Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Georges Kovacs Ribeiro

Utilização da Arquitetura Peer-to-Peer Para Marketplaces

São Paulo
2018

Georges Kovacs Ribeiro

Utilização da Arquitetura Peer-to-Peer Para Marketplaces

Dissertação de Mestrado apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT, como parte dos requisitos para a obtenção do título de Mestre em Engenharia da Computação.

Data de aprovação: ___/___/_____

Prof. Dr. Marcelo Novaes de Rezende (Orientador)
FIPT – Fundação Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Membros da Banca Examinadora:

Prof. Dr. Marcelo Novaes de Rezende (Orientador)
FIPT – Fundação Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Prof. Dr. Plínio Roberto Souza Vilela (Membro)
UNICAMP – Universidade Estadual de Campinas

Prof. Dr. Alexandre José Barbieri de Sousa (Membro)
Mestrado Engenharia da Computação

Georges Kovacs Ribeiro

Utilização da Arquitetura Peer-to-Peer Para Marketplaces

Dissertação de Mestrado apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT, como parte dos requisitos para a obtenção do título de Mestre em Engenharia da Computação.
Área de concentração: Engenharia de Software

Orientador: Dr. Marcelo Novaes de Rezende

São Paulo
Julho/2018

Ficha Catalográfica
Elaborada pelo Departamento de Acervo e Informação Tecnológica – DAIT
do Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT

R484u Ribeiro, Georges Kovacs
Utilização da arquitetura peer-to-peer para marketplaces. / Georges Kovacs Ribeiro. São Paulo, 2018.
91p.

Dissertação (Mestrado em Engenharia de Computação) - Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Área de concentração: Engenharia de Software.

Orientador: Prof. Dr. Marcelo Novaes de Rezende

1. Arquitetura de marketplace 2. Arquitetura peer-to-peer 3. WebRTC Web - Real Time Communications 4. BATON - Balanced Tree Structure for Peer-to-Peer Networks 5. OGP - Overlay Gateway Protocol 6. Blockstack. 7. Tese I. Resende, Marcelo Novaes de, orient. II. IPT. Coordenadoria de Ensino Tecnológico III. Título

18-86 CDU 004.41(043)

Bibliotecária responsável: Maria Darci Cornellas Narciso – CRB 8/3569

Dedicatória

Este trabalho é dedicado à memória da minha querida avó
Magda Kovacs

AGRADECIMENTOS

Ao Prof. Dr. Marcelo Novaes de Rezende pela sua paciência, sabedoria e experiência. Seus apontamentos tornaram este trabalho possível.

Aos professores Dr. Alexandre José Barbieri de Sousa e Dr. Plínio Roberto Souza Vilela por sugestões imprescindíveis à primeira versão do trabalho.

Aos secretários Mary Yoshioka Pires de Toledo e Adilson Feliciano Nascimento, que ajudaram em todas as questões burocráticas.

À minha esposa, Larissa Mayumi Sano, por estar sempre ao meu lado, sempre me dando força para conseguir atingir meus objetivos. Suas sugestões tornaram o trabalho muito mais inteligível.

“A thing isn’t quite real until you name it.“ (Jordan B. Peterson)

RESUMO

Os principais sistemas de *marketplace* utilizam, tipicamente, a arquitetura cliente-servidor. Essa arquitetura tem como característica centralizar toda a responsabilidade para um dado número de servidores, que fornecem seus recursos computacionais para os usuários do sistema. Por isso, a alocação desses recursos deve ser previamente disponibilizada, e a centralização dificulta o acesso ao sistema geograficamente. Já em uma arquitetura *peer-to-peer*, esses problemas podem ser contornados, já que sua natureza permite a escala e replicação do sistema, pois cada nó age tanto como provedor, quanto como consumidor de recursos, e, dada a sua distribuição, o nó também age como um ponto de distribuição geográfico. Neste trabalho é proposta uma arquitetura de *marketplace* baseada em *peer-to-peer*. Além da arquitetura *peer-to-peer*, é apresentada uma implementação referencial para exemplificação de todos seus componentes, demonstrando como é proposto o funcionamento de um *marketplace peer-to-peer*, baseando-se na tecnologia WebRTC e nas redes *overlays* BATON, OGP e *Blockstack*. A implementação referencial atendeu aos requisitos funcionais elencados e também aos requisitos não funcionais de performance e disponibilidade.

Palavras-chave: Marketplace; P2P; Arquitetura P2P; WebRTC; OGP; Blockstack; BATON.

ABSTRACT

Peer-to-Peer Architecture For Marketplaces

The major marketplace systems typically use the client-server architecture. The client-server architecture has the characteristic of centralizing all responsibility for a given number of servers, which provide its computational resources to system users. Therefore, the allocation of these resources must be previously made available, and centralization makes it difficult to access the system geographically. In a peer-to-peer architecture, these problems can be circumvented, since their nature allows scaling and replication of the system, since each node acts both as a provider and as a consumer of resources, and it's nodes are geographically distributed. In this work, a peer-to-peer-based marketplace architecture is proposed. In addition to the peer-to-peer architecture, a referential implementation is presented for exemplifying all of its components, demonstrating how a peer-to-peer marketplace, based on WebRTC technology and the overlays BATON, OGP and Blockstack networks. The referential implementation met the functional requirements listed and also the non-functional requirements of performance and availability .

Key-words: Marketplace; P2P; Arquitetura P2P; WebRTC; OGP; Blockstack.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BATON	Balanced Tree Structure for Peer-to-Peer Networks
BTC	Bitcoin
DoS	Denial-of-Service
DHT	Distributed Hash Table
OGP	Overlay Gateway Protocol
P2P	Peer-to-Peer
NAT	Network Address Translation
WebRTC	Web Real Time Communications

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

As atividades pertinentes a *e-commerce* e *marketplaces* já são parte integrante do cotidiano, o que é evidenciado pelo fato de existirem milhares de transações online ocorrendo a cada segundo. Por exemplo, o valor total de bens vendidos pelo *marketplace* eBay em 2016 ultrapassa os 80 bilhões de dólares (SHANG et al., 2017). Isso corresponde a 2 mil dólares por segundo. Um *marketplace* bem sucedido tem de encarar o desafio de criar um sistema que processe esse grande número de transações e também deve armazenar uma grande quantidade de dados. Geralmente, para solucionar esses problemas e satisfazer os usuários que demandam serviços de alta qualidade, empresas de *marketplace* criam um sistema cliente-servidor com diversos servidores e uma rede robusta. Essa arquitetura cliente-servidor com servidores centrais é a mais comum entre as aplicações, ou serviços, de *marketplace* (ALAMI; RODRÍGUEZ; JANSEN, 2015).

Embora a arquitetura cliente-servidor forneça uma base sólida para um grande número de usuários, com baixo custo de uso, manutenção e segurança, é difícil que ela supra todos os requisitos das aplicações de *marketplace*. A quantidade de produtos sendo vendida online está crescendo rapidamente e cada produto pode ter mais de uma transação de compra ou venda. A diversidade geográfica dos usuários que utilizam essas plataformas também produz um impacto na distribuição de conteúdo. Esses desafios evidenciam a dificuldade que qualquer sistema de *marketplace* tem de enfrentar. Um sistema de *marketplace* em uma arquitetura de servidores centrais deve absorver esses desafios e contorná-los por meio de uma arquitetura preparada pra tanto. A arquitetura cliente-servidor pode estar sujeita a falhas devido a sua centralização.

Uma alternativa que surgiu nos últimos anos é a utilização de serviços de nuvem. Eles possuem recursos computacionais sob demanda, que têm elasticidade rápida, ou seja, a aquisição de novos recursos, como, por exemplo, mais banda na rede ou mais armazenamento, é feita na medida da necessidade da aplicação. Além disso, os recursos podem ser facilmente descartados. Com tal facilidade, o modelo de cobrança geralmente é feito pelo uso. Tais recursos são compartilhados entre vários clientes, o que permite um uso eficiente. Ademais, estão distribuídos pelo planeta, o que permite o seu acesso por um ponto mais próximo, com os mecanismos já existentes de rede.

No entanto, para se utilizar um serviço de nuvem, a aplicação deve estar obrigatoriamente na Internet e há um custo relacionado a cada incremento na sua utilização. Já em um serviço *peer-to-peer*, cada novo usuário incrementa também a

quantidade de recursos disponíveis. Além disso, em uma arquitetura *peer-to-peer*, não há a dependência da Internet, o que pode ser importante em ambientes sem acesso à Internet ou hostis à comunicação livre na Internet, como ocorrido, por exemplo, nos governos da Líbia e Egito em 2011, quando o acesso à Internet foi desligado em diversas ocasiões entre janeiro e março daquele ano (DAINOTTI et al., 2011).

Sistemas *peer-to-peer* (P2P) têm sido usados para distribuir infraestrutura, para a distribuição de dados e para se obter a descentralização. Comparando a arquitetura cliente-servidor para o mesmo fim, os sistemas P2P conseguem suportar mais usuários, com menor custo. Muitas técnicas têm sido desenvolvidas para tecnologias P2P. A maioria delas tem potencial para ser utilizada comercialmente. Por exemplo, Chord (STOICA et al., 2001) tem sido amplamente adotado por projetos que necessitam de uma rede P2P na camada de aplicação, por implementar um DHT (*Distributed Hash Table*), pode ser utilizado com um índice distribuído de nós e arquivos.

O histórico do P2P possui mais de quarenta anos, começou com arquiteturas distribuídas, como, por exemplo, a Usenet, desenvolvida pela Duke University. Desde a década de 1990, redes com estruturas P2P começaram a ser utilizadas e desenvolvidas por diversas empresas, tais como a Boeing e a Intel. A própria concepção da Internet foi originalmente pensada como um sistema P2P. Os primeiros computadores ligados à ARPANET eram considerados nós com a mesma responsabilidade ao invés de adotarem papéis de cliente ou servidor, os servidores de nomes (DNS) estão em uma estrutura P2P até hoje. No entanto, quando a conexão à Internet via modem tornou-se amplamente utilizada e a disponibilidade de recursos na rede tornou-se assimétrica, passou a ser necessária a introdução dos papéis de cliente e servidor.

Em 1999, uma estrutura P2P chamou atenção, o Napster, que foi originalmente desenvolvido como um serviço P2P de compartilhamento de arquivos pela Internet, e, para isso, introduziu técnicas que permitiam o compartilhamento de áudio digital. Essa técnica envolvia um servidor central, que continha informações de metadados (nome do arquivo, data da criação, qualidade do arquivo) dos arquivos e sobre quem era o nó que disponibilizava essa informação, também havia informação da qualidade de conexão do nó. Por isso, apesar de a transferência de arquivos ser P2P, o sistema ainda era centralizado, o que permitiu seu fácil fechamento por uma ordem de justiça em 2001. Em seu pico, o Napster chegou a ter cerca de 26 milhões de usuários.

Em 2000, o projeto Gnutella, desenvolvido por Tom Pepper e Justin Frankel dentro da companhia Nullsoft, que fora adquirida recentemente pela AOL, introduziu a primeira rede P2P descentralizada. Diferentemente do Napster, o Gnutella não necessitava de um controle central, a própria rede se organizava, o que dificultou seu fechamento. Isso é evidenciado pelo fato de que, após o cancelamento do projeto Gnutella pela AOL, a rede P2P que suportava o Gnutella continuou existindo. Além

disso, devido a uma engenharia reversa do protocolo, outras versões do Gnutella puderam ser criadas para que o protocolo melhorasse e sua rede crescesse. (BERKES, 2003)

Essa descentralização do Gnutella é possível devido à introdução da rede *overlay*, uma camada acima das camadas padrões de rede, introduzida pelo protocolo Gnutella, que tem o intuito de providenciar os recursos necessários para que a rede P2P funcione. Esse protocolo disponibiliza o funcionamento de busca de nós, busca e transferência de arquivos de maneira descentralizada.

Outro proeminente sistema P2P de compartilhamento de arquivos é o Freenet, uma implementação de código aberto de um sistema descrito por Ian Clarke em 1999. O Freenet difere do Gnutella no fato de que seu principal intuito é a criação de uma rede global não censurável de armazenamento de informações. E, por isso, sua arquitetura foi desenvolvida levando em consideração o anonimato e a tolerância a falhas. (BERKES, 2003)

Outros sistemas proprietários, como Kazaa e Morpheus, introduziram o conceito de super nós, isto é, nós que se tornam responsáveis, dinamicamente, por uma parcela da rede, indexando e fazendo cache da parcela da rede das quais esse nó (ou super nó) é responsável. Não existe documentação detalhada de como esses sistemas operavam. (CHAUDHARY, 2014)

O protocolo BitTorrent, criado por Bram Cohen, em 2001, é um sistema P2P que tem como foco a distribuição de grandes arquivos. Para tanto, os nós pertencentes à rede tentam transferir um único arquivo, compartilhando partes desses arquivos. Para se juntar a essa rede, os usuários necessitam se conectar a um *tracker*, um servidor central que age como um diretório que lista os nós interessados nesse mesmo arquivo, e, assim, eles compartilham essas partes, dando mais prioridade para as mais raras na rede. O *tracker* pode ser substituído por outro sistema P2P, uma DHT, que guarda essa lista de nós. O BitTorrent é muito popular, chegou ter cerca de 150 milhões de usuários em 2014, mas hoje em dia seu uso está diminuindo, devido a serviços de streaming, como Netflix.

Além de compartilhamento de dados, existem sistemas P2P para *streaming* de vídeo, como o PPLive ou PPStream, popular na China (SILVERSTON et al., 2009). Essas aplicações combinam tecnologias P2P e de Internet TV para transmissão de vídeos ao vivo na Internet.

Também existem exemplos de voz sobre IP (VoIP), como o Skype, que combina a transmissão de voz a uma arquitetura híbrida de super nós. Mas, em 2012, o Skype deixou de ser P2P, devido à heterogeneidade dos dispositivos e conexões, dada a entrada de dispositivos móveis em redes 3G ou 4G.

Na área de comércio existem as moedas virtuais, ou criptomoedas, como o Bitcoin (NAKAMOTO, 2008), que é um sistema descentralizado, P2P, de pagamento, baseado em criptografia, em especial no algoritmo SHA-256. Os pagamentos são feitos gerando transações que transferem a moeda do Bitcoin, chamada de *"Bitcoin coins"* (BTCs), entre dois usuários da rede. Os usuários participam da rede utilizando pseudônimos virtuais, denominados endereços Bitcoin. Costumeiramente, um usuário possui centenas de endereços bitcoins, que são guardados em uma carteira virtual. Cada endereço é relacionado a uma função de transformação, que está atrelada a um par de chaves (pública e privada). Esses pares são utilizados para transferir BTCs entre os endereços. Todas as transações são publicadas na rede do Bitcoin e só são validadas caso os outros integrantes da rede P2P considerem-nas válidas.

Existem outras moedas virtuais, como o Litecoin, que utiliza como base para a criptografia o algoritmo Scrypt. Várias criptomoedas derivam dos algoritmos Scrypt e SHA-256, mas possuem pequenas diferenças, como a disponibilidade total de moedas ou pequenas mudanças na arquitetura P2P, o que ocorre com Feathercoin, Terracoin, P2PCoin, BitBar, ChinaCoin e BBQCoin.

O Bitcoin e suas ideias congêneres geraram também um modelo para outras ideias no mundo P2P. Como o voto pela Internet (KUBJAS, 2017) ou a assinatura de documentos (WAN; DENG; LEE, 2015).

Tais desenvolvimentos de sistemas P2P culminaram na existência de uma rede chamada *overlay*, cuja característica central consiste em uma arquitetura distribuída e eventualmente descentralizada, definida pelo protocolo da aplicação. Nessa arquitetura, os nós, isto é, os elementos pertencentes a essa rede, agem tanto como consumidores quanto como provedores de serviços e recursos, e todo o comportamento é definido pelo modo de comunicação. Dessarte, elimina-se o papel de servidor, e todos os pertencentes à rede podem fazer uso do mesmo software para utilizar o sistema. Essas redes podem ser classificadas como centralizadas, como o Napster, ou descentralizadas, tais como a Freenet. Na categoria descentralizada, a arquitetura P2P pode ser classificada sob dois pontos de vista, pelo grau de descentralização e pela topologia da rede *overlay*, cada categorização apresentando diversos modelos P2P. Essa gama de tecnologias propõe um desafio para a criação de um sistema P2P. É necessário investigar as características da aplicação para escolher qual tipo de arquitetura deve ser empregado.

Dentre as redes *overlays* estruturadas, o Chord (STOICA et al., 2001) é o mais popular (MALATRAS, 2015) e resolve o problema de localizar nós na rede P2P, utilizando DHT. Em particular, ele utiliza **hashing consistente** (KARGER et al., 1997) para associar chaves a nós e dados, de modo que cada nó é responsável por um número aproximadamente igual de chaves. O Chord é resistente a *churn* (intermitência

da participação de nós), é totalmente descentralizado e distribuído. Apesar de ser resistente a *churn*, foi notado que, caso ocorra um número massivo de *churn*, a performance medida de acordo com o número de mensagens extras trafegadas de uma rede P2P com N nós cai de $O(log(n))$ para $O((log(n))^2)$, isso se dá devido ao tempo que leva para o Chord atualizar a tabela de roteamento depois da saída e entrada de diversos nós. Para melhorar esse tempo, surgiu o BATON (JAGADISH; OOI; VU, 2005), que tem o custo amortizado de $O(log(n))$, mesmo sob *churn*. Essa rede *overlay* utiliza árvores binárias balanceadas, na qual cada nó da árvore é mantido por um participante da rede.

A utilização de P2P não se restringe a aplicações *standalone*. Com o surgimento do WebRTC, uma tecnologia que funciona nos navegadores e originalmente serve para comunicação de vídeo e áudio P2P, pode-se utilizar uma parte da especificação que trata de um canal de dados. Já existem diversos experimentos bem sucedidos que introduzem redes P2P de distribuição de conteúdo via *Browser*, como o de Voigt et al. (2013).

Apesar dos benefícios das arquiteturas P2P, existem diversos empecilhos que restringem sua adoção. A intermitência da participação de nós (*churn*), que é voluntária, dificulta a disponibilidade dos recursos e aumenta os custos para manter a rede *overlay* (STUTZBACH; REJAIE; SEN, 2008). A escala de um sistema P2P e a desconfiança de seus participantes prejudica os mecanismos de descoberta de recursos (DUMITRIU et al., 2005). A natureza autônoma dos sistemas P2P torna a segurança um desafio, deixando-os vulneráveis a ataques de DoS (*Denial-of-Service*) e *Sybil* (quando há a entrada de um número muito grande de nós maliciosos com o mesmo objetivo) (DOUCEUR, 2002). A dinâmica de mudança de nós e padrões de busca díspares podem introduzir uma carga desigual em um grupo de nós e, consequentemente, aumentar o tempo de resposta .

Um fato corriqueiro de muitas aplicações P2P é seu tempo de vida curto ou a mudança de arquitetura para cliente-servidor. O tempo de vida curto muitas vezes é devido a questões legais, como no caso do Napster. Já a mudança de arquitetura ocorre para suportar uma gama de clientes bem heterogêneos, como visto, isso ocorreu com o Skype e com o Dropbox. Sendo assim, parece haver um limite tanto na arquitetura quanto na reflexão do sistema na sociedade para a existência de aplicações P2P.

Mesmo com toda a evolução dos sistemas P2P e de *marketplaces* supra mencionada, ainda não foi encontrado um trabalho apresentando uma arquitetura de um sistema P2P totalmente descentralizada, onde usuários sejam parte da rede P2P, que possa, de forma prática, executar as atividades padrões de *e-commerce*. Existem propostas centralizadas como a de (WANG, 2014), que utilizam aspectos de P2P para *load balancing*, mas, na realidade, a aplicação continua existindo em um ponto

centralizado. Além disso, não foi encontrada uma análise das características dessa potencial plataforma descentralizada e seus componentes.

1.2 Objetivo

Neste trabalho, o objetivo principal foi propor uma arquitetura P2P de um *marketplace* que possibilite usuários, sejam eles indivíduos ou organizações, a fazer contribuições a esse sistema, por meio de fornecimento de recursos computacionais, contrastando com os sistemas tradicionais, nos quais os recursos são centralizados, e, por isso, sujeitos a pontos de falha. A arquitetura proposta atendeu aos requisitos funcionais vindos da literatura.

Como objetivo secundário, foi desenvolvida uma implementação referencial da arquitetura, que foi usada como exemplificação da proposta e na validação dos requisitos.

1.3 Contribuição

O trabalho de Vogt, Werner e Schmidt (2013) expandiu o uso da tecnologia WebRTC para a distribuição de conteúdo via P2P. Eles utilizaram o WebRTC Data Channels e criaram uma rede *overlay* P2P estruturada, isto é, baseada em DHT, e mostraram como utilizar essa tecnologia para implementar arquiteturas P2P utilizando os principais navegadores da atualidade. Este trabalho estende a técnica utilizada por Vogt, Werner e Schmidt (2013), combinando o trabalho de Jagadish (2005), criador de um *overlay* estruturado, que utiliza árvores binárias, cuja principal vantagem é o maior desempenho, mesmo quando a rede P2P está com *churn* alto. A utilização desses trabalhos permitiu a esta dissertação apresentar uma arquitetura de *marketplace* P2P em navegadores baseada em requisitos funcionais básicos.

Como contribuição à indústria, foi desenvolvida uma implementação referencial da arquitetura, com objetivo de ser expandida a um *marketplace* P2P totalmente funcional.

1.4 Método de trabalho

Nesta dissertação, foi desenvolvida uma arquitetura de *marketplace* P2P. Para o usuário, foi transparente a utilização desse sistema. Seguem as principais atividades que foram desempenhadas:

1. **Revisão Bibliográfica**: Foi feita a revisão bibliográfica com foco em arquiteturas para *marketplace*, sistemas baseados em P2P e requisitos de sistemas de *marketplace*.

2. **Preparação dos requisitos**: Com base na atividade anterior, revisão bibliográfica, foram definidos os requisitos da arquitetura a ser proposta. Foram propostos requisitos funcionais e não funcionais.

3. **Proposta da arquitetura**: Elaborar uma arquitetura de *marketplace* que atendesse aos requisitos definidos e fosse uma arquitetura P2P. O desenvolvimento dessa arquitetura foi guiado pela criação do protocolo de comunicação P2P, seguido de algumas escolhas técnicas baseadas na revisão bibliográfica.

    - **Proposta da arquitetura - Descrição dos principais componentes**: Nesta fase, foram descritos os principais componentes da arquitetura em uma visão mais geral, descrevendo o papel e função de cada componente. Foram descritos os componentes da rede de distribuição de conteúdo, a integração com uma rede P2P de cooperação e como se deu a integração com a rede externa.
    - **Proposta da arquitetura - Descrição detalhada dos componentes**: Após apresentação da visão geral, foram analisados cada componente e o comportamento deles dentro da arquitetura.

4. **Implementação referencial**: Desenvolver uma implementação referencial da proposta da arquitetura.

    - **Implementação referencial - Adaptação da tecnologia WebRTC**: Seguindo a fase anterior, foi implementada a expansão da distribuição de conteúdo via WebRTC para se utilizar BATON e *overlay* OGP (NGO, 2013).
    - **Implementação referencial - Desenvolvimento**: Nesta fase, a implementação referencial foi desenvolvida atentando-se para o fato de que a arquitetura foi baseada em WebRTC, podendo rodar nos principais navegadores da atualidade.

5. **Avaliação da arquitetura proposta**: Com a arquitetura e a implementação referencial prontas, foi verificado o atendimento aos requisitos. Também foram avaliadas as considerações finais, o futuro dessa arquitetura e como ela poderia ser melhorada. A arquitetura foi avaliada de duas formas, primeiro, como uma arquitetura de *marketplace*, verificando se ela atendia aos requisitos definidos, depois, como uma arquitetura P2P, notando as mesmas métricas de um protocolo P2P. Como referência, foi utilizado o trabalho de Jagadish (2005), no qual foram analisadas as métricas: custo de *churn* e custo de inserção.

1.5 Organização do Trabalho

Na seção 2, Revisão Bibliográfica e Estado da Arte, há a definição, as características e as aplicações de P2P. Nela também foi feita uma revisão da evolução dos sistemas P2P e suas diferentes classificações e foram apresentadas as principais evoluções nos sistemas P2P, considerando as diversas formas de redes *overlay*, estruturada e não estruturada. Também foram citados os principais avanços nas arquiteturas de *marketplace*.

Na seção 3, Proposta da Arquitetura, foram definidos os requisitos para o funcionamento do *marketplace* P2P, foi abordada a adaptação do trabalho de Voigt (2013) de WebRTC, foram descritos os principais componentes da arquitetura e, posteriormente, foi apresentada uma descrição detalhada deles e do protocolo de comunicação do *marketplace*.

Na seção 4, Implementação Referencial, há a descrição de uma implementação protótipa da arquitetura proposta, apontando as dificuldades, desafios e as tecnologias existentes e futuras, que permitem um desenvolvimento mais fiel do que foi proposto. Nessa seção também estão os testes de aderência aos requisitos e os testes selecionados na rede P2P, de acordo com Jagadish (2005).

Na seção 5, Conclusão, há um balanço da introdução da arquitetura P2P para *marketplace* e foram indicadas possibilidades futuras de uso e pesquisa.

2 REVISÃO BIBLIOGRÁFICA

2.1 Introdução

Esta seção está dividida em duas partes, na primeira, são apresentados os principais conceitos de P2P, serão vistos aspectos como as diversas definições de P2P e suas principais tecnologias. Também serão apontados os principais conceitos de *e-commerce*. Em Estado da Arte, são apresentados os principais avanços em P2P e arquiteturas inovadoras de *e-commerce*, são esses avanços que permitiram a existência deste trabalho.

2.2 Principais Conceitos

A arquitetura proposta foi construída utilizando um sistema P2P, cuja rede *overlay* tenha escalabilidade, tolerância a falhas e boa performance. Nesta seção, serão vistas as principais redes *overlay* de P2P.

2.2.1 Definições de P2P

Antes de mencionar P2P (*Peer-to-Peer*), é necessário definir o termo e como ele é utilizado no presente trabalho. Existem muitas definções que norteiam as mesmas características. Seguem algumas delas:

A IETTF/RFC 5694 (CAMARILLO et al., 2009), define P2P como um sistema em que seus elementos, também chamados de nós, compartilham recursos com o objetivo de prover algum serviço, de modo que os elementos provêm e consomem serviços de outros elementos. Essa definição, como descrito pelos autores, tem a intenção de ser a mais abrangente possível, o que pode incluir sistemas com serviços distribuídos, mas não necessariamente tem esses serviços descentralizados, já que cada elemento do sistema pode prover um serviço cliente-servidor.

Já Schollmeier (2002), define P2P como: "Uma arquitetura de rede distribuída pode ser chamada de rede P2P, se os participantes compartilham parte de seus recursos computacionais (processamento, armazenamento ou banda). Esses recursos compartilhados são necessários para que o serviço ou conteúdo oferecido na rede esteja disponível. Esses serviços ou recursos podem ser acessados diretamente, sem que haja uma entidade intermediadora. Os participantes, então, dessa rede, são consumidores e provedores de recursos".

Uma outra definição (ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004), que inclui a caracterização anterior, a simetria, define sistemas P2P como aqueles que

possuem as seguintes características, sintetizadas por ROUSSOPOULOS et al. (2004):

- **Auto-organizados**: Os nós possuem organização própria no processo de busca de outro nó. Não possuem diretório central de busca.
- **Simétricos**: Os nós são semelhantes, todos consomem e oferecem serviços.
- **Controle descentralizado**: Os nós definem o nível de participação na rede. Eles coordenam suas próprias ações.

Além dessas características, a definição de ANDROUTSELLIS-THEOTOKIS; SPINELLIS (2004) também dá ênfase a:

- **Estabilidade**: O sistema P2P é resistente a falhas. Também deve ficar estável em relação à entrada e saída de nós no sistema.
- **Compartilhamento de recursos**: Serviços são providos por algum nó e são compartilhados os recursos de hardware (ciclos de CPU, capacidade de armazenamento, banda de rede).

Essas características, acima definidas, são as consideradas por este trabalho como sendo a definição de P2P.

Um nó **A** é dito **vizinho** de um nó **B**, caso o nó **B** tenha possibilidade de estabelecer contato com o nó **A**.

2.2.2 História do P2P

Em 1969, a ARPANET, a rede precursora da Internet, foi concebida como um sistema P2P para o compartilhamento de recursos computacionais nos Estados Unidos. Seus primeiros participantes, universidades e órgãos governamentais, eram conectados como pares iguais.

Inicialmente, a Internet era aberta e sem a necessidade de *firewalls*. Os pacotes trafegavam sem intervenção. Já existiam aplicações cliente-servidor, como FTP ou Telnet, mas elas estavam disponíveis de maneira simétrica, em todos os pertecentes à rede. Por outro lado, também surgiam serviços P2P, como a USENET e o DNS (*Domain Name System)*.

A USENET foi desenvolvida em 1979 para compartilhamento de informação na comunidade Unix. Uma máquina podia chamar outra e trocar arquivos, baseado no protocolo de cópia entre computadores com Unix, o UUCP (*Unix-to-Unix copy protocol*). Na USENET os usuários também podiam compartilhar mensagens e publicações, similarmente aos fóruns de Internet, utilizando o NNTP (*Network News Transfer Protocol)*.

A importância da USENET se dá na maneira descentralizada de controle, sendo uma rede precursora dos sistemas P2P atuais.

Em 1983, o DNS foi feito para distribuir informações de *hostname* dos IPs na Internet. O DNS combina uma rede P2P com uma arquitetura hierárquica de posse de informação, dividida por seções. Cada seção é responsável por uma gama de *hostname*, e quando é necessária uma informação de uma seção que está acima na hierarquia, é feita uma requisição. Sendo assim, conforme sua posição, os servidores DNS, ou servidores de nomes, agem tanto como servidores e clientes. Esses conceitos, como a distribuição de responsabilidade sobre os dados, o fato de um servidor também agir como cliente e a comunicação entre pares, são características das redes P2P atuais.

2.2.2.1 Arquitetura P2P

As aplicações P2P possuem algumas características arquiteturais em comum, pois o sistema P2P advém da definição dada. Essas características independem do serviço prestado pelo sistema, elas são comuns aos sistemas P2P. (CAMARILLO et al., 2009):

- **Ingresso**: Os nós precisam ingressar no sistema P2P, portanto, é necessária uma funcionalidade de autorização e autenticação, pois o sistema P2P deve ser *auto-organizado* e *estável*.
- **Descoberta**: O sistema P2P precisa de um mecanismo de descoberta de nós para que eles se comuniquem entre si. Uma vez que o sistema P2P deve ser *simétrico*, o serviços precisam se achar para consumir e prover seus serviços, conseguindo *compartilhar os recursos*.

Outras características em comum podem depender da funcionalidade do sistema, pois um pode estar interessado em distribuir dados, e outro, só em processar informações ou fazer cálculos. São elas:

- **Busca e armazenamento de dados**: Os dados de um sistema P2P podem ser buscados, indexados e armazenados.
- **Processamento**: Certos nós em um sistema P2P podem processar dados e/ou fazer cálculos.
- **Mensageria**: O sistema P2P pode implementar um protocolo em comum de mensageria.

Essas funcionalidades podem ser agrupadas em uma arquitetura de sistema P2P de três camadas (NGO, 2013):

- **Rede subjacente**: A rede de comunicação que roteia pacotes de dados entre dois nós (TCP/IP, UDP).
- **Rede *overlay* P2P**: Implementa as referidas características arquiteturais em comum, ficando responsável por implementar a parcela que caracteriza o sistema como P2P.
- **Aplicação**: Responsável por implementar as funcionalidades específicas da aplicação, utilizando a Rede P2P.

É vantajosa essa visão, pois permite analisar separadamente a rede P2P, podendo classificar de acordo com a organização da distribuição dos nós.

2.2.3 Sistemas P2P

Como visto, um sistema P2P é composto por seus elementos, chamados de nós. Cada nó tem uma lista de endereço de outros nós, esses são os nós vizinhos. Uma das principais funções de um sistema P2P é localizar recursos, dada a requisição de um nó participante. Esses recursos estão distribuídos entre os nós desse sistema.

Essa atividade de localização se dá por uma busca distribuída no sistema. O primeiro passo da busca distribuída é a criação da requisição de busca, cada uma dessas requisições possui um identificador. Os algoritmos de busca distribuída são determinados pela estrutura da rede *overlay*. Por exemplo, um nó em uma rede desestruturada sempre divulga a requisição de busca para todos seus nós vizinhos. Como se nota, um algoritmo desses produz um grande tráfego na rede. Por outro lado, um nó em um sistema P2P estruturado envia a requisição de busca para os vizinhos que satisfazem uma condição previamente conhecida a respeito do identificador da requisição de busca. Como é de se esperar, esse tipo de busca performa melhor e usa menos banda da rede.

Existem sistemas P2P hieráquicos, estruturados, desestruturados ou híbridos.

2.2.3.1 Sistemas P2P Estruturados

Os sistemas P2P estruturados possuem uma arquitetura que utiliza uma DHT (*Distributed Hash Table*) para armazenar uma tabela que contém os nós, e a informação de qual recurso computacional cada nó possui. Essa tabela é armazenada, de forma distribuída, em cada nó do sistema P2P. Esse método reduz o custo de busca e minimiza o número de mensagens trafegadas.

As principais implementações, como Chord, Kademlia e Pastry, utilizam o conceito de tabela de hash distribuída (**DHT** - Distributed Hash Table). Nesse conceito, cada elemento dessa estrutura é representado por um par de **chave** e **valor**, sendo que a chave é única para toda a rede, e qualquer nó da rede pode buscar valor de um determinado par contendo uma dada chave. A implementação deve ditar a organização que os nós devem ter entre si para atingir a unicidade das chaves e a distribuição dos pares.

A principal vantagem dessa rede é a habilidade de, mesmo para conteúdo impopular, garantir um certo número de troca de mensagens na rede, assegurando escalabilidade. No entanto, como desvantagem, essa rede é menos robusta sob *churn*. Ademais, a garantia de limite na troca de mensagens não leva em conta que a rede subjacente pode forçar um roteamento mais ineficiente, por exemplo, um nó pode ser adjacente na rede *overlay*, mas geograficamente distante.

2.2.3.1.1 Sistemas P2P desestruturados

Os dados armazenados em cada nó também não seguem nenhuma regra, cada nó decide o que é armazenado e de que modo.

As redes *overlay* P2P caracterizam-se pelo processo aleatório de aquisição de nós vizinhos (STUTZBACH; REJAIE; SEN, 2008), o que randomiza a topologia da rede. Os nós vizinhos podem, por exemplo, ser adquiridos por um mecanismo externo, como uma lista em um *website*, o que ocorre no sistema *Overnet*. A lista de nós vizinhos pode ser revista a cada reinicialização do nó para a verificação de entrada e saída de nós na rede.

A busca de dados e nós, geralmente, é feita por *flooding* (a mensagem de busca é enviada para todos os nós vizinhos), busca aleatória no grafo da rede ou pelo algoritmo *hill climbing*.

Esse tipo de rede possui as seguintes vantagens:

- Como o conteúdo é local, suporta buscas mais complexas, a depender de cada nó.
- A rede comporta alta taxa de entrada e saída dos nós, devido à falta de estrutura lógica, o que torna a saída e entrada de nós facilitada.

Como desvantagem, a falta de estrutura lógica da rede impacta na escalabilidade, uma vez que a busca por dados e nós torna-se mais complexa à medida que a rede cresce.

Os principais sistemas que apresentam rede *overlay* P2P desestruturada são: Gnutella (KLINGBERG; MANFREDI, 2002), Freenet (CLARKE et al., 2001), Overnet/e-Donkey2000, Kazaa e BitTorrent.

2.2.3.1.2 Rede *overlay* P2P hierárquica

Nesse tipo de rede P2P, os nós não possuem todos o mesmo papel. Existem responsabilidades diferentes para cada nó. Geralmente existem os que são referidos como super nós, que têm mais recursos disponíveis e também ficam mais tempo na rede, e há os nós comuns, que se conectam a esses super nós. A versão 0.6 do Gnutella é um exemplo desse tipo de rede.

2.2.3.1.3 Chord

Chord é um sistema P2P estruturado que utiliza uma rede *overlay* com topologia circular. Nele, os endereços dos nós, assim como os recursos computacionais, que são geralmente dados, são obtidos por uma função *hash*. Com esse método, cada nó ou dado tem um identificador no espaço da imagem da função *hash*. A imagem da função deve ser grande o suficiente para garantir uma relação um-a-um de forma probabilística, isto é, cada nó ou dado deve ter um identificador único a menos de uma probabilidade negligenciável.

Na topologia circular do Chord, os nós são dispostos em um anel por ordem de seu identificador (o endereço do nó após a função *hash*). Cada nó possui um nó sucessor e um antecessor. Um nó guarda em si os dados e suas respectivas chaves dadas pela função *hash*, mas com um detalhe, os nós e seus dados utilizam *hashing* consistente, isto é, o nó somente guarda os dados cujos identificadores estão, segundo a ordem desse espaço, entre o identificador do nó antecessor e seu próprio identificador.

2.2.3.1.4 BATON

Os nós nessa rede *overlay* estruturada estão dispostos em uma estrutura de dados de árvore. Cada nó tem uma única posição lógica na árvore. Essa posição é identificada por um nível, começando de 0, e um identificador (ID), que vai de 1 a 2^l, em que l é o nível, sendo assim, cada nível pode comportar no máximo 2^l nós. Existem quatro tipos de conexões:

- Conexão de nó pai: É a conexão entre um nó pai e um nó filho.
- Conexão de nó filho: Ainda é uma conexão relativa ao nó pai, no entanto, essa conexão provê informações lógicas e físicas de todos seus nós filhos.

- Conexões presentes na tabela de roteamento: A tabela de roteamaneto armazena a informação de cada conexão, incluindo as informações lógicas e físicas. Essa tabela também armazena informações dos nós vizinhos, que são os nós com ID menor que 2^l.
- Conexão de nó adjacente: Um nó tem conexões também com seus vizinhos. Como os nós podem entrar e sair da rede, essas conexões mudam dinamicamente.

Para entrar na rede BATON, um nó precisa conhecer ao menos o endereço de um nó pertecente a ela. Para alocar o nó entrante na sua nova posição, os nós replicam a mensagem na rede *overlay*. Quando um nó p recebe uma mensagem com o pedido de entrada na rede do nó entrante q, o nó p primeiro checa se pode alocar o nó entrante q como um de seus filhos, caso contrário, p envia essa mesma mensagem para o nó pai. Quando um nó pai aceita o nó entrante q como um de seus filhos, ele divide a responsabilidade entre os nós no mesmo nível.

2.2.3.1.5 Tipos de chamadas

Em uma rede P2P, existem duas formas principais de busca por recursos. A primeira é a busca exata, nessa forma, o nó tem previamente o identificador do objeto ou recurso que deseja, manda uma mensagem na rede e obtém como resposta o contato responsável por esse objeto. Essa chamada apresenta as seguintes vantagens:

- Cada objeto possui um identificador único;
- Protocolo pode garantir resultados, caso o objeto exista;
- Como cada objeto tem um identificador único, é possível atribuí-los de maneira eficiente.

E as seguintes desvantagens:

- É necessário saber o identificador único do objeto;
- É necessário manter uma estrutura para endereçar os nós.

A outra forma de procurar objetos na rede P2P é por busca textual. A busca textual consiste em enviar uma mensagem com uma sequência de palavras e obter como resultado uma lista de nós que possuem objetos relacionados ao nome buscado. Esse método possui como principal vantagem a não obrigação de existência de um identificador único, porém, é difícil esse tipo de chamada ser eficiente, já que a busca é exaustiva no pior caso.

2.2.4 Bootstrap

O processo de Bootstrap refere-se ao método de entrada de um nó na rede P2P, tem como fim a descoberta dos nós na rede P2P pelo nó entrante e pode depender de um mediador que tenha conhecimento dos nós entrantes e dos nós da rede. Mas esse mediador não é inteiramente obrigatório, já que existem outras maneiras de se atingir esse objetivo em redes P2P, como, por exemplo, mandar mensagens aleatoriamente na rede.

2.2.5 WebRTC

WebRTC é uma API em desenvolvimento elaborada pela *World Wide Web Consortium* (W3C) e pelo *Internet Engineering Task Force* (IETF) para permitir aos navegadores executar aplicações de chamada telefônica, video chat e compartilhamento P2P sem a necessidade de plugins. Sendo assim, essa tecnologia permite uma comunicação direta em tempo real entre navegadores. O fato de ser direto é o pricipal diferencial dessa tecnologia, para comunicação indireta, com um sevidor mediando, pode-se utilizar a tecnologia *WebSockets*.

O WebRTC introduz protocolos, padrões e uma API de Javascript que são otimizados para dar suporte à transferência de áudio e vídeo em uma maneira P2P. Além de oferecer essas tecnologias, também é oferecido o que é chamado de *data channel*, ou canal de dados, um canal de comunicação que oferece comunicação de dados arbitrários. Esse canal tem o potencial de mudar os padrões de comunicação entre navegadores, já que existe a possibilidade de mudar a topologia para algo mais centrado em P2P. Isso permitirá maior liberdade na engenharia de tráfego das aplicações. Esses canais de dados são parte de um complexo protocolo, envolvendo sessões, chaves e gerenciamento de conexão e, por isso, é importante estudá-lo detalhadamente.

Uma limitação de serviços P2P é o fato de que muitas máquinas estão atrás de NAT (*Network Address Translation*), que filtra e previne comunicação direta P2P. E, dessarte, o padrão WebRTC utiliza o protocolo ICE (*Interactive Connectivity Establishment*), que tem dois tipos de servidores: *Session Traversal Utilities for NAT* (STUN) e *Traversal Using Relays around NAT* (TURN).

O ICE é o protocolo que permite ao WebRTC a conexão entre pares, ele atinge esse objetivo informando ao cliente os acessos do servidor de STUN e TURN. Primeiro, tenta-se conectar por STUN, o servidor de STUN basicamente informa ao cliente qual é seu IP público, e assim é feita a primeira tentativa de conexão entre os pares. Caso não haja sucesso, a conexão será estabelecida pelo servidor TURN, que transmite ativamente todos os dados entre os pares, a conexão não é totalmente P2P. Existem

servidores TURN nos protocolos TCP, UDP e também TLS sobre TCP, o último serve para tentar passar por certos *firewalls*.

O próximo passo é a sinalização por meio do protocolo SDP (*Session Description Protocol*). Essa sinalização começa por meio da transferência de dados das capacidades do navegador que está querendo se conectar. Esses dados são chamados de oferta, e ela não tem como ser enviada ao navegador da outra ponta sem um intermediário. Logo, sempre que se deseje fazer uma comunicação P2P via WebRTC, é necessário transferir essas ofertas. O padrão não estabelece um meio oficial para trocar essa informação, e, portanto, deve-se criar esse serviço.

2.3 Estado da Arte

Esta seção contém os principais avanços acadêmicos que permitem construir uma aplicação P2P utilizando *WebRTC*.

2.3.1 Rede P2P por WebRTC

O trabalho de Vogt et al. (2013) estabeleceu um modo de formar uma rede P2P via WebRTC. Esse trabalho estabeleceu os seguintes objetivos:

- Criar uma arquitetura cuja implementação permita aos browsers formar uma rede P2P;
- A biblioteca gerada deve ser facilmente utilizável, podendo ser simplesmente adicionada a qualquer arquitetura de *site*;
- A biblioteca deve somente utilizar APIs disponíveis nos principais navegadores, para que possa ser utilizada de forma independente.
- A comunicação deve ocorrer por meio dos canais de dados do WebRTC;
- A parcela de código no servidor deve ser a mínima possível;

A arquitetura pode ser visualizada em 4 grandes blocos: Mensageria inicial do WebRTC, conexão com o nó de *Bootstrap*, conexão com a rede P2P e a troca de mensagens.

2.3.1.1 Mensageria inicial do WebRTC

O protocolo escolhido foi o WebRTC, para tanto, deve-se utilizar a mensageria inicial que promove a conexão entre pares. Sendo assim, as decisões a respeito da arquitetura de redes P2P em WebRTC devem levar em conta essas mensagens.

A conexão entre dois navegadores por WebRTC envolve uma negociação elaborada, dividida em vários passos. Essa negociação está descrita no padrão JSEP (*Javascript Session Establishment Protocol*) do IETF, por Uberti et al. (2012). O ponto crucial dessa negociação são as mensagens que contêm a informação do protocolo SDP. O protocolo não estabelece uma maneira oficial de mandar essa informação. O trabalho de Vogt et al. (2013), para resolver essa questão, utilizou um *WebSocket* para a troca da informação SDP.

A utilização do *WebSocket* não é a única possível. É possível utilizar *long pooling* ou até mesmo enviar um arquivo físico com a informação SDP. No entanto, *long pooling* tem de abrir sempre uma conexão HTTP, e, às vezes, fazer todo procedimento de TLS, tendo performance inferior a *WebSocket* (LIU; SUN, 2012).

2.3.1.2 Conexão com o nó de *Bootstrap*

Para adentrar à rede P2P, novos nós estabelecem uma conexão *WebSocket* com um servidor de *Bootstrap*, que já estava conectado a outros nós. Essa conexão é utilizada para a mensageria inicial do WebRTC, resultando em conexão direta entre o novo nó e, pelo menos, um outro nó da rede P2P. Quando o nó entrante estabelece uma conexão com outro nó, ele já pode se desconectar do servidor de *Bootstrap*. Mesmo que não haja conexão direta entre nós, é possível a troca de mensagem entre todos pares, desde que a mensagem seja passada até chegar ao destinatário.

2.3.1.3 Conexão com a rede P2P

Depois de iniciada a conexão com alguns nós da rede P2P, deve-se decidir, baseando-se no tipo de rede *overlay* utilizada, quais são os nós que devem se conectar entre si. Uma forma trivial de resolver esse problema é cada nó se conectar a todos os outros nós, mas claramente essa solução não escala bem. Para tanto, em (Vogt et al.; 2013), foi escolhido utilizar uma rede P2P estruturada, o *Chord*. Essa rede *overlay* provê um crescimento logarítmico, garantindo que a rede possa crescer mais.

Aqui não foi prevista a queda de performance que o algoritmo *Chord* traz quando a rede está sob *churn*. Por isso, neste trabalho, será utilizado a rede BATON.

2.3.1.4 Troca de mensagens

A troca de dados em WebRTC depende da conexão prévia feita pelo ritual de mensageria, portanto, para comunicação entre dois nós, somente há duas alternativas. A primeira consiste em estabelecer todo o ritual de mensageria com o nó com o qual se deseja comunicar. A segunda, é mandar a mensagem através de nó previamente conectado a ambos. Como foi escolhida a utilização de um DHT, a troca de mensagens entre nós não é direta, logo, a segunda opção é mais utilizada.

2.3.1.5 Componentes

Tendo em vista os passos anteriores, foi descrita uma arquitetura dividida em componentes. São eles:

- *Router*: Esse componente é responsável por disseminar mensagens na rede, decidindo quando deve rotear uma mensagem a outro nó. Também recebe mensagens do componente de *Bootstrap* e mantém uma tabela de roteamento.
- *Connection Manager*: Esse componente é responsável pela parte específica de WebRTC, cuidando da abertura canais de dados do WebRTC entre nós da rede P2P. Na fase de *Bootstrap*, esse componente atua junto ao *Router* para ofertar conexões e adicioná-las à tabela de roteamento.
- *Bootstrap*: Esse componente fica apartado dos nós P2P, ele é um servidor que tem duas responsabilidades: servir os documentos estáticos da aplicação (HTML, Javascript e CSS) e agir como um servidor de *WebSocket*, que é utilizado para iniciar a rede, enviando uma oferta de conexão WebRTC para algum nó aleatório, também pode ser utilizado como uma alternativa ao WebRTC.

2.3.1.6 Utilização no trabalho

A inovação deste trabalho está na utilização dessas camadas para prover P2P sobre o *WebRTC*, este é o único trabalho a fazê-lo. Entretanto, só foi testado o Chord. Nesta dissertação, foi adaptado esse sistema de camadas por *PeerManager*, no entanto, foi utilizado o BATON, que possui a vantagem de ter melhor desempenho na atualização das tabelas de roteamento.

2.3.2 Blockstack

O armazenamento de dados e a verificação de posse desses dados de maneira descentralizada foram introduzidos junto às tecnologias de *Blockchain*. Logo após o surgimento do *Bitcoin*. Essa maneira criptograficamente auditável de proceder fez com que surgissem aplicações descentralizadas baseadas na tecnologia de *Blockchain*, uma vez que essas cadeias não possuem ponto central de confiança. Uma dessas tecnologias é o *Namecoin*, que age como uma espécie de DNS sem hierarquia. O *Namecoin* é diretamente baseado nos códigos do *Bitcoin*, adicionando a funcionalidade de registro de pares nomes e valor.

Uma dificuldade desses *Blockchains* com funcionalidades mais complexas é que a construção das máquinas virtuais que rodam esse cima dessas redes são muito acopladas ao próprio *Blockchain*. Logo, caso exista uma falha no *Blockchain*, toda a aplicação descentralizada construída sobre essa tecnologia está fadada a ter os

mesmos problemas. Com isso em mente, Ali et al. (2016) propuseram uma tecnologia, *Blockstack*, que pode rodar em cima de qualquer *Blockchain*, desde que seja mantido o ordenamento total, conforme a teoria dos conjuntos, de suas operações.

Os autores estabeleceram os seguintes objetivos:

- Registro de nomes descentralizado: Os usuários dessa rede P2P devem poder registrar nomes e descobrir recursos atrelados a outros nomes, sem precisar confiar em uma entidade centralizadora;
- Armazenamento descentralizado: Os usuários podem usar essa rede P2P como um sistema de armazenamento descentralizado;
- Performance equivalente: A performance dessa arquitetura deve ser comparável a serviços centralizados.

Essa arquitetura foi concebida para suportar falhas da tecnologia *blockchain*, podendo migrar um sistema entre *blockchains*. Isso permite uma vida longa a sistemas baseados nessa tecnologia. Também é possível a utilização de vários *blockchains* simultaneamente. O *Blockstack* trata os *blockchains* como canais de comunicação que transmitem operações totalmente ordenadas. Sendo assim, pode-se adicionar ou remover esses canais

O *Blockstack* introduz o conceito de *virtualchains*, que são máquinas de estado arbitrárias que rodam em cima de um *blockchain* sem necessitar de modificações. Isto é, a lógica mais complexa fica fora da cadeia, o que permite introduzir de maneira facilitada novas técnicas tanto no nível dos *virtualchains*, quanto no nível do *blockchain* subjacente. Uma outra vantagem é que usuários do *blockchain* não precisam executar programas não confiáveis somente para participar da rede, como ocorre no *Ethereum*.

É introduzida uma nova rede P2P desestruturada, chamada de rede Atlas, que introduz as capacidades de armazenamento de dados pequenos e na qual um índice global é mantido. Cada nó pertencente à rede Atlas mantém um réplica dos dados. Isso se dá para evitar ataques do tipo *Sybil* e escapar do problema de *churn*.

2.3.2.1 Camadas do *Blockstack*

A arquitetura do *Blockstack* é dividida em três camadas, *Blockchain* subjacente, rede P2P e armazenamento. A primeira camada é responsável pelo controle e as outras duas, pelos dados. O controle lida com volumes pequenos de dados e está focado em manter confiança, definindo uma relação entre nomes e recursos de rede. A parte responsável pelos dados contém informações de como um deles pode ser descoberto e como cada um é armazenado. Os dados são replicados, não importando

de qual nó o cliente o obtém, pois os clientes podem, independentemente, verificar se os dados recebidos são corretos ou não.

2.3.2.2 Camada *Blockchain* subjacente

Nessa arquitetura, a camada *Blockchain* subjacente é a camada mais baixa, ela serve a dois propósitos: prover um local para armazenar as operações e prover um consenso da rede P2P da ordem como essas informações foram incluídas. O *virtualchain* codifica as operações dos dados no *blockchain* subjacente. Essa camada encapsula complexidades do *blockchain*, como mineração de blocos, algoritmos de consenso, flutuação da criptomoeda.

Essa camada também inclui novas operações para se adaptar às novas funcionalidades, o *virtualchain*. O funcionamento é equivalente ao funcionamento de uma máquina virtual em uma máquina física. Sendo assim, diferentes tipos de *virtualchains* podem rodar na rede subjacente. As operações do *virtualchain* são codificadas em operações reais da criptomoeda como sendo metadados.

As regras para aceitar ou rejeitar as operações em uma *virtualchain* são específicas, isto é, cada um possui sua própria máquina de estado. Nessa arquitetura, a *virtualchain* tem o propósito de construir uma base de dados que armazena informações em blocos da criptomoeda.

2.3.2.3 Camada da rede P2P

Nessa camada são armazenados os dados chamados de *zonefiles*, arquivos que contêm informação de roteamento, eles são armazenados na rede Atlas. A rede P2P permite o armazenamento desses arquivos apenas se eles forem previamente anunciados na rede *blockchain*. Esse anúncio é feito pelo cálculo *hash* na rede da criptomoeda subjacente. Os arquivos armazenados têm cerca de 4KB, isso introduz um custo pequeno para guardar informações, considerando que o *blockchain* tem geralmente tamanho na ordem de GBs.

2.3.2.4 Camada de Armazenamento

Os dados de interesse são guardados na camada mais ao topo, a camada de armazenamento. Todos os dados são assinados pela chave do proprietário, definida na parte de controle, isto é, na rede *blockchain*. Fazendo isso, essa tecnologia permite armazenar dados arbitrariamente grandes em uma variedade de provedores de serviço, como serviços de cloud. Nessa camada também é implementado o serviço de nomes, BNS (*Blockchain Name System*).

2.3.2.5 Utilização no trabalho

Em uma aplicação *e-commerce* P2P é importante que os dados possam ser armazenados de forma a manter a sua propriedade, mesmo com o *churn* alto de uma rede baseada nos navegadores dos usuários. Utilizar o *blockstack*, permite ao usuário entrar na rede e armazenar dados da loja, produtos e pedidos.

2.3.3 Cooperação em P2P

Um dos avanços em P2P se dá no nível com que os sistemas P2P cooperam entre si. Essa cooperação tem como objetivo aumentar a compatibilidade entre eles para que os recursos compartilhados possam atingir um maior número de nós.

Tal compatibilidade será utilizada para várias redes P2P se comunicarem. Pois, como será proposto, a arquitetura poderá ser distribuída em vários sistemas P2P separados. Esses sistemas todos vão utilizar separadamente uma arquitetura baseada no que foi visto anteriormente.

Essa cooperação pode ser dividida entre as seguintes categorias (NGO, 2013):

- **Cooperação por compartilhamento de recursos**: Os sistemas podem permitir que recursos de um sistema P2P sejam acessíveis em outro sistema P2P.
- **Cooperação por fusão**: Sistemas P2P que tenham o mesmo objetivo podem se fundir, utilizando nós de forma única, eliminando nós que competem entre si, por estarem em sistemas distintos.
- **Roteamento entre sistemas**: Os sistemas P2P envolvidos podem criar um tipo de roteamento entre si para que os nós consigam enviar dados de um sistema a outro.

A cooperação por compartilhamento de recursos ou por fusão aumenta o tamanho do sistema P2P. Já o roteamento entre sistemas tem como objetivo aumentar o desempenho na comunicação entre sistemas P2P. (NGO, 2013)

Para aplicações com um grande número de transações é recomendado utilizar a cooperação por compartilhamento de recursos (NGO, 2013), portanto, este trabalho a utilizará.

2.3.3.1 Cooperação P2P por compartilhamento de recursos

Esse tipo de cooperação foi estudado tanto para redes P2P estruturadas quanto desestruturadas. Para as estruturadas, é utilizada a rede *overlay* P2P para que seja feito um roteamento neste nível. Já para as desestruturadas, como apresentando por Konishi

et. al (2006), é feita uma publicação por *flooding*. Como será utilizado neste trabalho um sistema P2P estruturado, já que eles apresentam melhor desempenho no roteamento (NGO, 2013), serão analisados os trabalhos de cooperação por compartilhamento de recursos para esse tipo de sistema.

O trabalho de Cheng (2007) apresenta uma solução para sistemas P2P estruturados baseados em DHT. Nesse trabalho os nós de um sistema P2P (A) são alocados em outro sistema P2P (B), quando o sistema B não tem algum serviço do sistema A. Essa alocação por demanda introduz uma complexidade desnecessária, pois é preciso monitorar os dois sistemas e ter esse conhecimento. As outras soluções mencionadas não necessitam desse passo extra.

O trabalho de Liquori (2009) apresenta um protocolo, o **Babelchord**, específico para cooperação entre os sistemas P2P Chord. Os nós de um sistema P2P Chord, que estejam também aceitando o Babelchord, agem como se fossem os *gateways* do sistema, a fim de que seja possível acessar alguma outra rede P2P Chord externa. Depois, apresentam o protocolo Synapse, que expande o Babelchord para outros tipos de *overlay* também.

2.3.3.2 Protocolo de cooperação

Como visto, o compartilhamento de recursos entre redes P2P é um problema devido às diferenças entre os diversos tipos de *overlay* P2P. Tendo isso em vista, o trabalho de Ngo (2013) cria o conceito de *super-overlay* na descrição do protocolo OGP (*Overlay Gateway Protocol*). O protocolo OGP é uma extensão do protocolo Kademlia, permitindo roteamento eficiente entre as redes P2P.

Os principais objetivos desse trabalho são:

- Nós nativos em uma rede P2P devem operar normalmente, mesmo na presença de novos protocolos.
- Nós que têm a ciência de outros protocolos podem encontrar recursos em diferentes sistemas.

O conceito de *super-overlay* consiste em uma rede principal onde existem duas formas de participação. A primeira, chamada de completa, é a rede de nós que participam em *overlay* onde se quer cooperar e na rede *super-overlay*. A segunda é composta por participantes das redes que não fazem parte da rede principal, mas podem se comunicar com nós que participam de maneira completa. Dessa forma, a participação completa age como um *gateway* entre redes P2P.

2.3.3.3 Utilização no trabalho

Para criar um *marketplace* P2P pode ser necessária a inclusão de mais uma rede P2P. Além disso, o sistema deve prever introdução de novas tecnologias P2P. Sendo assim, a cooperação entre redes P2P, pelo protocolo OGP, permitiu a este trabalho a utilização de pelo menos duas redes, a primeira definida pelos navegadores, via WebRTC, e a segunda definida pelo *Blockstack*, onde informações poderão ser guardadas.

2.3.4 Marketplaces

O trabalho de Giovanoli et al. (2014) apresenta um mercado eletrônico para serviços de *cloud* e, para tanto, foram buscados modelos existentes de mercados eletrônicos. O trabalho é dividido em duas partes, a primeira apresenta a pesquisa dos modelos e a segunda constrói um modelo específico para o mercado eletrônico de *cloud*. Este trabalho se baseia na primeira parte para definir os requisitos do *marketplace* P2P.

O trabalho define um modelo de referência para *marketplaces* como sendo o entendimento das relações entre entidades de comércio que dêem base a uma especificação. Essa descrição pode ser utilizada para educar e explicar a não-especialistas a respeito de *marketplaces*, formando uma base para engendrar modelos mais específicos.

O estudo procede identificando processos principais na literatura e suas atividades. Os processos são gerenciamento de catálogo, *marketing*, venda, logística e finalização. Para cada um desses processos, o trabalho (GIOVANOLI et al; 2014) estabelece atividades pertinentes com uma breve descrição.

2.3.4.1 Utilização no trabalho

Para se construir um *marketplace*, é necessário definir os requisitos funcionais, para que seja feito um sistema que atenda minimamente algum usuário. Baseando-se na utilização de um modelo já proposto, fica garantida a criação, pelo menos mínima, dos requisitos.

2.3.5 Princípios de *design* de *e-commerces* sociais

Um *e-commerce* social tem como foco a interação com o consumidor, suporte na decisão e encorajamento de retorno (HELANDER; 2000), sendo que a usabilidade é um dos princípios mais importantes em um *e-commerce* (LI; 2011). Segundo o ISO, usabilidade se refere à efetividade, eficiência e satisfação com que um específico usuário atinge certos objetivos em um dado contexto.

O trabalho de Huang (2013) define os seguintes importantes princípios de *design*, para *e-commerces* sociais, divididos entre as camadas: comércio, comunidade, conversação e individual.

A camada individual envolve características como perfil, conteúdo do perfil e atividade de um usuário, discutindo como a informação deve se apresentar maneira clara e correta.

A camada conversação tem relação com notificação, criação de conteúdo, como comentários ou resenhas, e engajamento através de compartilhamento. Aqui são descritas atividades do usuário interagindo com conteúdo estático.

Já a camada comunidade tem como objetivo construir uma comunidade dentro do sistema. Para tanto, são previstas comunicações entre usuários, lojas e outros participantes, fóruns de comunicação, comunidades a respeito de uso de produtos.

A última camada, comércio, expõe características básicas de *e-commerce*, como pagamento, carrinho de compras e rastreamento, e adiciona algumas facilidades sociais, como compra em grupo e sistema de recompensas.

Além disso, são apresentadas certas noções que são ortogonais às categorias, essas noções mais gerais guiaram este trabalho a buscar as características ideais para a usabilidade de um *e-commerce*.

2.3.5.1 Utilização no trabalho

Um *marketplace* deve ser feito com características que garantam a usabilidade e relevância. Para tanto, a extração de requisitos dos princípios acima descritos são fundamentais para garantir que o *marketplace* tenha funcionalidades sociais.

3 PROPOSTA DA ARQUITETURA

3.1 Introdução

Esta seção está divida em duas partes. A primeira trata dos requisitos funcionais e não-funcionais de um e-commerce. A segunda trata da proposta da arquitetura de *e-commerce* P2P.

3.2 Requisitos

3.2.1 Requisitos Funcionais

Um marketplace eletrônico refere-se a um ambiente online onde compradores e vendedores agem como atores para negociar serviços ou produtos. Nesta subseção são apresentados os requisitos, baseando-se no modelo de *marketplaces* sumarizado por Giovanoli (2014). Casos de uso baseados nesses requisitos encontram-se no apêndice A.

Os *marketplaces* eletrônicos são categorizados de acordo com sua governança (Sundareswaran et al, 2012). Há *marketplaces* privados, públicos ou por consórcio. Uma quarta categoria, que tem relação com este trabalho, é o *marketplace* comunitário, em que não há um único dono e vários agentes são responsáveis pelo serviço. Mas no caso de um serviço P2P, todos os participantes são responsáveis pelo sistema, sendo que há alteração de participantes a todo momento.

As principais funções de um marketplace são:

- **Pareamento de compradores e vendedores**: Essa é a maior funcionalidade de um *marketplace*. Os produtos devem ser listados com suas características, especificações e valores, para que o comprador ache algum produto de acordo com sua necessidade.
- **Facilitar a compra**: Um *marketplace* deve fornecer funcionalidades de logística, pagamento, comunicação e rastreamento.
- **Infraestrutura legal**: O *marketplace* deve estar de acordo com leis locais e regulamentos governamentais.

Os requisitos devem, minimamente, observar as fases do modelo de referência para *marketplaces* eletrônicos, o RM-EM, por Lindemann e Schmid (1998). As fases são:

- **Fase da informação**: É a fase da busca pela informação na plataforma. Um comprador busca produtos que satisfaçam sua necessidade em vários vendedores, e assim procede em uma análise competitiva de preços e especificações de produto. Essa fase termina quando o comprador acha um vendedor que tem seu produto no preço desejável.
- **Fase da negociação**: Nessa fase são negociados os preços e fatores ligados à logística. Essa fase termina quando os acordos são definidos.
- **Fase da compra**: Essa é a fase na qual é feita a finalização da aquisição do produto pelo comprador. Aqui ocorre o pagamento e envio da mercadoria.

Para a criação de um *marketplace*, os atores envolvidos devem passar por todas essas fases. Portanto, para a descrição dos requisitos, o processo de aquisição em um *marketplace* deve conter essas três fases e os atores principais: comprador e vendedor.

O modelo de Fingar, Kumar e Sharma (1999) estabelece uma lista de processos pertinentes a *marketplaces*: gerenciamento de catálogo, *marketing*, venda, logística e finalização. Desse modelo, Giovanoli (2014) extrai suas principais atividades, delas serão derivados os requisitos necessários para *marketplace* P2P.

O processo de gerenciamento de catálogo refere-se às diversas maneiras de agregar diferentes produtos de um vendedor. Um vendendor que está tentando vender seu produto em um *marketplace* virtual deve poder ter uma interface pela qual ele possa inserir produtos de maneira ontologicamente estruturada. Na visão dos compradores, ele deve poder buscar produtos de acordo com sua necessidade. Essa busca pode ser simples ou mais elaborada, com uma ontologia mais complexa. Desse processo, são extraídos os requisitos abaixo, para uma organização melhor, os quais são prefixados com GC (gerenciamento de catálogo).

- Requisito GC1 - **Criação de loja**: Um vendedor deve poder criar uma ou mais lojas.
- Requisito GC2 - **Gerenciamento de catálogo**: Um vendedor deve poder gerenciar um catálogo de produtos de cada uma de suas lojas. Esse catálogo será uma lista de produtos com palavras-chave, descrição e valores.
- Requisito GC3 - **Busca de produtos**: Um comprador pode buscar produtos por palavras-chave. Caso haja um produto de acordo com as necessidades do comprador, deve aparecer cada produto localizado com a descrição, preço e o vendedor.

No processo de venda, os compradores finalmente encontram um produto de acordo com sua necessidade e iniciam a transação. Esse processo começa quando o produto é adicionado ao carrinho de compras. Nessa fase, o comprador é autenticado e suas informações de pagamento e de envio são coletadas. Ademais, são calculadas as taxas e outros custos com relação ao envio. Posteriormente, é perguntado ao comprador se ele está de acordo e a compra é confirmada. Ao confirmar, o processo de venda deve enviar um recibo eletrônico ao comprador. Também é parte desse processo prover uma forma de acompanhamento de suas compras. Os requisitos extraídos desse processo são prefixados com V. São eles:

- Requisito V1 - **Carrinhos de compra**: Um comprador, após navegar pelos catálogos, tem a opção de mover certos produtos para o seu carrinho. Esse carrinho de compra é editável: é possível adicionar ou remover produtos, ou alterar sua quantidade.
- Requisito V2 - **Finalização de compra**: O comprador pode optar por seguir com a compra dos produtos no carrinho. Essa finalização deve apresentar a lista de produtos, com taxas e frete.
- Requisito V3 - **Coleta de informação de pagamento**: Para finalizar a compra, o comprador dever fornecer, ou já ter fornecido, os dados de pagamento.
- Requisito V4 - **Coleta de informação de frete**: O comprador deve fornecer as informações de envio.
- Requisito V5 - **Autorização**: O comprador pode autorizar ou não a finalização de sua compra.
- Requisito V6 - **Estimativa de frete**: A fase de finalização deve fornecer uma estimativa de frete.
- Requisito V7 - **Rastreabilidade da compra**: Todas as compras devem poder ter seu estado rastreado com relação a seus interessados.

O processo *marketing* envolve atividades como identificar segmentos, produtos, criar notoriedade à marca e atrair consumidores. Dentre as atividades desse processo, foram selecionadas todas que envolvem o catálogo de um vendendor. Assim, são extraídos os seguintes requisitos, prefixados com M:

- Requisito M1 - ***Cross-selling***: A página de um produto deve mostrar produtos correlatos.

- Requisito M2 - ***Upselling***: A página de um produto deve propor produtos semelhantes com uma qualidade superior.

Quando um produto é vendido, inicia-se o processo de logística. Esse processo contém empacotamento, envio e rastreamento de produtos. Dele, os seguintes requisitos, prefixados com L, foram levantados:

- Requisito L1 - **Atualização de inventário**: Após a venda, o inventário do vendedor deve ser atualizado conforme a venda.
- Requisito L2 - **Integração com serviços de logística**: O sistema deve prover alguma API básica para integração com sistemas de logística de terceiros.
- Requisito L3 - **Rastreamento**: O usuário deve poder obter informação de rastreamento dos seus produtos.

O processo de finalização envolve finalizar transações financeiras, processar os pagamentos dos produtos. Nessa fase também é envolvida a comunicação com terceiros, como, por exemplo, no caso de revenda. A finalização também concerne o pós-venda, como suporte técnico, suporte a problemas no pagamento e devoluções. Seguem os requisitos, prefixados com F:

- Requisito F1 - **Processar pagamentos**: O sistema deve poder processar pagamentos, sendo agnóstico a *gateways* de pagamento.
- Requisito F2 - **Pós-venda - Devolução**: Devem existir mecanismos de devoluções de produtos.
- Requisito F3 - **Pós-venda - Suporte ao pagamento**: Caso o usuário tenha problema no pagamento, o sistema deve prover um canal de comunicação com o lojista para que esse problema seja resolvido.

Esses são os requisitos de um e-commerce completo. No entanto, para simplificar a elucidação da arquitetura, uma versão reduzida desse *marketplace* foi considerada, com menos requisitos. Os demais podem ser completados em um trabalho futuro. Apenas o bloco de requisitos GC e os requisitos V1 e V2 foram atendidos.

Os casos de uso derivados desses requisitos estão presentes no anexo A.

3.2.2 Requisitos Não Funcionais

Nesta subseção são apresentados os requisitos não funcionais, o foco será a usabilidade e qualidade do sistema, baseando-se no trabalho de Huang (2013).

Segundo esse trabalho, os aspectos de *design* que concernem qualidade de informação, qualidade do sistema, qualidade de serviço, usabilidade e características lúdicas devem estar de forma pervasiva no sistema.

O aspecto de qualidade do sistema envolve principalmente segurança e performance, desses aspectos foram derivados os seguintes requisitos não funcionais:

- Requisito NF1 - **Acesso restrito**: O sistema deve garantir que acessos não autorizado, acidental ou não intencional não sejam possíveis.
- Requisito NF2 - **Comunicação encriptada**: Nenhuma comunicação entre participantes pode ser acessada por terceiros.
- Requisito NF3 - **Acesso disponível**: O sistema sempre deve estar disponível para acesso.
- Requisito NF4 - **Performance**: A performance deve ser equiparável a outros *marketplaces*.

Da mesma forma que se reduziu o escopo dos requisitos funcionais para dar enfoque na parte P2P, apenas os requisitos NF3 e NF4 foram considerados neste trabalho.

3.2.3 Relação de requisitos

A tabela 1 mostra a relação de requisitos levantados para um *marketplace* e quais foram considerados por este trabalho.

Tabela 1 – Relação de requisitos

Requisto	Funcional	Não funcional	Considerado para esse trabalho
GC1 - Criação de loja	Sim	Não	Sim
GC2 - Gerenciamento de catálogo	Sim	Não	Sim
GC3 - Busca de produtos	Sim	Não	Sim
V1 - Carrinhos de compra	Sim	Não	Sim
V2 - Finalização de compra	Sim	Não	Sim
V3 - Coleta de informação de pagamento	Sim	Não	Não
V4 - Coleta de informação de frete	Sim	Não	Não
V5 - Autorização	Sim	Não	Não
V6 - Estimativa de frete	Sim	Não	Não
V7 - Rastreabilidade da compra	Sim	Não	Não
M1 - *Cross-selling*	Sim	Não	Não
M2 - *Upselling*	Sim	Não	Não
L1 - Atualização de inventário	Sim	Não	Não
L2 - Integração com serviços de logística	Sim	Não	Não
L3 - Rastreamento	Sim	Não	Não
F1 - Processar pagamentos	Sim	Não	Não
F2 - Pós-venda - Devolução	Sim	Não	Não
F3 - Pós-venda - Suporte ao pagamento	Sim	Não	Não
NF1 - Acesso restrito	Não	Sim	Não
NF2 - Comunicação encriptada	Não	Sim	Não
NF3 - Acesso disponível	Não	Sim	Sim
NF4 - Performance	Não	Sim	Sim

Fonte: Elaborado pelo autor

3.3 Proposta de Arquitetura

3.3.1 Sobre a proposta

O *marketplace* P2P funciona armazenando produtos, lojas, carrinhos e transações em uma rede P2P estruturada genérica, e utiliza alguns nós com papéis específicos para cuidar da busca, entrada de novos nós e comunicação com sistemas externos. Para que essa rede genérica passe a dar suporte a um *marketplace*, são necessárias algumas adaptações em relação aos dados guardados na rede P2P e ao comportamento dos nós.

Os dados guardados na rede P2P devem seguir um modelo único, para que todos os nós consigam se comunicar na mesma linguagem. Os nós devem ter posse de identificadores na rede, por exemplo, um nó que crie uma loja deve manter a autoria sobre essa informação. A rede P2P deve garantir que os dados sejam replicados para que haja redundância da informação, pois os nós podem sair e entrar na rede a

qualquer momento.

O comportamento dos nós em uma rede P2P qualquer normalmente refere-se às disposições dos recursos e de seus nós. Essa arquitetura não altera esse comportamento, são adicionados novos comportamentos para a manutenção das informações de *marketplace* que estão armazenadas na rede P2P. Esse comportamento adicional é o que torna possível o *marketplace*. Um nó em uma rede P2P pode salvar uma informação qualquer, mas um nó do *marketplace* P2P salva uma informação específica com um intuito próprio, apenas nós que partilham do mesmo comportamento podem fazer algum sentido dessa informação. Dessa forma, é possível o cenário hipotético de que alguns nós estão apenas rodando o protocolo P2P original, mas outros estão na mesma rede para usufruir do *marketplace*.

A diferença entre um *marketplace* P2P e uma rede P2P comum é a introdução de um modelo de dados e de um comportamento específico de nós, que são divididos por papéis. Existem nós que têm o papel de fazer a busca por palavras-chave de produtos, lojas e transações, outros nós têm o papel de se comunicar com entidades externas à rede, os nós que cuidam da entrada na rede e o restante são os nós não especializados, que são os clientes e as lojas desse *marketplace*.

Sendo assim, o que define a arquitetura é um comportamento adicional dos nós, que manipulam um formato de dado específico e que têm papéis diferentes. Esses papéis apenas diferenciam-se em relação ao comportamento adicional.

A ideia geral é que a rede P2P funcione como um banco de dados distribuído por todos os nós, mas de maneira simplificada, sendo utilizado o mesmo mecanismo da rede P2P subjacente. Essa informação possui um identificador único. Para se obter um produto, loja, carrinho ou transação, é necessário saber o identificador dessa entidade. Os nós que oferecem a busca fornecem o serviço de buscar essas entidades por palavras-chave.

Alguns dados podem estar fora da rede P2P. Eles são acessados como todos os outros, mas existem nós com papel específico, que contatam sistemas exteriores e transformam esses dados em entidades que adotem o formato interno. Para os nós que consomem esses dados, não há indício de que eles sejam externos.

É pressuposto que os nós participantes da rede não sejam maliciosos e respeitem as regras de comportamento definidas por essa proposta. Caso se considerem nós maliciosos, seria necessária a introdução de mecanismos que coíbam essas ações, como, por exemplo, uma assinatura digital para garantir a autoria da informação.

Esse sistema torna-se P2P não pela definição de um novo protocolo P2P, mas pela utilização específica de algum protocolo dessa natureza. A sua existência satisfazendo os requisitos da arquitetura do *marketplace* é pressuposta.

3.3.2 Introdução

Em uma arquitetura tradicional de cliente-servidor, o cliente desse sistema age no servidor, que, por sua vez, pode executar essa ação, tendo em vista as regras de negócio, persistindo informações em um modelo de dados e se comunicando com sistemas exteriores. Essas ações no sistema de *marketplace* são, por exemplo, as compras ou vendas feitas. Já na arquitetura P2P, o cliente age em uma rede de pares que precisa executar as mesmas ações descritas. Para criar uma aplicação P2P, logo, é necessário que pelo menos esse sistema consiga executar a persistência de informação e a comunicação com o mundo exterior. Além disso, para que se possa comunicar com a rede P2P, e usufruir de seus serviços, é necessário tornar-se parte dessa rede, por meio do processo de *bootstrap*.

Para que haja encapsulamento das atividades P2P e de *marketplace*, foi necessário separar as regras de negócio das atividades P2P. Dessa forma, o componente que executa as regras de negócio não tem conhecimento do ambiente em que habita, isto é, não há diferença, para ele, se a arquitetura é cliente-servidor ou P2P.

Para garantir a performance da rede e estabelecer uma topologia ordenada dos nós, optou-se por utilizar protocolos P2P estruturados. A topologia ordenada é uma garantia de que as mensagens trafegadas na rede serão eventualmente respondidas. Esse protocolo P2P estruturado deve manter a posse de parcela da tabela *hash* distribuída para os nós que criem esses dados, para que somente o nó altere a entidade que ele insira. O protocolo deve manter a redundância dos dados na rede, assim, o nó pode sair e seus dados continuarão na rede. A posse do dado deve ser retornada ao nó. O nó pode retornar à rede e a posse da parcela de *hash* distribuída deve retornar a ele. O protocolo deve ser resistente a *churn* em relação ao número de mensagens trafegadas, isto é, a entrada e a saída repetida de nós deve ter a complexidade de $O(log(n))$, em que n é o número de nós na rede, para garantir a estabilidade da rede, pois a escala de visitantes únicos por mês de um *marketplace* grande, como o eBay, pode chegar à casa de dezenas de milhões (SUN; FANG; LI, 2017).

As redes P2P estruturadas possuem uma interface bem definida, com ações de roteamento, inclusão e remoção de nós e recursos. Essa interface pode ser traduzida a um modelo comum de P2P estruturado a ser utilizado por todos os componentes P2P. Com esse modelo, a arquitetura não depende da escolha do protocolo de P2P estruturado.

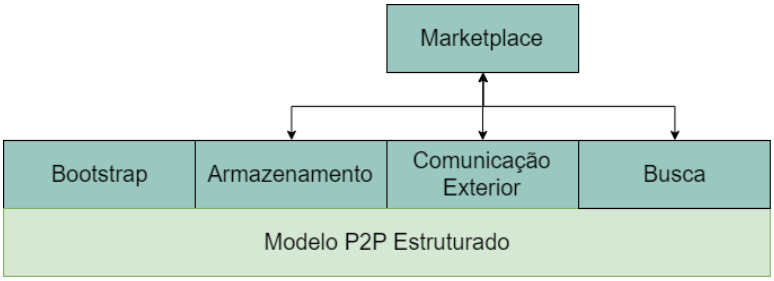
Geralmente, em sistemas tradicionais, a busca por algum dado pode ser feita a partir do próprio banco de dados do sistema. No caso de dados dispersos em uma rede P2P, isso não é possível. A busca poderia ocorrer por comunicação entre os próprios

nós, mas isso tornaria difícil a escolha de protocolos P2P estruturados. Portanto, adota-se um componente de busca separado, que cuida dessa indexação na rede e realiza busca por palavras-chave.

Essa arquitetura será apresentada primeiramente descrevendo os componentes, que desempenhem as funções descritas acima, de maneira estática, posteriormente, a dinâmica entre esses componentes será explicitada. Os componentes estáticos formam um mapeamento explícito, como exposto. Segue uma descrição breve de cada componente, separados entre a categoria de componentes que dependem da rede P2P:

- Componente de Bootstrap (P2P): Esse componente faz com que a entrada de novos nós na rede P2P seja possível. Ele fornece as informações e meios necessários para que o nó entre na rede;
- Componente de Armazenamento (P2P): Todos os dados do sistema devem passar por esse componente para serem armazenados;
- Componente de Comunicação Exterior (P2P): Torna possível a comunicação com sistemas exteriores a esse.
- Componente de Modelo P2P (P2P): Modelo de interface utilizado pelos componentes P2P;
- Componente de *Marketplace*: Todas as regras de negócio do *marketplace* são executadas por esse componente;
- Componente de Busca (P2P): Indexa a rede e fornece uma interface simples de busca por palavras-chave.

Figura 1 – Componentes principais da arquitetura



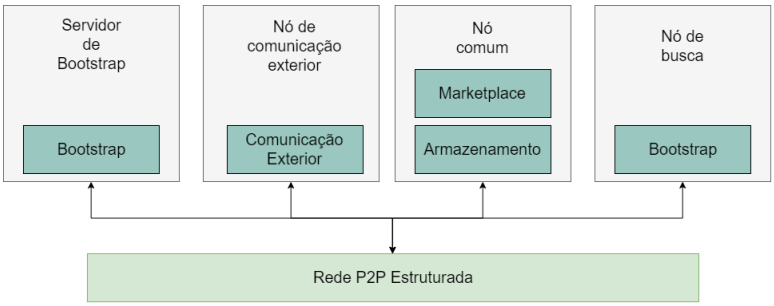
Fonte: Elaborado pelo autor

A figura 1 mostra os componentes P2P com sua dependência do modelo P2P e a troca informações feita com o componente de *marketplace*. Cada nó entra na rede pelo componente de *bootstrap*. Ao entrar na rede, passa a ser possível a utilização dos

componentes de armazenamento e comunicação exterior. Esses dois componentes são necessários para que as regras de negócio contidas no componente de *marketplace* sejam executadas.

As atividades pertinentes à comunicação exterior e de *bootstrap* são muito importantes para residirem em um nó com poucos recursos. Por isso, nessa arquitetura, a rede P2P estruturada é heterogênea, isto é, existem nós com papéis diferentes. Existem quatro papéis a serem desempenhados, ilustrados na figura 2, o servidor de *bootstrap* que cuida da entrada de novos nós na rede e contém a versão atual do sistema, o nó que tem habilidade de se comunicar com o mundo exterior, o nó comum, que executa as regras de negócio e cuida do armazenamento, e o nó de busca. Essa divisão indica essa necessidade de separação, mas não impede que um nó desempenhe todos os papéis simultaneamente. Mesmo que um nó com poucos recursos assim decida agir, seria inútil devido à concorrência gerada pelos nós com mais recursos, pois poucos clientes acessariam um servidor de *bootstrap* com problemas e as mensagens direcionadas aos nós de comunicação exteriores chegariam e seriam processadas primeiramente em um nó com mais recursos.

Figura 2 – Papéis principais da arquitetura



Fonte: Elaborado pelo autor

A ação de fazer o *bootstrap*, nessa arquitetura, consiste em fazer o nó entrar na rede P2P, mas também efetuar a conexão entre pares, para que se possa utilizar tecnologia que dependa disso. Por exemplo, quando um nó entra na rede, o *bootstrap* já definiu com quem o nó entrante deve se comunicar para entrar na rede, mas essa conexão pode não ser estabelecida de forma direta, sendo necessária uma troca adicional de informações mediada por esse servidor. Outra funcionalidade é fornecer a versão inicial do sistema ao cliente. O componente de *bootstrap*, então, tem as funções de selecionar um nó para ser ponto de contato ao nó entrante, estabelecer conexão mediada quando necessário e fornecer o sistema em si. O protocolo para acessar esse servidor pode ser um diferente do protocolo utilizado na rede P2P estruturada. Após a entrada do nó, ele só se comunicará com o servidor de *bootstrap* caso haja uma entrada de um novo nó e fique decidido que este ficará conectado com o nó que já estava na rede.

Figura 3 – Funcionalidades do componente *bootstrap*

Fonte: Elaborado pelo autor

O armazenamento dos dados pertinentes ao *marketplace* é feito pela tabela *hash* distribuída do protocolo P2P estruturado escolhido. Mas são estabelecidos requisitos para esse protocolo, cada nó deve ser responsável pela manipulação de certos identificadores da tabela. Por exemplo, as informações dos produtos presentes no carrinho do cliente devem ser criadas e armazenadas no próprio nó, para a garantia de quem é dono do dado. Para sua replicação, o protocolo deve garantir que alguns nós vizinhos possuam uma réplica dos dados criados por esse cliente. Caso o nó saia e entre novamente, ele deve obter o direito sobre os identificadores já utilizados por ele. Além de ser dono de uma faixa de valores, os nós também armazenam informações replicadas de outros nós.

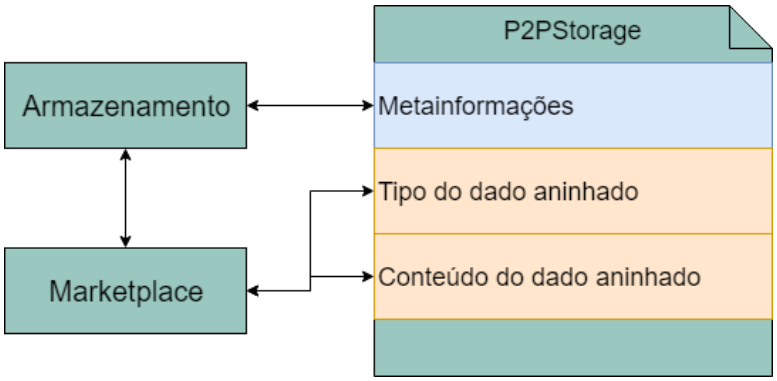
Cada dado guardado na tabela de *hash* distribuída tem informações pertinentes às características P2P do armazenamento do dado, como identificador e nós responsáveis, e também possui as informações relativas ao *marketplace*. Cada elemento armazenado possui um formato, denominado P2PStorage, com campos com metainformação desse dado e campos relacionados à informação buscada, como o tipo da informação e o conteúdo. Esse dado irá trafegar com dois propósitos, ele pulará de nó em nó até chegar ao destino e, então, os campos de metainformação serão utilizados, posteriormente, quando a informação chegar até o nó de destino, ele utilizará os campos relacionados a *marketplace*. O tipo de informação e o conteúdo no P2PStorage não dependem em nada do protocolo P2P, eles são definidos pelas regras de negócio da aplicação, por exemplo, "loja"é um tipo de informação e o conteúdo seria informação de uma loja.

O componente deve possuir uma interface para guardar entidades P2PStorage para salvar e recuperar dados. Por exemplo, o componente de *marketplace* chama esse componente para salvar lojas, produtos e carrinhos.

O componente de armazenamento é definido, então, como uma tabela *hash* distribuída que guarda valores no formato P2PStorage, o qual possui campos que são definidos pelo modelo de dados da aplicação. A figura 4 mostra essa divisão.

Para que o sistema seja extensível, o componente de acesso exterior garante

Figura 4 – Divisão do P2PStorage



Fonte: Elaborado pelo autor

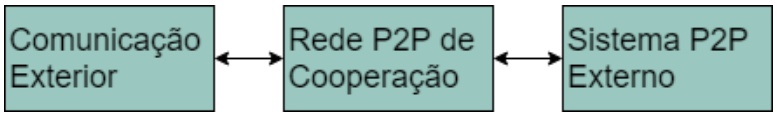
que outros sistemas possam se integrar com o *marketplace* P2P. Essa arquitetura foi concebida para ser descentralizada, sendo assim, o sistema acessado externamente idealmente é sempre pensado como um outra rede P2P. Assim minimizam-se os pontos de falha. Por exemplo, um sistema cliente-servidor integrado com o *marketplace* P2P pode virar parte integrante desse sistema cuidando do frete, mas, caso esse sistema fique fora ar, a descentralização estabelecida por essa arquitetura tornaria-se inútil, já que uma dependência exterior não descentralizada ficou fora do ar. Para fazer com que duas redes P2P comuniquem-se, isto é, para que exista cooperação entre duas redes P2P, a solução adotada foi a introdução de uma terceira rede, chamada de rede P2P de cooperação, que funciona como uma espécie de interface entre o sistema de *marketplace* P2P e o sistema externo, como observado na figura 5. Quando se utiliza uma rede de cooperação P2P, pode-se obter nós que sejam participantes das redes do *marketplace* e de cooperação, e vários nós podem se comunicar com sistemas externos, foi isto que motivou sua escolha.

Esse acesso externo deve ser transparente a nós na rede. Um recurso acessado externamente deve ser iniciado por uma requisição comum. O nó requisitante não precisa saber se o recurso é interno ou externo. Para simplificar o modelo, a conexão externa será feita somente para consumir dados externos, e não para enviá-los. A comunicação em duas vias não será discutida por este trabalho.

A utilização do componente de acesso exterior é importante mesmo para versões simples de *marketplace*. Arquivos, como imagens de produtos, devem estar externos ao sistema para não sobrecarregar a rede P2P com dados potencialmente grandes.

No componente de *marketplace* estão as regras de negócio. É esse componente que origina a criação de entradas na tabela *hash* distribuída, por meio do componente de armazenamento. Conforme os requisitos GC1, GC2, GC3, V1 e V2, o *marketplace* deve poder gerenciar lojas, produtos, carrinhos de compra e transações. Portanto, esse componente define essas entidades e faz as chamadas para armazená-las.

Figura 5 – Componente de comunicação exterior



Fonte: Elaborado pelo autor

O componente de busca deve funcionar como um função $B : [Termo] \rightarrow [Identificador]$, isto é, dada uma lista de termos, ele deve retornar uma lista de identificadores relacionados a esses termos. Essa simplificação ocorreu para dar liberdade aos mecanismos de indexação. Por essa lista de identificadores, o nó pode buscar na rede P2P estruturada o recurso desejado.

3.3.3 O componente de modelo P2P

O componente de modelo P2P é uma interface genérica de comunicação P2P. Nesse componente são definidos, abstratamente, quais são os elementos de uma rede P2P estruturada, como nós, recursos, identificadores e quais são as principais operações, como entrada, saída, busca e roteamento.

Um sistema P2P, em especial sua rede *overlay*, pode ser visto como um *middleware*. Esse *middleware* deve providenciar uma maneira simples, mas efetiva, de abstrair os conceitos e esconder detalhes de implementação do usuário, oferecendo controle e acesso a essa abstração. A maneira como o sistema é abstraído deve manter sua interface mesmo que a implementação seja outra.

O sistema de *marketplace* P2P pretendido é uma cooperação de sistemas P2P. Caso esses sistemas P2P utilizem um único modelo com mesma interface, pode-se trocar apenas a implementação. Para tanto, este trabalho adapta um modelo conceitual de Aberer et al. (2005) para a situação de cooperação, tendo em vista uma aplicação subjacente.

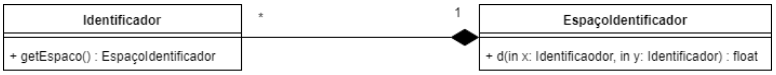
Nesse modelo de P2P, basta definir como são as operações de entrada, saída, busca, roteamento e seu espaço de identificadores para que se obtenha o mapeamento de um protocolo no modelo. O objetivo é fazer a dissociação entre a implementação do P2P e a arquitetura.

Como visto em Aberer et al. (2005), existem dois fundamentos em uma rede P2P. O primeiro, diz respeito aos recursos e nós dessa rede, como são associados identificadores a eles e de que modo esses identificadores estão dentro de um conjunto maior, um espaço de identificadores. O segundo, refere-se a como esses identificadores se comportam e se dividem entre os nós, formando o grafo da rede overlay. A relação entre esses dois fundamentos ocorre através da métrica do espaço de identificadores,

que define como cada nó está devidamente mapeado no espaço, pois o primeiro fundamento provê um identificador, e, através dessa métrica, é possível identificar onde esse nó se encontra na rede, podendo-se formar uma estratégia de overlay. Como a cada recurso também é dado um identificador, a métrica provê quem é o nó responsável pelo recurso, adotando uma estratégia a partir de uma distância dessa métrica.

Desses fundamentos, então, nota-se que toda rede P2P possui esse espaço de identificadores, logo, o primeiro elemento da arquitetura partirá dessa estrutura.

Figura 6 – Identificadores



Fonte: Elaborado pelo autor

Nota-se que essa estrutura, como pode ser visto na figura 6, abarca o conceito de que esse espaço de identificadores pertence a uma específica rede P2P, e isso será necessário, uma vez que existirão nós que podem pertencer a mais de uma rede. Portanto, foi necessária a introdução no Identificador de um método que retorne seu respectivo EspaçoIdentificador.

O próximo passo é definir quem são os elementos que podem ser identificados nesse espaço, como visto nos fundamentos, são dois: nós e recursos.

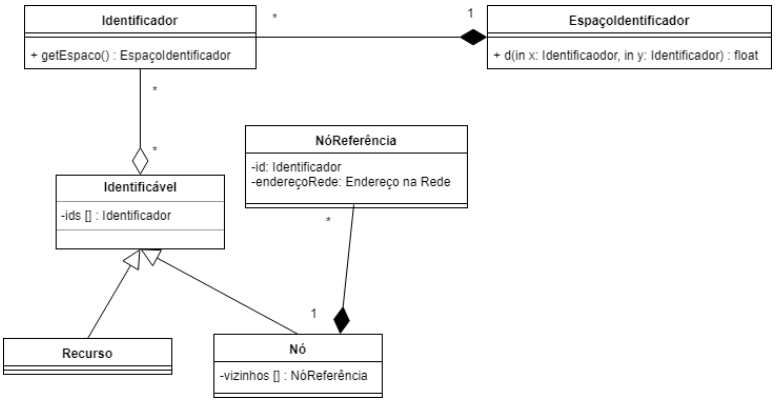
A começar pelos nós, eles possuem dois atributos essenciais, uma lista de identificadores e uma lista de vizinhos. A lista de identificadores deve conter um único identificador para cada uma das redes P2P a que o nó pertença. A tendência é pensar nos vizinhos como sendo uma lista de nós, mas é preciso notar, como visto na estrutura de camadas apresentada, que um vizinho não está somente mapeado em um espaço de identificadores, mas também possui um endereço físico, logo, uma estrutura um pouco diferenciada é necessária.

Os recursos são estruturas parecidas, eles também podem ser localizados no espaço identificador, no entanto, o conceito de vizinhança nem sempre é aplicável. Pode-se considerar, então, uma estrutura que abarque entidades com atributos de identificador, ou seja, tanto o recurso quanto o nó são identificáveis.

A figura 7 mostra os atributos dos nós e sua relação com os identificadores.

Na rede P2P existem interações entre esses elementos identificáveis. Há a interação entre Nó e Recurso, um nó pode inserir, remover, atualizar ou buscar um recurso. Essas ações envolvem referências a recursos, apenas para a busca é necessário introduzir um parâmetro de critério adicional. Existem as ações da interação entre o nó e seus vizinhos, um nó pode entrar em uma rede P2P, sair dessa rede, procurar o dono de algum dado ou rotear alguma mensagem. Para isso, são

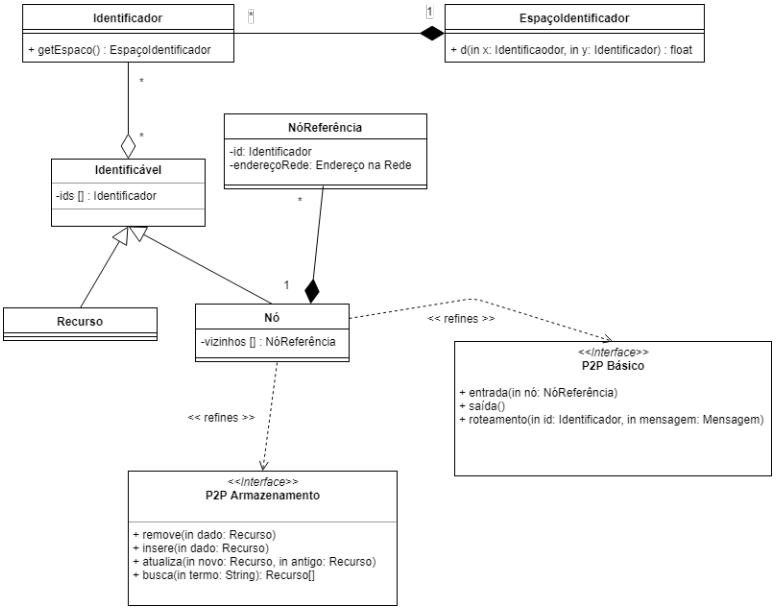
Figura 7 – Nós e identificadores



Fonte: Elaborado pelo autor

introduzidas duas interfaces que o nó deve satisfazer, a interface de armazenamento e a interface de roteamento. Como pode ser observado na figura 8.

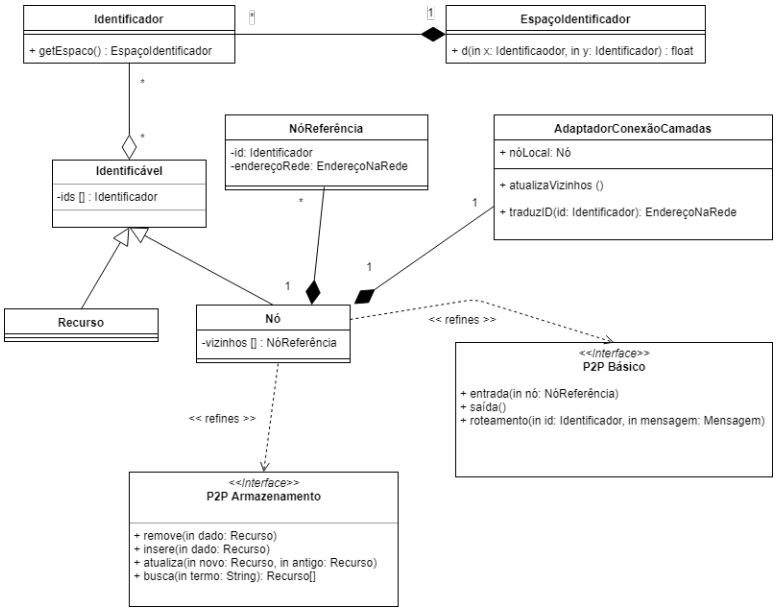
Figura 8 – Nós com interfaces



Fonte: Elaborado pelo autor

Para que as ações da interface sejam executadas, as mensagens devem trafegar na rede, para além dessa arquitetura, sendo assim, é necessário um dispositivo que faça com que seja possível enviar e receber mensagens ao endereço físico. Nesse modelo, foi introduzido um adaptador para essa função, chamado adaptador de conexão entre camadas. Esse adaptador tem a função de fazer a tradução entre a rede e a rede *overlay*, além de gerenciar as conexões ativas. Portanto, o adaptador gerencia as instâncias de vizinhos na instância do nó local.

Figura 9 – Diagrama com adaptador



Fonte: Elaborado pelo autor

3.3.4 O componente de *bootstrap*

O componente de bootstrap é o componente responsável pela entrada de nós na rede, pela seleção de um nó para ser ponto de contato ao nó entrante, por estabelecer conexão mediada quando necessário e fornecer o sistema. Esse fornecimento em uma implementação poderia se dar por HTTP, por exemplo.

Certos protocolos necessitam de uma troca de mensagens mediada para que haja uma conexão bem sucedida. Logo, não há a possibilidade de que uma conexão seja efetuada sem que exista um terceiro interessado fazendo a mediação, sendo indispensável a utilização de um servidor de *bootstrap* como mediador.

A principal função do servidor de *bootstrap* é fazer com que um cliente consiga entrar na rede P2P. Para isso, esse servidor deve fazer com que o cliente, por algum outro protocolo, consiga trocar as informações necessárias para estabelecer conexão com alguns nós já participantes na rede. Esse servidor também faz a mediação da conexão do protocolo de comunicação. Por exemplo, a implementação pode escolher utilizar o WebRTC.

Poderia existir um servidor para cada tipo de comunicação mencionada, mas a oportunidade de conexão de um nó entrante com esses serviços ocorre somente no momento da entrada na rede. Essas conexões são sequenciais e de pouco tempo de vida, com um único objetivo. Caso fossem separadas, o sistema teria um ponto a mais para falha, pois um nó só conseguirá entrar na rede se, de forma bem sucedida, conseguisse se comunicar com cada um desses serviços.

Após a aquisição dos arquivos pelo cliente, sucede a funcionalidade principal do *bootstrap*, ela foi dividida em duas fases. A primeira diz respeito à troca de protocolo, por algum outro, e a entrada na rede por um único nó, que é o mesmo servidor com o qual o cliente está se comunicando. A segunda fase consiste na comunicação com os outros nós pertencentes à rede P2P.

Com essa divisão estabelecida, torna-se interessante dividir esse componente do sistema em duas fases intercomunicáveis. A primeira será chamada de *Bootstrap* Inicial e a segunda de *Bootstrap* na Rede.

Na fase de *Bootstrap* Inicial, o componente necessita de uma conexão com algum nó participante da rede P2P, a qual é feita com o cliente entrante. Ele então faz a comunicação mediada entre esses dois nós. O objetivo é compartilhar entre os nós as mensagens de *handshake*.

A segunda fase, o *Bootstrap* na Rede, poderia ser feita sempre entre quaisquer nós da rede P2P. Mas também é possível que o próprio servidor de *bootstrap* aja como nó sempre presente. Isso é útil, pois a maioria dos nós está rodando em um navegador, o que pode tornar, com a entrada e saída repetida de nós, o *churn*, a tentativa de adentrar-se na rede frustrada, já que não se pode garantir que esses nós fiquem na rede.

O componente em sua segunda fase utiliza a interface de protocolo P2P apresentada na subseção anterior, sendo assim, esse modelo não depende do protocolo P2P escolhido, e, por sua vez, também não depende da finalidade da aplicação.

Quando o nó entrante está com a conexão estabelecida com algum outro nó da rede, está finalizada essa etapa de *bootstrap*. Por fim, o adaptador de conexão entre camadas é utilizado para se adicionar essa conexão estabelecida, então inicia-se o protocolo P2P.

Após esse início, é enviado ao nó entrante a mensagem de boas-vindas definida pelo componente de armazenamento.

3.3.5 O componente de armazenamento

O componente de armazenamento é definido como uma tabela *hash* distribuída que guarda valores no formato P2PStorage, o qual possui campos que são definidos pelo modelo de dados da aplicação.

Como o armazenamento dos dados é responsabilidade do protocolo de P2P estruturado, resta apenas definir como os dados armazenados são estruturados e a interface desse componente. Existem dois formatos de dados definidos para que o *marketplace* P2P funcione. O primeiro formato é enviado após a entrada do nó na rede. O segundo é um protótipo do que se contém na DHT. As mensagens serão descritas

em JSON Schema, dada uma descrição de cada campo. A escolha do JSON Schema é pela natureza desse sistema, que, principalmente, rodará em navegadores. Esses esquemas estão detalhados no apêndice B.

A primeira mensagem, nomeada *"MensagemInicial"*, que deve confirmar o identificador do nó entrante na rede através do campo "seuID" e do campo "listaNosBusca", contém uma lista de identificadores de nós de busca. Também são informadas duas faixas de identificadores, o campo "suaFaixa" é a faixa pela qual o usuário é responsável e a outra faixa, "suaFaixaDono", é a faixa que determina onde estão guardadas as informações do nó, como usuário, lojas, compras e vendas.

A ciência da lista de nós de busca é fundamental para que seja possível a utilização do sistema. Sem ela, o nó somente poderá fazer requisições por identificador na rede.

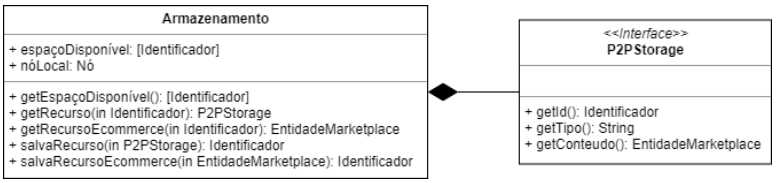
Essa faixa deve ser herdada pelo nó entrante pelo seu nó pai. Se o pai possui uma faixa de identificadores $minID <= id <= maxID$, o nó entrante possuirá identificadores dentro dessa faixa, e o nó pai deixará de ser responsável por esses valores, que serão entregues em uma lista de pares de limites. Por exemplo, caso seja entregue a lista $[4, 6, 7, 10]$, o nó entrante é responsável pelos identificadores 5, 8 e 9. Sendo responsável por esses identificadores, o nó entrante poderá alterá-los. Quando o nó sai da rede, esses elementos são distribuídos para o pai ou para um outro nó. Nesse caso, há risco de que a informação seja alterada, para tanto, deve-se adotar algum tipo de criptografia, que está longe do escopo deste trabalho. Quando o nó que era dono da informação voltar, ele deve se manter dono dos mesmos identificadores.

O segundo formato, denominado P2PStorage, que é o que se obtém como valor na tabela *hash* distribuída, foi feito tendo em mente a extensão desse protocolo. Essa mensagem contém o identificador do objeto na tabela *hash* distribuída, "id", o objeto como sendo um objeto qualquer em "valor", o tipo aninhado que esse objeto representa em "tipo". O campo "palavrasChave" define quais são as palavras-chave do dado, possibilitando a indexação pelo componente de busca.

Existe um campo opcional no P2PStorage, o "itensExteriores", que contém uma lista de elementos externos à rede que possuam alguma relação com esse objeto. Por exemplo, um produto pode ter uma lista de fotos, que estará armazenada na rede Blockstack. Os identificadores dessas fotos devem estar listados nesse campo. O campo "privado"com valor booleano é um indicativo de que aquele P2PStorage não deve ser indexado pelo motor de busca, caso tenha valor verdadeiro.

A figura 10 mostra a classe modelo desse componente. Ele deve armazenar referências dos identificadores disponíveis e do nó local. São expostas as funções de obter a lista de identificadores disponíves, salvar entidades do *marketplace*, *P2PStorage*

Figura 10 – Interface de Armazenamento



Fonte: Elaborado pelo autor

e buscá-las.

Quando um nó entra no sistema, cria algumas entidades e sai, a posse desses identificadores deve ser mantida. E, no eventual retorno, o nó informa que é um nó retornante que tinha posse de tais identificadores. Essa maneira de proceder não é ideal, pois um nó malicioso poderia obter a posse de qualquer identificador. No entanto, para o escopo desse trabalho, assume-se que os nós não são maliciosos.

3.3.6 O componente de comunicação exterior

O componente de comunicação exterior é o que possibilita a comunicação do *marketplace* P2P com outros sistemas, nessa descrição ele será arquitetado somente para obter informações fora do sistema, mas não para o exterior se comunicar com o *marketplace* P2P.

Uma porção do espaço identificador de recursos deve ser reservada para essas entidades. Elas devem ser acessadas pela interface do componente de armazenamento pelo seu identificador. Os nós responsáveis por essa comunicação dividem essa faixa entre si.

Quando existe uma informação fora da rede P2P, ela deve ser salva pelo nó responsável na rede P2P para que essa informação adquira um identificador interno.

Um nó desse tipo sempre está também em uma outra rede P2P, a rede de cooperação P2P. Isto é, todos os nós de comunicação exterior estão alojados em duas redes.

As informações cuidadas por esse componente são tratadas pela rede como quaisquer outras. Toda informação exterior possui um identificador único na rede P2P, e pode ser acessada por ele. Além disso, esse identificador possui o mesmo formato básico dos outros dados, o que permite a indexação pelo componente de busca, já que o formato básico, "P2PStorage", possui campos de palavras-chave.

Supondo que um sistema P2P externo forneça as imagens para o *marketplace*, a cada imagem será atrelada um identificador. O cliente, ao acessar o produto, por exemplo, pode descobrir que esse produto possui uma imagem. O nó então busca esse

componente na rede estruturada P2P, essa busca trafega de nó em nó até chegar em algum nó responsável por esse identificador. Esse nó utiliza a rede de cooperação P2P para obter o dado que está na outra rede, e, assim, que consegue retornar a imagem encapsulado em um *P2PStorage*.

3.3.7 O componente do *marketplace*

O componente de *marketplace* é o que executa e determina as regras de negócio do *marketplace*. Ele utiliza o componente de armazenamento para salvar as informações e não tem relação direta com a rede P2P.

Para que o *marketplace* funcione nesse modelo, basta definir o comportamento das ações com relação ao formato de dados P2PStorage. Sendo assim, todas as ações do *marketplace*, como criar usuário, lojas, produtos, devem ser definidas através de chamadas ao componente de armazenamento.

Para atender aos requisitos GC e o requisitos V1 e V2, o *marketplace* P2P terá os seguintes tipos de dados derivados dos requisitos: Loja, Produto, Carrinho, Compra e Venda. Esses dados serão guardados como *P2PStorage* na rede P2P. Cada nó é responsável por identificadores selecionados, caso ele deseje criar uma loja, ele guardará em algum de seus identificadores disponíveis, um objeto desse tipo. O carrinho deverá ter o campo "privado"do P2PStorage como verdadeiro, para não ser indexado pelos motores de busca.

Uma transação é completa quando o comprador cria uma entidade de compra e o vendedor confirma-a com a criação de uma entidade de venda.

Os esquemas de Loja, Produto e Carrinho foram simplificados para esse modelo e estão como *JSON Schema* no apêndice B. Esses dados são representados por classes que herdam o tipo "EntidadeMarketplace". Para guardar ou ler dados, basta utilizar a interface do componente de armazenamento para obter as entidades.

3.3.8 O componente de busca

As informações do sistema estão distribuídas na rede P2P. Cada informação tem um identificador único, seja ela interna ou externa. Por isso, para o funcionamento do *marketplace*, é necessário haver algum meio de busca dessas informações.

O componente de busca é o que possibilita que as informações sejam encontradas, ele recebe uma lista de palavras-chave e retorna uma lista de identificadores relacionados às palavras-chaves enviadas.

Alguns nós especiais são responsáveis pela busca e devem receber uma mensagem de busca contendo um termo e retornar uma lista de identificadores. Fica a

critério do desenvolvimento desse serviço de que forma a busca será feita, desde que a interface seja respeitada.

Quando se recebe a "MensagemInicial", o nó já tem conhecimento dos nós de busca disponíveis. Para efetuar a busca, basta utilizar a interface de roteamento definida no modelo com uma lista de termos.

Os nós de busca devem respeitar o campo "privado"das entidades "P2PStorage".

As transações também são buscadas por esse componente. As informações de compra e venda são guardadas como palavras-chave específicas que identificam qual é o produto vendido.

Um exemplo de busca básico é uma busca exaustiva por todos os nós, colhendo todas as entidades guardadas e gravando, criando imagens da rede P2P estruturada.

3.3.9 Exemplos de funcionamento

Essa subseção mostra o funcionamento da arquitetura para cada um do requisitos propostos e faz uma comparação em relação ao modelo cliente-servidor.

3.3.9.1 Entrada do cliente

Para que se possa utilizar o *marketplace* P2P, é necessário efetuar o *bootstrap*. Após a entrada do nó na rede, ele receberá a mensagem de boas-vindas "MensagemInicial", onde é atribuído a ele uma faixa de identificadores que ele pode modificar, podendo armazenar lojas, produtos, carrinhos e transações como P2PStorage.

A entrada do cliente em uma arquitetura cliente-servidor é mais simples, basta o usuário acessar uma página, por exemplo. No modelo P2P, o usuário precisa fazer parte da rede P2P para poder usufruí-la.

3.3.9.2 Saída do cliente

Nesse sistema P2P, quando um cliente sai, caso não haja a redundância dos dados que ele inseriu na rede, os dados podem ser perdidos. Por isso, a escolha de um protocolo P2P que garanta essa redundância é fundamental.

Não existe essa preocupação na arquitetura cliente-servidor, já que o cliente não guarda nenhum dado.

3.3.9.3 Busca por palavras-chave

A busca é feita fazendo requisições para nós responsáveis por indexar a rede P2P estruturada. O nó do cliente recebe na mensagem de boas-vindas quais são os identificadores desses nós de busca. O nó do cliente envia uma requisição no formato

de uma lista de palavras-chave para esse nó de busca. O nó de busca, então, retorna uma lista de identificadores relacionados a essa palavra-chave. Nesses identificadores, o cliente pode encontrar entidades de lojas e produtos relacionados à palavra-chave.

Em uma arquitetura cliente-servidor, a busca pode ser feita pelo próprio sistema, utilizando o banco de dados. A indexação prévia dos dados pode ser feita também, mas não é necessária. Já na arquitetura P2P proposta, devem existir nós específicos que fiquem indexando toda a rede.

3.3.9.4 Criação e manutenção de loja

Quando um cliente cria uma loja, ela é salva pelo componente de armazenamento em um dos identificadores que pertence ao nó criador. O protocolo de P2P estruturado deve manter a posse desse identificador, e, por conseguinte, da loja criada pelo usuário. Essa entidade de loja recém criada deve ser replicada em outros nós para que a redundância seja mantida. Por exemplo, se um indivíduo cria uma loja "Loja ABC" no identificador 123, ele pode alterar a qualquer momento a informação contida no identificador 123, e outros nós mantêm uma cópia dessa informação, mas não alteram a informação nesse identificador. No caso de saída desse nó, o protocolo deve manter a posse do nó faltante. No retorno desse nó, ele obtém o direito de modificar esse mesmo identificador. Para tanto, o nó deve comunicar no processo de *bootstrap* que ele está retornando. Pressupõe-se que não existem nós maliciosos que corrompam as informações.

Essa entidade é salva na rede com palavras-chave, e essas palavras-chave são indexadas pelos nós de busca. Depois dessa indexação, a loja está disponível para a busca pelos outros nós.

Para modificar a loja, o nó dono do identificador ao qual a loja pertence sobrescreve a mesma entrada.

Em uma arquitetura cliente-servidor, a loja criada não é armazenada inicialmente pelo próprio cliente, ela é salva em um banco de dados e fica sob inteira responsabilidade do servidor. A posse do dado não é real, mas sim uma concessão do servidor ao cliente.

3.3.9.5 Criação e manutenção de produtos

A criação e a manutenção de produtos procede da mesma forma que a criação de lojas, com a única diferença de que um produto deve fazer referência à loja a que pertence. Pode acontecer de um produto descrever que pertence a uma loja inexistente, nesse caso, o produto deve ser desconsiderado pelos nós.

Em uma arquitetura cliente-servidor, consegue-se manter uma melhor qualidade dos dados, colocando restrições a produtos sem lojas, por exemplo.

3.3.9.6 Criação e manutenção de carrinhos

A criação e manutenção de produtos procede da mesma forma que a criação de lojas e produtos, com a mesma observação de que os produtos devem existir na rede P2P para o carrinho ser válido. Essa entidade possui uma marcação de que é privada, isso garante que elas não sejam indexadas pelos nós de busca. Apesar disso, outros nós mantém cópia dessa entidade e a rede inteira pode acessá-la sabendo o identificador desse carrinho.

Em uma arquitetura cliente-servidor, a privacidade desse dado é mantida, já que o servidor controla o acesso a esse dado.

3.3.9.7 Transação de Compra/Venda

Após a criação do carrinho, o cliente pode optar por efetuar a compra desses produtos. Para cada produto no carrinho é criada uma entidade de transação desse produto pelo nó que está comprando, essa transação contém informação do identificador do produto, o preço pago e o identificador do cliente que está comprando. Essa entidade também é guardada em um identificador que seja de posse desse nó. As transações são salvas com duas palavras-chave que indicam que se trata de uma transação e qual identificador do produto comprado. Os nós de busca indexam essa informação. Posteriormente, o vendedor pode buscar pela palavra-chave de transação e o identificador do produto, se o produto foi vendido, salvando uma cópia desse *P2PStorage*. Essa cópia é o indicativo que o vendedor tem de que já tomou ciência dessa venda. Em seguida, o vendedor publica na rede P2P uma entidade de venda para confirmar a transação, informando a confirmação da compra, do usuário e do produto, e assim está efetivada a venda. Essa confirmação contém a informação sobre a qual venda se refere, assim, o comprador pode buscar na rede se sua compra foi efetuada. A confirmação também pode ser negativa, o vendedor pode não ter o produto e publicar na rede P2P que a compra efetuada foi mal sucedida.

Os clientes e lojas nunca são notificados. Um cliente cria uma compra na rede P2P, para que o nó responsável pela loja tome ciência, ele deve fazer uma busca, procurando transações relacionadas ao produto. Da mesma forma, após a confirmação da loja, essa informação precisa ser ativamente buscada na rede.

Em uma arquitetura cliente-servidor, as transações podem ser notificadas imediatamente ao vendedor. Nessa arquitetura P2P é necessário que cada vendedor procure cada produto vendido. Da mesma forma, um comprador precisa buscar na rede P2P se

sua compra resultou em uma transação, o que também não é necessário na arquitetura cliente-servidor.

3.3.10 Conclusão

Essa seção apresentou os requisitos funcionais e não-funcionais. A arquitetura foi descrita pela sua visão geral estática, e depois seguiu a descrição do funcionamento de seus componentes. Por fim, exemplos de funcionamentos da arquitetura foram elucidados.

4 IMPLEMENTAÇÃO REFERENCIAL

4.1 Introdução

Para engendrar a implementação da arquitetura de *marketplace* P2P, é necessário definir quais tecnologias utilizar como protocolo P2P estruturado e protocolo de cooperação P2P. Essas escolhas devem satisfazer os requisitos definidos pela arquitetura e ser adaptadas de acordo. Além disso, a arquitetura possibilita, pelo seu servidor de *bootstrap*, que protocolos de conexão mediada, como o WebRTC, sejam utilizados para a conexão entre nós. A arquitetura também define que o sistema possa se comunicar com outra rede P2P através de seu componente de comunicação exterior. Uma rede P2P dessas também foi escolhida para exemplificar. Esta seção mostra a escolha dessas tecnologias e quais decisões foram tomadas na implementação.

Esta seção também descreve testes dessa implementação referencial de acordo com o requisitos funcionais. Os requisitos não funcionais são testados a partir de testes de inclusão e de *churn*.

Tabela 2 – Arquitetura e tecnologias escolhidas

Item da arquitetura	Tecnologia Escolhida
Rede P2P estruturada	BATON
Rede P2P de cooperação	OGP
Rede P2P externa	Blockstack
Protocolo de comunicação entre os nós	WebRTC

Fonte: Elaborado pelo autor

A tabela 2 mostra a relação de tecnologias escolhidas. A motivação e possíveis modificações foram detalhadas nas próximas seções

O código-fonte está disponível na Internet como projeto de código-aberto em https://github.com/gribeiro/mktplacep2p.

4.2 O protocolo P2P estruturado escolhido

Para cumprir com os requisitos da arquitetura, a implementação deve escolher um protocolo P2P estruturado. Esse protocolo deve manter uma tabela *hash* distribuída que permita a posse de certos identificadores no espaço identificador. Esse protocolo deve manter a redundância das informações na rede e deve ter *churn* com complexidade $O(log(n))$.

Averiguou-se que o protocolo BATON consegue cumprir esses requisitos com algumas modificações. A sua escolha foi motivada pela facilidade com que ele poderia ser adaptado para essa implementação referencial.

4.2.1 Adaptando o BATON ao Modelo P2P

Foi descrito um modelo geral de P2P e como operar esse modelo para se utilizar o *bootstrap* nos protocolos especificados. Para que qualquer protocolo de P2P funcione em um navegador por esse método, basta implementar o protocolo dentro do modelo.

Foram descritas as principais funcionalidades de como a rede deve proceder. Foi escolhida uma adaptação do algoritmo BATON, que utiliza a estrutura de árvore binária balanceada, como sendo o meio de comunicação dentro da rede de *marketplace*.

O espaço de identificadores descrito pela rede BATON é uma aplicação entre uma árvore binária e um par de números (l, n), onde l é um inteiro de 0 até L, n é um inteiro de 1 até 2^L, e L é o número total de níveis da árvore. Para cada nó na árvore é associado um nível e um número. O nível começa pela raiz da árvore com o número 0. Cada nível de um nó é exatamente um número maior que seu nó pai. A cada nível L, cada nó recebe um número de 1 até 2^L, mesmo se ele não estiver presente. Portanto, o par (l, n) identifica qualquer nó nessa árvore e é o espaço identificador desse protocolo. Esse espaço identificador é considerado ordenado pela in-ordem da árvore binária. A rede BATON não especifica um espaço de identificadores para recursos, é estabelecido apenas que cada nó é responsável por uma faixa de valores de acordo com a ordem linear dos nós. Para isso, será adotado um inteiro de 64 bits como índice dos recursos, sendo que cada par (l, n) possui uma única faixa de valor (max, min). Dado um momento da rede, existe uma bijeção entre esses espaços, no entanto, a entrada e a saída de nós faz com que a faixa se modifique com a evolução da rede no tempo.

Cada nó nessa árvore mantém conexão com seu pai, filhos e nós adjacentes. Além dessas conexões, são mantidas duas tabelas de roteamento especiais, uma chamada de tabela de roteamento à esquerda e a outra chamada de tabela de roteamento à direita. Para adaptar esse protocolo à arquitetura, os nós pai e adjacente mantêm uma cópia dos dados.

Essas tabelas contêm nós que estão no mesmo nível e possuem diferença de potência de 2 para com o nó, ou seja, estão a essa potência de 2 à distância do nó. Sendo a diferença negativa à esquerda, e a positiva à direita. Por exemplo, dado um nó de número 6, no nível 3 de uma árvore, onde os nós possíveis são numerados de 1 a 8, tem na sua tabela à esquerda, os nós de número 5, 4, 2, e à direita os nós 7, 8. A tabela é preenchida mesmo quando não existem os nós, nesse caso, é colocado o valor nulo. É possível que a tabela não possua nós, no caso de um nó com número 1,

por exemplo, não existem nós que estão a uma potência de 2 à esquerda desse nó. Uma tabela de roteamento, à esquerda ou à direita, é considerada cheia se não possui entradas nulas.

Com isso, pode-se definir a operação de entrada do nó, para isso, o processo de *bootstrap* deve estar completo e uma requisição *JOIN* deve ser enviada ao nó conectado. Existem duas fases nessa entrada, a primeira serve para determinar onde esse novo nó se encaixa. A segunda parte é o próprio processo de inclusão. Um nó entrante é aceito como filho de outro nó se as seguintes condições são verificadas:

- (1) A tabela de roteamento à direita está cheia;
- (2) A tabela de roteamento à esquerda está cheia;
- (3) O nó não possui um filho à direita, ou à esquerda;
- (4) Não existem nós com informação armazenada nessa faixa.

Caso as condições 1 ou 2 não sejam satisfeitas, o nó entrante é encaminhado para o nó pai. Quando as condições 1 e 2 são satisfeitas, mas não a condição 3, busca-se um nó sem filhos na tabela de roteamento, caso não seja encontrado nenhum nó nessa condição, a mensagem de JOIN é encaminhada para o próximo nó adjacente, considerando a ordenação linear in-ordem. A condição 4 é a adaptação para garantir posse dos dados pelos nós. Nessa implementação, se o nó nunca mais voltar, a rede ficará prejudicada, podendo perder a capacidade de salvar dados em uma grande faixa de valores.

A saída de um nó na rede pode ser voluntária ou por falha. Quando a saída é voluntária, a intenção é que a árvore não fique desbalanceada. Se o nó que deseja sair não possuir nó vizinho com filhos, basta passar todo seu conteúdo ao nó pai, e o nó pai se encarregará de passar seu conteúdo ao nó vizinho. Caso contrário, deverá encontrar um substituto qualquer.

4.3 O protocolo de cooperação P2P escolhido

A arquitetura estabeleceu menos requisitos em relação à escolha do protocolo de cooperação. Apenas foi estabelecido que o nó que tem o papel de fazer a conexão entre as redes deve manter uma referência do dado na rede P2P do *marketplace*.

Escolheu-se o protocolo OGP, pois ele é baseado em Kademlia, e isso torna simples sua implementação. Para utilizar essa cooperação é necessário que haja uma outra rede P2P, essa rede é a Blockstack, ela será utilizada para armazenar as imagens dos produtos.

4.3.1 Adaptando o OGP ao Modelo P2P

Aqui será analisado como adaptar o modelo de P2P da mesma maneira que foi feito para o BATON.

Essa rede possui tipos de nós, o FOGP e o LOGP. Para o nó LOGP não há o que acrescentar, já que o protocolo BATON fará com que as mensagens trafeguem até o nó FOGP. Logo, as definições aqui propostas para se adequar o OGP ao modelo P2P sempre se referem a nós FOGP.

A primeira parte a se definir é o Espaço de Identificadores e a métrica desse protocolo. O identificador de um nó nessa rede é dado por uma sequência única de $(n + m)$ bits, a primeira parte, o número n, o netID, refere-se ao identificador da rede *overlay*, aqui limitado a 8 bits, pretendendo-se conectar até 256 redes *overlays*. Já o número m, FOGPID, refere-se ao identificador de 48 bits da rede OGP, gerado de forma randômica. A rede BATON será identificada pela sequência netID 00000000, a rede Blockstack é identificada pelo netID 00000001. Essa sequência única de 64 bits é chamada de OGPID.

A métrica é definida por distâncias de ou-exclusivo, XOR, entre OGPIGP, da mesma forma que o protocolo Kademlia. Essa distância XOR garante que nós com mesmo prefixo, isto é, pertencentes à mesma rede, sejam considerados próximos.

A lista de vizinhos que cada nó FOGP mantém é uma lista de tamanho fixo com nós que possuam o mesmo prefixo netID. Caso a lista fique cheia, o nó FOGP mantém uma lista de candidatos, removendo nós inativos e substituindo-os por nós aleatórios da lista de candidatos.

O esquema de roteamento se dá entre redes *overlay*. Originalmente, o protocolo OGP permite conexão em qualquer direção, *unicast*, *multicast* e *broadcast* entre as diferentes redes. Nessa implementação proposta, apenas o *unicast* foi utilizado, como toda a requisição parte da rede P2P BATON, há somente a necessidade de se buscar alguma informação em uma rede fora, e nunca ocorrerá o contrário.

4.4 A possibilidade de escolha de um protocolo de conexão mediada

A presença do componente de *bootstrap* da forma como foi definido permite a utilização de um protocolo de conexão mediada, como o WebRTC. A utilização do WebRTC é muito importante para que o sistema *marketplace* possa rodar dentro de navegadores, por onde a maioria dos *marketplaces* são acessados. Portanto, a implementação referencial mostra como isso é feito.

4.4.1 Bootstrap e WebRTC

O componente de bootstrap consiste em fazer com que nó entre na rede e se comunique com os outros nós. Para que haja conexão entre dois navegadores, a implementação faz a conexão mediada enviando contratos SDP. Primeiro o cliente contata o servidor de *bootstrap* por HTTP, esse servidor envia o sistema a esse usuário. Primeiro é feita uma conexão WebSocket entre cliente e servidor de *bootstrap*.

Segue o detalhamento da implementação da conexão mediada desse componente:

- Após o acesso inicial do navegador, o cliente se anuncia para o servidor de *bootstrap* por um dos protocolos de conexão ativa;
- Esse anúncio faz com que o servidor de bootstrap inicial gere um identificador único para esse possível nó entrante;
- Faz-se uma requisição ao nó local P2P, que se comunica por WebRTC, para que seja gerado um contrato SDP.
- O contrato SDP é enviado ao nó entrante, que retorna outro contrato SDP, que então é enviado ao nó com o qual o nó entrante se comunicará.

O contrato SDP é enviado pelo protocolo de conexão ativa para o nó entrante. O nó entrante cria uma resposta SDP e envia de volta por WebSocket. O cliente está agora conectado com o nó entrante na rede P2P. Não há mais utilidade em manter a conexão ativa, podendo ser terminada a conexão pelo protocolo WebSocket.

Vale notar que esse processo é o mesmo para conexão com qualquer nó da rede P2P. Mas, após a conexão com um nó, via WebRTC, esses contratos SDP podem trafegar pela conexão por esse nó.

Quando o nó entrante está com a conexão WebRTC estabelecida com o nó P2P do servidor de *bootstrap*, está materializada a conexão. O adaptador de conexão entre camadas é utilizado para se adicionar essa conexão com o servidor de bootstrap como um vizinho do nó entrante, e então inicia-se o protocolo P2P.

4.5 Tecnologias de desenvolvimento

Optou-se por utilizar tecnologias disponíveis nos navegadores. Os componentes foram escritos de acordo com as tecnologias disponíveis para estes. O servidor *bootstrap* é um servidor *web* que estabelece uma conexão *WebSocket* com o nó. Os nós são conectados via WebRTC.

Os componentes foram desenvolvidos em *Typescript* e posteriormente são compilados a *Javascript*, que também é uma tecnologia disponível nos navegadores. Essa escolha também se deu para que os tipos definidos na seção anterior não fossem perdidos, e a checagem de tipo estática presente na linguagem pudesse ajudar a coibir erros de programação.

O servidor de *bootstrap* foi desenvolvido utilizando a mesma tecnologia, mas *server-side*. Por isso, foi utilizada a versão do *NodeJS*.

Depois, seguindo o próprio guia estabelecido, foram verificados os itens na ordem da descrição. O primeiro item é a formulação de um modelo P2P genérico. As classes foram traduzidas para o *TypeScript* diretamente.

4.6 Partes da implementação

Todos os papéis específicos, como servidor de *bootstrap*, nó de comunicação exterior e nó de busca, ficaram concentrados em um mesmo nó, ou seja, responsável por contatar a rede *BlockStack* e pelo protocolo de cooperação P2P, o OGP.

A rede P2P estruturada escolhida para o *marketplace*, BATON, foi implementada separadamente.

A busca implementada no servidor de *bootstrap* foi a seguinte: Ele guarda toda a informação da rede em memória e faz uma busca exaustiva dos termos. Essa informação contém, em particular, todos os produtos disponibilizados na rede. Em uma versão real desse *marketplace*, seria utilizado um projeto como *ElasticSearch* para manter essa busca, e vários outros nós responsáveis por essa busca seriam distribuídos na rede.

4.6.1 Componentes utilizados

O processo decisório para escolha dessas bibliotecas foi o seguinte, adotou-se o projeto mais recente, com mais atualizações em plataformas de código-aberto como *github*. Foram feitos dois tipos de reutilização, a primeira, como biblioteca no projeto de *TypeScript*, a segunda, como modelo para portabilidade das funcionalidades principais.

4.6.1.1 Bibliotecas

Seguem as bibliotecas relevantes de JavaScript utilizadas no projeto.

- "express", versão $4.14.0$, foi utilizado como plataforma MVC para entregar os artefatos estáticos do *bootstrap*;
- "simple-peer", versão $9.0.0$, foi utilizado para comunicação via *WebRTC*;

- "ws", versão $5.0.0$, foi utilizado como biblioteca de transmissão via *WebSocket*.

4.6.1.2 Portabilidade

Como visto, houve a portabilidade de sistemas já estabelecidos para criar essa implementação referencial.

Para a implementação do BATON, não foi encontrada nenhuma implementação funcional, no entanto, o projeto *"d-p2p-sim"*, que é um simulador de redes P2P, tem a simulação de BATON. Essa simulação foi adaptada do Java para o código base em TypeScript.

O protocolo OGP não é muito popular, e, por isso, foi difícil achar alguma implementação já disponível. Mas, esse protocolo é fortemente baseado no protocolo Kademlia, que possui diversas implementações. Buscou-se uma implementação simples para que ela fosse adaptada a funcionar como OGP. Foi escolhido projeto “kademlia-dht”.

Para o BlockStack, utilizou-se o cliente oficial, $blockstack.js$, versão $17.2.0$, em JavaScript dessa rede, sem nenhuma alteração.

4.7 Interface do usuário

A interface do usuário consiste na tela inicial, a tela de criação de loja e a tela de administração da loja. Todas essas telas são feitas em HTML.

A tela inicial abarca mais de uma função, podendo ser efetuada a busca por nome ou por loja, o usuário também pode adicionar e remover produtos para o seu carrinho.

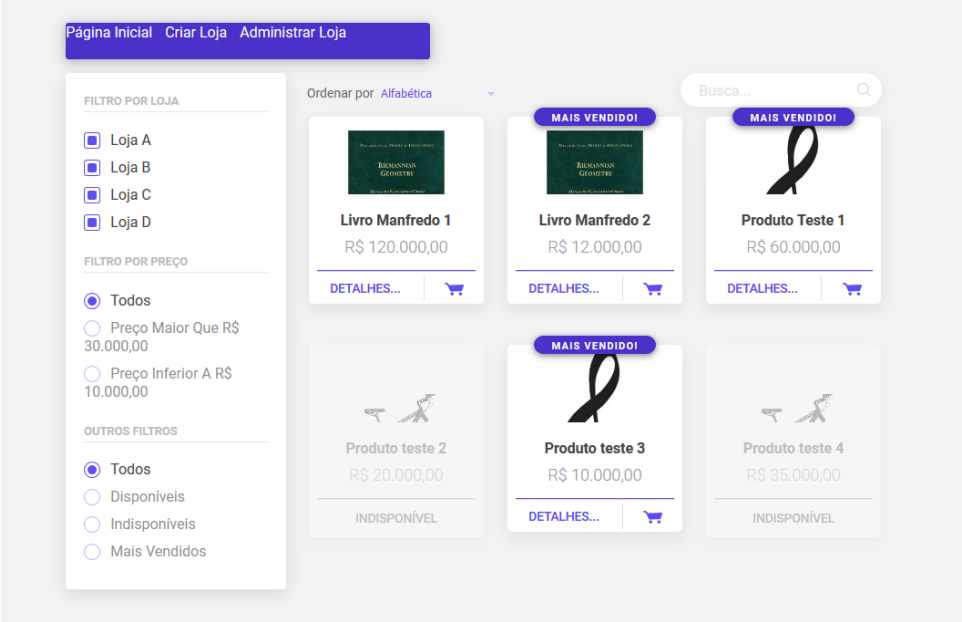
Como pode ser visto na figura 11, a tela inicial do usuário apresenta as principais opções do sistema: criar loja, administrar loja, pesquisar produtos e buscar por loja, além de filtros adicionais de campos específicos da entidade produto. Nessa tela o usuário pode efetuar todas as ações propostas para essa implementação referencial. Ao adicionar produtos ao carrinho, eles ficarão disponíveis abaixo do menu com o ícone de carrinho, para consulta. Na figura 12, nota-se que foram adicionados dois itens.

Essa tela mostra todos os produtos disponíveis na rede. A decisão foi adotada para facilitar a visualização dos dados propagados na rede.

Caso o usuário acesse o carrinho, aparecerá a tela informando os dados desse carrinho, com o sumário e um totalizador, como na figura 13. Nessa tela o usuário poderá remover os itens adicionados.

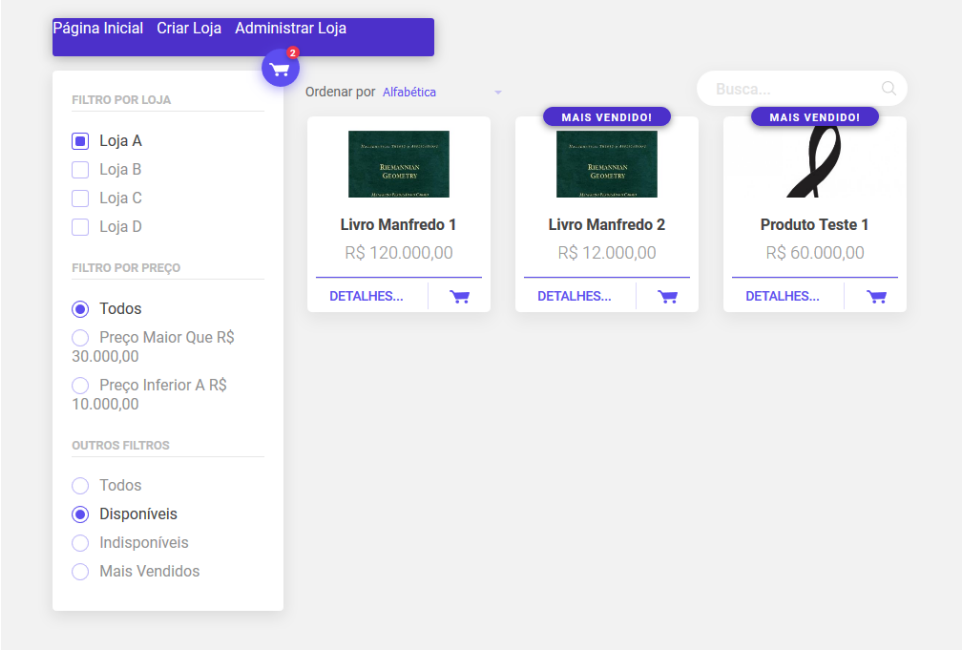
O usuário pode optar por visualizar os detalhes do produto na mesma tela, como na figura 14.

Figura 11 – Tela inicial da implementação referencial



Fonte: Elaborado pelo autor

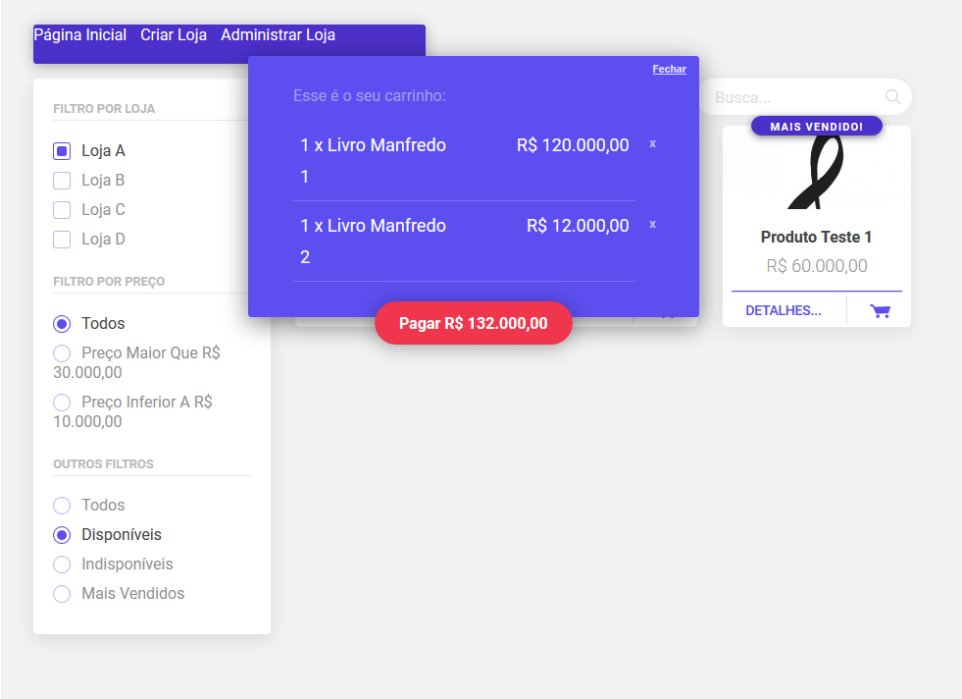
Figura 12 – Tela inicial com ícone do carrinho



Fonte: Elaborado pelo autor

A tela de criação de loja pode ser acessada por meio do menu no topo. Essa tela contém um formulário para que se adicione uma loja. A tela de administração de loja possui um campo para selecionar qual a loja deseja-se editar e um formulário para editar a relação de seus produtos. Na tela de administração de loja é possível visualizar as transações efetuadas, tanto para compradores como para vendedores, ali aparecem

Figura 13 – Tela do carrinho



Fonte: Elaborado pelo autor

todas as transações realizadas.

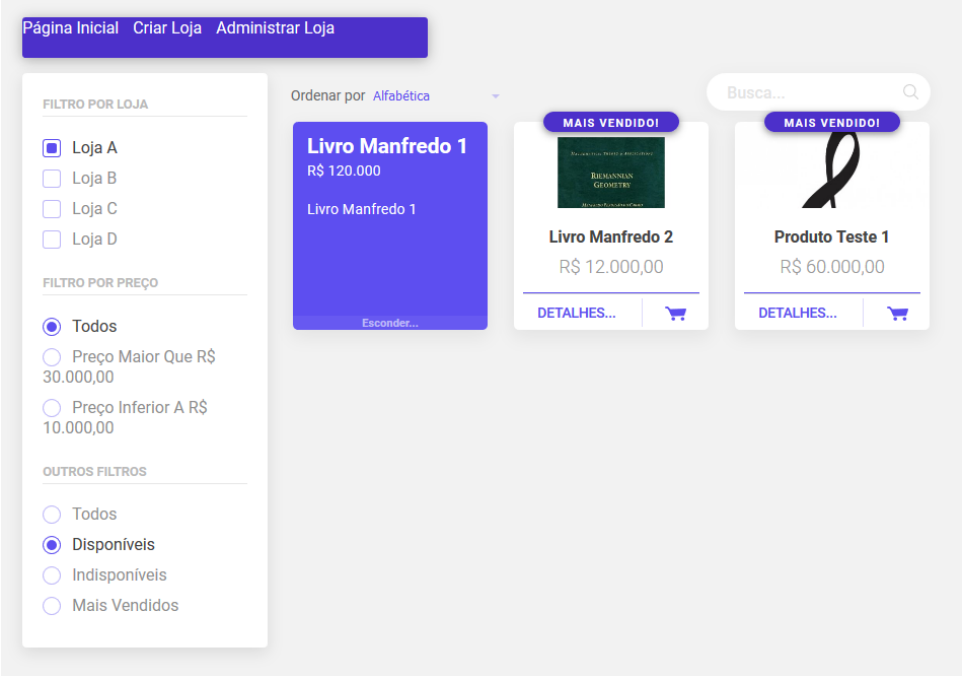
4.8 Atendimento aos requisitos funcionais

Para cada requisito do escopo reduzido, isto é, o bloco de requisitos GC e os requisitos V1 e V2, foram feitos testes para verificar o atendimento. Para verificar se cada requisito foi atendido, foram adotados critérios de sucesso. Esses critérios são resultados que o sistema deve apresentar, portanto adotou-se uma linguagem que envolvesse a interface e o requisito abstrato. Esses testes são baseados em testes de caso de uso.

Como se trata de um sistema P2P, os critérios devem ser verificados em múltiplos nós, isto é, certos resultados são esperados igualmente para todos os participantes. Mas outros resultados podem ser diferentes entre participantes, pois certas ações só podem ser executadas pelos nós que, no momento, são donos dos blocos P2PStorage, como, por exemplo, o carrinho de compras, a interface do usuário só mostra o carrinho que ele possui. Outras informações como lojas e produtos são públicas. As informações de compra e venda também são públicas, mas o sistema deve mostrar somente aquelas em que o comprador ou vendedor é parte da transação.

- **Critério para o requisito GC1 - Criação de loja**: A tela inicial deve mostrar novas lojas no filtro à esquerda. O nome da loja pode ser alterado e deve

Figura 14 – Detalhes do produto



Fonte: Elaborado pelo autor

aparecer na tela de administração de lojas apenas para o nó responsável por ela.

- **Critério para o requisito GC2 - Gerenciamento de catálogo**: Esse critério depende do anterior. O catálogo de uma loja pode ser modificado. Deve-se avaliar se um produto pode ser modificado e se essa modificação é populada. Também é necessário verificar a inserção e remoção de produtos da loja.
- **Critério para o requisito GC3 - Busca de produtos**: Um comprador pode buscar produtos por palavras-chave. Caso haja um produto de acordo com as necessidades do comprador, deve aparecer cada produto localizado com a descrição, preço e o vendedor.
- **Critério para o requisito V1 - Carrinhos de compra**: Deve-se testar a inserção e remoção de produtos no carrinho. O carrinho deve mostrar alguns detalhes dos produtos.
- **Critério para requisito V2 - Finalização de compra**: Deve-se testar uma transação de compra/venda. Um produto deve ser comprado, e, caso seja confirmado pelo vendedor, essa informação deve estar disponível tanto na visão do cliente quanto na visão do vendedor.

4.8.1 Procedimento de verificação dos requisitos funcionais

Para cada critério estabelecido foram feitos testes manuais no sistema para comprovar sua validade. Cada teste foi validado em duas pontas, primeiro foi verificado se o resultado esperado ocorreu para o nó que executou a ação e depois foi verificado em um nó diferente se o mesmo resultado estava disponível nesse nó. O nó que gera a ação será chamado de nó-gerador e um outro nó diferente deste será chamado de nó-verificador. Cada nó estava em uma máquina diferente. Na mesma máquina do nó-gerador estava o servidor de bootstrap. Seguem as ações realizadas:

- **Ação para verificar o requisito GC1**: Foi criada uma loja no nó-gerador, através do formulário na tela "Criar Loja". Primeiro foi verificado se a loja estava disponível no filtro da tela "Página Inicial". Depois, na tela "Administrar Loja", foi verificada a condição de que o nome da loja recém criada deveria estar disponível. No nó-verificador foi certificado, da mesma forma, se a nova loja estava disponível no filtro na tela "Página Inicial". Também foi verificado se na tela "Administrar Loja"se o nó-verificador a loja recém adicionada não estava na listagem.
- **Ação para verificar o requisito GC2**: Foi adicionada uma loja pelo nó-gerador. Depois, na tela "Administrar Loja", produtos foram adicionados, modificados e removidos. Conforme cada ação era feita, a repercussão dessas modificações foi avaliada na tela "Página Inicial"
- **Ação para verificar o requisito GC3**: Em ambos os nós, foram feitas buscas de produtos que estavam listados na "Página Inicial", uma vez que todos os produtos disponíveis estão listados nessa página.
- **Ação para verificar o requisito V1**: Foram adicionados produtos diferentes nos carrinhos de ambos os nós, depois foi verificado se cada carrinho estava com os mesmos produtos pré-selecionados.
- **Ação para verificar o requisito V2**: Uma compra foi feita no sistema, e foi verificado se as informações apareciam para o comprador e para o vendedor.

4.9 Testes de performance e disponibilidade

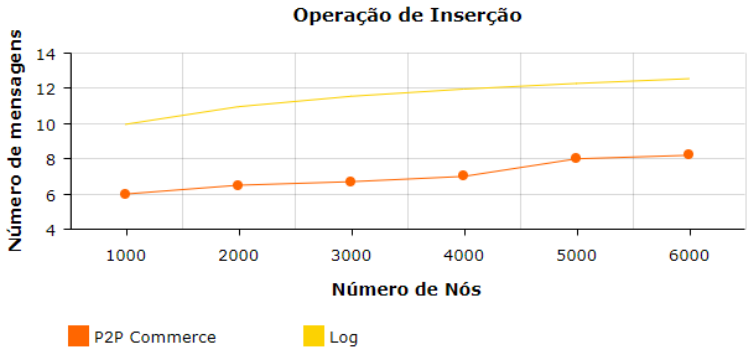
Para testar a viabilidade de performance do sistema, o que verifica o requisito não funcional NF4, foram executadas ações de inserção dos dados na rede P2P, conforme o crescimento da rede. Também foi testado o *churn* da rede. Para testar de forma efetiva, deve-se considerar que uma rede P2P é heterogênea, não estando garantidos tempos de respostas de pacotes ou acessos. Tendo em vista essa heterogeneidade descrita, a métrica avaliada foi a de número de mensagens trafegadas, assim como

no trabalho de Jagadish (2005), pois caso as mensagens se mantenham-se dentro dos limites teóricos dos algoritmos adotados, e presumindo-se uma razoabilidade dos recursos, como conexão, latência e performance computacional dos nós, pode-se inferir que, em geral, a performance é garantida. Em um ambiente real, essa presunção não é válida. Essa viabilidade é limitada a essa presunção. Seria possível a utilização de outra métrica, como tempo de resposta, mas essa métrica só serviria no caso em que todos os nós têm a mesma configuração, o que não é possível em uma rede P2P pública.

Para ambos os testes foi criado um servidor que recebia todas as mensagens trafegadas na rede e mantinha a média de mensagens trafegadas por inserção. Cada nó informava se estava entrando, saindo ou inserindo alguma informação.

No provedor de serviços de nuvem da Amazon, foram utilizadas 6 máquinas $c4.xlarge$. Em cada máquina, subiam 1000 nós, esses nós inseriam informações em um período de tempo aleatório, com probabilidade uniforme, entre 0 e 10 segundos . Primeiro foi criada uma rede com 1000 nós, depois uma outra máquina foi ligada, totalizando 2000 nós. Sucedeu-se dessa maneira até atingir a marca de 6000 nós. Antes de conectar mais um lote de nós, as informações de número de mensagens médias trafegadas por cada informação inserida foi coletada.

Figura 15 – Operação de inserção

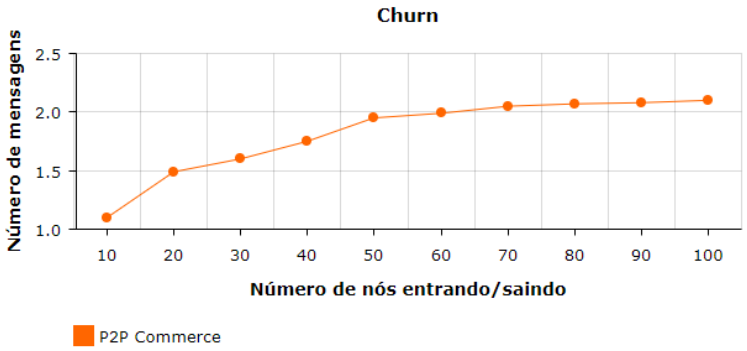


Fonte: Elaborado pelo autor

A figura 4.9 mostra a média de mensagens trafegadas para a inserção. A operação de inserção manteve-se abaixo do limite teórico da rede BATON e de outras redes P2P, que é de $O(log(n))$. Essa figura também mostra a comparação da média de mensagens trafegadas com a função $log(n)$.

O *churn* foi testado pela entrada e saída simultâneas de nós na rede, uma das formas de avaliá-lo. O número de mensagens refere-se ao número médio de mensagens que teve de ser reenviado devido a essa mudança. Por exemplo, um nó poderia estar no processo de inserir uma informação na rede, e mais uma mensagem

Figura 16 – *Churn*



Fonte: Elaborado pelo autor

teve de ser enviada. O *churn* não foi totalmente testado, é necessário avaliar mais métricas, como a saída e a entrada massiva, por exemplo.

Nesse teste, todas as 6 máquinas foram ligadas, formando uma rede de 6000 nós, com o mesmo algoritmo de inserção de dados em um período aleatório. Primeiro, mediu-se a média de mensagens trafegadas para se obter qual seria o normal, sem nenhuma ação dos nós, para estabelecer um parâmetro. Esse parâmetro pode ser visto na figura , para 6000 nós, em média trafegam cerca de 12 mensagens por inserção. Depois, em um segundo teste, a cada minuto, 10 nós aleatórios eram retirados da rede e entravam mais 10, depois de 5 minutos, a média de mensagens foi retirada. Assim foi feito até chegar em 100 nós nesse estado de entrada e saída simultânea. A figura mostra a diferença da média entre cada teste e o parâmetro. Por exemplo, para 100 nós entrando e saindo simultaneamente, cerca de 14 mensagens, em média, trafegavam por inserção feita, 12 do parâmetro somadas a 2 mensagens extras causadas pela mudança na rede, um aumento de cerca de 16%. O número de entrada e saída de nós, quando não é massiva, não altera significativamente a performance da rede P2P.

Durante os dois testes, o sistema ficou disponível, o que cumpre o requisito NF3.

Os testes mostram que os requisitos não-funcionais NF3 e NF4 foram atendidos. Mas, apesar disso, mais testes com números mais próximos da realidade de um *marketplace* são necessários para que se confirme essa expectativa.

4.10 Relação de requisitos e testes executados

Tabela 3 – Relação de requisitos e testes executados

Requisito	Teste executado
GC1 - Criação de loja	Ações executadas na interface de usuário
GC2 - Gerenciamento de catálogo	Ações executadas na interface de usuário
GC3 - Busca de produtos	Ações executadas na interface de usuário
V1 - Carrinhos de compra	Ações executadas na interface de usuário
V2 - Finalização de compra	Ações executadas na interface de usuário
NF3 - Acesso disponível	Teste de *churn* e de inclusão de dados
NF4 - Performance	Teste de *churn* e de inclusão de dados

Fonte: Elaborado pelo autor

A tabela 3 mostra uma relação dos requisitos e os testes que foram executados. Foram executadas ações na interface do usuário para testar a compatibilidade com os requisitos funcionais GC1, GC2, GC3, V1 e V2, e testes em relação às mensagens trafegadas foram feitos para averiguar os requisitos não funcionais NF3 e NF4.

4.11 Conclusão

Nessa seção foram vistos de que forma se procedeu a implementação, que tecnologias foram utilizadas e que testes foram executados nessa implementação referencial.

Os testes executados na rede P2P garantem uma performance semelhante a teórica dos protocolos subjacentes. Pode-se afirmar que, para essa versão reduzida de *marketplace*, esse sistema de *marketplace* P2P mostrou-se tecnicamente viável, de acordo com o que foi proposto por Jagadish (2005).

Os requisitos foram testados por casos de uso. Esses testes basearam-se nos casos de uso derivados dos requisito GC1, GC2, GC3, V1 e V2.

Os requisitos elencados, um subconjunto dos requisitos funcionais, os requisitos não funcionais de performance e de disponibilidade foram atendidos. Porém, o uso em escala massiva desta proposta arquitetural deve ser antes submetido a testes mais amplos, em condições mais reais de utilização, algo fora do escopo deste trabalho.

5 CONCLUSÃO

5.1 Resumo

Os sistemas P2P podem ser beneficiados pela sua descentralização, pelo compartilhamento de recursos e pela distribuição geográfica. Já existem muitos sistemas que estão utilizando essa arquitetura. Como sistema compartilhamento de arquivos, serviços de mensageiro instantâneo e armazenamento de dados.

Os *marketplaces* são importantes ferramentas de comércio digitais, movimentando grandes quantias monetárias anualmente. A maioria dos *marketplaces* está na arquitetura cliente-servidor. Eles podem possuir pontos de falhas devido a sua centralização, demandam muitos recursos e precisam recorrer a tecnologias para serem distribuídos geograficamente.

A introdução de uma arquitetura P2P para *marketplaces* pode criar um tipo de abordagem possivelmente interessante para os problemas mencionados. Esta dissertação documentou o trabalho realizado para descentralizar um *marketplace* utilizando tecnologia P2P, em que recursos podem ser compartilhados e a distribuição geográfica pode ser mais facilmente atendida. Para a base dessa arquitetura foram propostos os requisitos básicos funcionais e não-funcionais de um *marketplace*. Esses requisitos mostraram-se muito detalhados para o enfoque na descentralização, e, por isso, posteriormente foram selecionados alguns deles para compor o corpo deste trabalho.

O objetivo principal foi propor uma arquitetura P2P de um *marketplace*, contrastando com os sistemas tradicionais, nos quais os recursos são centralizados. A arquitetura proposta utilizou requisitos propostos na literatura. E, como objetivo secundário, foi desenvolvida uma implementação referencial da arquitetura, que foi utilizada para testá-la.

A proposta da arquitetura foi feita descrevendo primeiro estaticamente os principais componentes, depois mostrando o comportamento de cada um dentro do sistema.

A implementação foi feita escolhendo soluções pré-existentes. Essa implementação se deu com a intenção de ser uma implementação referencial, que foi testada, então, em duas métricas. Uma de inserção e outra de *churn*. Essas métricas demonstraram que, a uma primeira vista, a implementação referencial está adequada.

Algumas decisões importantes nortearam o formato do trabalho, muitas delas para limitar o seu escopo. Descreveram-se os requisitos fundamentais de uma plataforma de *marketplace*, depois uma gama menor de requisitos foi selecionada

para que a complexidade de uma aplicação *marketplace* não ofuscasse a proposta de descentralização. Os testes adotados também foram limitados, restringindo a mostrar um indício de viabilidade.

Outras decisões impactaram na arquitetura. Optou-se por separar o componente de *bootstrap* do componente de modelo P2P. Essa decisão foi importante para permitir que o processo de entrada na rede não influenciasse esse modelo P2P.

Após a descrição da arquitetura e o engendramento da implementação referencial, foram executados os testes propostos para testar a viabilidade da rede P2P. Também foram feitos testes de aceitação baseados nos casos de uso. Esses testes foram propostos adequando-se a linguagem estabelecida pela descrição da implementação referencial.

Neste trabalho propôs-se a fazer um sistema descentralizado. No entanto, algumas dificuldades foram encontradas. A complexidade dos requisitos de um *marketplace* não torna possível mostrar esse sistema descentralizado na sua totalidade, com todos os requisitos básicos. Além dessa redução, a descentralização na literatura e nos projetos encontrados não são focados para funcionar nos navegadores, as soluções encontratadas tinham complexidade muito menor e serviam apenas para distribuir arquivos ou documentos. O protocolo BATON não possui implementações de grande relevância, apesar de ser melhor do que o protocolo Chord em alguns aspectos. A utilização do protocolo BATON foi muito importante, pois ele é pouco impactado por *churn*. Essa falta de recursos relacionados ao BATON resultou em uma implementação baseada em uma solução feita para um simulador. Seria muito melhor se houvesse uma versão mais madura desse protocolo. A tecnologia WebRTC, apesar de fornecer os canais de dados necessários para este trabalho, tem como principal funcionalidade estabelecer comunicação de áudio e vídeo entre navegadores. Isso pode ser notado quando se deseja utilizar tais canais, pois seu acesso é restrito a poucas interfaces da tecnologia.

5.2 Contribuição

O trabalho descreveu um *marketplace* descentralizado satisfazendo alguns requisitos mínimos. Além disso, esta dissertação é um relato de como utilizar P2P e WebRTC, em geral. Também foram descritos requisitos de um *marketplace*, podendo ou não ser P2P. Sendo assim, a contribuição deste trabalho deu-se em três campos: plataforma *marketplace* P2P, utilização de cooperação entre redes P2P e P2P em navegadores.

5.3 Limitações do trabalho

A proposta da arquitetura de um *marketplace* P2P tem indícios de ser viável, no entanto, só foram considerados alguns dos requisitos propostos para um *marketplace* totalmente funcional. Através da implementação referencial, foi demonstrado que as tecnologias dos navegadores são suficientes para que se construa uma aplicação P2P de *marketplace*. Mas não há indício de que o uso além do apresentado por esta implementação referencial é possível.

Os testes executados demonstram que a rede está dentro de parâmetros ideais de redes P2P. Porém mais testes poderiam ter sido feitos, testando, por exemplo, a falha geral quando grande parte da rede acaba desconectando da rede por falha técnica.

A complexidade de um *marketplace* vai muito além do proposto. Esse escopo reduzido de requisitos atendidos não são os mais complexos. Ações de compra eletrônica como o processamento de um cartão de crédito ou a emissão de notas fiscais, com certeza seriam um desafio a essa arquitetura.

5.4 Trabalhos Futuros

Os trabalhos futuros podem se basear em dois aspectos faltantes. O primeiro diz respeito a completar todos os requisitos de um *marketplace* e não utilizar o escopo reduzido, como este trabalho fez. O segundo aspecto é testar os requisitos não funcionais em condições mais reais de utilização em relação à quantidade de nós e informações.

REFERÊNCIAS

ALAMI, D.; RODRÍGUEZ, M.; JANSEN, S. Relating health to platform success: Exploring three e-commerce ecosystems. In: ACM. **Proceedings of the 2015 European Conference on Software Architecture Workshops**. [S.l.], 2015. p. 43.

ALI, M. et al. Blockstack: A global naming and storage system secured by blockchains. In: **USENIX Annual Technical Conference**. [S.l.: s.n.], 2016. p. 181–194.

ANDROUTSELLIS-THEOTOKIS, S.; SPINELLIS, D. A survey of peer-to-peer content distribution technologies. **ACM computing surveys (CSUR)**, ACM, v. 36, n. 4, p. 335–371, 2004.

BERKES, J. E. Decentralized peer-to-peer network architecture: Gnutella and freenet. **University of Manitoba Winnipeg, Manitoba, Canada**, 2003.

CAMARILLO, G. et al. **Peer-to-peer (p2p) architecture: definition, taxonomies, examples, and applicability**. [S.l.], 2009.

CHAUDHARY, M. N. A survey on peer to peer system applications. 2014.

CLARKE, I. et al. Freenet: A distributed anonymous information storage and retrieval system. In: SPRINGER. **Designing Privacy Enhancing Technologies**. [S.l.], 2001. p. 46–66.

DAINOTTI, A. et al. Analysis of country-wide internet outages caused by censorship. In: ACM. **Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference**. [S.l.], 2011. p. 1–18.

DOUCEUR, J. R. The sybil attack. In: SPRINGER. **International workshop on peer-to-peer systems**. [S.l.], 2002. p. 251–260.

DUMITRIU, D. et al. Denial-of-service resilience in peer-to-peer file sharing systems. In: ACM. **ACM SIGMETRICS Performance Evaluation Review**. [S.l.], 2005. v. 33, n. 1, p. 38–49.

FINGAR, P.; KUMAR, H.; SHARMA, T. 21st century markets: From places to spaces. **First Monday**, v. 4, n. 12, 1999.

GIOVANOLI, C.; PULIKAL, P.; GRIVAS, S. G. E-marketplace for cloud services. **CLOUD COMPUTING**, p. 76–83, 2014.

HELANDER, M. G.; KHALID, H. M. Modeling the customer in electronic commerce. **Applied Ergonomics**, Elsevier, v. 31, n. 6, p. 609–619, 2000.

HUANG, Z.; BENYOUCEF, M. From e-commerce to social commerce: A close look at design features. **Electronic Commerce Research and Applications**, Elsevier, v. 12, n. 4, p. 246–259, 2013.

JAGADISH, H. V.; OOI, B. C.; VU, Q. H. Baton: A balanced tree structure for peer-to-peer networks. In: VLDB ENDOWMENT. **Proceedings of the 31st international conference on Very large data bases**. [S.l.], 2005. p. 661–672.

KARGER, D. et al. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In: ACM. **Proceedings of the twenty-ninth annual ACM symposium on Theory of computing**. [S.l.], 1997. p. 654–663.

KLINGBERG, T.; MANFREDI, R. **The Gnutella RFC, version 0.6**. [S.l.]: June, 2002.

KUBJAS, I. Using blockchain for enabling internet voting. 2017.

LI, F.; LI, Y. Usability evaluation of e-commerce on b2c websites in china. **Procedia Engineering**, Elsevier, v. 15, p. 5299–5304, 2011.

LINDEMANN, M. A.; SCHMID, B. F. Framework for specifying, building, and operating electronic markets. **International Journal of Electronic Commerce**, Taylor & Francis, v. 3, n. 2, p. 7–21, 1998.

LIQUORI, L.; TEDESCHI, C.; BONGIOVANNI, F. Babelchord: a social tower of dht-based overlay networks. In: IEEE. **Computers and communications, 2009. iscc 2009. ieee symposium on**. [S.l.], 2009. p. 307–312.

LIU, Q.; SUN, X. Research of web real-time communication based on web socket. **International Journal of Communications, Network and System Sciences**, Scientific Research Publishing, v. 5, n. 12, p. 797, 2012.

MALATRAS, A. State-of-the-art survey on p2p overlay networks in pervasive computing environments. **Journal of Network and Computer Applications**, Elsevier, v. 55, p. 1–23, 2015.

NAKAMOTO, S. **Bitcoin: A peer-to-peer electronic cash system**. 2008.

NGO, H. G. **From inter-connecting P2P overlays to co-operating P2P systems**. Tese (Doutorado) — Université Nice Sophia Antipolis; Hanoi University of sciences (Hanoi), 2013.

ROUSSOPOULOS, M. et al. 2 p2p or not 2 p2p? In: SPRINGER. **International Workshop on Peer-to-Peer Systems**. [S.l.], 2004. p. 33–43.

SCHOLLMEIER, R. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In: IEEE. **Peer-to-Peer Computing, 2001. Proceedings. First International Conference on**. [S.l.], 2001. p. 101–102.

SHANG, G. et al. How much do online consumers really value free product returns? evidence from ebay. **Journal of Operations Management**, Elsevier, v. 53, p. 45–62, 2017.

SILVERSTON, T. et al. Traffic analysis of peer-to-peer iptv communities. **Computer Networks**, Elsevier, v. 53, n. 4, p. 470–484, 2009.

STOICA, I. et al. Chord: A scalable peer-to-peer lookup service for internet applications. **ACM SIGCOMM Computer Communication Review**, ACM, v. 31, n. 4, p. 149–160, 2001.

STUTZBACH, D.; REJAIE, R.; SEN, S. Characterizing unstructured overlay topologies in modern p2p file-sharing systems. **IEEE/ACM Transactions on Networking**, IEEE, v. 16, n. 2, p. 267–280, 2008.

SUN, D.; FANG, L.; LI, J. Research on the development of cross-border e-commerce in port cities—a case of manzhouli city. In: SPRINGER. **Proceedings of the Fourth International Forum on Decision Sciences**. [S.l.], 2017. p. 589–596.

SUNDARESWARAN, S.; SQUICCIARINI, A.; LIN, D. A brokerage-based approach for cloud service selection. In: IEEE. **Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on**. [S.l.], 2012. p. 558–565.

UBERTI, J.; JENNINGS, C.; RESCORLA, E. Javascript session establishment protocol. **draft-ietf-rtcweb-jsep-01**, 2012.

VOGT, C.; WERNER, M. J.; SCHMIDT, T. C. Leveraging webrtc for p2p content distribution in web browsers. In: IEEE. **Network Protocols (ICNP), 2013 21st IEEE International Conference on**. [S.l.], 2013. p. 1–2.

WAN, Z.; DENG, R. H.; LEE, D. Electronic contract signing without using trusted third party. In: SPRINGER. **International Conference on Network and System Security**. [S.l.], 2015. p. 386–394.

WANG, S. **uopStore: an E-commerce Platform with a Peer-to-Peer Infrastructure**. Tese (Doutorado) — Université d'Ottawa/University of Ottawa, 2014.

Apêndices

APÊNDICE A – CASOS DE USO DO *MARKETPLACE*

A.1 Introdução

Esse apêndice apresenta os casos de uso de acordo com cada requisito utilizados pela arquitetura. Foram utilizados os seguintes requisitos:

- Requisito GC1 - **Criação de loja**: Um vendedor deve poder criar uma ou mais lojas.
- Requisito GC2 - **Gerenciamento de catálogo**: Um vendedor deve poder gerenciar um catálogo de produtos de cada uma de suas lojas. Esse catálogo será uma lista de produtos com palavras-chave, descrição e valores.
- Requisito GC3 - **Busca de produtos**: Um comprador pode buscar produtos por palavras-chave. Caso haja um produto de acordo com as necessidades do comprador, deve aparecer cada produto localizado com a descrição, preço e o vendedor.
- Requisito V1 - **Carrinhos de compra**: Um comprador, após navegar pelos catálogos, tem a opção de mover certos produtos para o seu carrinho. Esse carrinho de compra é editável: é possível adicionar ou remover produtos, ou alterar sua quantidade.
- Requisito V2 - **Finalização de compra**: O comprador pode optar por seguir com a compra dos produtos no carrinho. Essa finalização deve apresentar a lista de produtos, com taxas e frete.

A.1.1 Gerenciamento de loja (Requisitos GC1 e GC2)

Esses são os casos de uso da parte de gerenciamento de loja do *marketplace*.

A.1.1.1 Criar loja

- **Pré-condição**
  O usuário deve estar presente no sistema e identificado de alguma maneira.
- **Fluxo**
  Um usuário pode criar uma nova loja para operar uma loja online. Ele deve fornecer as seguintes informações:
  - Nome da loja;

  - Descrição da loja;
  - Produtos disponíveis.

- **Pós-condição**
  Uma nova loja associada a esse usuário será criada e armazenada. Posteriormente, o usuário poderá gerencia sua loja e seus produtos.

A.1.1.2 Atualizar loja

- **Pré-condição**
  O usuário deve estar presente no sistema, identificado de alguma maneira e possuir uma loja.

- **Fluxo**
  O usuário poderá editar as informações de sua loja, nome e descrição. Também poderá gerenciar seus produtos.

- **Pós-condição**
  As informações devem ser atualizadas no sistema. O usuário poderá escolher se notifica os outros usuários dessa modificação.

A.1.1.3 Remover loja

- **Pré-condição**
  O usuário deve estar presente no sistema, identificado de alguma maneira e possuir a loja que desejar remover.

- **Fluxo**
  O usuário pode remover sua loja do sistema. No entanto, as vendas processadas devem ser mantidas.

- **Pós-condição**
  A loja é removida do sistema, e essa informação é propagada.

A.1.2 Gerenciamento de produtos (Requisito GC2)

Esses são os casos de uso da parte de gerenciamento de produtos do *marketplace*.

A.1.2.1 Adicionar Produto a Loja

- **Pré-condição**
  O usuário deve estar presente no sistema, identificado de alguma maneira e possuir a loja em que deseja adicionar os produtos.

- **Fluxo**
  Um usuário pode criar uma novo produto para vender em sua loja. Ele deve fornecer as seguintes informações:
  - Nome do produto;
  - Descrição do produto;
  - Indicativo de que é um produto mais vendido ou não;
  - Quantidade Disponível.
- **Pós-condição**
  Um novo produto associada a esse usuário e sua loja será criado e armazenado no sistema. Posteriormente, o usuário poderá modificar essas informações.

A.1.2.2 Atualizar produto

- **Pré-condição**
  O usuário deve estar presente no sistema, identificado de alguma maneira, possuir a loja em que deseja adicionar os produtos e a loja deve possuir o produto que se pretende modificar.
- **Fluxo**
  O usuário poderá editar as informações de produtos associados a sua lojas. Essas informações são o nome, a descrição do produto, Indicativo de que é um produto mais vendido ou não e quantidade disponível.
- **Pós-condição**
  O produto é atualizado na loja. Posteriormente, o usuário poderá modificar essas informações.

A.1.2.3 Remover produto

- **Pré-condição**
  O usuário deve estar presente no sistema, identificado de alguma maneira, possuir a loja em que deseja adicionar os produtos e a loja deve possuir o produto que se pretende excluir.
- **Fluxo**
  O usuário pode remover o produto selecionado de sua loja. No entanto, as vendas processadas devem ser mantidas.
- **Pós-condição**
  O produto é removido.

A.1.3 Gerenciamento de carrinho (Requisito V1)

A.1.3.1 Adicionar produtos ao carrinho

- **Pré-condição**
  O usuário deve estar presente no sistema, identificado de alguma maneira. O usuário deve selecionar o produto de uma loja. A loja deve existir no sistema e o produto também deve existir.
- **Fluxo**
  O usuário seleciona o produto, e esse produto é adiciona a um carrinho relacionado a ele. No carrinho está presente o valor total de produtos presentes
- **Pós-condição**
  Um produto foi adicionado ao carrinho. O produto pode ser removido do carrinho. O valor total foi atualizado.

A.1.3.2 Remover produtos do carrinho

- **Pré-condição**
  O usuário deve estar presente no sistema, identificado de alguma maneira. O usuário deve selecionar o produto do carrinho que deve remover. A loja deve existir no sistema e o produto também deve existir.
- **Fluxo**
  O usuário seleciona o produto do carrinho que deseja remover. Esse produto é removido do carrinho. O valor total do carrinho deve ser atualizado.
- **Pós-condição**
  Um produto foi removido carrinho. O valor total foi atualizado.

A.1.4 Busca de produtos (Requisito GC3)

A.1.4.1 Busca por palavra-chave

- **Pré-condição**
  O usuário deve estar presente no sistema, identificado de alguma maneira.
- **Fluxo**
  O usuário informa palavras-chave de busca ao sistema. O sistema retorna uma lista de produtos.
- **Pós-condição**
  O usuário possui uma lista de produtos relacionadas à palavra-chave. Ele pode adicionar cada um desses produtos ao carrinho.

A.1.5 Transação (Requisito V2)

A.1.5.1 Transação de compra

- **Pré-condição**
  O usuário deve estar presente no sistema, identificado de alguma maneira. O carrinho deve conter produtos de alguma loja. Tanto produto e loja devem existir no sistema.

- **Fluxo**
  O usuário pode efetuar a compra de todos os produtos efetuados no carrinho. O usuário informa ao sistema que deseja comprar todos os produtos listados pelo valor total informado no carrinho.

- **Pós-condição**
  Para cada produto do carrinho, o usuário possui uma compra relacionada a ele. O vendedor pode consultar essa informação no sistema.

A.1.5.2 Transação de venda

- **Pré-condição**
  O usuário deve estar presente no sistema, identificado de alguma maneira. Algum outro usuário efetuou a compra de um produto da loja. O produto deve estar disponível.

- **Fluxo**
  O usuário recebe que uma compra foi efetuado de um produto de uma de suas lojas. Essa compra pode ser confirmada se houve disponibilidade do produto. O vendedor pode criar uma transação de venda bem-sucedida ou malsucedida.

- **Pós-condição**
  O produto que tinha transação de compra associada, agora tem a sua resposta do vendedor, com a venda autorizada ou não.

APÊNDICE B – ESQUEMAS JSONSCHEMA

B.1 Introdução

Nesse é apêndice são apresentados os principais tipos de dados da arquitetura *marketplace* P2P. Esses sãos os tipo utilizados no componente de armazenamento.

B.2 MensagemIncial

Essa é mensagem que todo nó recebe após entrar no sistema.

```
{
    "title": "MensagemIncial",
    "type": "object",
    "properties": {
        "suaFaixa": {
            "type": "array", "items": { "type": "integer" }
        },
        "suaFaixaDono"": {
                 "type": "array", "items": { "type": "integer" }
        },
        "listaNosBusca" : {
                "type": "array":, "items": {"type" : "integer" }
        },
        "seuId": {
                "type": "integer"
        }
    },
    "required": ["suaFaixa", "suaFaixaDono", "listaNosBusca", "seuId"]
}
```

B.3 P2PStorage

Todos os dados possuem estão dentro desse formato. As entidade de *marketplace* são guardadas em "valor". Seu tipo é descrito em "tipo".

```
{
    "title": "P2PStorage",
    "type": "object",
    "properties": {
```

```
        "id": {
            "type": "integer"
        },
        "tipo": {
            "type": "object"
        },
        "valor": {
                "type": "object"
        },
        "itensExteriores": {
                "type": "array", "items": { "type": "integer" }
        },
        "privado": {
                "type": "boolean"
        }
        "palavrasChave": {
                 "type": "array", "items": { "type": "string" }
        },
    },
    "required": ["id", "tipo", "valor", "privado", "palavrasChave"]
}
```

B.4 Loja

Esse é um modelo de exemplo de uma loja, com descrição, nome e produtos.

```
{
    "title": "Loja",
    "type": "object",
    "properties": {
        "id": {
            "type": "integer"
        },
        "descricao": {
            "type": "string"
        },
        "nome": {
            "type": "string"
        },
        "produtos": {
```

```
                "type": "array", "items": { "type": "object" }
        }
    },
    "required": ["descricao", "nome"]
}
```

B.5 Produto

Esse é um modelo de exemplo de um produto, com descrição, nome, quantidade e indicador de mais vendido.

```
{
    "title": "Produto",
    "type": "object",
    "properties": {
        "id": {
            "type": "integer"
        },
        "descricao": {
            "type": "string"
        },
        "nome": {
            "type": "string"
        },
        "quantidade": {
                "type": "integer"
        },
        "maisVendido": {
                "type": "boolean"
        }

    },
    "required": ["descricao", "nome", "quantidade"]
}
```

B.6 Carrinho

Esse é um modelo de exemplo de um carrinho, com produtos e total.

```
{
    "title": "Carrinho",
```

```
    "type": "object",
    "properties": {
        "id": {
            "type": "integer"
        },
        "produtos": {
                "type": "array", "items": { "type": "object" }
        },
        "total": {
            "type": "number"
        },
    },
    "required": ["produtos"]
}
```

B.7 Compra

Esse é um modelo de exemplo de transação de compra. Apenas é necessário informar o produto.

```
{
    "title": "Compra",
    "type": "object",
    "properties": {
        "id": {
            "type": "integer"
        },
        "produto": {
                "type": "object"
        },
    },
    "required": ["produto"]
}
```

B.8 Venda

Esse é um modelo de exemplo de confirmação da transação de compra. Apenas é necessário informar qual é a compra, e se ela está confirmada.

```
{
    "title": "Venda",
```

```
    "type": "object",
    "properties": {
        "id": {
            "type": "integer"
        },
        "compra": {
                 "type": "integer"
        },
        "confirmada": {
                "type": "boolean"
        }
    },
    "required": ["produto", "confirmada"]
}
```