Freechains: Disseminação de Conteúdo Peer-to-Peer

Francisco Sant'Anna, Fabio Bosisio, Lucas Pires

[1] UERJ - Departamento de Ciência da Computação

francisco@ime.uerj.br, fbosisio@gmail.com, lucasampires@gmail.com

Abstract. *O Freechains é um sistema peer-to-peer para disseminação de conteúdo: um usuário posta uma mensagem em um tópico e seus assinantes eventualmente recebem a mensagem. As postagens são estruturadas em um grafo direcionado acíclico criptográfico que é imune a modificações (*Merkle DAG*). O grafo é disseminado par a par na rede por* gossip *de maneira não estruturada. O Freechains suporta múltiplos arranjos de disseminação pública e privada entre grupos e indivíduos, sendo possível modelar desde conversas privadas por e-mail, até debates entre desconhecidos em fóruns públicos. Cada tópico público conta com um sistema descentralizado de reputação para combater abusos, tais como SPAM e notícias falsas. O Freechains executa como um* daemon *nos pares da rede e permite interações pela linha de comando, API em Kotlin ou sockets.*

1. Introdução

Apesar do crescimento contínuo da internet ao longo dos anos, o seu conteúdo disponível está cada vez mais sob o controle de poucas empresas [Zittrain 2018]. Consideramos como conteúdo qualquer tipo de informação ou interação na internet, tais como trocas de e-mails, consumo de notícias, interações em redes sociais, ou até mesmo backup de documentos. As empresas de internet tipicamente são somente intermediárias entre seus usuários, mas não criam conteúdo original. Por um lado, elas são essenciais, oferecendo interfaces amigáveis, armazenamento grátis e conectividade permanente. Por outro lado, essas empresas concentram mais poder do que o necessário para operarem, uma vez que controlam nossos dados, coletam informações privadas, "algoritimizam" o consumo, e ainda dificultam a portabilidade entre serviços através de formatos e protocolos proprietários.

Sistemas de disseminação *peer-to-peer* [Theotokis and Spinellis 2004] oferecem uma alternativa aos serviços centralizados, eliminando intermediários e movendo para os usuários finais toda a responsabilidade pela transferência, armazenamento, disponibilidade e validação do conteúdo disseminado. No entanto, há diversos novos desafios quando o sistema fica espalhado pelas bordas da rede, tais como distribuir adequadamente a infraestrutura de conectividade e armazenamento, lidar com pares (*peers*) maliciosos na rede, e investir na usabilidade do sistema sem um modelo de negócio.

Neste trabalho, apresentamos o *Freechains* como uma ferramenta para a disseminação de conteúdo peer-to-peer sem intermediação ou controle de acesso centralizado, e sem a necessidade de confiança entre seus usuários. No Freechains, um usuário posta uma mensagem em um tópico, também chamado de cadeia, e todos os outros assinantes daquela cadeia na mesma rede peer-to-peer eventualmente recebem a mensagem. O Freechains apresenta duas contribuições principais: (1) um protocolo mínimo para

múltiplos arranjos de disseminação de conteúdo e (2) um sistema de reputação autônomo e descentralizado. Sobre a primeira contribuição, o Freechains possibilita diversos arranjos de disseminação de conteúdo entre seus usuários:

- Arranjos públicos $1{\rightarrow}N$ e $1{\leftarrow}N$: Uma identidade pública (ex., pessoa ou organização) dissemina conteúdo autenticado para um público alvo ($1{\rightarrow}N$) com feedback opcional ($1{\leftarrow}N$). Exemplos são sites de notícias, serviços de streaming e perfis públicos em redes sociais.
- Arranjos privados $1{\leftrightarrow}1$, $N{\leftrightarrow}N$ e $1{\leftarrow\!\!\supset}$: Grupos de confiança (ex., amigos ou familiares) trocam mensagens privadas entre si. A comunicação pode ser em pares ($1{\leftrightarrow}1$), grupos ($N{\leftrightarrow}N$) ou até mesmo individuais ($1{\leftarrow\!\!\supset}$). Exemplos são e-mails, grupos de família e backup de documentos.
- Arranjos públicos $N{\leftrightarrow}N$: Grupos heterogêneos se comunicam publicamente. Exemplos são fóruns de perguntas e respostas, chats, e compra & venda online.

Sobre a segunda contribuição, o sistema de reputação do Freechains rege a qualidade das postagens e autores dentro de cadeias públicas e tem os seguintes objetivos:

- Combater o excesso de conteúdo restringindo o número de postagens por autor.
- Destacar o conteúdo de qualidade com um mecanismo de *likes & dislikes*.
- Combater SPAM, notícias falsas e conteúdo ilícito demandando reputação prévia de autores e removendo postagens com proporção muito baixa entre likes e dislikes.

O código fonte, documentação e vídeos introdutórios sobre o Freechains estão disponíveis em `https://github.com/Freechains/README`.

2. Comparação com Outros Sistemas

Os middlewares publish-subscribe desacoplam produtores (*publishers*) de consumidores (*subscribers*) através de um intermediário (*broker*). Exemplos de sistemas pubsub incluem o *XMPP*, *AMQP*, *WebSub* e *ActivityPub*. Uma limitação chave em *pubsubs* é que, embora produtores e consumidores se comuniquem sem conhecimento mútuo, os brokers ainda possuem um papel centralizador na rede. Eles são necessários para autenticar e validar as postagens, por exemplo, além de servirem as filas de mensagens que conectam produtores e consumidores.

Em protocolos federados, múltiplos servidores se sincronizam para permitir a comunicação externa entre seus usuários, mas a identidade de um usuário ainda é atrelada a um servidor específico. O serviço de e-mail (SMTP) é provavelmente o protocolo federado mais popular e permite que usuários de domínios diferentes troquem mensagens de maneira transparente. Mais recentemente, os sistemas *Diaspora*, *Matrix* e *Mastodon* cobriram aos cenários de redes sociais, chats e *microblogging* com arquiteturas federadas. Em sistemas federados, todo conteúdo poder ser gerenciado localmente em suas instâncias e a sincronização externa é um passo em separado, mas que não é necessário para o funcionamento local. No entanto, sem o controle de sua própria identidade, um usuário fica refém do seu servidor. Por exemplo, o servidor pode ser desligado pelos seus administradores ou banido pelo resto da federação, ou ainda o próprio usuário pode ficar insatisfeito com o serviço prestado. Em qualquer um desses casos, o usuário terá que exportar todo o seu conteúdo e histórico para outro servidor e anunciar a sua nova identidade aos seus seguidores.

Em sistemas peer-to-peer, todos os pares da rede desempenham as mesmas funções e atuam ora como clientes, ora como servidores. O *Bitcoin* [Nakamoto 2019] é provavelmente a rede peer-to-peer de maior sucesso, mas serve a um propósito muito específico. O *Scuttlebutt* [Tarr et al. 2019] e *Aether*[1] atendem aos cenários de comunicação entre amigos e grupos, focando respectivamente, na comunicação pública $1 \rightarrow N$ e $N \leftrightarrow N$. O Scuttlebutt é baseado em identidades de usuários que se seguem par a par e formam um grafo que se reflete tanto nas conexões de rede quanto no armazenamento redundante dos seus dados. Já o Aether se organiza em comunidades em torno de tópicos, com postagens replicadas na rede, mas que são efêmeras considerando uma janela de tempo. O Aether usa prova de trabalho nas postagens para evitar SPAM e elege moderadores para prevenir comportamento abusivo.[2] O Freechains também se baseia em tópicos, mas como um tópico pode ser de propriedade de uma identidade, então é possível modelar o comportamento do Scuttlebutt. Freechains introduz um sistema de reputação distribuído para lidar com abuso conforme discutido na Seção 3.3.

3. Visão Geral do Freechains

3.1. Funcionamento Básico

O Freechains é um sistema publish-subscribe baseado em tópicos: um usuário posta uma mensagem em um tópico e seus assinantes eventualmente recebem a mensagem. A Figura 1 ilustra os quatro conceitos básicos do Freechains: *cadeias*, *blocos*, *autores* e *pares*.

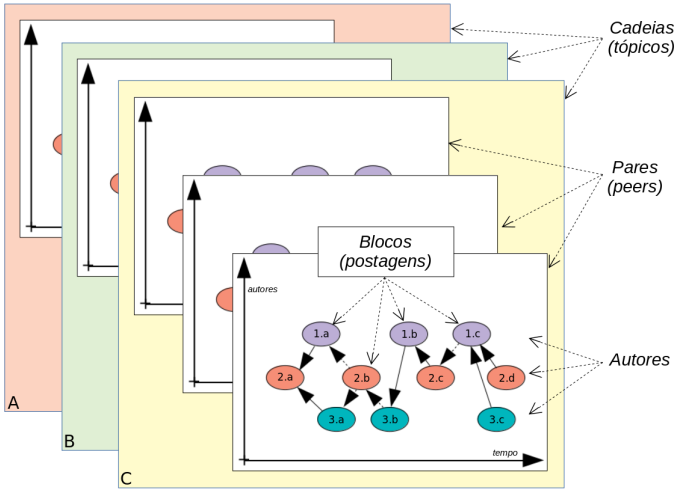

Figura 1. Conceitos básicos do Freechains: cadeias, blocos, autores e pares.

As *cadeias*, destacadas em quadrados coloridos (vermelho/A, verde/B e amarelo/C), são independentes umas das outras e podem representar, por exemplo, os *tweets* de uma personalidade (arranjo $1 \rightarrow N$ público), um grupo de *Whatsapp* de amigos (arranjo $N \leftrightarrow N$ privado) ou um *subreddit* em torno de um interesse em comum (arranjo $N \leftrightarrow N$ público).

Cada cadeia possui um conjunto de *blocos* com suas postagens, que estão destacadas como elipses coloridas (roxo/1.x, vermelho/2.x e verde/3.x), em que cada cor

[1] `https://getaether.net/`

[2] `https://blog.getaether.net/post/187829323237/aether-p2p-dev14-released-with-modera`

representa um *autor*. Os blocos da cadeia formam um grafo de causalidade indicando a relação temporal entre elas. Por exemplo, a primeira mensagem do autor vermelho (`2.a`) ocorreu antes das primeiras mensagens dos autores roxo e verde (`1.a` e `3.a`). O grafo é um DAG e expressa a ordem parcial entre todos os blocos na cadeia. Toda cadeia tem um bloco *gênesis* preexistente (não exibido na imagem) que, por construção, é alcançável por todos os blocos da cadeia.

O grafo da cadeia é inteiramente replicado em todos os pares. Na figura, a cadeia amarela/C é replicada em três pares. A disseminação do grafo pela rede é feita por *gossip*, ou seja, os pares se conectam dois a dois para sincronizar os seus grafos. Dessa maneira, o recebimento de mensagens no pares é eventual, pois depende de um roteamento par a par da origem ao destino.

O Freechains executa como um servidor ou *daemon* e escuta requisições de usuários locais, por exemplo para postar ou ler mensagens, e também se comunica com outros pares da rede peer-to-peer para sincronizar as suas cadeias de interesse. O daemon pode ser acessado de três formas equivalentes:

- Pela linha de comando, que usamos no resto do artigo.
- Por uma API em Kotlin, linguagem da implementação de referência.
- Por um protocolo textual, que permite usar outras linguagens via sockets.

A sequência de comandos a seguir ilustra o funcionamento básico do Freechains:

```
$ freechains start /var/fcs/ &      # inicia daemon em background
$ freechains crypto pubpvt "senha"  # cria identidade publica
EB172E... 96700A...                 # (retorna chaves publica e privada)
$ freechains chains join "#chat"    # se inscreve na cadeia #chat
$ freechains chain "#chat" post inline "Bom Dia!" --sign=96700A...
```

Após iniciar o daemon, o usuário cria uma identidade para si. A chave pública pode ser compartilhada e a chave privada deve ser guardada em segredo. Em seguida o usuário se inscreve na cadeia `#chat` e posta uma mensagem assinando-a com a sua chave privada. Até aqui, tudo ocorre localmente na máquina com o daemon em execução. A comunicação com outros pares deve ser feita explicitamente:

```
$ freechains peer 10.0.0.2 send "#chat"  # envia  #chat para 10.0.0.2
$ freechains peer 10.0.0.2 recv "#chat"  # recebe #chat  de  10.0.0.2
```

Para ler as postagens de uma cadeia, é necessário primeiro identificar os blocos não lidos:

```
$ freechains chain "#chat" genesis                # hash id do genesis
0_10EEB7...
$ freechains chain "#chat" traverse all 0_10EE... # do genesis em diante
1_A5EF... 2_1B5C... 2_2144...
$ freechains chain "#chat" get payload 1_A5EF...  # conteudo do bloco
Bom dia!
```

O comando `traverse` retorna todos os blocos mais novos que o(s) bloco(s) passados como parâmetro e serve para identificar as postagens que ainda não foram lidas. Inicialmente, usamos a identificação do bloco gênesis como âncora, já que nenhum bloco ainda foi lido pelo usuário. Em seguida, é possível usar o comando `heads` para identificar os blocos mais novos na cabeça do grafo e usá-los como âncora na próxima varredura. Uma outra forma é checar o conteúdo em tempo real através do comando `listen`:

```
$ freechains chain "#chat" listen   # escuta novos blocos em tempo real
1_A5EF...                           # 1o bloco recebido
2_1B5C...                           # 2o bloco recebido
2_2144...                           # 3o bloco recebido
```

O comando é bloqueante e exibe uma nova linha sempre que um novo bloco chega com a sua identificação. Esse comando pode ser usado em conjunto com um *pipe* para reagir a novas mensagens em tempo real.

Os comandos `send` e `recv` devem ser executados para cada cadeia e para cada par da rede de interesse. Por exemplo, se o usuário segue 10 cadeias e se liga a 5 pares, os comandos deverão ser executados 100 vezes a cada sincronização ($2 \times 10 \times 5$). O Freechains inclui uma ferramenta para automatizar o processo inteiro de sincronização. É possível registrar pares e cadeias e a ferramenta usa internamente os comandos `listen`, `send` e `recv` para se sincronizar assim que uma nova mensagem é postada ou recebida.

Mais tecnicamente, um bloco é uma estrutura de dados que persiste uma única mensagem na cadeia. Ele se encadeia às postagens anteriores e é encadeado pelas próximas postagens. A Figura 2 ilustra o encadeamento de blocos no grafo da cadeia e mostra a representação interna de um bloco com quatro campos enumerados:

msg: O conteúdo da mensagem propriamente dita.
sign: Uma assinatura opcional com a chave pública do autor e o código da assinatura.
meta: Os metadados da mensagem: tempo de criação, se está criptografada, hash da mensagem, se é um like/dislike, e os elos para blocos anteriores.
hash: O hash criptográfico dos metadados do bloco que o identificam univocamente. O hash é prefixado com a altura do bloco na cadeia. Por exemplo, o bloco `5_F700CC98A6BA6A562CF6272AFC1044CB0F049E2E71D1076DA3391E85EE2CE2B8` possui altura `5` e hash `F700CC....`

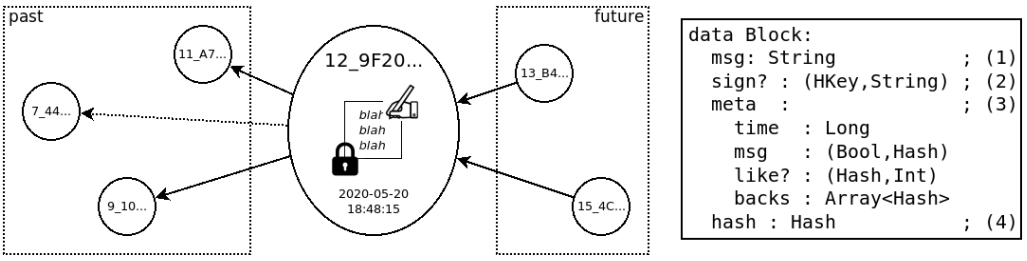

Figura 2. **Bloco (centro) se encadeia aos anteriores (esq.) e é encadeado pelos próximos (dir.).**

O encadeamento dos blocos forma um grafo direcionado acíclico que é imune a modificações (*Merkle DAG*). Ao se criar um novo bloco, calcula-se o hash dos seus metadados, que por sua vez incluem os hashes da mensagem em si e dos blocos anteriores. Esse novo hash identifica o bloco em todas as operações, inclusive nos elos dos próximos blocos, e pode a qualquer momento ser recalculado para verificar se os dados sofreram modificações. Como os elos anteriores também são identificados da mesma forma, o grafo pode ser verificado até o seu gênesis, cujo hash depende somente dos parâmetros da cadeia, sendo igualmente verificável.

3.2. Arranjos de Disseminação de Conteúdo

O Freechains suporta três tipos de cadeias, cada uma com um propósito diferente:

- **Fóruns Públicos**: Arranjo $N \leftrightarrow N$ público. Comunicação pública entre participantes sem confiança mútua. Exemplos: fóruns de perguntas e respostas, chats, e compra & venda online.

- **Grupos Privados**: Arranjos privados $1{\leftrightarrow}1$, $N{\leftrightarrow}N$ e $1{\leftarrow}$. Comunicação privada entre pares, grupos ou individuais. Exemplos: e-mail, grupos de WhatsApp, backup.
- **Identidade Pública**: Arranjos públicos $1{\rightarrow}N$ e $1{\leftarrow}N$. Uma identidade pública (ex., pessoa ou organização) dissemina conteúdo para um público alvo ($1{\rightarrow}N$) com feedback opcional ($1{\leftarrow}N$). Exemplos: sites de notícias, serviços de streaming e perfis públicos em redes sociais.

O tipo da cadeia é determinado pelo prefixo em seu nome:

- **`#`**: fórum público (ex., `#chat`)
- **`$`**: grupo privado (ex., `$familia`)
- **`@`**: identidade pública (ex., `@B2853F...`)

Em fóruns públicos, as mensagens circulam entre usuários e pares sem confiança mútua. Por essa razão, cadeias desse tipo dependem do sistema de reputação do Freechains para serem viáveis sob completa descentralização. Em grupos privados, todas as postagens são automaticamente criptografadas usando uma chave compartilhada entre os pares de confiança. Nesse caso, o comando `join` recebe um parâmetro extra com a chave, e deve ser executado da mesma forma em todos os pares:

```
$ freechains chains join "\$familia" 8889BB...
```

Todos os usuários de grupos privados têm reputação infinita e nem é necessário assinar as mensagens. Para cadeias de identidade pública, o nome deve ter o prefixo `@` seguido pela chave pública do autor proprietário da cadeia. O proprietário tem reputação infinita e deve assinar todas as mensagens com a sua chave privada:

```
$ freechains chains join "@EB172E..."
$ freechains chain "@EB172E..." post inline "Bom Dia!" --sign=96700A...
```

3.3. Sistema de Reputação

Fóruns públicos descentralizados são um convite para o abuso de usuários maliciosos com SPAM, notícias falsas e conteúdo ilícito. O sistema de reputação do Freechains permite que somente usuários com reputação prévia postem conteúdo novo em uma cadeia. Caso contrário, a postagem fica retida no par de origem e precisa ser aprovada por algum usuário com reputação para ser disseminada na rede. Cada cadeia é controlada por um sistema autônomo que contabiliza a quantidade de likes e dislikes a autores e postagens. Como cada cadeia é independente, a reputação de um autor pode variar entre elas. A unidade de reputação é conhecida como rep e pode ser gerada, consumida e transferida de diversas formas:

1. Geração:
   (a) A primeira postagem de uma cadeia adiciona $+30$ $reps$ ao autor.
   (b) Qualquer postagem mais antiga que $24h$ conta $+1$ rep ao autor, mas limitada a uma por dia. Se o autor tem 10 postagens nos últimos 7 dias, ele recebe somente $+7$ $reps$.
2. Consumo:
   (a) Qualquer postagem mais jovem que $24h$ conta -1 rep ao autor.
3. Transferência:
   (a) Um *like* partindo do autor *A* à postagem *P* do autor *B* conta -1 rep para *A* e $+1$ rep para *B*.

   (b) Um *dislike* partindo do autor *A* à postagem *P* do autor *B* conta -1 rep para *A* e -1 rep para *B*. Se uma postagem alcança pelo menos 5 dislikes que totalizem o dobro do número de likes, então o seu conteúdo é bloqueado na rede.
4. Regras Adicionais:
   (a) Postagens de usuários sem reputação ficam retidas até receberem um like, não sendo nem encadeadas nem retransmitidas.
   (b) Usuários ficam limitados a $+30$ $reps$.
   (c) Somente postagens mais novas que 90 dias são consideradas.
   (d) Em cadeias privadas, todos os usuários têm reputação infinita.
   (e) Em cadeias de identidade pública, o proprietário tem reputação infinita.

A primeira regra de geração de $reps$ (`1.a`) é essencial para fazer o "bootstrap" de uma cadeia, uma vez que seria impossível realizar postagens se ninguém possui reputação nenhuma. Assim, o autor da primeira postagem molda a cultura inicial da cadeia ao transferir sua reputação a outros autores, que por sua vez transferem a outros autores, expandindo a comunidade em alguma direção. Note que cadeias de mesmo nome mas com primeiros autores diferentes são incompatíveis e o protocolo se recusa a sincronizá-las. Isso pode acontecer quando duas redes independentes (ex., UERJ e PUC-Rio) seguem uma cadeia de nome usual (ex., `#computacao`) e, de algum jeito, acabam se unindo através de um par em comum.

Os comandos de `like` e `dislike` atuam sobre uma postagem já existente na cadeia:

```
$ freechains chain "#chat" like 2_12AB5C... --sign=96700A...
```

Nesse caso, o usuário que assinou o like transfere 1 rep seu para o autor da postagem referenciada (regra `3.a`). Já o comando `reps` verifica a reputação passando a chave pública do autor ou identificador hash da postagem a ser consultada:

```
$ freechains chain "#chat" reps 2_12AB5C...
1   <-- reputacao da postagem
```

A qualidade das postagens é subjetiva e cabe aos usuários as julgarem com likes, dislikes ou simples abstenções. Um usuário pode desgostar de uma postagem por considerá-la ofensiva, SPAM, falsa, ilícita, ou por simples desacordo. Por um lado, como os $reps$ são finitos, os usuários devem ponderar e evitar o seu gasto indiscriminado. Por outro lado, os $reps$ também expiram após 3 meses (regra `4.c`), então os usuários tem incentivos para cooperar com a qualidade das cadeias. Um conteúdo pode ser banido quando o número de dislikes supera em muito o número de likes (regra `3.b`). Considerando que os $reps$ são escassos, o banimento de postagens não tem o objetivo de eliminar discordâncias de opinião, mas sim de evitar a atuação de usuários maliciosos.

O sistema de reputação do Freechains busca oferecer oportunidades minimamente justas de participação nas cadeias. Por isso, restringe o número de postagens de um dado autor de duas maneiras: (1) penaliza postagens com menos de $24h$ (regra `2.a`); e (2) limita o ganho de reputação por postagens novas em 1 rep por dia (regra `1.b`). A primeira regra previne que um mesmo autor poste muitas mensagens em sequência sob a pena de consumir a sua própria reputação muito rapidamente. A segunda evita o acúmulo de reputação simplesmente por postar com muita frequência.

O tamanho da "economia" de uma cadeia é a sua quantidade de postagens consolidadas (regra `1.b`), dado que postagens com mais de $24h$ são a única forma de gerar $reps$.

Note que likes e dislikes apenas transferem reputação e a reputação inicial do primeiro autor se torna insignificante com o passar do tempo. A economia também depende muito da quantidade de autores ativos, uma vez que a geração de $reps$ por autor é limitada a 1 por dia. Esse mecanismo incentiva o acolhimento de novos autores a contribuírem com a cadeia, ao mesmo tempo que desincentiva dislikes pois estes drenam $2\ reps$ da economia para o limbo (regra `3.b`). Por um lado, esse desincentivo contribui para discussões com um nível razoável de desacordo, pois evita o colapso da cadeia com um surto de dislikes. Por outro lado, conteúdos claramente indesejados como SPAM de usuários que pouco contribuíram são banidos rapidamente da cadeia com poucos dislikes (regra `3.b`).

4. Conclusão

O Freechains é um protocolo peer-to-peer para disseminação de conteúdo com duas contribuições principais. A primeira é um protocolo mínimo para múltiplos arranjos de disseminação: arranjos públicos $1{\rightarrow}N$ e $1{\leftarrow}N$, Arranjos privados $1{\leftrightarrow}1$, $N{\leftrightarrow}N$ e $1{\leftarrow\!\!\shortmid}$, e arranjos públicos $N{\leftrightarrow}N$. A segunda é um sistema de reputação autônomo e descentralizado para combater o abuso e destacar conteúdo de qualidade. O Freechains pode ser operado pela linha de comando, por uma API em Kotlin ou ainda via sockets por um protocolo textual.

Em testes inicias, simulamos a execução de diversos pares em uma mesma máquina para verificar a corretude e escalabilidade do protocolo. Simulamos uma topologia com 21 pares, cada um conectado de 1 a 4 outros nós, com diversos ciclos e caminho máximo de 10 saltos. Executamos 2 aplicações simultâneas em cada par, uma com mensagens curtas e pouco espaçadas no tempo para simular uma aplicação de chat (50 bytes a cada 20 segundos, em média) e outra com mensagens longas e espaçadas para simular uma aplicação estilo *Instagram* (5 megabytes a cada 5 horas, em média). Verificamos que nos testes os pares chegam ao mesmo estado final sem perda de mensagens e com performance adequada. Essas duas aplicações simulam os arranjos públicos $N{\leftrightarrow}N$ e $1{\rightarrow}N$, respectivamente.

Ainda não temos uma avaliação da eficácia do sistema de reputação, uma vez que ele depende do interação subjetiva entre humanos nas cadeias de conteúdo, o que não permite uma simulação automatizada. Pretendemos desenvolver aplicações mais reais, tais como um chat e um wikipedia distribuído para analisar qualitativamente a evolução da reputação dos autores e postagens.

Referências

Nakamoto, S. (2019). Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot.

Tarr, D., Lavoie, E., Meyer, A., and Tschudin, C. (2019). Secure scuttlebutt: An identity-centric protocol for subjective and decentralized applications. In *Proceedings of the 6th ACM Conference on Information-Centric Networking*, pages 1–11.

Theotokis, S. A. and Spinellis, D. (2004). A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371.

Zittrain, J. (2018). Fixing the internet.