Suporte ao Desenvolvimento de Novas Aplicações Descentralizadas

Tomás Galvão, Rafael Domingues Matos, Felipe Rossi, Diogo Paulico, and João Leitão

NOVA LINCS & DI-FCT-UNL
{t.galvao,rd.matos,fp.carmo,d.paulico}@campus.fct.unl.pt,jc.leitao@fct.unl.pt

Resumo Os sistemas descentralizados de grande escala estão a ganhar popularidade devido ao sucesso de aplicações baseadas em cripto-moedas, ou na tecnologia blockchain na sua generalidade, bem como ao aumento da desconfiança nos grandes operadores de cloud e preocupações com a privacidade dos dados. Esses sistemas são frequentemente desenvolvidos combinando vários protocolos descentralizados ao nível aplicacional, um processo moroso e dispendioso. A framework Babel foi recentemente proposta para simplificar o desenvolvimento desses sistemas, oferecendo um modelo próximo do utilizado por investigadores da área, lidando ainda de forma semi-automática com aspetos de baixo nível.
No entanto, os novos sistemas descentralizados impõem requisitos com aspetos pouco abrangidos por ferramentas práticas com o Babel, tais como a configuração automática de membros, a reconfiguração dinâmica do sistema durante a sua vida útil e aspetos relacionados com segurança, como a autenticação descentralizada de elementos e a proteção dos dados trocados entre os mesmos. Para endereçar estes desafios, neste artigo, propomos um conjunto de novas técnicas que podem ser utilizadas por diferentes protocolos descentralizados e sistemas, e materializamos estas técnicas numa nova versão da framework Babel enriquecida com suporte nativo para endereçar todos estes desafios. A nossa avaliação experimental mostra que é simples integrar estes mecanismos em protocolos existentes e que o custo adicional de execução dos mesmos é reduzido.

Keywords: Sistemas descentralizados · Gestão automática · Reconfiguração · Segurança · Ferramentas de Desenvolvimento.

1 Introdução

O interesse nos sistemas descentralizados tem aumentado nos últimos anos, devido a estes constituírem uma alternativa a suportarem aplicações na Internet mais resistentes a censura, democráticas, e independentes de operadores de plataformas de nuvem que para além de controlarem grande parte do serviços usados no dia a dia, também têm acesso a dados de operadores e utilizadores. Neste contexto, o movimento Web3 [13] (incluindo serviços como IPFS [3,9]) e a popularidade de sistemas baseados em blockchains [2,23,1] assim como a crescente de popularidade de sistemas que garantem o anonimato de utilizadores [11] têm contribuído significativamente para o aumento da popularidade destes serviços.

É expectável que várias aplicações peer-to-peer sejam cada vez mais complexas para atender às crescentes exigências.

Estes sistemas são muitas vezes compostos por implementações de protocolos distribuídos (e descentralizados) e que cooperam entre si para fornecer funcionalidades ou abstrações quer a outros protocolos como às aplicações. Por exemplo, uma aplicação que apenas propaga mensagens para todos os utilizadores de uma rede sem um servidor central, apesar de conceptualmente simples uma aplicação destas requer um protocolo que forneça informação (dinâmica) sobre a filiação do sistema (tipicamente uma rede sobreposta [26,19,8], um protocolo de propagação de rumor [4,6,18], e potencialmente se pretendermos a receção de mensagens emitidas enquanto um nó se encontra temporariamente desligado da rede, um protocolo de anti-entropia [10,24], em que os dois últimos protocolos operam sobre informação providenciada pelo protocolo de filiação. Notamos que seria possível também fazer uma implementação monolítica em que todas estas funcionalidades eram implementadas em conjunto, no entanto esta exigiria uma implementação muito especializada e pouco flexível. Outra motivação importante para o desenho modular passa pela possibilidade de reutilizar protocolos já implementados na construção de diferentes aplicações (de forma a que as implementações de diferentes protocolos funcionem como uma biblioteca).

A consequência de ter tantas componentes lógicas é que o desenvolvimento de aplicações e sistemas descentralizados se torna repetitiva, complexa, e propensa a erros. De forma a mitigar este problema, no passado propusemos a framework Babel [12] que oferece um conjunto de abstrações para simplificar o processo de desenvolvimento de protocolos e sistemas distribuídos e descentralizados, em grande parte por permitir que a implementação de um protocolo seja muito semelhante às típicas apresentações de artigos em algoritmos baseados em eventos, escondendo complexidades inerentes à execução paralela de programadores e endereçando necessidades comuns de baixo nível como gestão de ligações TCP ou de temporizadores.

Apesar do Babel ter sido utilizado com sucesso não só em atividades letivas como também para desenvolver vários tipos de sistemas e protocolos distribuídos as necessidades de novas aplicações práticas descentralizadas continuam a aumentar e esta framework não fornece suporte adequado às mesmas. Estas necessidades prendem-se não só com o esforço do programador mas também com o conhecimento técnico por parte do utilizador para operar um nó de um sistema descentralizado. Estas necessidades incluem: *i*) a correta configuração e arranque de novos nós num sistema já em operação, nomeadamente quando uma configuração inicial fixa é inadequada para juntar um nó a um sistema em operação; *ii*) preocupações de segurança, desde gestão de identidades e material criptográfico à necessidade de canais seguros, autenticação entre nós, e uso de primitivas criptográficas; e *iii*) mecanismos que permitam de forma automática modificar o comportamento do sistema perante mudanças significativas no seu ambiente de execução (i.e., gestão autonómica), por exemplo, modificar parâmetros associados à operação de protocolos descentralizados quando o número total de nós no sistema aumenta ou reduz de forma significativa.

Neste artigo endereçamos estes desafios emergentes através de um conjunto de técnicas que podem ser utilizadas por diferentes protocolos e sistemas descentralizados. De forma a demonstrar a viabilidade destas técnicas adicionámos

suporte para as mesmas numa nova versão da framework Babel enriquecida com suporte nativo para endereçar estes desafios. Para além disso, modificámos alguns protocolos anteriormente implementados recorrendo ao Babel para extrair benefício destas técnicas, de forma a podermos validar o impacto do uso das mesmas numa simples aplicação de exemplo. A nossa avaliação experimental mostra que é simples integrar estes mecanismos em protocolos existentes e que o custo adicional de execução é aceitável considerando os benefícios trazidos.

O resto deste artigo está organizado da seguinte forma. Na secção 2 apresentamos uma breve introdução à framework Babel. A secção 3 discute brevemente o trabalho relacionado. Na secção 4 apresentamos as nossas propostas para fornecer suporte a mecanismos para a simplificação da configuração automática de protocolos e sistemas descentralizados, suporte a mecanismos de segurança e mecanismos de gestão autonómica, discutindo a sua implementação no Babel. A secção 5 apresenta o nosso ambiente experimental e resultados experimentais. Concluímos o artigo na secção 6.

2 Preliminares: A Framework Babel

O Babel [12] é uma *framework* Java concebida para simplificar o processo de prototipar e implementar protocolos e sistemas distribuídos. Esta *framework* permite que os protocolos nela desenvolvidos sejam executados num contexto real, afastando-se desta forma dos simuladores utilizados no passado para prototipar e avaliar protocolos distribuídos.

De modo a permitir que o programador se possa focar em desenvolver o seu protocolo ou sistema distribuído o Babel abstrai e gere aspetos de baixo níveis inerentes ao desenvolvimento destes sistemas, nomeadamente, aspetos de comunicação entre protocolos, gestão de temporizadores e concorrência intra e inter-protocolo.

Esta abstração de aspetos de baixo nível é oferecida ao programador em parte através um modelo de programação baseado em eventos, semelhante à forma como os protocolos são descritos na literatura, em que cada protocolo possuí a sua própria fila de eventos onde o Babel coloca eventos relativos a temporizadores, eventos de rede e mensagens vindas de outros protocolos, sendo apenas necessário que os programadores que utilizem a *framework* definam a função de *callback* para cada tipo de evento.

De forma a abstrair certos aspetos de comunicação, o Babel oferece algo a que chama de *channels* com os quais os protocolos podem interagir através de uma interface concisa e que oferecem diferentes garantias e capacidades dependendo do seu tipo, sendo um exemplo destes o *TCPChannel* que permite estabelecer conexões TCP. Cada protocolo poderá utilizar o número de *channels* que necessitar, e é possível a partilha de *channels* entre vários protocolos.

O facto de cada protocolo possuir a sua própria *thread*, permite a um único processo de Babel ser responsável por vários protocolos que executam em simultâneo e gerir aspetos de concorrência protegendo o programador de anomalias que derivam da concorrência.

3 Trabalho Relacionado

Nesta secção discutimos brevemente contribuições existentes relacionados com o objetivo deste trabalho. Dividimos esta discussão em quatro aspetos complementares relacionados com protocolos e sistemas descentralizados, frameworks de desenvolvimento, auto-configuração, segurança e gestão autonómica.

Framework de desenvolvimento para sistemas e protocolos descentralizados. O libp2p [15] é talvez a solução que mais se aproxima dos objetivos deste trabalho. Esta é uma biblioteca desenvolvida pela Protocol Labs (disponível em várias linguagens) especificamente para suportar o desenvolvimento de aplicações e sistemas descentralizados. Existem aspetos considerados pelo libp2p que interceptam com o nosso trabalho, todas as comunicações são cifradas, os nós possuem uma identidade baseada numa chave publica, e a sua identidade é autenticada no estabelecimento de canais de comunicação. No entanto o libp2p é ele próprio composto por vários protocolos sendo esta biblioteca muito pouco flexível, por exemplo não é possível substituir (trivialmente) a rede sobreposta usa pela biblioteca. Notamos que apesar do libp2p não considerar aspetos como auto-configuração, estes foram implementados pelo IPFS [3] que por sua vez opera sobre o libp2p.

Configuração automática de sistemas descentralizados. Muitos artigos já discutiram propostas diferentes para a configuração automática de sistemas. A descoberta por multicast, difusão (i.e., broadcast) e o uso de DNS já foi abordada para procura de dispositivos IoT [14]. Neste trabalho recorremos ao DNS não apenas para localizar nós que possam facilitar o acesso à rede mas também para obter parâmetros específicos para governar a operação de protocolos descentralizados. Há outras sugestões no campo do IoT descentralizado [7], fazendo uso de uma combinação de difusão em redes locais e DHTs para uma rede alargada. O sistema IPFS [3] explora o uso de mDNS, no entanto a configuração para participar na rede é copiada de um servidor central (ou distribuída com o binário).

Outras técnicas conhecidas incluem o *Bonjour* da Apple que permite a descoberta de serviços e dispositivos na mesma rede através do uso do mDNS e DNS-SD. É um sistema interessante de procura e descoberta, mas que só funciona para serviços e dispositivos singulares. Neste trabalho o nosso focus é mais alargado incluindo a localização de nós que facilitem a entrada de novos elementos no sistema, mas também para obter configurações iniciais, potencialmente de outros nós da rede.

Segurança em sistemas descentralizados. Um dos grandes desafios em sistemas descentralizados é conseguir identificar corretamente e autenticar cada nó face à ausência de uma autoridade central de certificação, que traz centralização e reduz a autonomia do sistema, mitigando as qualidades desejáveis de sistemas descentralizados.

Li et. al [20] estudam e discutem diversos modelos populares de confiança que podem ser adaptados para sistemas descentralizados, como: *i*) usar uma paralela

empenhar o papel de uma autoridade de certificados de forma distribuída; *ii*) adotar um modelo de *Web of Trust* (popularmente utilizado pelo PGP); *iiii*) adotar um modelo de confiança estatística, em que se confia num nó que prova ter a confiança de um determinado numero de outros nós; ou utilizar *Certificateless Public Key Cryptography* em que cada nó cria e gere suas próprias identidades, que são derivadas de suas chaves públicas; Esta última é adotada pelo sistema IPFS.

Sistemas descentralizados autonómicos. Muitas características de sistemas autónomos modernos são analogias diretas aos sistemas autónomos observados na biologia, como o sistema nervoso humano [25]. Esses sistemas biológicos servem de inspiração para a criação de tecnologias que replicam sua eficiência e adaptabilidade. Assim como o sistema nervoso humano coordena e responde a estímulos de maneira autónoma, os sistemas tecnológicos que pretendemos devem ser projetados para operar de forma independente, processando informações e tomando decisões sem a necessidade de intervenção humana constante.

A ideia de um sistema completamente autónomo prende-se então em obter propriedades, para além da auto-configuração, como *recuperação automática de falhas*, que é fornecido pela maioria de sistemas descentralizados [25,19,26]; *Auto-proteção* que permite a um sistema identificar e reagir de forma automática a uma ameaça; e *reconfiguração automática* em que o sistema muda o ser comportamento, nomeadamente parâmetros da sua configuração, em reacção a mudanças ao seu ambiente de execução (como carga no sistema ou número de nós) [22]. Uma forma tradicional de obter este último comportamento é recorrendo à arquitetura MAPE-K [16], onde se monitoriza a operação do sistema e baseado nessas métricas e conhecimento de domínio, se aplicação operação de reconfiguração. Tipicamente este modelo assume um modele de reconfiguração com elevada coordenação, que não é adequada a sistemas descentralizados.

4 Evolução Arquitetural do Babel

Nesta secção discutimos as técnicas que adicionamos a framework babel para fornecer as características discutidas anteriormente. Apesar do focus deste trabalho ser sobre sistemas descentralizados, notamos que estas técnicas são agnósticas ao tipo de arquitetura distribuída e por isso podem ser aplicadas aos componentes de qualquer sistema distribuído.

4.1 Configuração automática

A configuração automática apresenta várias dimensões. No contexto de sistemas descentralizados, que tipicamente recorrem a uma rede sobreposta para gerir a filiação do sistema [26,19], um problema frequente quando um novo nó se junta à rede é identificar um *contacto*, um nó já presente no sistema, que possa auxiliar o novo nó a juntar-se. Para além disso, outro desafio fundamental é a obtenção de uma configuração inicial para o nó que se junta a rede. Apesar de se poder recorrer a uma configuração por defeito, em sistemas complexos isso pode não ser suficiente, visto que o sistema poderá ter evoluído e essa configuração já

não ser adequada. Endereçamos estes desafios de vários formas que passamos a descrever.

Contacto Explorámos duas formas complementares de fornecer suporte no Babel para localizar um nó de contacto numa rede local: por *broadcast* e por *multicast*. Ambos trazem as suas vantagens e desvantagens. No uso da procura com *multicast*, é possível criar grupos separados por meio dos endereços IP multicast, ou seja, um grupo pode ter atribuído um IP multicast e outro grupo ter outro IP multicast atribuído. No entanto, nem todas as redes e interfaces de rede suportam pacotes multicast, limitando o seu uso.

Criámos dois protocolos de descoberta que integramos no núcleo do Babel, sendo que o programador pode definir outros utilizando uma interface definida para esse fim através da classe abstrata apresentada no listagem 1.1. A escolha da estratégia a usar é feita através de configuração bastando para isso indicar o nome da classe java que implementa o mecanismos de descoberta. Ao iniciar, o Babel instancia e executa esse protocolo recorrendo a reflexão. De forma a indicar ao núcleo do Babel que um protocolo requer um contacto, este deve estender um classe abstrata apresentada na Listagem 1.2. Esta última requer que o protocolo implemente um pequeno conjunto de métodos que servem para, respetivamente, iniciar a execução do protocolo quando este tem informação necessária para iniciar; indicar ao núcleo do babel se o protocolo está pronto a iniciar e se precisa de um contact; receber um contacto e expor o contacto atual.

Listing 1.1: Classe abstrata para serviços de descoberta

```
public abstract class
    DiscoveryProtocol extends
    GenericProtocol {
    public abstract void
        registerProtocol
        (DiscoverableProtocol
             dcProto);
    public abstract void
        uponRequestDiscovery
        (RequestDiscovery
            request, short
            sourceProtocol);
}
```

Listing 1.2: Classe abstrata para protocolos que desejem procurar um contacto caso não o tenham

```
public abstract class
    DiscoverableProtocol
    extends GenericProtocol {
    private Host myself;
    public abstract void
        start();
    public abstract boolean
        readyToStart();
    public abstract boolean
        needsDiscovery();
    public abstract void
        addContact(Host host);
    public abstract Host
        getContact();
}
```

Neste caso, o Babel irá automaticamente registar todos os protocolos que estendam esta classe e não tenham contacto. É esperado também que todas as implementações do *DiscoveryProtocol* iniciem a execução do protocolo assim que este estiver pronto e acrescentem os contactos à medida que os descobre.

4.2 Definição dos parâmetros

Depois de encontrado um contacto (se necessário), é preciso verificar se há uma configuração adequada. Os parâmetros para os protocolos estão definidos como variáveis de instância das classes. Para indicar ao Babel que há parâmetros que podem ser auto configurados (obtendo essa configuração de outro nó ou de um registo DNS), foi criada uma classe abstrata que estende a classe *Discoverable-Protocol* que identifica um protocolo deste tipo. Na implementação é necessário anotar todas as variáveis com um *@AutoConfigureParameter* que se desejam ter uma configuração da rede. A framework ainda espera que todos os parâmetros anotados tenham um setter e um getter associado com um nome definido e acordado. Estes getters e setters são obtidos por meio de *Reflection*.

Além desta nova classe, foi criada a abstração para a criação de um protocolo para procura de configurações em rede. A framework tratará de chamar estes métodos abstratos para fornecer a informação necessária ao protocolo do que procurar e depois os getters e os setters correspondentes.

O Babel, quando vai iniciar um protocolo, vai verificar se o protocolo estende esta classe. Caso estenda, vai verificar o valor de cada parâmetros chamando o respetivo getter. Caso o getter devolva null, ele vai passar este parâmetro para o *addProtocolParameterToConfigure*, caso contrário, passa o parâmetro para o *addProtocolParameterConfigured*. É esperado que todos os getters e setters devolvam e aceitem Strings. Feito estes passos, o *SelfConfigurationProtocol* escolhido deverá tratar do resto, incluindo arranque dos protocolos caso estes recebam uma configuração completa válida. A seguir, identificamos as implementações realizadas:

Cópia e verificação: Este mecanismo contact um nó ja presente e ativo na rede e copia cada um dos seus parâmetros de forma automática.

DNS: Este mecanismo obtém as entradas TXT de um registo DNS (indicado como parâmetro) e analisa essas entradas para extrair os valores dos parâmetros. As entradas TXT devem ter o formato parâmetro=valor.

4.3 Mecanismos de Segurança

Geração de identidades e de confiança Como referido anteriormente, a falta de uma autoridade central em sistemas descentralizados é um grande desafio no seu desenvolvimento e sem uma solução definitiva. Embora existam modelos de confiança descentralizados mais populares do que outros, todos exigem sacrifícios e a melhor escolha depende do sistema a ser construído. Por isso foi preciso encontrar um equilíbrio entre minimizar o trabalho do utilizador e limitar sua liberdade sobre que tipo de modelo de confiança implementar e como.

Para isso fornecemos uma implementação predefinida do modelo da criptografia de chave pública com certificados auto-assinados referida anteriormente na secção 3. Escolhemos esta por ser uma solução popular, utilizada com sucesso por sistemas como os baseados no libp2p [15] e cria o suporte fundamental para ferramentas de gestão de identidades e confiança definidas pelo utilizador. O utilizador que deseje substituir o modelo de confiança padrão pode especificar num ficheiro de configuração as classes que são invocadas para gerir e verificar identidades dinamicamente.

Canais seguros Ao adicionar suporte para a gestão de identidades e de confiança, passa a ser possível criar canais seguros autenticados na camada de rede do Babel. Estes canais usam o gestor de identidades do Babel para que possam utilizar as identidades especificadas na autenticação. Os canais também consultam o gestor de confiança para que possam verificar, de acordo com as políticas padrão ou pelas definidas pelo programador, se deve aceitar a conexão com base no certificado recebido e identidade anunciada pelo outro nó no *handshake* inicial.

Primitivas Criptográficas Outra funcionalidade fornecida pela componente de segurança do Babel é gestão em runtime de material criptográfico como chaves privadas com certificados associados usados para identificação, certificados de nós confiados e chaves simétricas. Cada uma dessas categorias é guardada em um par de KeyStores Java diferente, com cada par consistindo em uma KeyStore efémera que existe apenas na memória do programa, e uma persistente que é guardada em disco. Quando são acedidas pela primeira vez, as KeyStores são carregadas/criadas e o utilizador acede-as direta ou indiretamente através da API da componente de segurança do Babel, podendo, por exemplo, requisitar um objeto capaz de fazer operações criptográficas com uma interface simples e direta em cima de uma chave privada ou secreta específica, como fazer assinaturas, MACs, e cifras de blocos. Também existem facilitadores para gerar segredos aleatoriamente ou a partir de uma palavra passe. A API fornecida recorre extensivamente às funcionalidades da *Java Cryptography Architecture* com o suporte estendido das bibliotecas *Bouncy Castle*[5].

4.4 Gestão autonómica

Sendo o objetivo auxiliar a implementação de sistemas descentralizados com gestão autonómica ao nível dos protocolos que os compõem, ajudando nas suas reconfigurações em tempo de execução, focamo-nos na mudança de parâmetros, numéricos ou não, que é um tipo de reconfiguração mais simples, no entanto não trivial de resolver, que pode impactar bastante a performance de um nó, e por sua vez do sistema como um todo. Alguns exemplos incluem a alteração do número de vizinhos mantidos por um nó ao nível de um protocolo de filiação, ou a alteração do *fanout* de um protocolo de disseminação por rumor *(gossip)*. Estas alterações podem ser despoletadas localmente, com valores calculados pelo próprio nó dada a sua perceção local da rede, ou pelos vizinhos do nó que visam convergir para uma configuração comum global.

Foram então implementadas duas novas classes abstratas no babel, *AdaptiveProtocol* e *AdaptiveMembershipProtocol*, que fornecem ao protocolo mecanismos de reconfiguração de parâmetros identificados pela anotação do Java *@Adaptive*. Estas classes obrigam o progamador a definir métodos de alteração de parâmetros que são automáticamente identificados e executado aquando de um pedido genérico de reconfiguração, local ou remoto, que é feito por um controlador externo. Estas abstrações são bastante suportadas pelos mecanismos de ***Reflexão*** do Java. Apesar da reflexão ter um custo de performance inerente à sua utilização, foi feito um esforço para que este mecanismo fosse apenas usado no *bootstrap* dos protocolos, pelo que o uso não afeta de forma significativa o funcionamento do protocolo em condições normais ou em momentos de reconfiguração

Controlador Autonómico e Interface Para permitir a reconfiguração de parâmetros de protocolos em tempo de execução, foi criado um controlador autonómico que é responsável por monitorizar o estado do sistema e decidir se, e quando, deve reconfigurar um protocolo. O controlador autonómico é instanciado e executado pelo núcleo do Babel, e aguarda por anúncios de métricas de suporte à decisão que são enviadas por protocolos dedicados a esse fim, seguindo uma interface definida por mensagens e pedidos predefinidos, podendo estas ainda serem estendidas caso necessário.

O controlador autonómico pode ser configurado para reagir a diferentes métricas, e para cada métrica, o programador pode definir um conjunto de regras de alto nível que determinam se e como o protocolo deve ser reconfigurado. O programador deverá ter atenção à ordem de dependência entre os parâmetros adaptáveis nas regras que o próprio criou, de forma a garantir que as reconfigurações são feitas de forma consistente. No entanto, neste trabalho apenas consideramos reconfigurações locais que não requerem coordenação com outros nós.

Coleção e Estimativa de Métricas de Suporte à Decisão Para que o controlador autonómico possa tomar decisões informadas, é necessário que as métricas de suporte à decisão sejam estimadas e recolhidas. Sendo uma das mais importantes dessas métricas o número de nós na rede, ou mais especificamente o diâmetro da rede, que pode ser usado para ajustar o *fanout* de um protocolo de disseminação por rumor, por exemplo. Assim, foi implementado um protocolo baseado no método Random Tour [21]. Este protocolo é executado periodicamente em paralelo com os protocolos da aplicação por todos os nós, conseguindo estimar o tamanho da rede com uma margem de erro aceitável e as suas estimativas são enviadas para o controlador autonómico.

5 Avaliação Experimental

Nesta secção descrevemos a nossa metodologia de avaliação da solução proposta. Para esse fim desenvolvemos uma aplicação de teste simples, mas que combina a operação de três protocolos distribuídos. Avaliamos o esforço de desenvolvimento contando o número de linhas adicionais para tirar partido das abstrações fornecidas no contexto de um protocolo de filiação. De seguida verificamos que o desempenho da aplicação não é afetado de forma significativa pelo uso dos novos mecanismos, e finalmente avaliamos o consumo de recursos (CPU e RAM) desta aplicação simples (baseline) e quando tirando partido de cada um dos mecanismos propostos.

5.1 Aplicação de Teste

Consideremos por exemplo uma aplicação brinquedo mencionada no início deste artigo em que o objetivo é a disseminação de mensagens entre um grande número de utilizadores, potencialmente dinâmico, e sem recorrer a qualquer plataforma centralizada. Uma aplicação destas pode ser construída recorrendo a protocolos de rumor (do inglês *gossip*) [4,17] de forma a que os vários participantes da rede

cooperem entre si para difundir estas mensagens. Usamos ainda um protocolo de filiação baseado em redes sobrepostas não estruturadas HyParVIew [19] e de forma a recuperar mensagens transmitidas durante períodos transientes de desconexão, implementamos um protocolo simples de anti-entropia, que guarda e retransmite mensagens por um período de 10 minutos. Nas nossas experiências, cada nó da rede envia uma mensagem com mais de 63000 bytes a cade dez segundos. Sobre a aplicação e protocolos base fizemos as seguintes modificações para explorar o uso de cada uma das metodologias indicadas acima.

Variante com auto-configuração: Nesta variante modificámos o protocolo de filiação HyParView para obter o contacto por multicast e para obter os seus parâmetros de operação (num total de sete diferentes parâmetros) por registos DNS TXT (no host: `hyparview.leitao.rocks`).

Variante segura: Na variante segura recorremos a um canal de comunicação que autentica os participantes quando uma ligação TCP é criada, recorrendo a identidades dos nós em certificados auto-assinado (sendo que quando este é desconhecido os nós trocam os seus certificados durante o handshake), todas as mensagens trocadas têm a integridade protegida por um MAC, e finalmente as mensagens enviadas pelo protocolo de rumor são ainda assinadas criptográficamente.

Variante com gestão autonómica: Nesta variante modificámos o protocolo HyParView de forma a poder mudar o seu número de vizinhos dinamicamente, e o protocolo de rumor de forma a mudar o seu *fanout*. Estas modificações são guiadas por um controlador local que opera com base em estimativas do tamanho da rede produzidas por uma implementação do protocolo Random Tour.

5.2 Ambiente experimental

Executámos experiencias recorrendo a containers dockers, que instrumentamos de forma a obter valores de latência entre os nós de acordo com uma distribuição realista num sistema global. Executámos as nossas experiencias num grupo de servidores recorrendo ao docker-swarm. Os nossos resultados capturam múltiplas execuções independentes de cada experiência. Realizámos experiências com 25, 50, e 100 nós. Recorremos a uma feature do Babel que nos permitiu instrumentar os protocolos de forma a coletarmos o número de mensagens transmitidas por eles. Como configuração de base, cada nó tenta obter 5 vizinhos e usa um fanout de 4. O protocolo de anti-entropia recorre a bloom filters configurados para uma taxa de falsos positivos abaixo de 0.001%.

Em cada experiência, os nós foram iniciados sequencialmente com um intervalo de 30 segundos e começaram a transmitir mensagens aplicacionais assim que iniciaram. Após todos os nós terem iniciado, o sistema ficou em execução por 2 horas. finalmente os nós foram desativados sequencialmente. Durante a desativação, cada nó para imediatamente de transmitir mensagens aplicacionais, mas permanece ativo e a receber mensagens por 10 minutos.

Nós	Versão	Mensagens Totais	Latência (ms)	Fiabilidade
100	Baseline	87287	188185.1354	0.9886
	Self-Config	87159	178072.0071	0.9774
	Adaptive	87457	187639.8497	0.9876
50	Baseline	39827	58944.6533	0.9975
	Self-Config	39833	56779.1417	0.9979
	Adaptive	39815	54474.0753	0.9985
25	Baseline	18955	15411.4852	0.9999
	Self-Config	18937	14939.3678	1.0
	Adaptive	18909	15865.0191	1.0

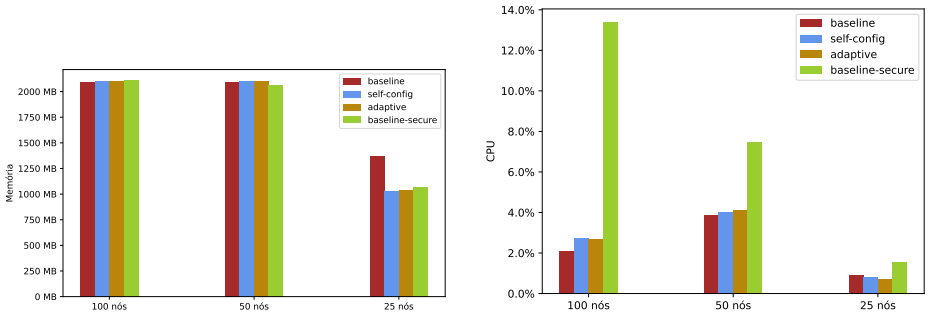
Tabela 1: Média de mensagens enviadas, latência e fiabilidade utilizando os diferentes mecanismos para a mesma aplicação.

5.3 Resultados Experimentais

Complexidade de Código/Usabilidade De forma a quantificar o esforço em usar as novas funcionalidades implementadas no Babel, avaliamos o número de linhas de código adicionais para fazer as alterações a uma implementação do protocolo HyParView (originalmente com 504 linhas) como descritas anteriormente. De forma a suportar a auto-configuração, foram necessárias 180 linhas de código adicionais. Para tirar partido dos mecanismos de segurança foram adicionadas 63 linhas de código. E finalmente, o suporte à auto-gestão necessitou de 35 linhas de código adicionais (neste processo contabilizámos anotações java como correspondendo a uma linha de código). O número de linhas adicionais é relativamente baixo o que indica um esforço relativamente baixo de implementação. A maior diferença encontra-se no caso do suporte à auto configuração, que necessitou da implementação de um conjunto de métodos adicionais como explicado anteriormente.

Impacto no desempenho na aplicação. A tabela 1 reporta um conjunto de indicadores de desempenho, incluindo, o número total de mensagens aplicacionais disseminadas, a latência média (que corresponde ao tempo entre a transmissão de uma mensagem pela aplicação e a entrega dessa mensagem à camada aplicacional mais tardia) e a fiabilidade que corresponde à taxa de entrega. Podemos observar que conforme o sistema aumenta de dimensão mais mensagens são transmitidas, o que é natural visto que nestas experiencias todos os nós transmitem mensagens a um ritmo constante. Naturalmente a latência da disseminação aumenta com o tamanho do sistema, devido às mensagens terem de cruzar um caminho maior entre dois nós. No entanto notamos que os valores apresentados entre as várias soluções são semelhantes, evidenciando que as nossas técnicas não afetam a correção do sistema.

Duas notas relevantes, a fiabilidade não é completa nos sistemas de maiores dimensões pois, quando o último nó entra na rede, o protocolo de anti-entropia já removeu as primeiras mensagens transmitidas da sua cache. Podia ser expectável que a aplicação que usa mecanismo de auto-gestão pudesse apresentar um melhor desempenho, no entanto para dimensões de sistema relativamente



(a) RAM (b) CPU

Figura 1: Consumo de recursos dos vários mecanismos para a mesma aplicação.

pequenas, a variabilidade de latência deve mascarar qualquer pequena melhoria no desempenho.

Consumo de recursos A tabela 1 mostra o consumo de recursos (médio) das várias soluções no meio da experiencia em termos de CPU (fração do CPU) e de memória consumida para as várias alternativas comparada com uma solução que não recorre a nenhum mecanismo reportado neste artigo. Em termos de RAM o consumo entre todas as alternativas é semelhante quando consideramos sistemas com a mesma dimensão. O que indica que o custo adicional de memória dos diferentes mecanismos é desprezável (o aumento de consumo de RAM com o tamanho do sistema deve-se à cache do protocolo de anti-entropia). No entanto em relação a CPU podemos observar um pequeno aumento em relação ao baseline para os mecanismos de auto-configuração e auto-gestão e um aumento muito significativo para a variante com segurança. Isto deve-se naturalmente ao custo computacional das primitivas criptográficas utilizadas.

6 Conclusão

Neste artigo discutimos um conjunto de funcionalidades avançadas que consideramos relevantes para suportar o desenvolvimento de novas aplicações descentralizadas. Estas funcionalidades endereçam necessidades de segurança, auto-configuração e auto-gestão. Para endereçar as dificuldades que estas necessidades representam desenvolvemos vários mecanismos que integrámos na framework Babel. A nossa avaliação experimental usou uma aplicação simples que foi adaptada para tirar partido de todos os mecanismos, e que revelou que os mesmos funcionavam como se esperava, sendo que a análise da quantidade de código adicional necessária para os usar estes mecanismos revelou um custo adicional baixo. As nossas experiências mostram ainda que os mecanismos propostos não trazem nenhum impacto negativo sobre o desempenho de aplicações, sendo que o seu custo adicional em termos de memória e CPU é desprezável a menos do uso adicional de CPU quase se usa mecanismos de segurança baseados em criptografia.

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. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the thirteenth EuroSys conference. pp. 1–15 (2018)
2. Baumann, A., Fabian, B., Lischke, M.: Exploring the bitcoin network. WEBIST (1) **2014**(369-374), 3 (2014)
3. Benet, J.: Ipfs - content addressed, versioned, p2p file system (7 2014), https://arxiv.org/abs/1407.3561v1
4. Birman, K.P., Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M., Minsky, Y.: Bimodal multicast. ACM Transactions on Computer Systems (TOCS) **17**(2), 41–88 (1999)
5. of the Bouncy Castle Inc., L.: Bouncy castle open-source cryptographic apis. https://www.bouncycastle.org/
6. Carvalho, N., Pereira, J., Oliveira, R., Rodrigues, L.: Emergent structure in unstructured epidemic multicast. In: 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07). pp. 481–490. IEEE (2007)
7. Cirani, S., Davoli, L., Ferrari, G., Léone, R., Medagliani, P., Picone, M., Veltri, L.: A scalable and self-configuring architecture for service discovery in the internet of things. IEEE Internet of Things Journal **1**(5), 508–521 (2014). https://doi.org/10.1109/JIOT.2014.2358296
8. Costa, P.Á., Fouto, P., Leitao, J.: Overlay networks for edge management. In: 2020 IEEE 19th International Symposium on Network Computing and Applications (NCA). pp. 1–10. IEEE (2020)
9. Costa, P.Á., Leitão, J., Psaras, Y.: Studying the workload of a fully decentralized web3 system: Ipfs. In: IFIP International Conference on Distributed Applications and Interoperable Systems. pp. 20–36. Springer (2023)
10. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic algorithms for replicated database maintenance. In: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing. pp. 1–12 (1987)
11. Dingledine, R., Mathewson, N., Syverson, P.F., et al.: Tor: The second-generation onion router. In: USENIX security symposium. vol. 4, pp. 303–320 (2004)
12. Fouto, P., Costa, P.Á., Preguiça, N., Leitão, J.: Babel: A framework for developing performant and dependable distributed protocols. In: 2022 41st International Symposium on Reliable Distributed Systems (SRDS). pp. 146–155. IEEE (2022)
13. 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)
14. Jara, A.J., Martinez-Julia, P., Skarmeta, A.: Light-weight multicast dns and dns-sd (lmdns-sd): Ipv6-based resource and service discovery for the web of things. In: 2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. pp. 731–738 (2012). https://doi.org/10.1109/IMIS.2012.200
15. labs, P.: libp2p. https://libp2p.io/, (Accessed on 06/11/2023)
16. Lara, E., Aguilar, L., Sanchez, M.A., García, J.A.: Adaptive security based on MAPE-K: A survey. Applied Decision-Making: Applications in Computer Sciences and Engineering pp. 157–183 (2019)

17. Leitão, J.: Gossip-Based Broadcast Protocols. Master's thesis, Faculdade de Ciências da Universidade de Lisboa (May 2007)
18. Leitao, J., Pereira, J., Rodrigues, L.: Epidemic broadcast trees. In: 2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007). pp. 301–310. IEEE (2007)
19. Leitao, J., Pereira, J., Rodrigues, L.: Hyparview: A membership protocol for reliable gossip-based broadcast. In: 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07). pp. 419–429. IEEE (2007)
20. Li, Z., Xu, X., Shi, L., Liu, J., Liang, C.: Authentication in peer-to-peer network: Survey and research directions. In: 2009 Third International Conference on Network and System Security. pp. 115–122 (2009). https://doi.org/10.1109/NSS.2009.30
21. Massoulié, L., Merrer, E., Kermarrec, A.M., Ganesh, A.: Peer counting and sampling in overlay networks. pp. 123–132 (07 2006). https://doi.org/10.1145/1146381.1146402
22. Rosa, L., Rodrigues, L., Lopes, A.: Appia to r-appia: refactoring a protocol composition framework for dynamic reconfiguration (2007)
23. Tikhomirov, S.: Ethereum: state of knowledge and research perspectives. In: Foundations and Practice of Security: 10th International Symposium, FPS 2017, Nancy, France, October 23-25, 2017, Revised Selected Papers 10. pp. 206–221. Springer (2018)
24. Van Renesse, R., Dumitriu, D., Gough, V., Thomas, C.: Efficient reconciliation and flow control for anti-entropy protocols. In: proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware. pp. 1–7 (2008)
25. Van Roy, P., Haridi, S., Reinefeld, A., Stefani, J.B., Yap, R., Coupaye, T.: Self management for large-scale distributed systems: An overview of the selfman project. vol. 5382, pp. 153–178 (10 2007). https://doi.org/10.1007/978-3-540-92188-2_7
26. Voulgaris, S., Gavidia, D., Van Steen, M.: Cyclon: Inexpensive membership management for unstructured p2p overlays. Journal of Network and systems Management **13**, 197–217 (2005)