Sprinkler: Um protocolo de disseminação probabilística para fornecer interação fluida do usuário em ecossistemas de vários disposit

Adrien Luxey, Yérom-David Bromberg, Fábio M Costa, Vinícius Lima, Ricardo da Rocha, François Taïani

Sprinkler: Um protocolo de disseminação probabilística para fornecer interação fluida do usuário em ecossistemas de vários dispositivos

Adrien Luxeyÿ , Yerom-David Bromberg ÿ , † Fábio M. Costa , Vin´ÿcius Lima† , Ricardo CA da Rocha† , François Ta¨ÿaniÿ
ÿUniv. de Rennes 1 / IRISA / Inria, Rennes, França, †Univ. Federal de Goiás, Goiânia-GO, Brazil
ÿ{nome.sobrenome}@irisa.fr, † fmc@inf.ufg.br, viniciusb50,@gmail.com, rcarocha@ufg.br

Resumo— Oferecer interações fluidas de vários dispositivos aos usuários enquanto protege sua privacidade continua sendo um desafio contínuo. As abordagens existentes geralmente usam um design ponto a ponto e inundam as informações da sessão pela rede, resultando em soluções caras e muitas vezes pouco práticas. Neste artigo, propomos o SPRINKLER, um protocolo de disseminação probabilística descentralizado que usa um algoritmo de aprendizado baseado em fofoca para propagar de forma inteligente as informações da sessão para os dispositivos que o usuário provavelmente usará em seguida. Nossa solução permite que os projetistas troquem com eficiência os custos de rede pela fluidez e, por exemplo, é capaz de reduzir os custos de rede em até 80% em relação a uma estratégia de inundação, mantendo uma experiência de usuário fluida.

I. INTRODUÇÃO

Nas últimas décadas assistiu-se a uma proliferação exponencial de dispositivos ligados à Internet. Estamos entrando em uma nova era, mais um passo em direção à visão de Mark Weiser de computação ubíqua [39], onde o paradigma de um único usuário para um único dispositivo não se aplica mais. Os usuários possuem vários dispositivos (PCs, smartphones, relógios inteligentes, tablets, notebooks) e não gastam mais seu tempo em uma única tela de desktop para realizar suas atividades digitais diárias (navegação, pesquisa, compras e jogos online, streaming de vídeo) [20] , [25].

Esta tendência emergente está longe de ser isenta de desafios. Acessar tudo, a qualquer hora, em qualquer lugar, de forma contínua em vários dispositivos heterogêneos, na hora certa e no lugar certo, pode se tornar uma tarefa assustadora para os usuários.
Particularmente, o suporte para simplificar a experiência do usuário está ficando para trás. Conforme introduzido por Levin [33], os aplicativos direcionados a um ecossistema multidispositivo devem ser projetados com três conceitos fundamentais em mente: Consistência, Continuidade e Complementaridade (para abreviar os 3Cs). Por exemplo, um aplicativo pode não ter uma interface de usuário consistente em dispositivos heterogêneos, prejudicando a interação do usuário com vários dispositivos e, portanto, aumentando a frustração do usuário. Da mesma forma, com aplicativos não projetados para continuidade, os usuários podem ter que gerenciar a si mesmos como retomar seu contexto de interação de seu dispositivo atual para o próximo. Isso implica, na maioria das vezes, que os usuários tenham que transferir manualmente os dados atualizados do aplicativo, para frente e para trás entre os dispositivos, a fim de recriar as sessões de interação, comprometendo a fluidez da interação. Finalmente, vários dispositivos podem ser usados simultaneamente para complementar um ao outro, por exemplo, um dispositivo pode atuar como um controle remoto para pilotar outro dispositivo.

Este artigo aborda explicitamente a questão de fornecer interação contínua. De fato, com os últimos avanços nas tecnologias da Web, consideramos que o problema de projetar aplicativos consistentes está quase resolvido. Em particular, por meio do uso de HTML, CSS e JAVASCRIPT incorporados nativamente em navegadores da Web, os aplicativos baseados na Web tornaram-se consistentes em dispositivos heterogêneos [2], [4], [8] e agora são os candidatos mais adequados para suporta a experiência de vários dispositivos dos usuários [11], [13], [21], [22], [30], [34], [37].

Além disso, a partir de um estudo recente conduzido pelo Google [20], 90% dos usuários que cresceram em um ecossistema multi-dispositivo, alternam entre os dispositivos de forma sequencial. Em outros termos, um determinado aplicativo só pode estar ativo em um único dispositivo em um determinado momento. A prevalência do uso sequencial nos leva a: (i) focar nosso trabalho em interações sequenciais, e (ii) considerar o uso simultâneo de dispositivos, ou seja, o aspecto complementar da interação dos usuários, para trabalhos futuros.

Por fim, o problema de fornecer interação contínua entre dispositivos heterogêneos é atualmente abordado de forma massiva, contando de fato com provedores de nuvem que atuam como um ponto central de sincronização. Por exemplo, aplicativos populares como EVERNOTE [6], 1PASSWORD [1], WUNDERLIST [9], CHROME [7], AMAZON KINDLE [3], etc. dependem do GOOGLE DRIVE, DROPBOX, ICLOUD, AMAZON S3 e/ou seus próprios servidores para realizar o handoff de sessão, ou seja, para salvar e transferir a sessão interativa de um aplicativo no ecossistema multidispositivo. No entanto, essas soluções sofrem de uma deficiência importante: a atividade dos usuários e suas informações pessoais e privadas relacionadas são registradas pelos provedores de aplicativos.

Os usuários não esperam ter sua vida digital rastreada e analisada por terceiros que coletam dados em seu nome.

Até onde sabemos, apenas algumas abordagens procuraram proteger a privacidade dos usuários em um ecossistema de vários dispositivos. Essas abordagens normalmente exploram uma arquitetura peer-to-peer para salvar e transferir sessões de aplicativos apenas em dispositivos pertencentes aos usuários, evitando a necessidade de confiar em terceiros [17], [18], [32]. No entanto, como o comportamento dos usuários não é conhecido antecipadamente, as soluções mencionadas inundam cegamente a sessão interativa em andamento para todos os dispositivos do ecossistema, pois nenhum dispositivo é capaz de prever qual dispositivo será usado em se
Tal mecanismo de transmissão de força bruta, apesar de sua simplicidade, leva a desempenhos muito ruins, pois implica: (i) redundante

mensagens, (ii) consumo excessivo de largura de banda da rede, (iii) uma latência mais alta que afeta inerentemente a fluidez da interação do usuário e (iv) esgotamento de energia mais rápido. Além disso, não se adapta bem ao número de dispositivos. Embora o número atual de dispositivos pertencentes a um usuário seja em média de 4 dispositivos [20], espera-se que esse número aumente com o advento da Internet das coisas.

Neste artigo, apresentamos o SPRINKLER, uma nova abordagem para executar a transferência de sessão preditiva aprendendo de maneira descentralizada como o usuário se comporta. O SPRINKLER combina um protocolo de disseminação probabilística e um mecanismo proativo de transferência de sessão para fornecer interações de usuário perfeitamente fluidas em um ecossistema de vários dispositivos. Nossa solução: (i) não depende de nenhum ponto de centralização, (ii) tem um consumo de recursos de rede limitado e conhecido, (iii) é capaz de prever qual dispositivo tem maior probabilidade de ser usado em seguida, permitindo a transferência do sessão de interação sem inundação cega, (iv) respeita o direito do usuário à privacidade e (v) escala para um número arbitrário de dispositivos.

Nossas contribuições são as seguintes:

- Desenhamos o SPRINKLER, um protocolo baseado em dois algoritmos: o SPRINKLER Gossiper, que permite aos dispositivos conhecer o comportamento do usuário de forma distribuída; e o mecanismo de Handoff de Sessão do SPRINKLER , que usa esse conhecimento para enviar proativamente pedaços da sessão atual para dispositivos que provavelmente serão usados em seguida; • Avaliamos nossa abordagem com 8 diferentes modelos de Markov de tempo discreto para emular comportamentos de usuários; • Demonstramos que não existe um protocolo de divulgação ideal. É tudo sobre a compensação entre precisão de previsão, latência e consumo de rede. Mostramos em particular que o desempenho do SPRINKLER depende muito do comportamento do usuário. Quanto mais previsível for o usuário, melhor será o desempenho do SPRINKLER ; • O SPRINKLER permite que os projetistas troquem com eficiência os custos de rede por fluidez e, por exemplo, é capaz de reduzir os custos de rede em até 80% em relação a uma estratégia de inundação, mantendo uma experiência de usuário fluida.

A seguir, detalhamos os conceitos e a abordagem da SPRINKLER (Seção II). A Seção III então avalia a abordagem proposta quanto ao desempenho e diferentes modelos que emulam diferentes tipos de comportamento do usuário. Finalmente, a Seção IV considera trabalhos relacionados e a Seção V discute trabalhos futuros e conclusões.

II. CONCEITOS E ABORDAGEM DA SOLUÇÃO

Nosso objetivo é fornecer um protocolo de comunicação tal que toda vez que o usuário (que chamamos de Alice) abrir um de seus dispositivos, ele encontre seus aplicativos da Web como os deixou, independentemente do dispositivo que usou anteriormente. Como as informações de uso diário dos aparelhos de uma pessoa são um bem privado, desejamos evitar a dependência de um sistema de armazenamento externo, o que ameaçaria o direito à privacidade do usuário. Para tanto, nosso protocolo deve envolver apenas os dispositivos de Alice, aproveitando estratégias de comunicação distribuída. Em segundo lugar, queremos que nosso protocolo seja leve, para evitar a drenagem de energia dos dispositivos móveis que o executam.

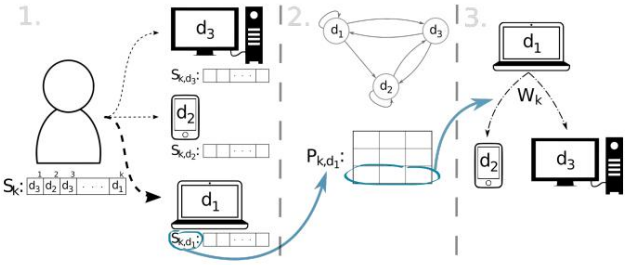

Fig. 1. Pipeline do Handoff da Sessão: do comportamento do usuário à inferência do modelo e ao compartilhamento dos dados da sessão.

O estado dos aplicativos do usuário é armazenado em um blob, arbitrariamente pesado em tamanho, a partir de então chamado de sessão. Nosso protocolo, SPRINKLER, compartilha proativamente partes (blocos) da sessão do usuário sempre que ele sai de um dispositivo, prevendo o dispositivo que ele usará em seguida. Essa previsão é obtida permitindo que os dispositivos aprendam o comportamento do usuário fofocando informações entre si. Queremos minimizar, por um lado, o tempo que Alice tem que esperar para que sua sessão anterior seja buscada quando ela abre um novo dispositivo e, por outro lado, o tráfego de rede induzido por trocas de sessão.

Para isso, nosso protocolo é constituído por dois algoritmos: o SPRINKLER Gossiper permitirá que os dispositivos conheçam o comportamento do usuário; o algoritmo de transferência de sessão do SPRINKLER usará essas informações para enviar proativamente partes da sessão atual para os dispositivos que mais provavelmente será usado a seguir.

A. Abordagem geral

Consideramos uma usuária, Alice, que usa um número de N dispositivos, D = {d1, ..., dN }. Podemos modelar o uso de seus dispositivos por como uma sequência de interações: S = {r1, r2, ..., interação r_i Alice , ...}. Cada ri é caracterizada por um par (dri , tri) ÿ D × R, o que significa que Alice começou a usar o dispositivo dri no tempo tri (e parou de usá-lo antes de tri+1). Assumimos que Alice usa apenas um dispositivo por vez, de modo que duas interações não compartilhem o mesmo carimbo de data/hora.

Para efeito de nossos experimentos, consideramos que os dispositivos de Alice possuem acesso a um relógio físico sincronizado, criando uma ordem total na sequência. A mesma ordem total pode ser obtida com relógios lógicos, por exemplo, timestamps Lamport [29], uma vez que as interações nunca são concorrentes.

Nosso procedimento de transferência de sessão é ilustrado na Fig. 1.

Propagando o comportamento do usuário: Do ponto de vista global, a sequência Sk contém todas as k interações realizadas pelo usuário desde o início da execução do programa.
Localmente, porém, cada dispositivo d inicialmente conhece apenas a sequência Sk|d de interações que ocorreram nele, ou seja:

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

onde r ÿ D representa a projeção da interação r no conjunto de dispositivos, ou seja, o dispositivo no qual r ocorreu.

Para obter conhecimento sobre o comportamento do usuário, os dispositivos fofocam informações sobre as interações entre si. Desta forma, no passo k, todo dispositivo d conhece uma sequência local (possivelmente incompleta) Sk,d das ações do usuário, tal que:

$$Sk|d \ddot{y} Sk,d \ddot{y} Sk.$$

Inferindo um modelo probabilístico: Uma sequência ordenada S de interações pode ser usada para calcular uma cadeia de Markov de tempo discreto, representando a probabilidade de que o usuário troque de um dispositivo para outro, para cada par de dispositivos em D. Para fazer isso, primeiro precisamos para calcular a matriz de contagens de transição C = (cdi,dj)(di,dj)ÿD2 entre dispositivos:

$$cdi,dj = |\{rt, rt+1 \ddot{y} S, (rt \ddot{y} D) = di \ddot{y} (rt+1 \ddot{y} D) = dj\}| .$$

C captura o número de vezes que Alice alterna entre cada par de dispositivos (di , dj) em S. De C, podemos derivar a matriz de probabilidades de transição P = (pdi,dj)(di,dj)ÿD2 , (ou seja, o pesos das arestas da cadeia de Markov):

$$pdi,dj = \frac{cdi,dj}{dk\ddot{y}D\ cdi,dk} = P\ [di \ddot{y} dj\ ] ,$$

onde di ÿ dj significa “Alice usa o dispositivo dj logo após di”. pdi,dj representa assim a probabilidade de Alice mudar de di para dj , de acordo com a sequência de interações do usuário, S. Chamamos de ÿ a operação de geração de uma matriz de transição de Markov P a partir de uma sequência S: P = ÿ(S).

Um dispositivo dcurr que está sendo usado atualmente por Alice na etapa k (ou seja, d1 na Fig. 1) usa sua sequência local Sk,dcurr para calcular a matriz de transição Pk,dcurr = ÿ (Sk,dcurr). Isso fornece dcurr com o vetor de transição pk que contém, para cada d ÿ D, a probabilidade de que o usuário mude de dcurr para d:

$$pk = Pk,dcurr\ (dcurr, \ddot{y}) = (P\ [dcurr \ddot{y} d \mid Sk,dcurr\ ])d\ddot{y}D . \quad (1)$$

Observe que pk (dcurr) geralmente não é nulo: o usuário às vezes volta para o mesmo dispositivo.

Executando a transferência de sessão: depois que Alice fecha seu dcurr de dispositivo atual, queremos enviar proativamente o blob contendo o estado de seu aplicativo — sua sessão — para o próximo dispositivo que ela usará: dnext.

Ao longo do restante do artigo, presumimos que cada sessão pesa bytes wsess . Como dcurr não pode ter certeza de qual dispositivo será usado em seguida, ele envia partes de sua sessão para vários pares selecionados. Chamamos essas porções de blocos de sessão e assumimos que podem pesar qualquer tamanho de 0 a wsess. Por fim, definimos o conjunto D de dispositivos para os quais dcurr pode potencialmente enviar blocos de sessão: D = D \ {dcurr}.

dcurr envia wk,d ÿ [0, wsess] bytes da sessão para cada dispositivo d ÿ D , resultando no vetor Wk de todos os blocos de dados enviados por dcurr na etapa k:

$$Wk = (wk,d \ddot{y} [0, wsess])d\ddot{y}D = f\ (pk). \quad (2)$$

O desempenho do Session Handoff depende da precisão de pk, que depende da sequência local de dcurr, Sk,dcurr .

A seguir, apresentamos primeiro o SPRINKLER Gossiper, que propaga de forma confiável a sequência de interações de um usuário para todos os dispositivos, antes de discutir o SPRINKLER Session Handoff, que lida com a migração proativa de uma sessão de usuário.

B. Agregação de conhecimento descentralizada

Inicialmente, um dispositivo só pode observar as interações ocorrendo localmente. Para prever o comportamento futuro de um usuário, todos os dispositivos devem, no entanto, obter uma visão global do comportamento passado do usuário e, assim, agregar seu conhecimento local em uma sequência de interação global. Propomos realizar essa agregação com um protocolo de disseminação probabilística [12], [15], [26] que denominamos SPRINKLER Gossiper.

1) Intuição: O Gossiper implementa uma agregação reativa e incremental que envolve todos os dispositivos de um usuário. O protocolo é invocado toda vez que o usuário (digamos, Alice) sai de um dispositivo para ir para outro, sinalizando o fim de uma interação e o início de uma nova. O SPRINKLER Gossiper é baseado em fofocas, ou seja, utiliza trocas aleatórias de mensagens entre dispositivos para propagar a sequência de interações realizadas por Alice. Essa abordagem aleatória torna nosso protocolo leve e robusto, duas propriedades que são centrais para a transferência de sessão descentralizada. Utilizamos uma estratégia push-pull [23] para propagar informações, ou seja, quando um dispositivo p entra em contato com um dispositivo q, p envia novas informações para q (push), mas também solicita qualquer nova informação que q possa ter (pull).

A fim de evitar rodadas de comunicação redundantes, o Gossiper ainda mantém o controle da percepção local de cada dispositivo do conhecimento de outro dispositivo usando um mecanismo inspirado em relógios vetoriais [31] combinados com diffs incrementais.

2) Algoritmo: O pseudo-código do SPRINKLER Gossiper é mostrado nas Figuras 2 e 3. Para facilitar nossa explicação, a parte de requisição da troca push/pull é mostrada na Fig. 2 do ponto de vista de p, enquanto a resposta parte é mostrada na Fig. 3 do ponto de vista de q. (Todos os dispositivos executam ambas as partes na prática.) Assumimos que p e q são de propriedade de Alice e que ela está usando o dispositivo p no momento. o conhecimento atual de p sobre a sequência de interações de Alice é armazenado na variável Sp, enquanto o array RVp[·] armazena a visão remota de p do conhecimento de outros dispositivos sobre a sequência de Alice (Tabela I).

O algoritmo inicia quando Alice inicia uma nova interação abrindo o dispositivo p (linha 1). O algoritmo insere um novo registro de interação p, timestamp na visão de interação local Sp (linhas 3-4), e inicia a disseminação probabilística, implementada em GOSSIPUPDATE().

GOSSIPUPDATE() primeiro seleciona um pequeno conjunto aleatório de f outros dispositivos da sequência local de p Sp (linhas 6-8). Como consequência, os únicos dispositivos que participam do SPRINKLER são aqueles que Alice já usou pelo menos uma vez.

Esses dispositivos aleatórios são selecionados de Sp, ou seja, a sequência de interações já aprendidas por p, da qual excluímos p e o dispositivo mais recente encontrado em Sp (que provavelmente está atualizado). p inicia uma troca push/pull com cada par selecionado q que não é conhecido por saber pelo menos tanto quanto p (linhas 9-13). Essa verificação de conhecimento é realizada nas linhas 10-11, usando RVp[q], a ideia de p das interações que são conhecidas por q. Por construção, e no

TABELA I
VARIÁVEIS E PARÂMETROS DE ASPERSORES

	Variáveis mantidas pelo dispositivo p **pertencente a Alice**
SP	o conhecimento de p sobre a sequência de interação de Alice, ou seja, sua sequência local. Sp é inicializado com um pequeno número de dispositivos centrais (possivelmente apenas um) que Alice usa regularmente.
RVp[·]	RVp[q] contém a ideia de p do que é conhecido pelo dispositivo q. Inicialmente RVp[q] = ÿ para todo q ÿ D \ {p}.
	Parâmetros do algoritmo
f	O fanout da transmissão probabilística, ou seja, o número de dispositivos com os quais cada dispositivo comunica novas informações.

```
1: no evento Alice abre dispositivo pr ÿ p,timestamp
2:
                                                        Nova interação r
3:      Sp ÿ Sp ÿ {r}                                   Atualizando a visualização local de p
4:      GOSSIPUPDATE()                                  Acionando a divulgação

5: função GOSSIPUPDATE(excluir[= ÿ])
6:      último dispositivo ÿ dispositivo mais recente em
7:      Sp exclui ÿ exclui ÿ {p, último dispositivo} peers
8:      ÿ f dispositivos de {dispositivos de Sp} \ exclude for q ÿ peers do
9:                                                      Percorrendo pares aleatórios
10:         Sdiffpush ÿ Sp \ RVp[q] se |
11:         Sdiffpush| > 0 então                        Apenas enviando novos dados
12:             envie REQ : Sdiffpush para q            Empurrar/puxar para q
13:             RVp[q] ÿ RVp[q] ÿ Sdiffpush             Rastreamento q

14: ao receber ANS : Sdiffpull de q:                    Puxe a resposta
15:     Sp ÿ Sp ÿ Sdiffpull
16:     RVp[q] ÿ RVp[q] ÿ Sdiffpull
```

Fig. 2. Solicitação push/pull do SPRINKLER (no dispositivo p)

```
17: ao receber REQ : Sdiffpush de p:                    Rastreamento de
18:     RVq[p] ÿ RVq[p] ÿ Sdiffpush se                  solicitação
19:     Sdiffpush Sq então                              push/pull p O Sdiffpush é novo?
20:         Sq ÿ Sq ÿ Sdiffpush
21:         GOSSIPUPDATE({p})                           Propagando novos dados
22:     Sdiffpull ÿ Sq \ RVq[p] if |
23:     Sdiffpull| > 0 então envie                      Alguma novidade para p?
24:         ANS : Sdiffpull para p                      Atender puxar
25:         RVq[p] ÿ RVq[p] ÿ Sdiffpull
```

Fig. 3. Resposta push/pull do SPRINKLER (no dispositivo q)

ausência de falhas de comunicação, RVp[q] subestima o conhecimento real de q (ou seja, RVp[q] ÿ Sq) , o que significa[1] que nenhuma nova informação é perdida.

A operação **de envio** na linha 12 inicia a troca push/pull real com q: a atualização incremental Sdiffpush é enviada para q.
Ao receber Sdiffpush (linha 17, Fig. 3), q primeiro processa a atualização incremental de p (linhas 18-21), (i) adicionando-a à ideia de q da visão de p (linha 18), (ii) atualizando sua própria visão se necessário (linha 20) e (iii) lançar uma disseminação em cascata caso o diff contenha informações novas para q. A condição na linha 19

1Isso ocorre porque quaisquer interações adicionadas a RVp[q] por p foram enviado para q (linhas 13 e 25) ou recebido de q (linha 16).

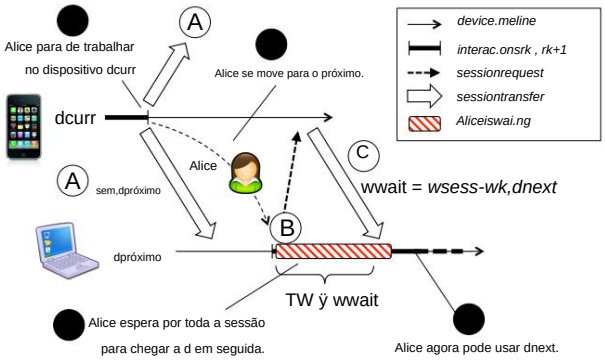

Fig. 4. Linha do tempo da transferência da sessão do dispositivo dcurr para dnext.

garante que a disseminação eventualmente pare, já que os incrementos nunca são fofocados duas vezes pelo mesmo dispositivo. {p} é passado como parâmetro para GOSSIPUPDATE() para aumentar a probabilidade de atingir dispositivos não informados.

As linhas 22-25 implementam a resposta de q ao pull request de p. Novamente, q só responde a p se este possuir novas informações (teste da linha 23), para reduzir a comunicação. A (possível) resposta de q para p é processada por p nas linhas 14-16 (Fig. 2).

3) Bootstrap e confiabilidade do SPRINKLER Gossiper: Como um dispositivo p apenas fofoca com outros dispositivos encontrados em sua exibição de sequência Sp, a exibição de p precisa ser inicializada com um valor padrão de alguns dispositivos principais usados regularmente por Alice, por exemplo {a, 0, b, 0}, onde 0 é um timestamp de bootstrapping arbitrário. A utilização de Sp como fonte de candidatos fofoqueiros evita que dispositivos não utilizados por Alice sejam envolvidos no protocolo. Por outro lado, quando um novo dispositivo é usado por Alice, ele propaga atualizações contendo pelo menos ele mesmo (linhas 2-3) e se torna automaticamente conhecido pelo restante do sistema.

A confiabilidade geral da difusão da fofoca é governada por seu coeficiente fanout f (que aparece na linha 8). Em uma rede confiável, um fanout f ligeiramente acima de log (N) garante que todos os dispositivos serão alcançados com uma probabilidade muito alta [27].
Na prática, portanto, usamos f = log (N).

Embora assumamos uma rede confiável, falhas transitórias de comunicação podem ocorrer. Quando isso acontece, o protocolo pode falhar temporariamente em alcançar todos os nós, mas a propagação completa será retomada quando a rede se recuperar. Enquanto a rede estiver degradada, RVp[q] pode divergir e conter informações não incluídas em Sq. Isso ocorre porque não garantimos que a mensagem REQ : Sdiffpush enviada na linha 12 seja recebida com sucesso por q antes de modificar a visão remota de p RVp[q]. Na maioria das situações, no entanto, outros nós fornecerão q com as informações perdidas quando a rede se recuperar. Essa escolha favorece a leveza da comunicação em detrimento da confiabilidade, mas acaba funcionando bem na prática (conforme mostrado na Seção III-B1).

C. O algoritmo Session Handoff A Fig. 4

mostra as diferentes etapas realizadas pelo SPRINKLER Session Handoff quando Alice se move do dispositivo dcurr (seu celular aqui) para outro dispositivo dnext (seu laptop) entre as interações rk (no dispositivo dcurr) e rk+ 1 (no dispositivo dnext).

Quando Alice sai de seu celular (dcurr, label no final da interação rk, o SPRINKLER envia proativamente um estado de sessão parcial para seus outros dispositivos (label A). (Assumimos aqui que podemos detectar o fim de uma interação, por exemplo, usando algum mecanismo de reconhecimento de atividade.) A quantidade de estado que cada dispositivo recebe é decidida usando o modelo de comportamento do usuário construído até agora pelo SPRINKLER Gossiper . (Detalhamos isso abaixo.) Observamos wk,d a quantidade de estado de sessão recebida proativamente pelo dispositivo d no final da interação rk. Na Fig. 4, o laptop de Alice recebe proativamente wk,dnext da sessão de Alice em seu celular.

Quando Alice alcança seu laptop 2 , A transferência de sessão do SPRINKLER solicita de forma reativa o restante do estado da sessão que ainda não atingiu o laptop B. [2] Enquanto o estado de sessão restante wwait = wsess ÿ wk,dnext é baixado de dcurr, Alice deve esperar (3 barra de h m na linha do tempo do laptop) até que ela possa finalmente usar seu dispositivo 4 . Assumindo que a latência é insignificante em comparação com o tempo de download de wwait, o tempo de espera de Alice TW é proporcional a wwait.

O algoritmo de handoff de sessão perfeito sempre forneceria ao usuário o último estado de seus aplicativos quando ele abrisse um dispositivo, qualquer um de seus dispositivos que ele tivesse usado anteriormente, e sem nenhum tempo de espera. Além disso, dado que pelo menos alguns de seus dispositivos são ativos móveis com recursos limitados, esse algoritmo não deve consumir mais do que o mínimo: ele teria enviado a sessão do aplicativo uma vez, do dispositivo que Alice acabou de sair para o que ela está prestes a usar.

Tal algoritmo não é viável, pois ninguém pode prever o futuro. Em vez disso, o dispositivo atual dcurr infere quais dispositivos são mais prováveis de serem usados em seguida, para enviar proativamente a eles pedaços do blob da sessão quando o usuário sair do dcurr. Se o dcurr enviar mais blocos de sessão para os outros dispositivos, o tempo de espera TW diminuirá ao custo de um tráfego de rede mais alto. Por outro lado, se dcurr não enviar nenhum estado da sessão atual, o custo de espera será máximo, pois dnext terá que buscar reativamente toda a sessão ao abri-la. Chamamos esse custo de rede de CN .

Existe claramente um trade-off entre o tempo de espera TW e o tráfego de rede CN .

1) Formulando o custo de handoff: Graças ao Gossiper, dcurr conhece um subconjunto da sequência de interações do usuário Sk,dcurr . Calculando o vetor de transição pk (ver Equação 1) a partir de Sk,dcurr , o algoritmo Session Handoff gera o vetor de dados Wk (ver Equação 2), que contém a quantidade de bytes da sessão a ser enviada para cada outro dispositivo.

Formulamos o tempo de espera TW e o custo da rede CN descrito anteriormente:

• TW ÿ wwait = wsess ÿ wk,dnext : O tempo de espera TW é proporcional à quantidade da sessão atual que o próximo dispositivo dnext ainda precisa baixar. No caso particular em que dcurr = dnext, a sessão já está totalmente no dispositivo, levando a um custo de espera nulo TW = 0; • CN = wk,d: o custo da rede CN é a soma de todos os dÿD

os pedaços de sessão enviados proativamente de dcurr para os outros dispositivos ·

em D 2) Computando Wk: O objetivo do algoritmo Session Handoff é descobrir o melhor vetor de dados enviados Wk com base no vetor de transição pk. dcurr quer minimizar TW dado que enviará um total de bytes CN da sessão para seus pares.
Introduzimos o parâmetro ÿ, que controla a quantidade de dados que o dispositivo dcurr irá enviar para os outros appliances.

Propomos duas soluções diferentes para o cálculo de Wk: • **Uniforme:** Como linha de base, nossa primeira solução é enviar a mesma quantidade de sessão para cada dispositivo em D , independentemente do pk:

$$\ddot{y}d \ddot{y} D, wk,d = wsess \ddot{y} \frac{\ddot{y}}{N \ddot{y} 1}.$$

• **Proporcional:** Nossa segunda solução é tornar Wk proporcional a pk. Dessa forma, dispositivos com alta probabilidade de serem usados em seguida receberão naturalmente uma parcela maior da sessão do usuário. O cálculo mais simples seria o seguinte:

$$D, wk,d = wsess \ddot{y} \min(\ddot{y} \ddot{y} pk(d), 1). \ddot{y}d \ddot{y}$$

Entretanto, um dispositivo dlow pode ter uma probabilidade muito baixa pk(dlow) de ser escolhido. O cálculo anterior resultaria em um wk,dlow insignificante em comparação com o custo da troca de rede. Portanto, calculamos um novo vetor de transição p k , tal que qualquer probabilidade inferior a um certo limiar ÿ é nula. Na prática, definimos arbitrariamente ÿ = 1/(N ÿ 1). Introduzimos um segundo parâmetro ÿ para garantir que p k some um. É calculado da seguinte forma:

$$\ddot{y} = \frac{1}{\sum_{\substack{d \ddot{y} D \\ pk(d) > \ddot{y}}} pacote(d)};$$

$$\ddot{y}d \ddot{y} D, p\,k\,(d) = \begin{cases} 0 & \text{se } pk(d) \ddot{y} \ddot{y} \ddot{y} \\ \ddot{y}\,pk(d) & \text{senão} \end{cases}.$$

O vetor de dados enviados é então simplesmente proporcional a este novo vetor podado de probabilidades:

$$\ddot{y}d \ddot{y} D, wk,d = wsess \ddot{y} \min(\ddot{y} \ddot{y} p\,k\,(d), 1).$$

Dessa forma, enviamos blocos de sessão apenas para dispositivos que tenham uma probabilidade alta o suficiente de serem usados.

Propusemos duas soluções para despachar blocos de sessão para os dispositivos na transferência de sessão. O primeiro (uniforme) nos dará um ponto de comparação para observar a influência do conhecimento comportamental inferido com o SPRINKLER Gossiper (na abordagem proporcional). O parâmetro ÿ nos permitirá ajustar a quantidade total de dados que dcurr enviará para seus

[2]Para buscar esse estado restante, dnext deve saber o endereço de dcurr. Isso pode ser alcançado por vários meios: se dcurr enviou pedaços da última sessão para dnext (o que não é o caso quando wk,dnext = 0) ou observando a penúltima interação em Sk+1,dnext (embora a sequência local de dnext possa estar incompleto). Além disso, quando dnext erroneamente acredita que um determinado dispositivo d é o dispositivo usado na interação rk e solicita a última sessão, d usa seu próprio conhecimento para redirecionar dnext para o dispositivo certo, dcurr. Se dnext ainda não conseguisse localizar dcurr, a transferência da sessão falharia completamente: a sessão anterior de Alice nunca seria carregada em dnext. Consequentemente, avaliamos um fracasso transferência na Seção III-B3.

peers: de ÿ = 0, que depende apenas de handoff reativo, até um ÿ máximo, que consiste em enviar a totalidade da sessão para cada dispositivo com probabilidade não nula de ser usado próximo.

III. AVALIAÇÃO

A. Abordagem experimental

Avaliamos nossas contribuições lançando vários dispositivos virtuais usados por um usuário emulado. Em nosso cenário futurista, imaginamos um usuário controlando uma dezena de dispositivos sequencialmente. A seguir, primeiro propomos vários modelos comportamentais que emulam a atividade de um usuário fictício, antes de apresentar nossa configuração experimental.

1) Modelos de comportamento do usuário propostos: Na Seção II-A, representamos o comportamento de um usuário como uma sequência crescente S de interações com seus aparelhos. Para emular essas interações, propomos o uso de vários modelos de Markov de tempo discreto M, a partir dos quais geramos as sequências de interações que conduzem à avaliação de nossas contribuições. Dado o conjunto D = {d1, . . . , dN } dos dispositivos de um usuário, um modelo de usuário Markov de tempo discreto assume a forma:

PM = pdi,dj (di,dj)ÿD2 st pdi,dj = P [di ÿ dj] .

Propomos usar 5 estratégias diferentes para gerar esses modelos de usuário, e para 3 delas, criamos duas variantes, levando a um total de 8 modelos de usuário. Esses modelos diferem em termos de densidade (representando quantos próximos dispositivos em potencial podem ser escolhidos após o atual) e uniformidade (representando até que ponto a seleção de um próximo dispositivo é tendenciosa ou não).

A Figura 5 mostra exemplos das matrizes de transição PM geradas pelos 8 modelos, representadas como mapas de calor (usando N = 12 dispositivos). As estratégias de criação são explicadas abaixo: 1) **uniforme:** O pior cenário para nossa estrutura é um modelo completamente uniforme, onde Alice escolhe seu próximo dispositivo com uma probabilidade par de 1/N. Nessa situação, o dispositivo atualmente usado não pode adivinhar qual dispositivo será usado em seguida, tornando a transferência de sessão praticamente aleatória;

2) **cíclico:** O melhor padrão de uso é quando Alice usa seus dispositivos em uma ordem cíclica (fazendo uma cadeia de Markov circular), pois os dispositivos sempre conseguem prever o próximo dispositivo que ela usará. Neste modelo, todo vetor de transição PM(d, ÿ) é constante;

3) **sequência:** Este modelo é calculado a partir de uma sequência de modelo aleatório SM contendo l interações. SM é preenchido por dispositivos selecionados aleatoriamente com uma probabilidade desigual Pdevices ÿ [0, 1]N . Pdevices favorece assim o uso de determinados dispositivos (por exemplo, Alice usa seu smartphone com mais frequência do que o laptop de sua mãe). Calculamos a matriz de transição PM = ÿ(SM) conforme mostrado na Seção II-A. Quanto mais longa for a sequência do modelo SM, mais densa será a matriz de saída. Assim, geramos dois modelos de sequência: **3.1,** com l = 2 ÿ N, e **3.2,** com l = 10 ÿ N; 4) **zipf:**

Muitos processos na vida real seguem uma lei de Zipf [36]: ocorrências de palavras, citações de artigos científicos,

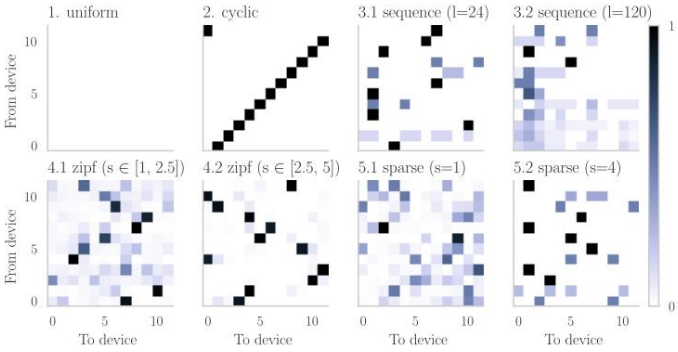

Fig. 5. Heatmaps da matriz de transição PM de cada modelo proposto, com n = 12 dispositivos. Para cada modelo, a linha i do mapa de calor representa o vetor de probabilidades de transição de di: PM(di, ÿ). Modelo 1. uniforme tem uma probabilidade constante de 1/N.

riqueza por habitante... A lei discreta de probabilidade z(k; n, s) de Zipf é definida por um tamanho populacional constante n ÿ N, um posto k ÿ N + e um expoente constante s ÿ [1, +ÿ[. Ela afirma que: z(k; n, s) = z(1; n, s) ÿ k ÿs , ou seja, a probabilidade de ocorrência do k elemento mais frequente z(k; n, s) é igual à probabilidade de ocorrência do elemento mais frequente z(1; n, s) vezes k ÿs Quanto maior o expoente s, mais rápido a função se aproxima zero, e quanto mais z(1; n, s) domina as outras probabilidades.

Em nosso cenário futurístico em que um usuário possuiria e usaria com frequência uma dúzia de dispositivos, a suposição de que a probabilidade de transição entre seus dispositivos segue uma lei Zipf parece plausível.

Propomos um modelo onde, para cada vetor de transição PM(d, ÿ) do dispositivo d, atribuímos uma permutação aleatória do PMF da lei de Zipf usando n = N e um expoente aleatório s. Propomos duas variantes: no modelo **4.1,** escolhemos s de [1, 2,5], que mantém a maior probabilidade z(1; n, s) ÿ [0,32, 0,75]. No modelo **4.2,** desenhamos s de [2.5, 5], tal que z(1; n, s) ÿ [0.75, 0.96]. Observe que PM é sempre denso usando o modelo zipf, mas a heterogeneidade das probabilidades cresce com o expoente s;

5) **sparse:** Uma matriz de transição esparsa contém probabilidades nulas: existem certos dispositivos di e dj tais que dj nunca será usado após di . Isso é realista, por exemplo, dois computadores desktop de dois locais distantes nunca serão acessados em sequência. Para o handoff do SPRINKLER , essa esparsidade evita que o dispositivo usado tenha que escolher entre muitos appliances para os quais enviar blocos de sessão. Para cada d ÿ D, calculamos o vetor de transição PM(d, ÿ) desenhando amostras de uma lei Zipf Z(n, s) com um n “grande” (por exemplo, 1000) e um s fixo:

$$PM(d, \ddot{y}) = \frac{x}{\sum_{x \ddot{y} X} x} \text{ st } \begin{cases} \ddot{y} X = \{Z(n, s) \ddot{y} 1\}^{N} \\ \ddot{y}_{\ddot{y}^{x \ddot{y} X}} x = 0 \end{cases}$$

Quanto maior o expoente s, maior a probabilidade de que Z(n, s) produza um, ou seja, que uma transição de saída seja igual a zero. Propusemos dois modelos **5.1** e **5.2** com

s = 1 e s = 4 respectivamente. Vemos que a dispersão de PM é proporcional ao expoente s.

Ao criar esses modelos, sempre garantimos que o grafo de Markov esteja fortemente conectado, a fim de ver efetivamente o usuário alternar entre os N dispositivos, em vez de percorrer um pequeno subconjunto de D.

Para gerar sequências da atividade do usuário para cada modelo, caminhamos aleatoriamente sobre o grafo de Markov derivado do PM, partindo de um dispositivo aleatório. A sequência Stot resultante é então usada para conduzir a avaliação da transferência da sessão.

2) Testbed Experimental: Para avaliar nosso sistema, implantamos N dispositivos virtuais implementando o algoritmo SPRINKLER al. Realizamos uma avaliação por modelo comportamental apresentado acima. De cada um deles, obtemos uma sequência de interação Stot = {r1, . . . , rL} de tamanho L. É dividido em dois: a primeira subsequência Sinit = {r1, . . . , rLinit} de tamanho Linit < L é alimentado aos dispositivos no bootstrap (cf. Seção II-B3) para que eles conheçam seus respectivos endereços e para dar aos dispositivos um conhecimento inicial do comportamento do usuário. A segunda subsequência Sexp = {rLinit+1, . . . , rL} (de tamanho Lexp = L ÿ Linit) fornece as interações realizadas durante o experimento.

Em nossos experimentos, a interação do usuário é atômica: abrir e fechar um dispositivo é instantâneo e gera uma nova sessão. Embora o algoritmo SPRINKLER Gossiper tenha sido efetivamente implementado pelos dispositivos, o Session Handoff é apenas simulado: com base na sequência local do dispositivo atual (genuíno), determinamos a quantidade de dados da sessão que ele envia para seus pares e, finalmente, calculamos o custo da rede CN e o tempo de espera TW para esta interação (cf. Seção II-C1).

Parâmetros experimentais: Definimos o número de dispositivos como N = 12. Argumentamos que esse número de dispositivos já é três vezes superior aos comportamentos de uso atuais [20]. O comprimento inicial da sequência Linit é definido como 30. Consideramos que um usuário com fome de tecnologia, possuindo e alternando regularmente entre doze dispositivos, alcançaria facilmente 30 interações por dia. Esse comprimento de sequência é grande o suficiente para conter a maioria dos dispositivos, mas pequeno o suficiente para fornecer apenas uma estimativa grosseira do comportamento real do usuário. A segunda subsequência é definida para um comprimento de Lexp = 70.

O protocolo SPRINKLER possui três parâmetros: o fanout f do Gos siper, o parâmetro SessionHandoff ÿ que controla a quantidade total de dados de sessão enviados e ÿ que controla a probabilidade de uso abaixo do qual não enviamos nenhum dado de sessão para um dispositivo. Como já foi dito, fixamos f = log (N), pois esse valor se mostrou suficiente para que uma transmissão probabilística alcance todos os seus participantes com uma probabilidade muito alta [27]. ÿ foi definido arbitrariamente como 1/(N ÿ 1). ÿ é definido como 1 ao comparar os diferentes comportamentos de uso na Figura 6 (assim limitando o custo da rede CN ao tamanho da sessão wsess); varia de 0 a N ÿ 1 na Figura 7.

De acordo com [37], uma sessão de aplicação web pode pesar entre 10kB a um valor ilimitado, dependendo do método de coleta de estado (por exemplo, instantâneos ou log de eventos) e, obviamente, da aplicação. Assim, consideramos que uma sessão pesa entre 10kB e 1MB.

B. Avaliação do SPRINKLER

Primeiro avaliamos o SPRINKLER Gossiper. Em seguida, discutimos o desempenho do handoff de sessão proativo e do fallback reativo.

1) Desempenho do SPRINKLER Gossiper: O objetivo do algoritmo Gossiper (Seção II-B) é propagar com sucesso uma sequência de interação geral do usuário St para todos os dispositivos.
Para avaliar a eficiência do Gossiper, comparamos o tamanho da sequência real St com a versão local da sequência Sd,t mantida por cada dispositivo naquele momento.

Agregamos os traços de todos os nossos experimentos (um por modelo de usuário), levando assim a 6391 sequências locais estudadas. 577 (9,0%) deles estão incompletos. Entre sequências locais incompletas, a diferença mediana da sequência real (|St| ÿ |St,d|) é 1, enquanto a diferença máxima é 7 (um décimo de Lexp).

Concluímos que nosso algoritmo é capaz de prop agar perfeitamente a sequência da atividade do usuário na maior parte do tempo.
Quando não é o caso, o desvio da sequência local é controlado: os dispositivos eventualmente obtêm as informações que faltam de outros pares, e acabam conhecendo perfeitamente o comportamento do usuário novamente.
No geral, acreditamos que as sequências locais geralmente são capazes de gerar um pk bastante imparcial para a transferência de sessão para compartilhar blocos de sessão.

Além disso, em termos de custo de rede, cada dispositivo recebe uma quantidade média de dados relacionados à atividade de 3,5 kB por interação do usuário, levando a um global de 42 kB para N = 12 dispositivos para cada interação. Assim, a quantidade mediana do tráfego global gerado pelo SPRINKLER Gossiper é de 2,9MB (42kBÿLexp) por experimento (usando um único modelo comportamental). Obviamente, aumentar o número de dispositivos diferentes que o usuário possui inerentemente tem um impacto direto no tráfego.

2) Handoff da sessão: Queremos entender o tipo de comportamento do usuário para o qual nosso algoritmo é mais adequado. Para isso, comparamos os tempos de espera obtidos com os diferentes modelos de usuários para um ÿ fixo. Assim, definimos o parâmetro ÿ do Sprinkler como 1, ou seja, no total, o dispositivo atualmente usado pode compartilhar proativamente apenas o tamanho da sessão wsess, distribuído entre os possíveis próximos dispositivos.

Lembre-se que uma solução de handoff de sessão centralizada funciona apenas de forma reativa: o tempo de espera do usuário normalizado TW é sempre um nesta situação. No entanto, ele sempre pontua um CN de custo de rede proativo igual a zero.

A Figura 6 mostra os boxplots dos tempos de espera normalizados (st TW = wwait/wsess) para cada modelo de usuário (ver Figura 5). As bordas da caixa mostram o primeiro e o último quartil, enquanto os bigodes representam 3/2 do intervalo interquartil. Os modelos são classificados por quartil inferior e mediana crescentes. Um tempo de espera igual a zero significa que toda a sessão foi enviada proativamente; TW = 1 significa que nada disso foi enviado e que toda a sessão teve que ser baixada de forma reativa quando o usuário abriu seu dispositivo. A linha pontilhada representa o tempo médio de espera da linha de base (Seção II-C2). É constante, porque a linha de base não leva em conta as probabilidades de transição.

Como esperado, o algoritmo Session Handoff funciona melhor com o modelo 2. cíclico, onde o próximo dispositivo usado dnext é

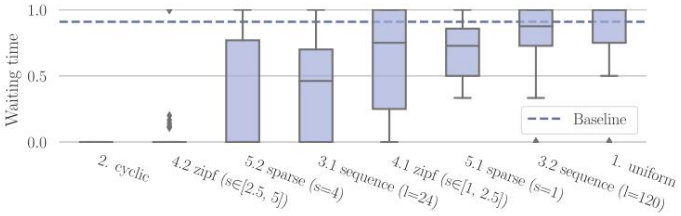

Fig. 6. Box plots do tempo de espera do usuário (normalizado) após o handoff da sessão, agrupados por modelo comportamental. Aqui, ÿ = 1: cada dispositivo não compartilha mais do que o tamanho da sessão com todos os outros dispositivos. Menor é melhor: um tempo de espera nulo significa que a sessão foi enviada proativamente em sua totalidade; um tempo de espera de um significa que o dispositivo atual precisa baixar toda a sessão de forma reativa. Vemos que o desempenho do handoff depende muito do modelo do usuário.

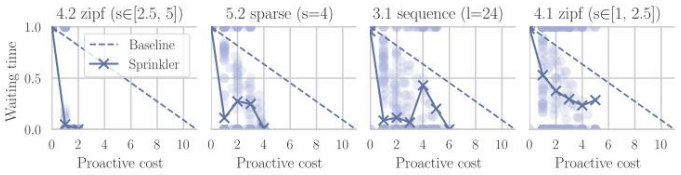

Fig. 7. Tempo de espera do usuário (normalizado) em função do custo de rede do handoff (normalizado por wsess), para diversos modelos comportamentais, usando ÿ ÿ [[0, N ÿ 1]]. Menor é melhor: o handoff é mais eficiente quando um pequeno aumento de consumo gera grandes ganhos de tempo de espera.

sempre conhecido. Como resultado, o modelo cíclico obtém um tempo de espera constante TW = 0, o que significa que a sessão é sempre totalmente enviada proativamente para o dispositivo correto. O algoritmo tem pior desempenho com o modelo 1. uniforme, onde dnext é imprevisível. Leva a um tempo de espera TW > 0,75 em 75% dos casos. O segundo melhor modelo para Handoff de Sessão é o 4.2 zipf com s ÿ [2.5, 5], onde uma probabilidade de transição supera as outras em cada linha (como pode ser observado na Figura 5). Este modelo é muito semelhante ao cíclico, com algum ruído adicionado: a transferência proativa é quase perfeita.

Os próximos dois modelos, esparso 5.2 (s = 4) e sequência 3.1 (l = 24), mostram resultados bastante semelhantes. Os bigodes superiores de ambos os modelos atingem um: o tempo de espera é altamente variável. A Figura 5 mostra matrizes de transição quase idênticas para esses dois modelos: principalmente determinísticas, exceto por algumas transições equiprováveis. Argumentamos que a diferença do tempo médio de espera (0,46 para o modelo 3.1 contra 0 para o 5.2) é causada pela probabilidade uniforme de alternar entre 7 dispositivos ao usar o dispositivo #1 no modelo 3.1. Para esses dois modelos, observamos que uma pequena perda na previsibilidade do comportamento do usuário causa resultados mais instáveis, mesmo que os escores de handoff da sessão ainda sejam bons na maioria das vezes.

Os últimos três modelos, 4.1 zipf (s ÿ [1, 2.5]), 5.1 esparso (s = 1) e 3.2 sequência (l = 120) apresentam um tempo de espera médio acima de 0,5 (resp. 0,75, 0,73 e 0,88). Os três apresentam matrizes de transição muito densas (além de alguns vetores constantes em 3.2): nós os consideramos como diferentes tipos de comportamento imprevisível. De fato, no modelo zipf 4.1, uma probabilidade de transição continua a dominar as outras, enquanto as outras duas apresentam muitas probabilidades uniformes de transição.
Isso faz com que o dcurr sempre envie pequenos pedaços da sessão para vários dispositivos, o que prejudica os resultados gerais.

Este experimento mostra que o desempenho da transferência de sessão pró-ativa depende muito do comportamento do usuário. Dado nosso modelo de inferência simples, obtemos melhores resultados quando o usuário tem hábitos previsíveis, apesar de alguma variabilidade (por exemplo, 4.2 zipf, 5.2 sparse, 3.1 sequence).

Realizamos um segundo estudo (Figura 7), desta vez variando o parâmetro ÿ para um modelo de usuário fixo, permitindo observar a influência do custo normalizado da rede proativa CN no tempo de espera TW . A figura mostra o tempo de espera normalizado TW = wwait/wsess em função do custo da rede CN , para quatro modelos de usuários diferentes, e para cada valor de ÿ ÿ[0, N ÿ 1]].

A linha de base, representada como uma linha pontilhada, mostra um tempo de espera linearmente decrescente à medida que o custo da rede aumenta. Cada ponto representa o resultado de uma única transferência. A linha cheia representa a média do tempo de espera em uma janela deslizante do custo da rede. Observe que, em nosso algoritmo Session Handoff, o custo proativo da rede CN é frequentemente menor que ÿ: de fato, o dispositivo atual só enviará sua sessão para dispositivos que tenham uma probabilidade não nula de serem usados a seguir de acordo com p k (cf. Seção II-C2). Neste gráfico, menor é melhor: o modelo 4.2 zipf (s ÿ [2.5, 5]) mostra uma transferência quase perfeita. Em seguida, exibimos os resultados para 5.2 sparse (s = 4), 3.1 sequence (l = 24) e 4.1 zipf (s ÿ [1, 2.5]), na mesma ordem em que aparecem na Figura 6.

Traços de handoff de sessão muito bons se parecerão com 4.2 zipf : um pequeno aumento no custo de rede permitido leva a uma diminuição drástica no tempo de espera. Além disso, o zipf 4.2 é muito curto: os dispositivos nunca enviam mais do que o dobro do tamanho da sessão. Por outro lado, 4.1 zipf apresenta resultados ruins: em CN = 1, o tempo médio de espera é superior a 0,5. A função decresce monotonicamente até CN = 5: apenas handoffs muito imprevisíveis causam tanta troca de dados, pontuando assim mais do que para CN = 4. Finalmente, 5.2 e 3.1 mostram uma combinação do primeiro e do último gráficos: eles atingem um tempo de espera muito baixo em CN = 1 (ou seja, handoffs determinísticos) e um TW mais longo à medida que o custo da rede aumenta (ou seja, handoffs imprevisíveis).

Também observamos muitos pontos em TW = 1: essa situação só ocorre quando um dispositivo está sendo usado pela primeira vez. Como não está na sequência de interações, o dispositivo anterior não poderia ter enviado nenhum chunk de sessão para ele, resultando na falha do handoff proativo. Felizmente, assim que esse dispositivo entrar na fofoca, ele poderá receber a sessão de forma proativa.

Olhando para este experimento, observamos dois possíveis casos de uso para o SPRINKLER: pode ser preferível manter um custo de rede limitado, levando a tempos de espera variáveis (como na Figura 6); ou pode ser preferível ter handoffs proativos perfeitos, mantendo os custos de rede em um nível razoável. De fato, combinando todos os nossos modelos, exceto o uniforme, vemos que o custo da rede proativa CN nunca excede 7 vezes o tamanho da sessão, o que está longe de ser uma inundação.

No geral, achamos nossa abordagem preditiva para transferência de sessão distribuída promissora: por um custo limitado, ela pode enviar a maior parte da sessão de um usuário para seu próximo dispositivo na maioria dos casos. O futuro trabalha no modelo comportamental (por exemplo, usando timestamps ou locais) e na decisão de transferência de sessão

O algoritmo tem mais potencial para reduzir drasticamente o tempo de espera do usuário.

3) Transferência reativa: Como nossa transferência de sessão proativa pode ser imperfeita, devemos propor um fallback reativo funcional. Em uma situação em que a sessão anterior foi baixada apenas parcialmente, dcurr precisa localizar dprev para poder recuperar o restante da última sessão dele. Assumimos que dprev está sempre conectado quando dcurr solicita a sessão.

Existem quatro maneiras pelas quais dcurr pode encontrar o endereço de dprev : 1) Se dcurr tiver uma sequência local atualizada Sk,dcurr contendo ing dprev em sua última interação (ou seja, dprev está em Sk,dcurr); 2) Se dprev enviou pedaços de sua sessão para dcurr (isto é, quando wkÿ1,dcurr = 0); 3) Se os dispositivos anteriores e atuais forem os mesmos (dcurr = dprev);

4) Se dcurr acreditava erroneamente que o dispositivo anterior era d, mas d (que conhecia o dprev correto) redirecionou dcurr para o dispositivo correto.

É somente quando nenhuma dessas soluções funciona que deixamos de fornecer a Alice sua última sessão.

TABELA II
COMO A dCURR SABE O ENDEREÇO DA dPREV ?

dprev in Sk,dcurr	wkÿ1,dcurr = 0	dcurr = dprev	Redirecionado	Fracassado
523 (94,7%)	407 (73,7%)	15 (2,71%)	16 (2,90%)	2 (0,36%)

A Tabela II mostra como dcurr identifica dprev em todas as ocorrências de handoff de nosso experimento (usando os 8 modelos e ÿ = 1). No total, existem 8 ÿ 69 = 552 pontos de dados. Observe que a primeira e a segunda categorias não são exclusivas. Vemos que, embora 5% dos dispositivos usados tenham uma sequência errada durante o handoff, o dispositivo atual quase sempre encontra o endereço de seu ancestral. Como resultado, o handoff reativo é bem-sucedido em 99,64% dos casos.

Ainda consideramos uma falha completa do handoff inaceitável. Para resolver esse problema, devemos evitar depender apenas do dprev para recuperar a sessão anterior: na verdade, outros dispositivos receberam partes dela. Pode-se, por exemplo, aceitar pedaços de sessão de qualquer dispositivo online, de forma semelhante ao protocolo de troca de arquivos BitTorrent [14].

4. TRABALHO RELATADO

O problema de fornecer handoff de sessão entre dispositivos sem interrupções para aplicativos da Web tem recebido atenção considerável tanto da comunidade de pesquisa quanto da indústria e, atualmente, representa um tópico importante de pesquisa. Os trabalhos de ponta podem ser divididos em diferentes categorias de acordo com as diferentes facetas da transferência de sessão que visam [19], [34]: (i) desencadear a transferência de uma sessão manualmente ou automaticamente; (ii) despejar e restaurar uma sessão; e (iii) encaminhar uma sessão de forma centralizada ou distribuída.

A primeira categoria de trabalhos é focada principalmente em mecanismos para salvar/restaurar o estado de um aplicativo de maneira otimizada [11], [16], [24], [28], [30], [37], [38]. O salvamento e a restauração do estado são obtidos pela injeção de código no caminho de dados do aplicativo, por meio de transformação de código ou por meio de proxies que interceptam solicitações HTTP para instrumentar o código na página da Web do aplicativo. Outras facetas da transferência de sessão são tratadas superficialmente. O estado da sessão é armazenado em um servidor baseado em nuvem, de onde é transferido para o dispositivo de destino por transferência direta ou fornecendo uma URL que o usuário pode carregar no dispositivo de destino. Além disso, a transferência de sessão é acionada explicitamente pelo usuário ou iniciada automaticamente de maneira reativa com base nas preferências e no contexto do usuário.

Uma segunda categoria de trabalhos aborda especificamente a migração de sessão entre um conjunto de servidores de uma perspectiva mais ampla (ou seja, sem considerar aplicações web) [10], [35]. Esses trabalhos contam com o protocolo de sinalização SIP para gerenciar a migração. Embora permita o encaminhamento de uma sessão entre diferentes servidores, requer uma arquitetura SIP tradicional baseada em servidores SIP (ou seja, registrador, proxy e servidor de redirecionamento) para gerenciamento de sessão, tornando-se uma solução centralizada. A transferência de sessão é acionada pelo usuário ou acionada automaticamente quando o endereço IP do usuário é alterado.

Uma terceira categoria de trabalhos aborda a transferência de sessão de maneira ad hoc, contando com uma arquitetura peer-to-peer para salvar e transferir sessões de aplicativos entre dispositivos pertencentes aos usuários [17], [18], [32]. No entanto, esses trabalhos não levam em consideração o comportamento do usuário e, como o dispositivo de destino não é conhecido com antecedência, eles dependem de inundação cega para propagar o estado da sessão.

Também vale a pena mencionar uma série de iniciativas do setor para fornecer uma experiência perfeita ao usuário final com aplicativos que funcionam em vários dispositivos, como Apple Handoff [5] ou aplicativos do Google Drive. No entanto, essas abordagens são proprietárias, centralizadas e isoladas umas das outras.

Comparado a todos os trabalhos mencionados, o SPRINKLER aproveita a primeira categoria, focada no dump de sessão para aplicativos da Web. No entanto, o SPRINKLER é baseado em uma abordagem totalmente descentralizada e não depende do usuário (ou preferências do usuário) para acionar o handoff da sessão: ele é executado de forma automática e proativa de acordo com um algoritmo de previsão distribuído. A força do SPRINKLER é sua capacidade de ser descentralizado e prever o comportamento do usuário para otimizar a transferência de sessão.

V. CONCLUSÃO E TRABALHOS FUTUROS

Apresentamos o SPRINKLER, um novo protocolo de disseminação probabilística que explora o aprendizado descentralizado para aumentar a fluidez da interação ponto a ponto do usuário em vários dispositivos, reduzindo custos de rede desnecessários. De forma mais geral, a SPRINKLER destaca o interesse potencial dos métodos de aprendizado descentralizados para possibilitar interações difusas privadas. No futuro, gostaríamos de investigar o uso de informações adicionais, como geolocalização e duração da atividade, para melhorar ainda mais a abordagem.

AGRADECIMENTOS

Os autores gostariam de agradecer a FAPEG e CAPES em Brasil, e CNRS na França por financiar parcialmente este trabalho.

REFERÊNCIAS

[1] 1 Senha. 1password. com.

[2] Adobe PhoneGap. https://phonegap.com.

[3] Amazon Kindle. www.amazon.com/kindle.

[4] Apache Cordova. https://cordova.apache.org.

[5] Transferência da Apple. https://developer.apple.com/handoff/.

[6] Evernote. evernote. com.

[7] Google Chrome. www.google.fr/chrome.

[8] Arcabouço iônico. https://ionicframework.com.

[9] Lista de maravilhas. www.wunderlist.com/.

[10] M. Adeyeye e P. Bellavista. Áreas de pesquisa emergentes em serviços convergentes baseados em SIP para clientes da Web estendidos. World Wide Web, 17(6):1295–1319, novembro de 2014.

[11] F. Bellucci, G. Ghiani, F. Paterno e C. Santoro. Engenharia de persistência de estado javascript de aplicativos da web que migram em vários dispositivos. No 3º ACM SIGCHI Sym. em Eng. Comp interativo. Sys., EICS, 2011.

[12] KP Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu e Y. Minsky. Multitransmissão bimodal. ACM ToCS, 1998.

[13] P.-YP Chi e Y. Li. Weave: Scripting de interação entre dispositivos vestíveis. Em ACM CHI, 2015.

[14] B. Cohen. A especificação do protocolo bittorrent. Relatório técnico, BitTorrent Inc., 2008.

[15] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart e D. Terry. Algoritmos epidêmicos para manutenção de banco de dados replicados. AC, 1987.

[16] J. Feng e A. Harwood. Browsercloud: Uma nuvem pessoal para migração e gerenciamento de sessão do navegador. Em WWW '15 Companion.

[17] ER Fisher, SK Badam e N. Elmqvist. Projetando interfaces de usuário distribuídas ponto a ponto: estudos de caso sobre a construção de aplicativos distribuídos. Int. J. Hum.-Comput. Stud., 72(1):100–110, janeiro de 2014.

[18] A. Gallidabino e C. Pautasso. Implantação de componentes da Web com estado em vários dispositivos com liquid.js para polímero. Em ACM CBSE'2016, páginas 85–90.

[19] A. Gallidabino e C. Pautasso. Modelo de maturidade para arquiteturas web líquidas. Em ICWE'17, páginas 206–224. Springer.

[20] Google. O novo mundo multitela: compreendendo o comportamento do consumidor multiplataforma. Relatório técnico, 2012.

[21] B. Hartmann, M. Beaudouin-Lafon e WE Mackay. Hydrascope: Criando meta-aplicações multi-superfície por meio de sincronização de visualização e multiplexação de entrada. In PerDis'13, páginas 43–48, 2013.

[22] M. Husmann, NM Rossi e MC Norrie. Análise de uso de aplicativos da Web entre dispositivos. Em PerDis '16, páginas 212–219, 2016.

[23] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec e M. Van Steen. Amostragem por pares baseada em fofocas. ACM ToCS, 25(3):8, 2007.

[24] D. Johansson e K. Andersson. Mobilidade de aplicativos adaptativos baseados na Web. Em 2012 IEEE 1º Int. conf. on Cloud Networking (CLOUDNET), páginas 87–94, novembro de 2012.

[25] F. Kawsar e AB Brush. Computação doméstica desconectada: por que, onde e quando as pessoas usam diferentes dispositivos conectados em casa. Em UBICOMP'13, páginas 627–636.

[26] A.-M. Kermarrec, L. Massoulie e AJ Ganesh. Comunicação Probabilística Confiável em Sistemas de Disseminação de Informação em Grande Escala. Relatório técnico, 2000.

[27] A.-M. Kermarrec, L. Massoulie e AJ Ganesh. Disseminação confiável probabilística em sistemas de grande escala. IEEE Transactions on Parallel and Distributed systems, 14(3):248–258, 2003.

[28] J.-w. Kwon e S.-M. Lua. Migração de aplicativo da Web com fechamento reconstrução. Em WWW'2017.

[29] L. Lamport. Tempo, relógios e a ordenação de eventos em um sistema distribuído. Communications of the ACM, 21(7):558–565, 1978.

[30] JTK Lo, E. Wohlstadter e A. Mesbah. Imagen: Migração de tempo de execução de sessões do navegador para aplicativos da web javascript. Em WWW'2013.

[31] F. Mattern. Tempo virtual e estados globais de sistemas distribuídos. Em MC et. al., editor, Parallel and Distributed Algorithms: anais do International Workshop on Parallel & Distributed Algorithms, páginas 215–226. Elsevier Science Publishers BV, 1989.

[32] J. Melchior, D. Grolaux, J. Vanderdonckt e P. Van Roy. Um kit de ferramentas para interfaces de usuário distribuídas ponto a ponto: Conceitos, implementação e aplicativos. Em The 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS '09, páginas 69–78, 2009.

[33] L. Mical. Projetando experiências para vários dispositivos. O'REILLY, 2014.

[34] T. Mikkonen, K. Systa e C. Pautasso. Rumo a aplicações web líquidas. Em ICWE'15, páginas 134–143. Primavera, 2015.

[35] W. Munkongpitakkun, S. Kamolphiwong e S. Sae-Wong. Mobilidade de sessão da web aprimorada com base em sip. No 4º Int. conf. on Mobile Technology, Applications, and Systems and the 1st International Symposium on Computer Human Interaction in Mobile Technology, Mobility '07, pages 346–350, 2007.

[36] MEJ Newman. Leis de potência, distribuições de Pareto e lei de Zipf. Cities, 30:59–67, fevereiro de 2013.

[37] J. Oh, J.-w. Kwon, H. Park e S.-M. Lua. Migração de aplicações web com execução perfeita. Em VEE '15, páginas 173–185, 2015.

[38] J. Oh e S.-M. Lua. Aceleração do tempo de carregamento baseada em instantâneo para aplicativos da web. Em The 13th Annual IEEE/ACM International Symposium on Code Generation and Optimization, CGO '15, páginas 179–189, 2015.

[39] M. Weiser e JS Brown. Além do cálculo. capítulo The Coming Age of Calm Technology, páginas 75–85. 1997.