablet-market-growth/.

[107] Steven J. Vaughan-Nichols, *Goodbye, Tablets*, July 2019, URL: https://www.computerworld.com/article/3405587/goodbye-tablets.html.

[108] Levin Michal, *Designing Multi-Device Experiences*, O'REILLY, 2014, ISBN: 1449340385.

[109] *Apache Cordova*, URL: https://cordova.apache.org.

[110] *NativeStript*, URL: https://www.nativescript.org/.

[111] *Facebook React Native*, URL: https://facebook.github.io/react-native/.

[112] *Microsoft Xamarin*, URL: https://visualstudio.microsoft.com/xamarin/.

[113] *Adobe PhoneGap*, URL: https://phonegap.com.

[114] *Google Flutter*, URL: https://flutter.dev/.

[115] *Electron*, URL: https://electron.atom.io.

[116] Eli Raymond Fisher, Sriram Karthik Badam, and Niklas Elmqvist, « Designing Peer-to-peer Distributed User Interfaces: Case Studies on Building Distributed Applications », *in*: *Int. J. Hum.-Comput. Stud.* 72.*1* (Jan. 2014), pp. 100–110, DOI: 10.1016/j.ijhcs.2013.08.011.

[117] Jérémie Melchior, Donatien Grolaux, Jean Vanderdonckt, and Peter Van Roy, « A Toolkit for Peer-to-peer Distributed User Interfaces: Concepts, Implementation, and Applications », *in*: *The 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '09, Pittsburgh, PA, USA, 2009, pp. 69–78, ISBN: 978-1-60558-600-7, DOI: 10.1145/1570433.1570449.

[118] Andrea Gallidabino and Cesare Pautasso, « Deploying Stateful Web Components on Multiple Devices with Liquid.js for Polymer », *in*: *ACM CBSE'2016*, pp. 85–90.

[119] *Apple Handoff*, URL: https://developer.apple.com/handoff.

[120] *Resilio Sync*, URL: https://www.resilio.com/individuals-sync.

[121] *Syncthing*, URL: https://syncthing.net.

[122] Leslie Lamport, « Time, clocks, and the ordering of events in a distributed system », *in*: *Communications of the ACM* 21.*7* (1978), pp. 558–565, URL: http://dl.acm.org/citation.cfm?id=359563 (visited on 03/31/2017).

[123] Kenneth P. Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky, « Bimodal Multicast », *in*: *ACM ToCS* (1998).

[124] A.-M. Kermarrec, L. Massoulié, and A. J. Ganesh, *Reliable Probabilistic Communication in Large-Scale Information Dissemination Systems*, tech. rep., 2000.

[125] Friedemann Mattern, « Virtual Time and Global States of Distributed Systems », *in*: *Parallel and Distributed Algorithms: proceedings of the International Workshop on Parallel & Distributed Algorithms*, ed. by M. Cosnard et. al., Elsevier Science Publishers B. V., 1989, pp. 215–226, URL: citeseer.nj.nec.com/mattern89virtual.html.

[126] A.-M. Kermarrec, Laurent Massoulié, and Ayalvadi J. Ganesh, « Probabilistic reliable dissemination in large-scale systems », *in*: *IEEE Transactions on Parallel and Distributed systems* 14.*3* (2003), pp. 248–258, (visited on 07/10/2017).

[127] M. E. J. Newman, « Power laws, Pareto distributions and Zipf's law », *in*: *Cities* 30 (Feb. 2013), pp. 59–67, ISSN: 02642751, DOI: 10.1016/j.cities.2012.03.001, URL: http://arxiv.org/abs/cond-mat/0412004 (visited on 07/27/2017).

[128] JinSeok Oh, Jin-woo Kwon, Hyukwoo Park, and Soo-Mook Moon, « Migration of Web Applications with Seamless Execution », *in*: *VEE '15*, Istanbul, Turkey, 2015, pp. 173–185, ISBN: 978-1-4503-3450-1, DOI: 10.1145/2731186.2731197.

[129] Michael Frigge, David C. Hoaglin, and Boris Iglewicz, « Some Implementations of the Boxplot », *in*: *The American Statistician* 43.*1* (1989), pp. 50–54, ISSN: 00031305, URL: http://www.jstor.org/stable/2685173.

[130] *Consul, by HashiCorp*, URL: https://www.consul.io.

[131] Adam Lipowski and Dorota Lipowska, « Roulette-wheel selection via stochastic acceptance », *in*: *Physica A: Statistical Mechanics and its Applications* 391 (Mar. 2012), pp. 2193–2196, ISSN: 03784371, DOI: 10.1016/j.physa.2011.12.004.

[132] Daniel Stutzbach and Reza Rejaie, « Understanding Churn in Peer-to-peer Networks », *in*: *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, Rio de Janeriro, Brazil, 2006.

[133] Tomasz Kalbarczyk and Christine Julien, « Omni: An Application Framework for Seamless Device-to-Device Interaction in the Wild », *in*: *Proceedings of the 19th International Middleware Conference on - Middleware '18*, Rennes, France: ACM Press, 2018, pp. 161–173, ISBN: 978-1-4503-5702-9, DOI: 10.1145/3274808.3274821.

[134] Charles T. B. Garrocho, Maurício J. da Silva, and Ricardo A. R. Oliveira, « D2D Pervasive Communication System with Out-of-Band Control Autonomous to 5G Networks: Project and Evaluation of a Middleware for Networking and Content Exchange to D2D Communication without Human Interaction », *in*: *Wireless Networks* (Aug. 2018), ISSN: 1022-0038, 1572-8196, DOI: 10.1007/s11276-018-1820-2.

[135] David Dearman and Jeffery S. Pierce, « It's on My Other Computer!: Computing with Multiple Devices », *in*: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, Florence, Italy: ACM, 2008, pp. 767–776.

[136] Timothy Sohn, Kevin A. Li, William G. Griswold, and James D. Hollan, « A Diary Study of Mobile Information Needs », *in*: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI'08, Florence, Italy: ACM, 2008, pp. 433–442.

[137] Richard Harper, Tom Rodden, Y Rogers, and Abigail Sellen, *Being Human: Human-Computer Interaction in the Year 2020*, Jan. 2008, ISBN: 978-0-9554761-1-2.

[138] Sangeun Oh, Hyuck Yoo, Dae R. Jeong, Duc Hoang Bui, and Insik Shin, « Mobile Plus: Multi-device Mobile Platform for Cross-device Functionality Sharing », *in*: *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '17, Niagara Falls, New York, USA, 2017.

[139] Nicolai Marquardt, Robert Diaz-Marino, Sebastian Boring, and Saul Greenberg, « The Proximity Toolkit: Prototyping Proxemic Interactions in Ubiquitous Computing Ecologies », *in*: *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, Santa Barbara, California, USA, 2011.

[140] Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg, « Side Channels in Cloud Services: Deduplication in Cloud Storage », *in*: *IEEE Security & Privacy* 8.*6* (2010), pp. 40–47.

[141] Jennifer Stisa Granick, « We Kill People Based on Metadata », *in*: *American Spies: Modern Surveillance, Why You Should Care, and What to Do About It*, Cambridge University Press, 2017, pp. 53–66, DOI: 10.1017/9781316216088.006.

[142] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong, « Freenet: A Distributed Anonymous Information Storage and Retrieval System », *in*: *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*, Berkeley, California, USA: Springer-Verlag New York, Inc., 2001, pp. 46–66.

[143] Michael J. Freedman and Robert Morris, « Tarzan: A Peer-to-peer Anonymizing Network Layer », *in*: *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, 2002.

[144] Marc Rennhard and Bernhard Plattner, « Introducing MorphMix: Peer-to-peer Based Anonymous Internet Usage with Collusion Detection », *in*: *Proceedings of the 2002 ACM Workshop on Privacy in the Electronic Society*, WPES '02, 2002.

[145] Michael J Freedman and Robert Morris, « Tarzan: A Peer-to-Peer Anonymizing Network Layer », *in*: (2002), p. 14.

[146] Arjun Nambiar and Matthew Wright, « Salsa: A Structured Approach to Large-scale Anonymity », *in*: *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, Alexandria, Virginia, USA, 2006, pp. 17–26, ISBN: 1-59593-518-5.

[147] David L. Chaum, « Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms », *in*: *Commun. ACM* 24.*2* (Feb. 1981), pp. 84–90.

[148] Stevens Le Blond, David Choffnes, Wenxuan Zhou, Peter Druschel, Hitesh Ballani, and Paul Francis, « Towards Efficient Traffic-Analysis Resistant Anonymity Networks », *in*: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, Hong Kong, China, 2013, pp. 303–314, ISBN: 978-1-4503-2056-6, DOI: 10.1145/2486001.2486002.

[149] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich, « Vuvuzela: Scalable Private Messaging Resistant to Traffic Analysis », *in*: *Proceedings of the 25th Symposium on Operating Systems Principles - SOSP '15*, Monterey, California: ACM Press, 2015, pp. 137–152, ISBN: 978-1-4503-3834-9, DOI: 10.1145/2815400.2815417.

[150] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis, « The Loopix Anonymity System », *in*: *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 1199–1216, ISBN: 978-1-931971-40-9.

[151] John P. Podolanko, Revanth Pobala, Hussain Mucklai, George Danezis, and Matthew Wright, « LiLAC: Lightweight Low-Latency Anonymous Chat », *in*: IEEE, Aug. 2017, pp. 141–151, ISBN: 978-1-5386-1027-5, DOI: 10.1109/PAC.2017.14.

[152] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson, « Touching from a Distance: Website Fingerprinting Attacks and Defenses », *in*: *Proceedings of the 2012 ACM Conference on Computer and Communications Security - CCS '12*, Raleigh, North Carolina, USA: ACM Press, 2012, p. 605, ISBN: 978-1-4503-1651-4, DOI: 10.1145/2382196.2382260.

[153] Stuart J. Russell and Peter Norvig, *Artificial Intelligence, A Modern Approach. Second Edition*, University of Michigan Press, Jan. 2003, p. 835.

[154] Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen, « CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays », *in*: *Journal*

of Network and Systems Management 13.*2* (June 2005), pp. 197–217, ISSN: 1064-7570, 1573-7705, DOI: 10.1007/s10922-005-4441-x, (visited on 10/25/2016).

[155] Amos Fiat and Moni Naor, « Broadcast Encryption », *in*: *Advances in Cryptology — CRYPTO' 93*, ed. by Douglas R. Stinson, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1993, pp. 480–491, ISBN: 978-3-540-48329-8.

[156] Douglas R Stinson, *Cryptography: theory and practice*, Chapman and Hall/CRC, 2005.

[157] Benoît Libert, Kenneth G. Paterson, and Elizabeth A. Quaglia, « Anonymous Broadcast Encryption: Adaptive Security and Efficient Constructions in the Standard Model », *in*: *Public Key Cryptography – PKC 2012*, ed. by David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Marc Fischlin, Johannes Buchmann, and Mark Manulis, vol. 7293, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 206–224, ISBN: 978-3-642-30056-1 978-3-642-30057-8, DOI: 10.1007/978-3-642-30057-8_13.

[158] Adam Barth, Dan Boneh, and Brent Waters, « Privacy in encrypted content distribution using private broadcast encryption », *in*: *International Conference on Financial Cryptography and Data Security*, Springer, 2006, pp. 52–64.

[159] J. A. Lockitt, A. G. Gatfield, and T. R. Dobyns, « A Selective Repeat ARQ System », *in*: *3rd International Conference on Digital Satellite Communications*, 1975, pp. 189–195.

[160] E. Weldon, « An Improved Selective-Repeat ARQ Strategy », *in*: *IEEE Transactions on Communications* 30.*3* (Mar. 1982), pp. 480–486, ISSN: 0090-6778, DOI: 10.1109/TCOM.1982.1095497.

[161] Larry L. Peterson and Bruce S. Davie, *Computer Networks: A Systems Approach, 3rd Edition*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003, pp. 97–110, ISBN: 978-1-55860-832-0.

[162] Xiaoyang Zhu and Youakim Badr, « Identity Management Systems for the Internet of Things: A Survey Towards Blockchain Solutions », *in*: *Sensors* 18.*12* (Dec. 2018), p. 4215, DOI: 10.3390/s18124215.

[163] *Nym: Building the next generation of privacy infrastructure*, https://nymtech.net/, 2019, URL: https://nymtech.net/.

Acknowledgements

My first thanks go to the long forgotten discoverers of caffeine. Most human societies are forever grateful, and I am indebted as well.

Thanks to Alexandra Elbakyan (who created SciHub) and all the freedom of information fighters, who enable societies to build knowledge and researchers to keep the conversation running, notwithstanding crooks and vultures.

A grateful thank you to Virginie Desroches, my beloved administration-kick-boxing specialist, who teaches us that all that paperwork is not so bad, and is a breeze of fresh air in this oh-so-serious world.

Thanks to all the WIDE's team: Michel, Davide, George, Erwann, and notably François Taïani, who is a precious team-dad and always up for a digression on literature or else. Louison, Quentin, Alex, Loïck, and the other colleagues who participated in making every noon meal a thrill and a moment to question ourselves. David Guyon, Benjamin Roussel, and the many traumatised PhD students who were always eager to share a coffee (!) break and whine about our respective supervisors.

I would also like to thank my academic mentors, mostly Isabelle Puaut and François Bodin, whose viewpoints fed my reflections while being a lot of fun.

I spare the worn-out family thank-you for their ears only, as they know I love them and—being as smug as myself—too much praise would only harm them. My sweetheart Axelle, deserves her praise, though, as she provided for me in every way during my redaction (and cooked me food you would be very jealous of).

A penultimate thank you to Vivien Quéma and Romain Rouvoy, who endorsed the time-consuming task of reviewing my manuscript.

Finally, I would like to give my warmest regards to David Bromberg, who guided me scrupulously throughout this PhD. His dedication to his students is examplary; he introduced me to academia without pretense, and kicked my butt in the right direction a number of times. By pointing out and acknowledging my weaknesses and strengths, he certainly made me a wiser man; I hope the partnership was reciprocal.

And to all the researchers, academic or not, past and future, that seek knowledge and study societies for the common goods, I am proud to be part of this community.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : Les e-squads : Un nouveau paradigme pour la conception d'applications ubiquitaires respectant le droit à la vie privée

Mot clés : réseaux distribués, vie privée, informatique ubiquitaire, apprentissage statistique

Résumé : L'émergence de l'Internet des Objets (IoT) menace actuellement le droit à la vie privée : une quantité croissante de périphériques connectés transmet continuellement des informations personnelles à des entités privées hors du contrôle des populations. Malgré tout, nous pensons que l'IoT pourrait tout aussi bien devenir un gardien de notre vie privée, s'il s'échappait du modèle client-serveur. Dans cette thèse, nous proposons un nouveau concept, celui d'*e-squad* : en permettant aux équipements connectés des utilisateurs de collaborer par voie de communication épidémique afin de construire de nouveaux services respectueux de la vie privée. Cette communication entraîne la création d'un savoir sur l'utilisateur, qui peut être utilisé par l'e-squad afin de proposer des applications prédictives. Nous démontrons que la collaboration permet de surmonter les piètres connectivité et performance des objets connectés. Enfin, en fédérant les e-squads, nous construisons un robuste réseau pour l'échange de fichier anonyme, prouvant le potentiel des e-squads au-delà des applications à un seul participant. Nous espérons que ce manuscrit inspirera la création de technologies plus agréables et humaines.

Title: E-squads: A novel paradigm to build privacy-preserving ubiquitous applications

Keywords: decentralised networks, privacy, ubiquitous computing, machine learning

Abstract: The emergence of the Internet of Things (IoT) has proved to be dangerous for the people's right to privacy: more and more connected appliances are sharing personal data about people's daily lives to private parties beyond their control. Still, we believe that the IoT could just as well become a guardian of our privacy, would it escape the centralised cloud model. In this thesis, we explore a novel concept, the *e-squad*: to make one's connected devices collaborate through gossip communication to build new privacy-preserving services. We show how an e-squad can privately build knowledge about their user and leverage this information to create predictive applications. We demonstrate that collaboration helps to overcome the end-devices' poor availability and performance. Finally, we federate e-squads to build a robust anonymous file exchange service, proving their potential beyond the single-user scenario. We hope this manuscript inspires the advent of more humane and delightful technology.