CASCADE: Reliable Distributed Session Handoff for Continuous Interaction across Devices

Yérom-David Bromberg, Adrien Luxey, François Taïani

CASCADE: Reliable Distributed Session Handoff for Continuous Interaction across Devices

Yérom-David Bromberg, Adrien Luxey, François Taïani
Univ Rennes, Inria, CNRS, IRISA — {firstname.surname}@irisa.fr.

***Abstract*—Allowing users to navigate seamlessly between their personal devices while protecting their privacy remains today an ongoing challenge. Existing solutions rely on peer-to-peer designs, and blindly flood the network with session messages. It is particularly hard to come up with proposals that are both cost-efficient and dependable while relying on poorly connected mobile appliances. We propose CASCADE, a distributed protocol to share applicative sessions among one's devices. Our proactive session handoff algorithm takes inspiration from the BitTorrent P2P file sharing protocol, but adapts it to the specific characteristics of our problem. It eschews in particular trackers, and limits the seeders of each session to the devices most likely to be used next, as computed by a decentralized aggregation protocol. A key aspect of our approach is to trade off network costs for reliability, while providing a faster session handoff than centralized solutions in the vast majority of the cases.**

I. INTRODUCTION

In recent years, our relations to digital technologies have become more symbiotic than ever. From desktops to smartphones to connected TVs and cars, the proliferation of Internet-connected devices among us is a step forward to Mark Weiser's vision of "ubiquitous computing" [36]. People nowadays own multiple devices (e.g. PCs, smartphones, smart watches, tablets, notebooks...) that they use interchangeably depending on the context (their location, the task at hand, their preference...) [21], [24].

This situation raises a major concern for application designers: how can we maintain a private, convenient and delightful interaction [7] with users when they switch back and forth between a variety of heterogeneous devices? The accidental complexity [14] of using different platforms must be taken out of the users' shoulders: applications need to provide a *consistent* and *continuous* experience across platforms. To this end, recent breakthroughs in web technologies and engineering have practically solved the *consistency* problem. Leveraging HTML, CSS, JavaScript, and native browsers embedded in modern appliances, developers can now provide a consistent output on any support using a single codebase [2], [4], [6], [10]. As a result, web technologies are becoming a standard for building device-agnostic applications [12], [15], [22], [23], [28], [32], [34].

To provide a *continuous* interaction across several devices, the state of a user's applications must be propagated between her devices. (This propagation is also called *session handoff*.) At the moment, the vast majority of existing solutions rely on cloud services to act as a central point of synchronization. For instance, Evernote [8], 1Password [1], Wunderlist [11], Chrome [9], Amazon Kindle [3], etc. all rely on either Google Drive, Dropbox, iCloud, Amazon S3 and/or other corporate data centers to store and retrieve the user's applicative session. Such an approach raises two concerns: firstly, it violates the right to privacy, by letting applications providers record and analyze their clients' behavior; secondly, it requires the newly opened device to be connected to the Internet to fetch the latest applications' state.

To tackle the privacy issue, a logical remedy is to keep the user's data on appliances that she owns. To the best of our knowledge, though, there exists no adequate solution proposing distributed session handoff among one's devices. A few studies have tackled applications' state sharing in the context of Distributed User Interfaces, by leveraging peer-to-peer (P2P) techniques [18], [31]. Alas, they require a constant interconnection between devices, and blindly flood the devices with state updates. Applied to session handoff on potentially mobile devices, this approach is neither dependable, scalable, nor economical.

To overcome these limitations, we propose CASCADE, a novel solution to deliver a continuous user experience across personal devices. CASCADE's objectives are to:

- respect the right to *privacy* by keeping all data on one's devices;
- be *reliable*: CASCADE should have no centralization point and function in any situation;
- be *fast*: the user should instantly find her previous session on her newly opened device;
- be *lightweight*: because most modern devices are mobile assets, CASCADE should use the minimum network and energy resources;
- *scale* with the number of devices: as demonstrated by the popularity of the Internet of Things (IoT), or by Cisco's prediction that there will be 50 billion connected objects by 2020, the amount of devices owned per person is likely to keep rising up. In that regard, we consider a user that would own and frequently use a dozen of devices.

To achieve these goals, CASCADE performs a *proactive session handoff* using a protocol that is loosely inspired from BitTorrent, a mature P2P file sharing protocol. CASCADE adapts the principles of BitTorrent to the specific characteristics of the session handoff problem: it eschews in particular trackers, and limits the seeders of each session to the devices most likely to be used next, as computed by a decentralized

aggregation protocol.

Our contributions are as follows:

- We propose CASCADE, a *decentralized session handoff protocol* that reinterprets some of the ideas of the P2P BitTorrent protocol in the context of personal device interactions. We model our protocol precisely using *communication automata*, a rigorous formalism capturing the behavior 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 devices most likely to be used next.
- We present an *in-depth evaluation* of our proposal based on a number of synthetic user models, in order to exercise our solution across a wide range of usage scenarios.

The remainder of this paper is structured as follows. Section II details the CASCADE concepts and its approach. Section III evaluates the proposed approach regarding the performances and different models emulating different types of user behavior. Section IV reviews related work. Finally, Section V discusses future work and concludes.

II. THE 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 behavior, 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 II-A. Then, Section II-B will cover how CASCADE gathers knowledge on the user to proactively exchange the user's session between devices.

A. Peer-to-peer session handoff

CASCADE is loosely based on the BitTorrent file-sharing protocol [16]. 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 [29], 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 $Sess = \left(\textcircled{\scriptsize t}, W_s, \mathcal{H}, h, bf\right)$. 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 $\textcircled{\scriptsize t}$, 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 of the session it refers to $\mathcal{H}$.

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 $S \triangleright H$.

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 behavior, and is used to propagate the session to relevant peers. The predictive multicast is one of the key contributions of the paper, and will be discussed in details in Section II-B. γ is used when a peer needs to do a traditional broadcast. Moreover, ϵ event enables to trigger a transition without the occurence 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.

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

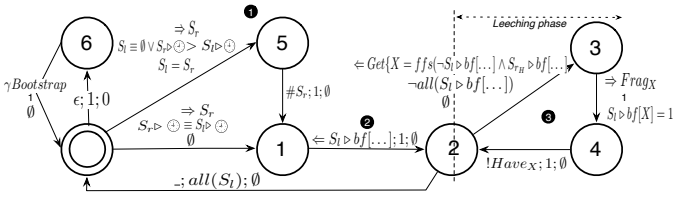


Fig. 1. Client behavior of CASCADE, i.e leecher

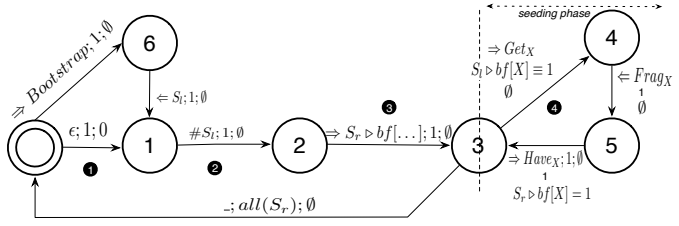


Fig. 2. Server behavior of CASCADE, i.e seeder

Additionally, $\mathcal{B}(\mathcal{M})$ is the set of constraint conjunctions on $\mathcal{M}$. Furthermore, $\rightarrow \subseteq Q \times Evt \times \mathcal{M} \times \mathcal{B}(\mathcal{M}) \times Act$ is the set of transitions.

Concretely, 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:

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

According to the aforementioned definitions, Figure 1 and Figure 2 describe the control flow that enables CASCADE to exchange sessions, and more particularly the behavior of a device when it acts either 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.

The user activity is modeled 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 sessions (See Figure 1 ❶, transition ❍→ ⑥). In a similar manner, A new session is created every time the user leaves a device d (See Figure 2 ❶). After chunking the new session into pieces, d advertises it to thoughtfully selected peers (See Figure 2 ❷). The recipients perform a predictive broadcast to advertise this new session only if they have never been notified of it yet (See Figure 1 ❶, transition ⑤ → ①). A new session is detected when its related metadata message S_r is received from a remote peer with a timestamp ⏲ 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 1 ❶, transition ❍ → 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. Whatever, as soon as *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 1 ❷). *Leechers* enter then in a *leeching phase* (See Figure 1 ❸): they first calculate the *First Find Set* (ffs) of the inversed bit field of their ongoing session (i.e. $ffs(\neg S_l \triangleright bf[\ldots])$), to find the index of the first bit that is set to 1 to identify the next chunk to request. Every time a peer d' completes the downloading of 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 2 ❸, ❹).

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

B. 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 behavior across time in a compact manner, (ii) propagating the user's behavior among devices, (iii) and finally selecting adequate peers to share the session, by inferring the future user's behavior.

ISSUE 1, representing the user's behavior: as in our previous work [30], 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 behavior: in order to predict the user's future behavior, all devices must gain a global understanding of the user's past behavior, 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 leverage on probabilistic dissemination protocols [13], [17], [25], precisely implementing a probabilistic distributed broadcast. This algorithm from earlier works in our team [30] has been called STORYTELLER in the remainder of the paper.

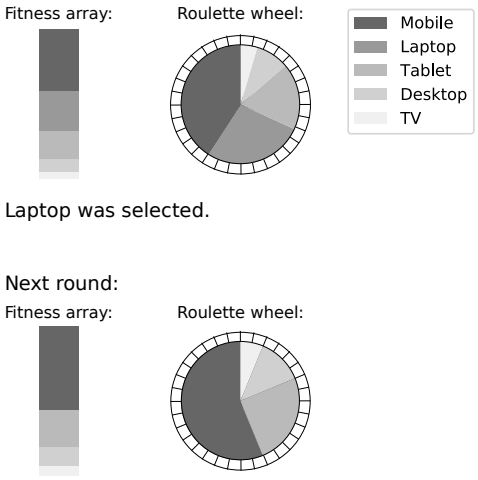


Fig. 3. Illustration of a roulette-wheel selection.

ISSUE 3, 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* $sc = \left[sc_{d_{\text{curr}} \to d}\right]_{d \in \mathcal{D}}$. $sc_{d_{\text{curr}} \to d}$ is simply the number of times that Alice has switched from d_{curr} to d in the past, and 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 would then be to preferably target the devices that have the highest score values, and hence 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, thus requesting reactive download from other peers. 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 [27] 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 (in our case, the score $sc_{d_{\text{curr}} \to d}$). The name comes from the analogy with a roulette wheel in a casino, where each individual is given a number of holes on the wheel proportional to its score. The roulette 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 CASCADE's *fanout*, a configuration parameter. Every time d chooses a device, it is removed from sc. In Figure 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 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 online 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 Figure 1 and Figure 2. By directing the flow of sessions exchanges, it drives the performance of the proactive session handoff, and guarantees the possibility of the reactive fallback.

We will now proceed to the evaluation, where we will analyze CASCADE's efficiency at performing proactive and reactive session handoff, and its network consumption.

III. EVALUATION

We first introduce several behavioral models in Section III-A, that will be used to evaluate the performances of CASCADE under different scenarii. In Section III-B, we introduce our experimentation methodology. In particular, we describe the different configuration parameters that are used during the experiments. In Section III-C, we show how the STORYTELLER sub-system behaves under our churn model. Finally, we evaluate the performances of our approach CASCADE in Section III-D.

A. Evaluation testbed

To the best of our knowledge, there exists no real-world dataset describing a user's behavior among its set of devices. In this context, we have designed an evaluation testbed to evaluate CASCADE's performance in different usage scenarii. The latter includes several standard user behavioral models to mimic the user behavior. These models are introduced in Section III-A1. Moreover in the specific context of this paper, to evaluate our protocol's resilience to devices' appearance and disappearance, we are introducing a model of churn covered in Section III-A2.

In our testbed, the user's interaction is considered atomic: the user provides a new session at the same time as she requests the previous one.

1) Proposed user behavior models: for our experiments, we need to craft sequences of user behavior that show various degrees of predictability. Towards this aim, we will design a discrete time Markov model per kind of desired behavior.

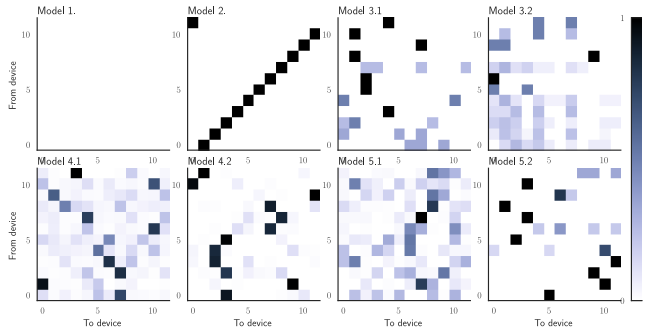


Fig. 4. Heatmaps of the transition matrix $\mathcal{P}_\mathcal{M}$ of each proposed model, with $N = 12$ devices. For each model, the i^{th} row of the heatmap represents the vector of transition from d_i: $\mathcal{P}_\mathcal{M}(d_i, *)$. Model *1. uniform* has a constant probability of $1/12$.

Each sequence will be generated by randomly walking on their respective Markov chain.

Indeed, we can consider the user's behavior as a stochastic process having a finite state space (the devices) and a discrete time. Further assuming that the next used device does only depends on the one that is currently used, and not on the previous ones (the Markovian property), the user device switching process is a discrete time Markov process.

Such a Markov process $\mathcal{M}$ can be entirely defined by its state space $\mathcal{S}$, the probability of its initial state Π, and the transition matrix $\mathcal{P}_\mathcal{M}$ that gives the probability to switch from a state to another. In our case, $\mathcal{S} = \mathcal{D}$, the set of $N = 12$ devices. We assume that each state is initially equiprobable: $\forall d \in \mathcal{D}, \Pi_d = 1/N$. In the end, each of our device switching models will be defined solely by its transition matrix $\mathcal{P}_\mathcal{M}$:

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

We propose five different ways to generate types of user behavior. For three of these, we create two variants, leading to a total of eight different models. Figure 4 shows an instance of each model's transition matrix $\mathcal{P}_\mathcal{M}$, using $N = 12$ devices, such that the i^{th} row $\mathcal{P}_\mathcal{M}(d_i, *)$ represents the transition probabilities from device d_i. We designed these models to show both dense transition matrices (that always generate a non-null probability to switch from any device to any other), and sparse ones (that make some device transitions impossible). We also tried to create transition probabilities that would be either very uniform (such that several devices have the same probability of being used) or dominated by one device's probability (that would always be chosen, except in rare exceptions). Their creation is 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 to predict the next appliance that she will use. In this model, every transition vector $\mathcal{P}_\mathcal{M}(d, *)$ is deterministic;
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} favors the use of certain devices (e.g. Alice uses her smart-phone more often than her mother's laptop). We create the transition matrix $\mathcal{P}_\mathcal{M}$ by building a Markov transition matrix out of the sequence of observed states $S_\mathcal{M}$.
The longer the input sequence $S_\mathcal{M}$, the denser the transition matrix. Thus, we generate 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 [33]: 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 seems plausible.
We propose a model where, to each row of the transition matrix, we assign a random permutation of Zipf's law's PMF using $n = N$ using a random exponent s.
We generate two variants: in model **4.1**, we pick s in $[1, 2.5]$, which puts the biggest probability $z(1; n, s)$ between 0.32 and 0.75. In model **4.2**, we draw s from $[2.5, 5]$, such that $z(1; n, s)$ lies between 0.75 and 0.96. In model 4.2, a transition probability dominates the others way more than in 4.1;
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: two desktop computers from two faraway locations will never be accessed in a row. In CASCADE, having null probabilities lowers the search space of our peers selection function (cf Section II-B).
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 an outgoing transition equals zero, the sparser the matrix. We thus generated two models **5.1** and **5.2** with $s = 1$ and $s = 4$ respectively.

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

For each model, we generate a sequence of the user's activity by randomly walking on the obtained Markov chain, starting from a random device. The output sequence $S_{\text{tot}} = \{r_1, \ldots, r_L\}$, of size L, drives the evaluation of our protocol.

The beginning of the sequence, $S_{\text{init}} = \{r_1, \ldots, r_{L_{\text{init}}}\}$ (such that $L_{\text{init}} < L$) is given to the devices on bootstrap: it provides them with an initial knowledge of the user's behavior. The remainder of the sequence, $S_{\text{exp}} = \{r_{L_{\text{init}}+1}, \ldots, r_L\}$, gives the order with which we use the devices during the experiment. As already stated, each device's usage is emulated by calling their REST function. For a given user sequence length L and a given bootstrap sequence length L_{init}, the experiment thus counts $L_{\text{exp}} = L - L_{\text{init}}$ interactions with the user.

2) Churn model: We call *churn* the act of devices leaving or entering the network. 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 churn model such that the devices enter and leave the system 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 behavior drives the churn: if our algorithm survives the former, it should handle the latter.

We randomly assign a target *average uptime rate* υ to each device d on bootstrap. Initially, d is connected with a probability υ. Stutzbach and Rejaie [35] showed that the Weibull distribution fit P2P churning behaviors 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:

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

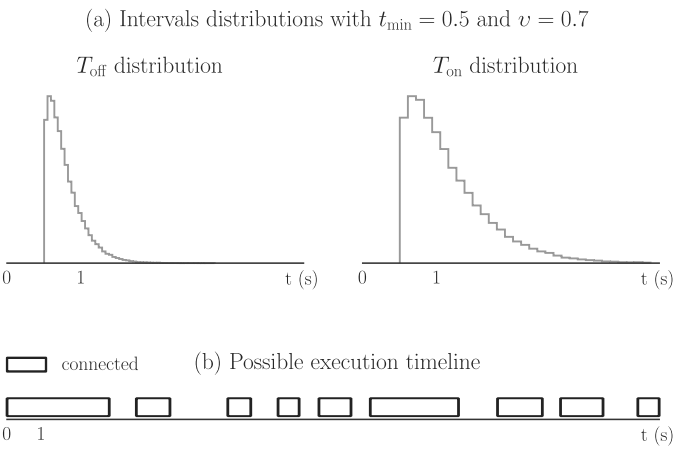


Fig. 5. Distribution of the random variables driving the disconnection and connection intervals (resp. T_{off} and T_{on}) using $t_{\text{min}} = 0.5$, and $\upsilon = 0.7$, and the timeline of a device's execution using these parameters.

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

Figure 5 (a) shows two histograms representing the distribution of the values taken by T_{off} and T_{on} for $t_{\text{min}} = 0.5$, 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 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.

In practice, our churn model makes the behavior 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), we crafted a simple oracle, thus avoiding the need to bloat our protocol with acknowledgments. 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 [5], although this elementary oracle was enough to suit our needs.

We have designed two behavioral 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 behaviors, and that our churning model is uncompromising enough to assess the efficiency of CASCADE.

B. Conducted experiments

Experiment methodology: an experiment is made of different steps. We first compute a behavioral 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 *experiments set* comprises an experiment per user behavioral model described in Section III-A1.

Experiment setting: we conducted 10 *experiments sets* with the same parameters twice: once enabling the churn, and another time with the churn deactivated for comparison.

Every session weights the same size $W_s = 1$MB, while the session pieces weight $W_p = 16$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 [26]. Further, we set the number of devices to $N = 12$. We argue that this number of devices is three times above modern usage behaviors [21]. 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 behavior, 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 interactions 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.

C. Evaluation of STORYTELLER

To provide a global overview of the user's past behavior to predict the future, we leverage on a previous contribution named STORYTELLER, which is based on a probabilistic dissemination protocol [30]. However, STORYTELLER was initially designed in a perfect world, i.e without any churns. Hence, we are re-evaluating performances of STORYTELLER and its impact on CASCADE according to our previously introduced churn model.

STORYTELLER *principles:* devices report their local sequence of the user's actions every time it is updated by STORYTELLER. At time t, a device d knows its local sequence $S_{t,d}$, whereas the user performed a sequence S_t of actions, a superset of $S_{t,d}$. The goal of STORYTELLER is to successfully propagate the real sequence S_t of the user's interactions to the devices.

Thus, to assess the protocol's efficiency, we compare, for each device d and each time t, the size of $S_{t,d}$ against that of S_t. By computing 10 experiments sets in two situations (with and without churn), we obtain the results presented in Table I. The number of collected local sequences is written in column "# sequences". Since most local sequences are complete, we show the number of incomplete in column "# incomplete". *Among these incomplete sequences*, the median difference $|S_t| - |S_{t,d}|$ is shown in column "median diff.". We finally show the maximal difference in the last column "max. diff."

Using the outputs from the experiments described in Section III-B, we computed the results found in Table I.

TABLE I
PERFORMANCE OF STORYTELLER

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

The amount of recorded sequences is 30% bigger without churning: devices being always online, they share information more often. The amount of incomplete sessions is way higher with the 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.

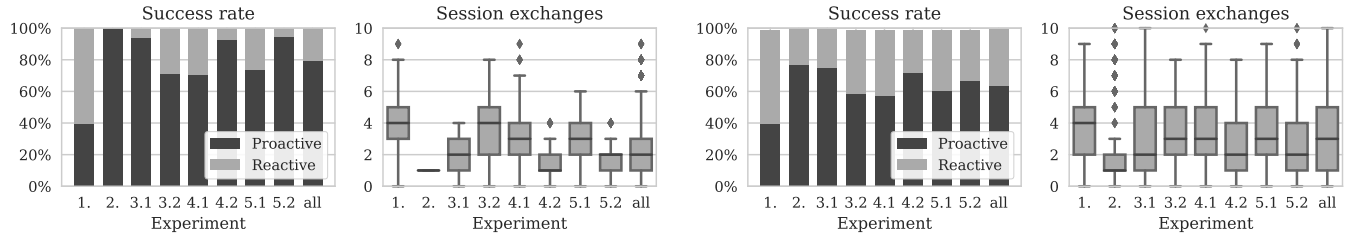
The local sequences are used for the prediction of CASCADE: a small sequence error can impair the session handoff more heavily. We show in the next Section that our predictive scheme remains efficient despite these incoherences.

D. 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.

It appears clearly that one of our objectives is to achieve the highest rate of proactive handoffs as possible: this is our protocol's purpose. However, successfully falling back to reactive handoff is not so bad: it still provides the same functionality as a centralized session handoff solution but in a distributed manner. However, a distributed alternative to cloud-based services must first and foremost be dependable to convince its userbase. Hence, our most prevalent objective is to reach as close as possible 0% of misses.

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



(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 $s = 4$.

Fig. 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.

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, though, we do not 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 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 behavioral model described in Section III-A1, 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 (similarly coined "all"), reading the distribution of the number of completed session exchanges per interaction. The central darker line represents the median, the box read the lower and upper quartiles, and the whiskers represent 3/2 of the interquartile range (they are Tukey plots [19]).

1) A world without churn: When devices are always on, the peers selection function presented in Section II-B can show its full potential: the session exchange decision is always made according to the sequence of the user's behavior (cf Section III-A1), and is not impeded by the remote devices' state. These results are shown in Figure 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 behavior.

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 behavior 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. It comes from the fact that: the user's behavior is deterministic, such that 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 irregular probabilities: a couple of devices always have a bigger probability of being chosen than the others. We conclude that the peers selection function works best when the user has a very predictable behavior.

The last three models, that score from 70 to 75%, propose different user behaviors: 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 behaviors still provide efficient proactive handoff: devices reactively download the session 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 behavior 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 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 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.

2) A world with churn: Now that we have thoroughly studied CASCADE's behavior in an ideal (yet power-wasting) situation where our user never turn off her devices, we can see how its performances hold when she does. These results are displayed in Figure 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 arise 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. 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 behavior models, apart from the *3.2* (which median session exchanges when down from 4 to 3): the one with the denser transition matrix. 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 behavior 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 unpredictable model like *3.2* (whose success rate is 60%, with a median of 3 session exchanges per interactions). More importantly, whatever the user's behavior, we believe that CASCADE would remain as dependable as it proved to be in the above experiments.

IV. RELATED WORK

The problem of providing seamless, cross-device migration capability for interactive applications, including those based on the Web, has received considerable attention from both the research community and the industry. This is due, at least partly, because such feature is an important ingredient to achieve ubiquity and to provide a more fluid user experience across the different devices through which applications can be accessed.

Oh et al. [34] present a framework for Web application migration that allows saving the session state of an application in a source device and restoring it on a Web browser running on a target device. The application then has its operation resumed from the point it was stopped in the source device.

Bellucci et al. [12] describe a server-based approach for persisting and restoring the state of Web applications, allowing seamless migration across heterogeneous devices. The mechanism is based on a proxy that intercepts HTTP requests and instruments received Web pages with code to call a migration server whenever session migration is necessary. Control of session migration is performed by the user via a control panel, from which the user can trigger the migration of a session. The control panel at each device also interacts with the migration server to keep a list of devices currently available as targets for migration. Similarly to our work, their approach does not require the original Web applications to be modified in order to enable their migration. However, their approach differs from ours as application state transfer is always performed through a dedicated server – in our approach, application state is propagated directly among devices, resulting in no single point of failure. Moreover, migration control is autonomic in our approach, whereas in Bellucci et al. every migration event requires explicit user intervention.

The Imagen system, proposed by Lo et al. [28], uses a similar mechanism of source code instrumentation to inject code to control migration. This can be performed in two ways: by requiring application developers to use a code transformation tool, or by requiring users to install an HTTP proxy that intercepts Web pages and instruments their code. Either way, the result is the addition of a GUI button at the end of the page, which the user can push to trigger the execution of injected code to perform state saving. Saved state is stored persistently and a URL is provided, which can be used to retrieve and load the application on the target device. Although the approach,

in principle, enables migration of any Web application, the user has to manually determine both the timing and the target device of migration.

Mikkonen et. al [32] coin the term *liquid Web applications* to refer to applications that "can seamlessly and dynamically migrate from one device to another, following the user attention and usage context". The approach is based on the more general concept of *liquid software* [20], which refers to software (not only end-user applications) that exists independently of the device where it is currently running, being able to work across devices, both simultaneously and at different times, without disruption of its operation. Similarly to our work, liquid Web applications explicitly distinguish between the application data maintained at the server side (e.g., on a SaaS cloud infrastructure) and the application state maintained at client side (in the browser). The latter, despite being transient in nature, is crucial to achieving seamless user experience when migrating application sessions. In fact, while application data mobility is well-supported by current SaaS systems, transient application state is lost if a proper session handoff mechanism is not in place.

It is also worth mentioning a number of industry initiatives on the way to provide seamless end-user experience with applications that work across devices. Apple Handoff[2], first introduced in iOS8 and macOS Yosemite, enables application interaction to automatically switch from one device to another across a local network link. It works with a number of native Apple applications and an SDK is provided for third-party developers to include the feature in their applications. Nevertheless, the feature is limited to the Apple ecosystem.

Similarly, examples that resemble application session migration can be found in the Google SaaS ecosystem. In Google Docs, for instance, a user may start editing a document on a device and later switch to another device and still have access to the same document contents and editing capabilities. Nevertheless, such a process requires a lot of user interaction (explicitly starting the Web application, selecting the same document, and moving to the right point in the document to resume editing it). The approach proposed in this paper can be seen as complementary to current SaaS Web applications, as it makes the process of moving the transient application state smooth and effortless, at the same time that it leverages on cloud to make application data content directly available for the target device.

V. Conclusion and Future Work

In this paper, we have presented CASCADE, a novel approach to provide a continuous interactive experience to users navigating between multiple personal devices, a problem known as *session handoff*. CASCADE avoids any remote service dependency and keeps a user's personal data on her own devices. As such, CASCADE is inherently private: no personal information is leaked to any third party. CASCADE revisits some of the ideas of the P2P BitTorrent file sharing protocol while adapting them to the context of personal mobile devices to implement a *proactive session handoff* protocol.

We have shown through an in-depth evaluation that CASCADE is dependable and communication efficient, and is able to proactively deliver session information to the correct device with a success rate of over 60% in realistic user models. Combining this result with its reactive fallback, CASCADE successfully transmits user sessions in 98.9% of the studied cases.

In the future, we envisage to perform user studies and gain more insights into the benefits of our strategy. We would also like to experiment with other peer selection strategies, for instance taking into account additional information such as the time of day of the user's location. Finally, leveraging local connectivity could open new areas of application.

This research was partially funded by the O'Browser ANR grant (ANR-16-CE25-0005-03).

[2]https://developer.apple.com/handoff/

References

[1] 1 password. 1password.com.
[2] Adobe PhoneGap. phonegap.com.
[3] Amazon kindle. www.amazon.com/kindle.
[4] Apache Cordova. cordova.apache.org.
[5] Consul, by hashicorp. https://www.consul.io/.
[6] Electron. electron.atom.io.
[7] Ethical design manifesto. ind.ie/ethical-design/.
[8] Evernote. evernote.com.
[9] Google chrome. www.google.fr/chrome.
[10] Ionic framework. ionicframework.com.
[11] Wunderlist. www.wunderlist.com.
[12] F. Bellucci, G. Ghiani, F. Paternò, and C. Santoro. Engineering javascript state persistence of web applications migrating across multiple devices. In *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '11, pages 105–110, New York, NY, USA, 2011. ACM.
[13] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal Multicast. *ACM Transactions on Computer Systems*, 17:41–88, 1998.
[14] F. P. J. Brooks. No silver bullet essence and accidents of software engineering. 20(4):10–19.
[15] P.-Y. P. Chi and Y. Li. Weave: Scripting cross-device wearable interaction. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 3923–3932, New York, NY, USA, 2015. ACM.
[16] B. Cohen. The bittorrent protocol specification. Technical report, BitTorrent Inc., 2008.
[17] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. *ACM*, 1987.
[18] E. R. Fisher, S. K. Badam, and N. Elmqvist. Designing peer-to-peer distributed user interfaces: Case studies on building distributed applications. *Int. J. Hum.-Comput. Stud.*, 72(1):100–110, Jan. 2014.
[19] M. Frigge, D. C. Hoaglin, and B. Iglewicz. Some implementations of the boxplot. *The American Statistician*, 43(1):50–54, 1989.
[20] A. Gallidabino, C. Pautasso, V. Ilvonen, T. Mikkonen, K. Systä, J. P. Voutilainen, and A. Taivalsaari. On the architecture of liquid software: Technology alternatives and design space. In *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 122–127, April 2016.
[21] Google. The new multi-screen world: Understanding cross-platform consumer behavior. Technical report, 2012.
[22] B. Hartmann, M. Beaudouin-Lafon, and W. E. Mackay. Hydrascope: Creating multi-surface meta-applications through view synchronization and input multiplexing. In *Proceedings of the 2Nd ACM International Symposium on Pervasive Displays*, PerDis '13, pages 43–48, New York, NY, USA, 2013. ACM.

[23] M. Husmann, N. M. Rossi, and M. C. Norrie. Usage analysis of cross-device web applications. In *Proceedings of the 5th ACM International Symposium on Pervasive Displays*, PerDis '16, pages 212–219, New York, NY, USA, 2016. ACM.

[24] F. Kawsar and A. B. Brush. Home computing unplugged: Why, where and when people use different connected devices at home. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '13, pages 627–636, New York, NY, USA, 2013. ACM.

[25] A.-M. Kermarrec, L. Massoulié, and A. J. Ganesh. Reliable Probabilistic Communication in Large-Scale Information Dissemination Systems. Technical report, 2000.

[26] A.-M. Kermarrec, L. Massoulié, and A. J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed systems*, 14(3):248–258, 2003.

[27] A. Lipowski and D. Lipowska. Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, 391:2193–2196, Mar 2012.

[28] J. T. K. Lo, E. Wohlstadter, and A. Mesbah. Imagen: Runtime migration of browser sessions for javascript web applications. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 815–826, New York, NY, USA, 2013. ACM.

[29] A. Loewenstern and A. Norberg. Dht protocol. Technical report, BitTorrent Inc., 2008.

[30] A. Luxey, Y.-D. Bromberg, F. Costa, V. Lima, R. Da Rocha, and F. Taïani. Sprinkler: A probabilistic dissemination protocol to provide fluid user interaction in multi-device ecosystems. In *PerCom 2018*, Athens, Greece, Mar. 2018.

[31] J. Melchior, D. Grolaux, J. Vanderdonckt, and P. Van Roy. A toolkit for peer-to-peer distributed user interfaces: Concepts, implementation, and applications. In *Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '09, pages 69–78, New York, NY, USA, 2009. ACM.

[32] T. Mikkonen, K. Systä, and C. Pautasso. *Towards Liquid Web Applications*, pages 134–143. Springer International Publishing, Cham, 2015.

[33] M. E. J. Newman. Power laws, Pareto distributions and Zipf's law. *Cities*, 30:59–67, feb 2013.

[34] J. Oh, J.-w. Kwon, H. Park, and S.-M. Moon. Migration of web applications with seamless execution. *SIGPLAN Not.*, 50(7):173–185, Mar. 2015.

[35] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, New York, NY, USA, 2006.

[36] M. Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 1999.