

# Avaliação de uma proposta para uma DHT evolutiva prática

James Furtado, Pedro Á. Costa, and João Leitão

NOVALINCS & NOVA University of Lisbon

`jh.furtado@campus.fct.unl.pt`, `pah.costa@campus.fct.unl.pt`,  
`jc.leitao@fct.unl.pt`

**Resumo** Os sistemas *Peer-to-Peer* (P2P) são uma alternativa cada vez mais atraente para suportar sistemas na Internet. O movimento web3 [5] tem contribuído para o aparecimento de uma ampla variedade destes sistemas. Um desafio frequentemente enfrentado por esses sistemas, especialmente em larga escala, é lidar com a evolução dos protocolos que suportam o sistema, em particular alterações significativas (i.e, atualizações que tornam o sistema incompatível com versões anteriores, em inglês comumente referido como *breaking changes*). Este desafio deriva de cada nó num sistema P2P ser operado por uma entidade diferente, o que torna impossível a atualização sincronizada de todos os processos. O IPFS, um sistema P2P de larga escala, não é uma exceção. Dados obtidos a partir do *Nebula crawler* mostra uma enorme diversidade de versões de nós do IPFS em execução. Esta situação tem sido gerida através da co-existência de várias versões de protocolos, no entanto, brevemente prevê-se que se torne inevitável a introdução de uma alteração significativa sob o modo de operação da sua *Distributed Hash Table* (DHT), o mecanismo primário que permite e facilita a interação descentralizada entre nós do IPFS. Com o objetivo de minimizar o impacto de futuras alterações significativas na DHT do IPFS, foi feita uma proposta, por investigadores da Protocol Labs, para alterar o design desta de forma a permitir a interoperabilidade entre versões distintas que, de outra forma, seriam incompatíveis. Este artigo apresenta uma análise inicial conduzida através do uso de um protótipo e de um significativo trabalho experimental. Os nossos resultados mostram que a proposta realizada é promissora, e poderá minimizar de forma significativa as anomalias causadas por futuras evoluções da DHT.

**Keywords:** IPFS · DHT · Sistemas P2P · Mudanças significativas.

## 1 Introdução

Durante a vida útil de um sistema informático, em particular de sistemas distribuídos de grande escala, estes passam por várias atualizações com diversas finalidades como a correção de erros (i.e., *bugs*), melhorias de segurança e alterações ou adição de novas funcionalidades. Estas atualizações podem conter mudanças significativas (i.e., em inglês *breaking changes*) que fazem com que partes do sistema se tornem incompatíveis com versões anteriores do mesmo.

Este tipo de mudanças, são geralmente extremamente complexas de se lidar em sistemas *Peer-to-Peer* (P2P) de larga escala visto que no pior cenário, essas mudanças podem levar a uma partição permanente na rede P2P em dois segmentos distintos devido à transição lenta por parte dos utilizadores para novas versões do sistema. Mesmo no melhor cenário, será necessário manter um mecanismo de retro-compatibilidade por períodos indeterminados, o que limita a evolução do software, tipicamente carrega penalizações não desprezáveis ao desempenho destes sistemas, e limita de forma severa a introdução de novas funcionalidades ou melhorias significativas no sistema.

O *Inter-Planetary File System* (IPFS) [2], que é um sistema de ficheiros distribuídos P2P cada vez mais popular e um dos pilares da Web3 [6], é atualmente utilizado por centenas de milhares de utilizadores, tendo regularmente mais de 20.000 nós ativos na rede a operar como servidor [15,14]. Devido a preocupações crescentes de privacidade, este sistema têm prevista uma evolução na sua arquitetura que vai resultar numa *breaking change* da sua *Distributed Hash Table* (DHT), que é o mecanismo primário que permite e facilita a interação descentralizada entre nós do IPFS.

A fim de mitigar as repercussões de futuras *breaking changes* na DHT, os engenheiros e investigadores da **Protocol Labs**<sup>1</sup> propuseram [8] um novo *design* para a DHT do IPFS (que até o final do artigo iremos referir como *Upgradable DHT*). Apesar desta solução ter o potencial para permitir que no futuro novas funcionalidades possam ser adicionadas à DHT, e que nós em diferentes versões (e consequentemente com diferentes funcionalidades disponíveis) possam co-existir na rede recorrendo a um conjunto de funcionalidades fundamentais definida na proposta referida acima, é pouco claro quais as implicações deste mecanismo.

Este artigo vem demonstrar a viabilidade da solução proposta pelos engenheiros e investigadores da ProtocolLabs e estudar o impacto que esta poderá vir a ter, face a versão atual do IPFS através da condução de uma avaliação experimental dessa proposta. Para tal, foi desenvolvido um protótipo da Upgradable DHT e usou-se como exemplo de *breaking change* o protótipo desenvolvido pela **Chain Safe** [13] (adaptadas a *Upgradable DHT*) do novo mecanismo de privacidade que será implementado na DHT do IPFS.

O resto deste artigo está organizado da seguinte forma. A secção 2 apresenta um conjunto de informação relevante a compreensão do trabalho apresentado neste artigo relativo ao funcionamento da IPFS e da sua DHT. A secção 3 apresenta a proposta da *Upgradable DHT* e as decisões de design que tivemos que tomar poder implementá-la. A secção 4 discute a implementação do nosso protótipo. A secção 5 discute o nosso trabalho experimental, incluindo o *setup* experimental e os resultados obtidos. Finalmente a secção 6 conclui o artigo.

## 2 Preliminares

Nesta seção, vamos apresentar: uma breve introdução ao IPFS (§ 2.1), a sua DHT (§ 2.2), a *breaking change* que esta vai sofrer e a sua motivação (§ 2.3).

<sup>1</sup> A empresa que desenvolve e mantém o IPFS.

## 2.1 IPFS

O IPFS [2] é um sistema de ficheiros distribuídos *Peer-to-Peer* (P2P), onde cada ficheiro é identificado por um *hash* criptográfico do seu conteúdo, denominado por *Content Identifier* (CID). Os ficheiros estão distribuídos por vários nós (computadores a correr uma instancia do IPFS) e a informação sobre que nó guarda que ficheiro é mantida numa estrutura de dados distribuída denominada de *Distributed Hash Table* (DHT).

Cada nó do IPFS possui um par de chaves pública/privada gerados por um algoritmo criptográfico assimétrico. Essas chaves são usadas para identificar e autenticar operações feitas pelo nó. O *hash* da chave publica do nó é o identificador único do nó no IPFS, e este é denominado por *Peer Identifier* (PID).

Quando um nó do IPFS quer publicar um ficheiro no IPFS, este calcula o CID do ficheiro e usa a DHT para anunciar na rede do IPFS que ele tem um cópia do ficheiro identificado pelo respetivo CID. Para se obter um ficheiro a partir do IPFS, este usa um outro protocolo chamado de *BitSwap* [12]. O *BitSwap* começa por interrogar aos vizinhos imediatos do nó em questão pelo conteúdo do ficheiro e em caso negativo usa a DHT para obter a informação sobre que nós é que tem uma copia do ficheiro. Depois o *BitSwap* procede à transferência do ficheiro que é modo bem similar ao *Bittorrent* [9].

## 2.2 DHT

A DHT é uma extensão das tabelas de dispersão convencionais para um ambiente distribuído. Onde a informação dos pares  $\langle chave, valor \rangle$  estão distribuídos e replicados em vários dispositivos. A DHT usada pelo IPFS é uma variante da *Kademlia* [7], que incorpora ideias do *S/Kademlia* [1] por questões de segurança e do *Coral DSHT* [4] - onde uma chave pode estar associada a mais de um valor.

Para poder entender como a DHT do IPFS funciona, iremos indicar os passos que se tem que dar para adicionar e pesquisar conteúdo na DHT. No contexto do IPFS a chave é o CID do ficheiro, e o valor é um registo assinado com chave privada de um determinado nó que declara que este nó contém uma cópia do ficheiro identificado pelo CID, que também estará presente no registo. Este registo chama-se *Provider-Record* (PR). Logo as entradas da DHT do IPFS são pares  $\langle CID, PR \rangle$  [11].

Para adicionar uma entrada na DHT do IPFS, um nó deve encontrar  $K$  (tipicamente 20) nós tal que o  $\mathbf{XOR}(PID_i, CID)$  tenham o menor valor possível, isto é, os  $K$  nós cujo identificador está mais próximo do CID [11,7]. Para obter o valor de uma entrada da DHT do IPFS, um nó recorre aos restantes nós para identificar aqueles que têm um PID mais próximo do CID pesquisado, e verifica se estes tem o PR desse CID. Se estes não tiverem, eles vão responder com os  $K$  nós mais perto do CID que eles conhecem permitindo ao nó continuar a sua pesquisa.

Para que estas operações sejam eficientes cada nó da DHT mantém uma *Routing Table* composta por  $K$ -buckets [7] que guardam informações sobre outros nós que fazem parte da DHT. Um dos objectivos principal desta *Routing Table* é

que permitir a cada nó conhecer mais nós que estão mais pertos deste (distância **XOR** dos PIDs) do que nós que estão mais distantes. Para tal os *K-buckets* que compõem a *Routing Table* são enumerados começando em 0, cada *K-bucket*  $i$  pode guardar no máximo informação de  $K$  nós e a informação só é guardada nesse *K-bucket* se o PID deste nó tenha um prefixo de tamanho pelo menos  $i$  bits em comum com o PID do nó que guarda os *K-buckets*.

Como explicado anteriormente, a DHT do IPFS permite com que mais de um valor esteja associado a uma chave. Um nó responsável por guardar informação de um CID pode guardar mais de um PRs associados a esse CID. Também a operação de pesquisa sobre um CID na DHT pode retornar apenas um subconjunto (eventualmente unitário) dos PRs conhecidos para este CID.

### 2.3 Double Hashing

Quando um nó IPFS consulta a DHT para obter os PR de um determinado CID, este nó obtém essa informação de outros nós. No entanto, ao fazer isso, revela a vários participantes o CID que está a procura. Estes nós por sua vez podem fazer a mesma pesquisa na DHT e determinar que o conteúdo do ficheiro que o nó inicial estava a tentar aceder.

Isto implica que a privacidade de utilizadores do IPFS pode ser trivialmente comprometida. Embora os IDs dos nós IPFS sejam opacos e não estejam diretamente associados às identidades dos utilizadores, é possível, através de vários métodos conhecidos, extrair essas informações (por exemplo os IPs são expostos e registos públicos podem permitir identificar um utilizador).

Para mitigar este problema, o IPFS vai implementar um mecanismo denominado *Double-Hashing* [10] que muda o que é guardado na DHT e como se extraí informação desta. Em vez de se guardar pares  $\langle CID, PR \rangle$  na DHT, vão ser guardados pares  $\langle Hash(CID)[N], PR_{CID} \rangle$ . Ou seja o um prefixo do *hash* do CID (que já é um *hash*) e o *Provider-Record* cifrado com o CID da entrada.

O nó que quer obter *Provider-Records* de um ficheiro (para poder descarregá-lo), tem de saber o CID do ficheiro em que está interessado. Depois este vai calcular o *hash* do CID, escolher um prefixo do *hash* que calculou e repetir o mesmo processo que se fazia com o CID mas com o prefixo de *hash(CID)*. A única diferença é que os nós contactados devem responder, não com uma entrada específica ou com  $K$  nós mais perto, mas com todas as entradas cuja chave tenha um prefixo igual ao solicitado e com os  $K$  nós que estes conhecem que estejam mais próximos desse prefixo. Desta forma todos os nós deixam de conhecer o CID pesquisado pelo nó inicial, e ainda que apenas uma entrada exista para esse prefixo, o nó que armazena essa entrada não consegue ver os conteúdos do *Provider-Record* pois o CID não foi exposto. Na nossa implementação, baseada na da **SafeChain**, os prefixos têm um tamanho constante de 64 bits.

## 3 Upgradable DHT

A ideia principal do *Upgradable DHT* [8] (também denominada de *Composable DHT*) proposta pelos engenheiros e investigadores da Protocol Labs é ter uma

DHT que suporta controlo de versões e que consiga oferecer interoperabilidade entre versões diferentes da DHT mesmo na presença de mudanças significativas.

Para tal foi proposta a noção de *funcionalidade*, que basicamente representa uma funcionalidade suportada por uma versão da DHT. Todos os nós partilham a lista de funcionalidades que suportam entre si e com base nesta os nós conseguem saber se podem realizar uma determinada operação num nó específico da DHT. A proposta também propõe o uso de uma função que será usada para determinar o quão semelhante/relevante a lista de funcionalidade de um nó é comparado com a de outro. Os nós devem preferir ter nos seus *K-buckets*, nós cuja lista de funcionalidade maximize a compatibilidade (valor da função referida) com as suas funcionalidades suportadas e que dê preferência a funcionalidades mais relevantes. Na nossa implementação, por que só tínhamos 2 variantes da *Upgradable DHT*, em que uma tem algumas funcionalidades a mais que outra, a função que usamos para tal simplesmente dava um ponto para cada funcionalidade em comum da duas listas de funcionalidade.

A proposta também sugere que todas as versões da DHT deveriam ter uma funcionalidade de base, que basicamente permite, dado uma chave/sequência-de-bytes, obter os nós cujo PID é o mais próximo da chave. Com obter os nós queremos dizer obter informações sobre este, desde o PID, o endereço e a lista de funcionalidades que este suporta. Um problema que não foi discutido nesta proposta mas que tivemos que resolver é: se usarmos a ideia do *Upgradable DHT* para ter versões do IPFS com o *Double-Hashing* (em que só se consegue publicar em nós que também suportam *Double-Hashing*) e outras versões em que essa funcionalidade não está disponível como é que nós das diferentes versões conseguem encontrar o nós a quem eles podem publicar uma entrada na DHT.

Uma solução seria, durante o processo de procura dos nós mais próximo da chave cada nó só comunica com nós que possui a mesma funcionalidade de publicar que este. O problema disto é que limita a interação entre nós com versões diferentes e não garante que o nó chegue ao nó mais próximo da chave. Para conduzir as nossas experiências fizemos com que a versão da DHT que suporta *Double-Hashing* também tenha a funcionalidade de responder a as pedidos do protocolo original. E quando estes forem publicar usando o *Double-Hashing*, façam uma pesquisa semelhante ao protocolo original, em que se localizam os nós mais próximo da chave pesquisado independentemente deste suportar *Double-Hashing*. Porém estes vão preferir comunicar com nós que suportam *Double Hashing*. Quando não é possível encontrar nós mais pertos do prefixo filtram os nós que encontraram para obter apenas aqueles que suportam *Double-Hashing*.

## 4 Implementação

O código do IPFS, foi escrito em *Go* e é composto por vários repositórios Git que implementam diferentes partes das funcionalidades do IPFS. Destacamos alguns deles:

### kubo

É o repositório principal que agrega os restantes por meio de dependências.

**go-libp2p**

Uma biblioteca que abstrai a rede, fornece uma maneira eficiente de registrar protocolos e suas respectivas *handling functions* e multiplexa conexões virtuais sobre uma ou mais conexões na rede física.

**go-libp2p-kad-dht**

A implementação da DHT do IPFS.

**go-libp2p-kbucket**

A implementação de K-buckets usados na DHT.

Para implementar a *Upgradable DHT*, migrar o código da DHT atual para a *Upgradable DHT*, adicionar os mecanismos de segurança discutidos na secção 2.3 (que foram migrados do protótipo da ChainSafe [13]) como funcionalidades a *Upgradable DHT* e adicionar *logs* para as experiências tivemos que mudar a lógica nos componentes destes 4 repositórios. Utilizamos a versão do IPFS (**kubo**) **0.19.1**.

Como explicado na secção 3 a ideia do *Upgradable DHT* é ter várias DHTs com várias funcionalidades. A proposta original não define concretamente o que seria uma funcionalidade portanto assumimos que as mesmas são capturadas por RPCs (*Remote Procedure Calls*) da DHT. São estes que na nossa implementação definem as funcionalidades que um nó suporta. Por isso, o *PeerID*, contém uma lista de funcionalidades, que neste caso será uma lista de strings que correspondem aos IDs de cada RPC da DHT.

Para testar se a nossa implementação usamos os testes unitários disponíveis no código original (com algumas alterações em alguns casos) e outros que nós adicionamos. Adicionalmente, verificamos a correção das experiências através de várias verificações sobre os *logs* obtidos como o PID que se obteve da DHT corresponder ao nó que publicou o conteúdo, entre outros.

Todo o código usado nas experiências está disponível em <https://github.com/JamesHertz/ipfs-tests>.

## 5 Avaliação Experimental

### 5.1 Versões da DHT

A fim de poder comparar o desempenho da *Upgradable DHT* face à DHT original, criou-se duas versões da *Upgradable DHT*. Em que a única diferença de uma para a outra é a existência de 2 RPCs adicionais que publicam e consultam (resolvem CIDs) na DHT usando *Double-Hashing*. O resto, em relação à DHT original, manteve-se inalterado, senão que esta versão da DHT anuncia as funcionalidades/RPCs que suporta e prefere ter nós nos seus *K*-buckets que tenham mais funcionalidades em comum consigo.

Das duas versões da *Upgradable DHT* que implementamos denominamos a versão que suporta o *Double-Hashing* de **SECURE** e a outra de **NORMAL**. E algumas regras se aplicam, no que toca a publicar e consultar conteúdo publico na *Upgradable DHT*: **(1)** Os nós **SECURE** publicam utilizando os RPCs que suportam *double hashing* pelo que o conteúdo que estes publicam não pode

ser consultado pelos nós NORMAL. (2) Os nós NORMAL publicam usando o algoritmo base, e como os SECURE suportam todas as funcionalidades que os NORMAL suportam, os SECURE conseguem consultar conteúdo publicado por um nó NORMAL. (3) Os nós SECURE guardam *Provider Records* para CIDs publicados por nós SECURE e por nós NORMAL. (4) Os nós NORMAL só guardam *Provider Records* guardados por nós do tipo NORMAL, por que estes não tem as funcionalidades que lhes permite suportar o algoritmo de pesquisa do *double hashing*.

Os comportamentos definidos acima para os nós NORMAL e os SECURE foram definidos a pensar no cenário que o IPFS iria passar caso este já tivesse implementado uma *Upgradable DHT* e pretendesse suportar a função de *Double-Hashing*.

## 5.2 Configurações

Para se realizar as experiências foram criados 600 directorias de trabalho do IPFS, que foram mapeadas para um nó distinto do IPFS, que guardam o par de chaves criptográficas assimétrica de cada nó juntamente com o seu PID. Também foram gerados 12.000 ficheiros, 20 por cada nó, de 10 MiB, calculou-se os CIDs e guardou-se num ficheiro. O parâmetro  $K$  dos  $K$ -buckets usado foi 10. E por ultimo atribui-se 20 CIDs para cada nó de IPFS, para publicação no IPFS durante a experiência, como explicado mais a frente. Em todas as experiências reportadas usamos esta configuração inicial. Isto permite garantir que a estrutura da rede sobreposta apresente pouca variabilidade e os nós que guardam os pares da DHT sejam semelhantes entre experiências.

Executámos cada experiência em 8 máquinas com 16 a 32 *hardware cores* e mais de 100GB de RAM. Cada nó de IPFS executa num container docker, sendo que usámos docker-swarm para ter uma rede virtual que liga todos os containers que estavam em máquinas físicas diferentes. Também usou-se a ferramenta **Traffic Control** do Linux para introduzir latencia artificial nas conexões entre todos os containers.

As experiências capturam 2 cenários distintos e cada experiência foi repetida 3 vezes. Em ambos os cenários usamos 600 nós, dentre os quais 10 serviram como *bootstrap*, ou seja, os nós que os restantes utilizaram para formar a rede sobreposta. No inicio da experiência todos os nós juntaram-se ao sistema através dos nós de *bootstraps*, e depois de 5 minutos todos os nós publicam 20 CIDs na DHT. Depois de 5 minutos, cada nó escolhe um CID random de entre todos os CIDs<sup>2</sup> (que foram pre-gerados) e consulta a DHT pelo PID do nó que a publicou durante um período de 20 minutos. Entre os dois cenários executados, a única diferença é que no primeiro os 600 nós executam a versão não alterada da DHT e no segundo 300 nós executam a versão NORMAL e 300 a versão SECURE da upgradeable DHT. Os *bootstraps* do segundo cenário foram 5 nós com a versão NORMAL e 5 com versão SECURE.

<sup>2</sup> Excepto nós da variante NORMAL que só escolhem CIDs publicado por nós da mesma versão.

O nosso objetivo com esta experiência foi observar como é que as operações a DHT principalmente as leituras se comportam face as mudanças, uma vez que a leitura é a operação mais frequente to IPFS [3].

### 5.3 Resultados

Nesta seção reportamos os resultados das experiências conduzidas. Todos os resultados apresentados são médias das 3 repetições que se fez por cada uma dos dois cenários capturados nas experiências. Em média durante cada experiência se publicou 12.000 CIDs e foram feitas 200.000 pesquisas (333,3(3) por nó).

Começamos por reportar a taxa de sucesso das operações de procura na DHT, comparando a versão original da DHT (denominada de Baseline) como o desempenho que a variante SECURE da Upgradable DHT, quando esta procura por conteúdo publicado por ambas as variantes da Upgradable DHT, e o desempenho que os nós da variante NORMAL da Upgradable DHT, quando estes procuram por conteúdo publicados por nós da sua variante da Upgradable DHT. A Figura 1 mostra que a taxa de sucesso foi praticamente de 100% para todos os CIDs que se tentou procurar pelo respetivo *Provider Record* (PR) na DHT. Isto mostra que, neste trabalho experimental, a Upgradable DHT manteve a disponibilidade de todos os CIDs.

De seguida reportamos o *throughput* das operações de pesquisa na DHT. Estas, como referido anteriormente na secção 5.2, são realizadas nos últimos 20 minutos de cada experiência. O que se pode observar é que o *throughput* da DHT quando se usa a *Upgradable DHT* é apenas ligeiramente inferior ao que se obtém com a DHT atual. Logo, com base neste trabalho experimental, pode-se concluir que o custo adicional de operação da Upgradable DHT em termos de desempenho é desprezável.

As Figuras 3 e 4 mostram, respetivamente, o tempo médio para concluir as operações de procura e de publicação de conteúdo na DHT em milissegundos. Em relação à procura, o que se pode observar é que os nós NORMAL quando pesquisam por CIDs publicados por nós NORMAL e os nós SECURE quando pesquisam por CIDs também publicados por nós NORMAL apresentam tempos de acesso similares aos apresentados pelo baseline.

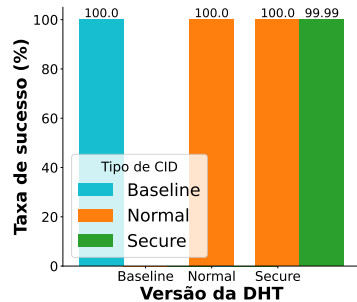


Figura 1: Taxa de sucesso nas operações de procura na DHT.

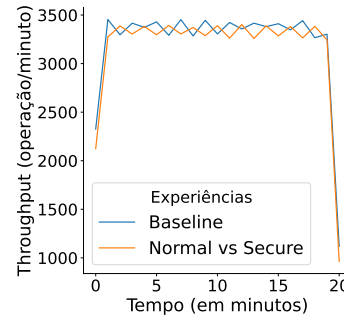


Figura 2: *Throughput* das DHTs.



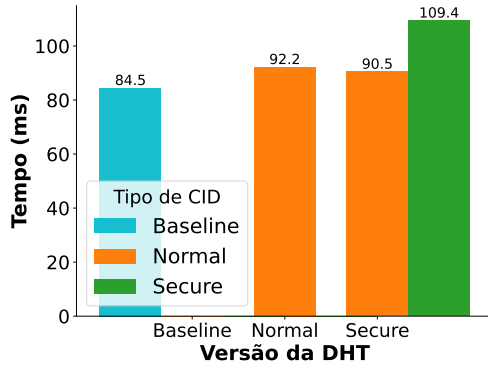


Figura 3: Tempo médio de procura.

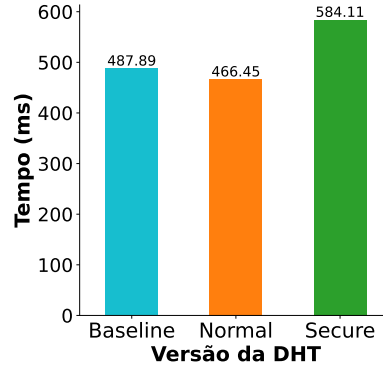


Figura 4: Tempo médio de publicação.

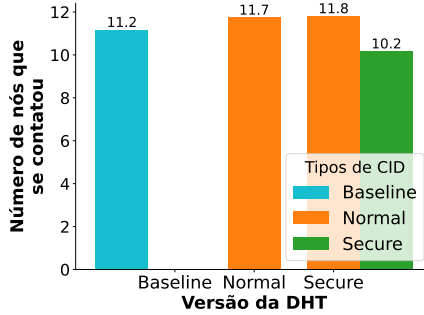


Figura 5: Número de contactos durante as pesquisas.

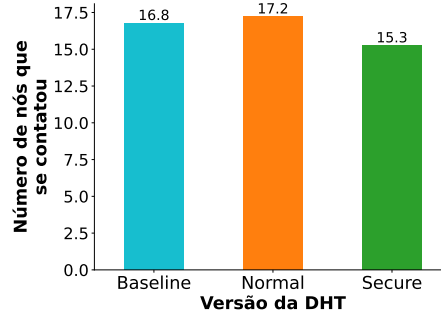


Figura 6: Número de contactos durante as publicações.

O tempo de procura dos nós SECURE por CIDs SECURE já apresentam um valor mais elevado em relação ao baseline. Este fenómeno é esperado, uma vez que o uso do mecanismo de *Double-Hashing* apresenta um custo criptográfico adicional. Uma justificação complementar é que apesar de que os nós SECURE preferirem ter outros nós SECURE nas suas tabelas de encaminhamento (*K-buckets*), estes ainda devem ter uma fracção de nós NORMAL e por isso tem que fazer um percurso médio maior até outros nós SECURE, o que pode levar a um ligeiro aumento da latência.

Em relação à publicação de conteúdos, nota-se que (naturalmente) apresentam um tempo de execução maior em relação às pesquisas, isto acontece porque a publicação envolve encontrar os  $K$  nós da DHT mais perto de uma chave ou um prefixo. Também nota-se que a publicação de CIDs feita por nós SECURE, que usam o mecanismo de *Double-Hashing*, demora mais tempo do que os restantes, pelas mesmas razões mencionadas acima. O leitor deve notar que o custo adicional do mecanismo de *Double-Hashing* nas pesquisas é de cerca de  $30ms$  mas para a publicação sobe para cerca de  $100ms$ . Isto por que a publicação usando o mecanismo de *Double-Hashing* requer a execução de uma cifra simétrica (AES na nossa implementação) que é significativamente mais lenta do que o uso de funções de dispersão criptográficas usadas na procura.

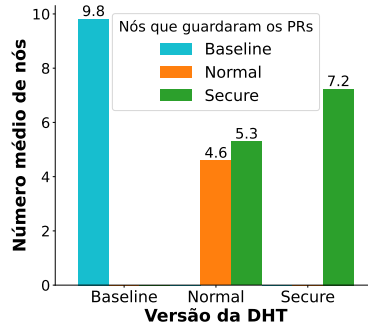


Figura 7: Número médio de nós que cada CID foi publicado.

Na Figura 5 é reportado a média do número de nós que se contactou ao consultar a DHT por um CID e até se obter o *Provider-Record* desse CID. Por sua vez, na Figura 6 observa-se a mesma coisa mas relativamente às operações de publicação de conteúdo na DHT. Pode-se observar que os nós SECURE quando pesquisam CIDs SECURE e publicam CIDs apresentam um número menor de contactos assim como o esperado. Isto acontece porque ao usar *Double-Hashing*, os CIDs são publicados e procurados usando o prefixo do *hash* do CID em vez do CID inteiro, logo é necessário contactar um menor número de nós na DHT para encontrar os nós mais próximos desse prefixo.

A Figura 7 mostra o número médio de nós a que o *Provider-Record* (PR) de um CID foi publicado, idealmente devem ser  $K$  nós distintos que guardaram a entrada  $\langle CID, [PR] \rangle$  na DHT. Uma vez que a publicação é feita durante os primeiros minutos da experiência, pode acontecer que a DHT não tenha estabilizado e não se conheça nós o suficiente (como se vai observar na explicação da Figura 9) para publicar em  $K$  nós. Esperara-se que estes casos sejam infrequentes e que a média de nós publicados seja próxima de 10 (o valor do  $K$  usado). Isto é o que se verifica quando olhamos para número de nós nos resultados obtidos para a baseline e dos nós com funcionalidade NORMAL mas não é o caso para os nós com a funcionalidade SECURE. Isto acontece porque assim como explicado anteriormente na secção 4, o algoritmo usado pelos nós SECURE para encontrar os nós mais próximos do prefixo do hash de um CID é semelhante ao utilizado para encontrar os nós mais perto de um CID, só que, no final são filtrados os nós obtido de forma a considerar apenas os nós SECURE.

Consideramos agora algumas métricas relativas ao grafo que a rede sobreposta da *Upgradable DHT* e a DHT *baseline* do IPFS apresentam. Começamos com o Coeficiente de agrupamento, que está ilustrado na Figura 8, e que mede a percentagem dos vizinhos em comum entre um nó e os seus vizinhos. Quanto menor for o valor melhor, pois quanto maior o coeficiente de agrupamento mais danos o grafo terá face a *churn* de nós. O que se pode observar é que este evolui ao mesmo ritmo em ambas das DHTs. No início apresenta um valor maior, isto por que o nós utilizam o conjunto de nós de *bootstraps* para se juntarem à rede e por isso vão ter mais vizinhos em comum, e com o tempo o valor vai diminuindo.

A figura 9 mostra o grau médio dos vértices do grafo, ou seja o número médio de vizinhos que cada nó tem. Novamente observa-se que este evolui de

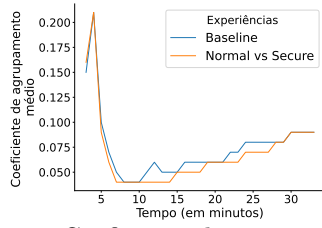


Figura 8: Coeficiente de agrupamento médio da rede sobreposta.

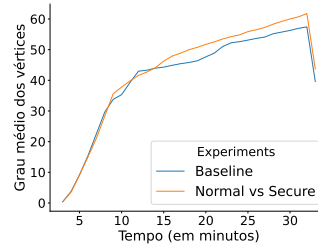


Figura 9: Grau médio de cada nó.

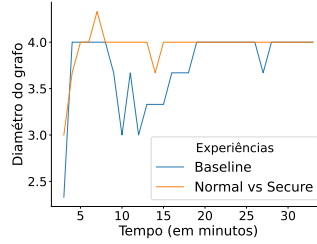
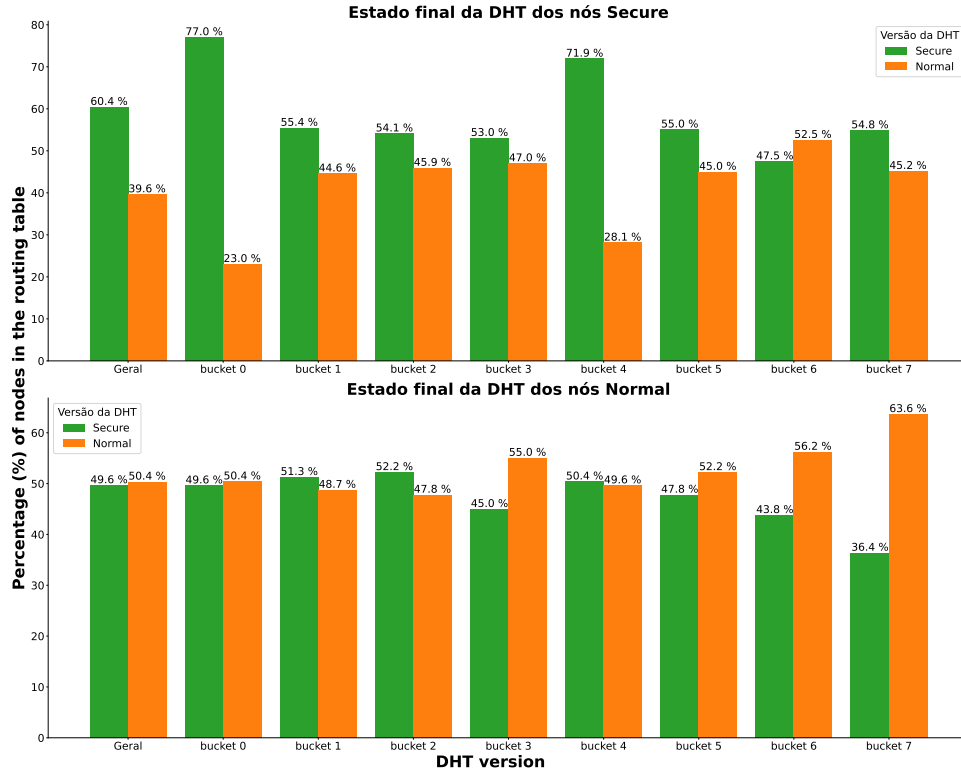


Figura 10: Evolução do diâmetro da rede sobreposta.

forma idêntica em ambas as DHT. No início este é pequeno, e com o tempo vai aumentando devido a varias interações entre os nós da DHT. Apesar do valor do grau chegar aos 50 num grafo com 600 nós, este é o comportamento esperado no Kademlia, sendo que o coeficiente de agrupamento continua baixo. É interessante observar que por volta dos 5 minutos nem todos os nós tem 10 vizinhos, sendo este o momento quando se começa a publicar os CIDs. Isto acontece por que apesar de todos os nós tentarem ligar-se aos bootstrap, estes são apenas 10 e a fim de minimizar a quantidade de conexões que os *bootstraps* tem que ter aberto, fizemos com que cada nó escolha um tempo aleatório nos 5 minutos iniciais para se ligar aos *bootstraps*.

Por fim, consideramos o diâmetro do grafo codificado pelas relações de vizinhança ao nível da DHT. Estes são apresentados na Figura 10, que é o maior de todos os menores caminhos entre quaisquer 2 vértices. Esta métrica representa o caminho mais longo entre dois nós da rede sobreposta e por isso quanto menor melhor. O que se pode observar é que que varia em torno de 4 e que o *Upgradable DHT* esteve mais estável durante a experiência toda, enquanto que o da baseline variou um pouco mas no final estabilizou em 4 também. Estes resultados mostram que a upgradeable DHT mantém as propriedades da rede lógica da solução original.

Para terminar reportamos o estado final dos *K-buckets* da *Upgradable DHT* distinguindo entre os buckets dos nós NORMAL e SECURE e o tipo de nós que se encontram nestes buckets também. Assim como podemos ver na Figura 11, em geral os nós SECURE têm mais nós SECURE do que NORMAL nos *K-buckets*, enquanto que os nós NORMAL são insensíveis ao tipo de nó, visto que os SECURE também possuem todas as funcionalidades que estes suportam.

Figura 11: Estado final os  $K$ -buckets da *Upgradable DHT*

## 6 Conclusões

Neste artigo realizámos uma avaliação experimental de uma proposta da Protocol Labs para um desenho alternativo de DHT que permite a introdução de mudanças significativas e novas funcionalidade na DHT sem que a rede lógica formada por essa DHT quebre, permitindo assim maior flexibilidade na evolução de sistemas descentralizados baseados em DHTs.

Os resultados obtidos mostram que a *Upgradable DHT* apresenta alguns custos de desempenho, mas que estes são bastante baixos. Em nenhuma das métricas, desde a disponibilidade dos CIDS, os tempos de procura e de publicação e por fim algumas métricas relativas à topologia da rede sobreposta que as DHTs formaram, foram encontradas diferenças significativas. Já em relação ao mecanismo de *Double-Hashing* observou-se que este de facto apresenta alguns custos adicionais mesuráveis, assim como esperado, mas também não são demasiado elevados e consideramos que face às garantias extras oferecidas, esse custo é aceitável.

**Agradecimentos:** O trabalho apresentado neste artigo foi suportado pela Fundação para a Ciência e Tecnologia através do laboratório NOVA LINCS (UIDP/04516/2020: DOI 10.54499/UIDP/04516/2020) e pelo projeto Horizon Europe TaRDIS financiado pela Comissão Europeia (Grant Agreement number 101093006).

## Referências

1. Baumgart, I., Mies, S.: S/kademlia: A practicable approach towards secure key-based routing. In: ICPADS. pp. 1–8. IEEE Computer Society (2007), <http://dblp.uni-trier.de/db/conf/icpads/icpads2007.html#BaumgartM07>
2. Benet, J.: Ipfs - content addressed, versioned, p2p file system (2014)
3. Costa, P.Á., Leitão, J., Psaras, Y.: Studying the workload of a fully decentralized web3 system: Ipfs (2022), <https://arxiv.org/abs/2212.07375>
4. Freedman, M., Freudenthal, E., Mazières, D.: Democratizing content publication with coral. In: First Symposium on Networked Systems Design and Implementation (NSDI 04). USENIX Association, San Francisco, CA (Mar 2004), <https://www.usenix.org/conference/nsdi-04/democratizing-content-publication-coral>
5. Gan, W., Ye, Z., Wan, S., Yu, P.S.: Web 3.0: The future of internet (2023), <https://arxiv.org/abs/2304.06032>
6. Guan, C., Ding, D., Guo, J.: Web3. 0: A review and research agenda. In: 2022 RIVF international conference on computing and communication technologies (RIVF). pp. 653–658. IEEE (2022)
7. Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the xor metric. In: Druschel, P., Kaashoek, F., Rowstron, A. (eds.) Peer-to-Peer Systems. pp. 53–65. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
8. Michel, G.: Composable dht design. <https://pl-strflt.notion.site/Composable-DHT-Design-704a1d9ccfdd40d8977f08b7258f6f3c>, (Acedido em 23/06/2024)
9. Pouwelse, J., Garbacki, P., Epema, D., Sips, H.: The bittorrent p2p file-sharing system: Measurements and analysis. In: Proceedings of the 4th International Conference on Peer-to-Peer Systems. p. 205–216. IPTPS’05, Springer-Verlag, Berlin, Heidelberg (2005). [https://doi.org/10.1007/11558989\\_19](https://doi.org/10.1007/11558989_19), [https://doi.org/10.1007/11558989\\_19](https://doi.org/10.1007/11558989_19)
10. Protocol labs: Double hash dht. <https://github.com/guillaumemichel/specs/blob/double-hashing-dht/IPIP/0373-double-hash-dht.md>, (Acedido em 06/11/2023)
11. Protocol labs: libp2p, (Acedido em 01/08/2024)
12. De la Rocha, A., Dias, D., Psaras, Y.: Accelerating content routing with bitswap: A multi-path file transfer protocol in ipfs and filecoin (2021)
13. Safe, C.: Double hashing dht prototype. <https://github.com/ChainSafe/go-libp2p-kad-dht>, (Acedido em 23/06/2024)
14. Trautwein, D.: A network agnostic dht crawler, monitor, and measurement tool that exposes timely information about dht networks. <https://github.com/dennis-tra/nebula>, (Accessed on 31/07/2024)
15. Trautwein, D., Raman, A., Tyson, G., Castro, I., Scott, W., Schubotz, M., Gipp, B., Psaras, Y.: Design and evaluation of ipfs: a storage layer for the decentralized web. In: Proceedings of the ACM SIGCOMM 2022 Conference. p. 739–752. SIGCOMM ’22, Association for Computing Machinery, New York, NY, USA (2022). <https://doi.org/10.1145/3544216.3544232>, <https://doi.org/10.1145/3544216.3544232>