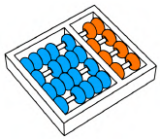

Balanceador de carga distribuído tolerante a falhas usando tv boxes reaproveitadas

T. G. Bannwart *L. F. F. Gonzalez* *Gustavo P. C. P. da Luz*
J. F. Borin

Technical Report - IC-26-02 - Relatório Técnico
July - 2026 - Julho

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Balanceador de carga distribuído tolerante a falhas usando tv boxes reaproveitadas

Tiago Godoi Bannwart [*] Luis Fernando Gomez Gonzalez [†]
Gustavo P. C. P. da Luz [‡] Juliana Freitag Borin [§]

Resumo

Ações de fiscalização conduzidas pela Anatel e pela Receita Federal têm retirado do mercado brasileiro um grande volume de TV Boxes sem homologação, equipamentos tradicionalmente destinados à destruição, gerando custos logísticos e resíduos eletrônicos. Como alternativa, a Receita Federal tem destinado parte desses dispositivos a instituições de ensino para projetos de reaproveitamento. Neste contexto, este trabalho investiga a viabilidade do uso de TV Boxes reaproveitadas, do modelo Youit TX2, como nós de um cluster distribuído para balanceamento de carga TCP tolerante a falhas. O sistema utiliza o subsistema IPVS (*IP Virtual Server*) do Linux em modo *Direct Routing* e emprega o algoritmo *Bully* para eleição dinâmica de líder, permitindo a substituição automática do nó coordenador em caso de falha. Um cluster composto por três TV Boxes foi implementado para avaliar a tolerância a falhas da solução. Adicionalmente, clientes e *workers* virtuais foram utilizados em testes de saturação para caracterizar o limite de desempenho do balanceador sob cargas elevadas. Os experimentos demonstraram recuperação em menos de 200 ms para falhas *fail-stop* e em poucos segundos para falhas *fail-silent*, sendo o tempo de recuperação determinado principalmente pelo intervalo de *heartbeat* configurado. Nos testes de saturação, o sistema atingiu vazão de aproximadamente 40 mil requisições por segundo, limitado pelo hardware do nó coordenador, enquanto o desempenho escalou de forma aproximadamente linear com o aumento do número de *workers* para a carga de trabalho avaliada. Os resultados demonstram que TV Boxes reaproveitadas constituem uma alternativa viável para a implementação de balanceadores de carga distribuídos tolerantes a falhas, oferecendo baixo custo, resiliência e potencial para reduzir o descarte de equipamentos eletrônicos no contexto da economia circular.

Palavras-chave: Balanceamento de carga; Cluster Distribuído; Tolerância a Falhas; TV Boxes; Economia Circular.

1 Introdução

Nos últimos anos, ações de fiscalização conduzidas pela Agência Nacional de Telecomunicações (Anatel) e pela Receita Federal têm retirado do mercado brasileiro um volume significativo de equipamentos eletrônicos sem homologação. Entre janeiro de 2025 e janeiro de 2026, foram apreendidos mais de 1,3 milhões de produtos irregulares, correspondendo a um valor de mercado superior a R$ 136,6 milhões [1].

[*] tiago.bannwart@gmail.com
[†] gonzalez@unicamp.br
[‡] g271582@dac.unicamp.br
[§] jufborin@unicamp.br

Um dos tipos de equipamentos mais apreendidos nas operações são as TV Boxes, que são dispositivos compactos projetados para converter televisores em equipamentos de *streaming*, mas frequentemente comercializados com acesso não autorizado a canais pagos e conteúdo sob demanda [2].

Além de serem comercialmente ilegais, TV Boxes não homologadas representam riscos ao usuário. Por não passarem pelos testes obrigatórios de segurança elétrica, podem provocar falhas técnicas e interferência em redes de telecomunicações. Ademais, investigações da Anatel identificaram ainda que parte desses dispositivos chega ao consumidor com *malwares* instalados de fábrica, como o BadBox 2.0, capazes de integrar o aparelho a redes usadas para fraudes digitais e interceptação de tráfego [3].

Historicamente, o destino padrão dos equipamentos apreendidos era a destruição, que além de gerar custos logísticos ao Estado, produz resíduo eletrônico em escala significativa [4]. Dentro desse contexto, a Receita Federal tem buscado formas de reaproveitamento desses dispositivos como alternativa à destruição, destinando lotes de equipamentos apreendidos a projetos de reconfiguração em parceria com instituições de ensino [5].

Essa iniciativa se insere nos princípios da economia circular, modelo que propõe manter produtos e materiais em uso pelo maior tempo possível, reduzindo a extração de novos recursos e a geração de resíduos [6]. No contexto de equipamentos eletrônicos, isso significa priorizar o reaproveitamento funcional sobre o descarte, e consequentemente transformar um problema logístico e ambiental em insumo para novos usos. No caso das TV Boxes, faz sentido reconhecer que, apesar de irregulares para a finalidade comercial original, esses dispositivos preservam capacidade computacional que pode ser direcionada a outros fins.

Nesse cenário, da Luz et al. [4] investigaram a viabilidade de reaproveitamento de TV Boxes para computação na borda em cidades inteligentes, avaliando resiliência e eficiência energética em um *testbed* de 20 dispositivos. Os resultados demonstram que esses equipamentos conseguem executar tarefas de processamento contínuo, mas com restrições uma vez que modelos computacionalmente mais exigentes sobrecarregam o hardware, exigindo versões mais leves para viabilizar a operação. Esse resultado evidencia uma limitação geral de recursos computacionais nesses dispositivos, que restringe as cargas de trabalho que um único nó pode atender.

Uma abordagem para superar essa limitação é organizar os dispositivos em cluster, combinando seus recursos para atender demandas que seriam inviáveis individualmente. Trabalhos anteriores demonstraram a viabilidade dessa estratégia com dispositivos reaproveitados de baixo custo.

Switzer et al. [7] propuseram o conceito de "*junkyard computing*", no qual *smartphones* descartados são reorganizados em clusters para tarefas como *cloudlets* de microsserviços e sensoriamento de monitoramento ambiental. Os autores desenvolveram a métrica de Intensidade de Carbono Computacional para comparar o impacto de ciclo de vida entre servidores tradicionais, laptops e *smartphones* reaproveitados, concluindo que, para determinadas cargas de trabalho, clusters de *smartphones* são simultaneamente mais baratos e mais eficientes em emissões de carbono do que a aquisição de servidores novos.

Libert [8] demonstrou a viabilidade prática de formar um cluster a partir de *smartphones* reaproveitados, utilizando uma distribuição leve do Kubernetes voltada para dispositivos de borda. O autor implantou no cluster uma aplicação real de classificação de imagens baseada em TensorFlow Lite, validando a capacidade do cluster de executar cargas de trabalho via requisições HTTP. Os experimentos demonstraram que o desempenho do cluster escala com o número de nós até um patamar específico do meio de rede utilizado, sendo viável tanto em Ethernet quanto em *Wi-Fi* 5 GHz. O trabalho evidencia, assim, que a principal barreira para a formação de clusters com esses dispositivos não está na capacidade de processamento individual, mas nas características de conectividade de rede de cada aparelho.

No contexto específico de TV Boxes apreendidas pela Receita Federal, Farias [9] construiu um

cluster de computação de alto desempenho voltado à democratização do ensino de computação de alto desempenho em cursos de engenharia, área em que o alto custo de infraestrutura tradicionalmente limita o acesso de instituições educacionais a ambientes de cluster. A arquitetura resultante organiza os nós em um Head Node e Compute Nodes, com compartilhamento de arquivos via NFS e gerenciamento de filas de trabalho através do OpenPBS. O resultado consolida o cluster como ferramenta didática funcional, demonstrando que TV Boxes reaproveitadas são tecnicamente capazes de operar em conjunto como uma plataforma computacional coesa, mesmo em tarefas de processamento intensivo.

No contexto específico de serviços de rede, onde o cluster precisa responder a requisições de múltiplos clientes simultaneamente, surge a necessidade de um mecanismo que distribua esse tráfego entre os nós disponíveis, papel exercido pelo balanceador de cargas. Contudo, abordagens centralizadas em que um nó mestre permanente coordena a distribuição do tráfego, apesar de simplificarem a implementação, introduzem um ponto único de falha. Esse risco se torna especialmente relevante em ambientes de hardware reaproveitado, onde a heterogeneidade dos dispositivos e o histórico de uso irregular tornam a ocorrência de falhas um cenário a ser considerado no projeto do sistema.

Com isso, este trabalho investiga a viabilidade de utilizar TV Boxes reaproveitadas como nós de um balanceador de carga TCP distribuído, tolerante a falhas e sem mestre fixo. O sistema proposto opera sobre o subsistema IPVS do *kernel* Linux em modo *Direct Routing* e implementa eleição dinâmica de líder por meio do algoritmo *Bully*, permitindo que qualquer nó do cluster assuma o papel de mestre em caso de falha. A detecção de falhas e a redistribuição de papéis ocorrem de forma autônoma entre os próprios nós, sem dependência de um coordenador fixo. Além da tolerância a falhas, este trabalho avalia o impacto dessa camada de coordenação distribuída sobre a vazão máxima do balanceador, de modo a verificar se a eliminação do mestre fixo é obtida sem custo relevante de desempenho frente a uma configuração de balanceamento centralizada equivalente.

2 Metodologia

2.1 Hardware da TVBox

Figura 1: TV Box modelo Youit TX2 usada no trabalho.

Os nós do cluster são TV Boxes apreendidas de modelo Youit TX2, conforme apresentado na Figura 1, equipadas com o SoC Rockchip RK322x, composto por quatro núcleos ARM Cortex-A7MP de 32 bits a 1,2 GHz, 2 GB de RAM LPDDR3 e 16 GB de armazenamento eMMC. A interface de rede é Fast Ethernet (100 Mbps).

Para situar esse perfil de hardware frente a dispositivos de referência em computação na borda, da Luz et al. [4] compararam o TX2 a dois modelos de Raspberry Pi amplamente utilizados na literatura: o Raspberry Pi 3B+ (Cortex-A53 de 64 bits, quatro núcleos a 1,4 GHz, 1 GB de RAM LPDDR2) e o Raspberry Pi 4B (Cortex-A72 de 64 bits, quatro núcleos a 1,5 GHz, 2 GB de RAM LPDDR4). Em termos de capacidade de processamento, os autores situam o TX2 como comparável ao Raspberry Pi 3B+, suficiente para tarefas de detecção de objetos que tolerem latências de até alguns segundos.

2.2 Sistema Operacional

As TV Boxes utilizadas neste trabalho são originalmente comercializadas com um sistema operacional baseado em Android, voltado ao consumo de mídia. Por isso, a primeira etapa da preparação de cada dispositivo consiste em substituir esse sistema original por uma distribuição Linux, para isso adotamos o Armbian.

Cada nó executa Armbian versão 25.11.1 Bookworm Minimal, baseado no *kernel* Linux 6.12.58. O Armbian é uma distribuição de código aberto, customizável e otimizada para hardware ARM, construída sobre uma base Debian. Essa distribuição foi escolhida por oferecer suporte a *kernel*s *mainline* para plataformas Rockchip, por manter compatibilidade com o repositório de pacotes Debian. A variante Bookworm Minimal, sem ambiente gráfico, foi adotada por seu menor consumo de recursos, adequado ao papel dos nós como servidores dedicados, e não como estações de uso geral. A instalação é realizada inicializando o dispositivo por um cartão SD com a imagem do Armbian e, em seguida, gravando essa imagem diretamente na memória eMMC, dispensando o cartão nas inicializações seguintes.

2.3 Topologia da Rede

O cluster está conectado a uma única rede local, gerenciada por um roteador Predator Connect W6X (OpenWrt 25.12.0), responsável por atribuir os endereços IP na faixa 10.0.0.0/24: o roteador ocupa o endereço 10.0.0.1, os nós do cluster ocupam o intervalo de 10.0.0.101 a 10.0.0.254 e os clientes de teste ocupam endereços entre 10.0.0.2 e 10.0.0.99. O tráfego de clientes é direcionado a um IP virtual (VIP), 10.0.0.100, na porta 80. O nó eleito *master* configura esse VIP localmente e distribui as requisições entre os demais nós via IPVS.

Como o roteador dispõe de apenas quatro portas Gigabit Ethernet, insuficientes para conectar simultaneamente os três nós do cluster, os clientes de teste e demais equipamentos, um *switch* Mercusys MS105G de cinco portas Gigabit Ethernet com negociação automática foi adicionado à rede apenas para suprir esse número de portas, sem introduzir segmentação. Como os nós do cluster possuem interface Fast Ethernet (100 Mbps), as portas Gigabit do *switch* e do roteador negociam automaticamente para 100 Mbps nessas conexões, operando em sua capacidade máxima apenas com os demais equipamentos da rede.

Para organizar fisicamente o cluster, foi projetado um rack próprio, impresso inteiramente em 3D e montado com parafusos, dimensionado especificamente para o formato das TV Boxes. O objetivo foi estruturar a topologia de rede descrita, fixando a posição de cada nó em relação ao roteador e ao switch, e padronizar o acesso físico aos dispositivos durante a condução dos testes, sobretudo os cenários de falha que exigem intervenção manual direta em um nó específico, como a desconexão do cabo de rede ou o desligamento da alimentação. Essa organização facilita a indução dessas falhas, garantindo que cada intervenção seja aplicada ao nó correto, e reforça, em menor escala, o mesmo princípio de baixo custo que motiva o reaproveitamento das TV Boxes. A Figura 2 mostra o rack montado, com o roteador e o switch, além das tv boxes.

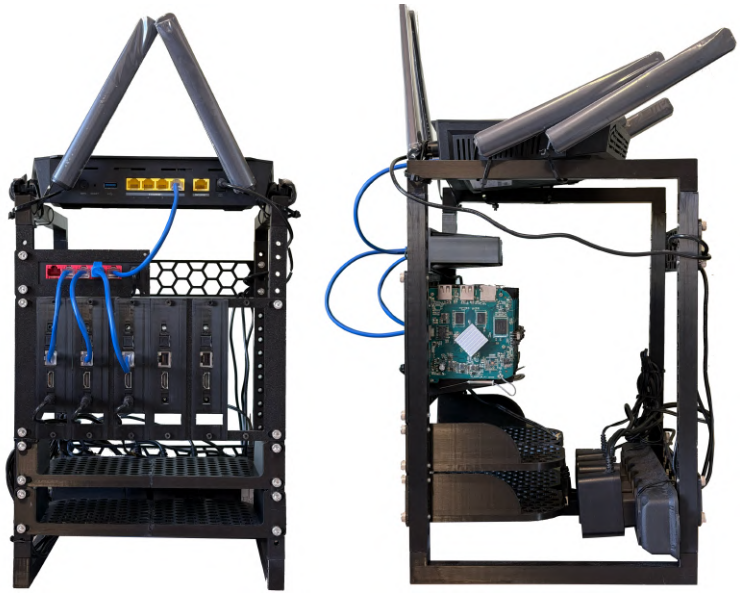

Figura 2: Rack impresso em 3D utilizado para alojar as TV Boxes, o roteador e o switch.

2.4 Implementação do Sistema

O DFTLB (*Distributed Fault-Tolerant Load Balancer*) é o programa desenvolvido neste trabalho, escrito em Go, que inicia como serviço `systemd` em todos os nós do cluster. Seu papel é coordenar os nós entre si, decidindo continuamente qual deles deve assumir a responsabilidade de encaminhar o tráfego. O papel do encaminhamento em si é delegado a um mecanismo do próprio sistema operacional, detalhado a seguir.

Balanceamento de carga: IPVS. Para realizar a distribuição de pacotes entre os nós, este trabalho adota o IP Virtual Server (IPVS), subsistema do *kernel* Linux para balanceamento na camada de transporte. Diferentemente de balanceadores em espaço de usuário (por exemplo, Nginx), que processam cada conexão como um *proxy* completo, terminando a conexão TCP do cliente e abrindo uma nova conexão para o *backend*, o IPVS encaminha pacotes dentro do próprio *kernel*, sem essa sobrecarga adicional de processamento em espaço de usuário. Essa característica é particularmente relevante neste trabalho, dado que os nós do cluster possuem recursos computacionais limitados.

O nó eleito *master* assume o papel ativo do IPVS: cria o serviço virtual associado ao VIP, com escalonamento *round-robin* entre os *backend*s, e atribui o endereço do VIP à sua própria interface de rede. O escalonamento *round-robin* foi escolhido por distribuir as conexões de forma cíclica e igualitária entre os *backend*s. Como os nós do cluster compartilham o mesmo hardware, essa política já é suficiente para equilibrar a carga entre eles, sem necessidade de algoritmos que atribuam pesos ou considerem outros fatores, como o número de conexões ativas por nó (por exemplo, weighted *round-robin* ou *least-connections*). Em seguida, envia mensagens de ARP gratuito (*gratuitous ARP*) para atualizar as tabelas ARP do roteador, do *switch* e dos clientes, redirecionando para si o tráfego destinado ao VIP, passo necessário para que a rede reconheça o novo *master* após um *failover*, sem intervenção externa. Por fim, registra cada *worker* como servidor real do serviço virtual, no modo de encaminhamento por *gateway* (*Direct Routing*).

Os demais nós, ao assumirem o papel de *worker*, também precisam de ajustes de rede para

operar nesse modo: o endereço do VIP é atribuído à interface de loopback, e não à interface física, permitindo que o *kernel* aceite pacotes destinados a esse endereço, enquanto as respostas ARP para o VIP são suprimidas via `sysctl` (arp_ignore e arp_announce). Essa supressão evita que o *worker* responda a requisições ARP pelo VIP e entre em conflito com o *master*, que deve ser o único a anunciar essa associação na rede.

Em modo *Direct Routing* (DR), o *master* reescreve apenas o endereço MAC de destino de cada pacote de entrada para o MAC do *backend* selecionado, sem alterar os endereços IP. O *backend* processa a requisição e envia a resposta diretamente ao cliente, sem que ela passe pelo *master*. Isso elimina a sobrecarga de processamento das respostas no *master*, mas implica que todo o tráfego de entrada (requisições e ACKs do cliente) necessariamente atravessa a NIC (Network Interface Card, a interface física de rede) do *master*, consequência relevante para a análise de desempenho apresentada na Seção 4.

Como aplicação de teste, todos os nós do cluster executam o `Nginx` servindo um arquivo estático de pequeno tamanho. Essa escolha se deve à sua simplicidade: como o balanceamento ocorre na camada de transporte, o DFTLB é agnóstico à aplicação executada sobre TCP, e o mesmo cluster seria capaz de encaminhar tráfego para qualquer outro serviço que opere sobre esse protocolo.

Eleição de líder: algoritmo *Bully*. A tolerância a falhas do DFTLB depende de um mecanismo capaz de determinar, de forma autônoma entre os nós, qual deles assume o papel de *master* quando o atual falha. Este trabalho adota o algoritmo *Bully* para essa tarefa por sua simplicidade de implementação e por sua adequação a clusters pequenos. Além disso, sua determinação de líder por identificador é suficiente para o requisito do sistema de designar um único coordenador, sem precisar de outros mecanismos mais robustos oferecidas por protocolos mais complexos (por exemplo, Raft).

Cada instância do DFTLB mantém uma lista de nós ativos, atualizada por meio de *heartbeat*s periódicos enviados entre os nós. Quando um nó detecta a ausência de *heartbeat*s do *master* atual por um número configurável de intervalos consecutivos, inicia uma eleição: envia mensagem de eleição contendo seu identificador aos nós com identificador superior e, se nenhum responder dentro de um *timeout*, declara-se *master* e notifica os demais com uma mensagem de coordenador. Caso receba resposta de um nó com identificador maior, aguarda o resultado daquela eleição.

2.5 Otimizações Realizadas nos Nós

Durante os experimentos, foram identificados alguns pontos no comportamento padrão do *kernel* e dos *drivers* de rede que poderiam ser melhorados para o cenário de uso do IPVS, além de outros parâmetros de sistema com valores padrão subótimos para essa carga de trabalho.

a) Módulo de *Wi-Fi*: embora a rede do cluster utilize apenas a interface Ethernet, o *driver* do módulo de *Wi-Fi* integrado ao hardware mantinha uma rotina interna em execução contínua, consumindo cerca de 45% da capacidade de um núcleo de CPU mesmo sem nenhuma conexão sem fio ativa. Como o *Wi-Fi* não é utilizado neste trabalho, o módulo foi desabilitado.

b) Distribuição de pacotes entre núcleos: o Linux distribui o processamento de pacotes recebidos entre os núcleos de CPU por meio do RPS (*Receive Packet Steering*). Um parâmetro de sistema mantinha habilitado, adicionalmente, o RFS (*Receive Flow Steering*), um refinamento do RPS que tenta direcionar cada pacote ao núcleo onde o processo que vai consumi-lo está em execução, mas esse direcionamento depende de uma tabela de fluxos que só é preenchida quando existe um *socket* local associado à conexão. Como o tráfego encaminhado pelo IPVS em modo *Direct Routing* não abre *sockets* locais, essa tabela nunca era preenchida, e o RFS acabava concentrando todo o tráfego em um único núcleo, em vez de distribuí-lo. A correção manteve o RFS desabilitado, permitindo que o RPS voltasse a distribuir os pacotes entre os núcleos como esperado.

c) Agregação de pacotes recebidos: o GRO (*Generic Receive Offload*) é uma otimização que agrupa múltiplos pacotes de uma mesma conexão em um só antes de entregá-los à pilha de rede, reduzindo o *overhead* de processamento por pacote. O *driver* de rede utilizado, no entanto, não preservava, após essa agregação, a informação usada pelo RPS para decidir a qual núcleo encaminhar cada pacote, o que fazia o RPS tratar o pacote como sem núcleo definido e, na prática, processá-lo sempre no mesmo núcleo, anulando o efeito da correção anterior. A correção foi desabilitar o GRO, preservando o funcionamento do RPS.

d) Separação entre recepção de rede e processamento: toda vez que a interface de rede recebe um pacote, ela gera uma interrupção de hardware que precisa ser atendida por um núcleo de CPU para iniciar o processamento desse pacote. Por padrão, essa interrupção era sempre atendida pelo mesmo núcleo que também processava os pacotes já recebidos, fazendo essas duas tarefas concorrerem pelo mesmo núcleo. A correção fixou essa interrupção em um núcleo dedicado exclusivamente à recepção, e excluiu esse mesmo núcleo da distribuição feita pelo RPS, reservando os demais núcleos para o processamento de IP e do IPVS.

e) Modo de operação da CPU: por padrão, o sistema ajusta dinamicamente a frequência de operação da CPU conforme a carga, para economizar energia. Essa variação de frequência introduz uma variância adicional na latência de processamento, indesejada durante os testes. A CPU foi fixada no modo de desempenho máximo, eliminando essa fonte de variância.

f) Tamanho da tabela de conexões do IPVS: o IPVS mantém uma tabela interna para rastrear as conexões ativas, cujo tamanho padrão pode ser insuficiente para um volume grande de conexões simultâneas, causando colisões que degradam o desempenho de busca nessa tabela. Por isso, o tamanho dessa tabela foi aumentado.

2.6 Metodologia dos Testes

Os experimentos realizados neste trabalho seguem duas configurações complementares. Na primeira, utiliza-se o *testbed* composto por três TV Boxes para avaliar a tolerância a falhas do cluster e caracterizar o teto de vazão limitado pela capacidade de processamento dos próprios *workers*. Na segunda, o *testbed* é composto por apenas uma TV Box atuando como *master*, com o *backend* substituído por um servidor de maior capacidade, denominado *Virtual Worker* (VW), uma máquina x86-64 com interface de rede GbE rodando Nginx, de modo a caracterizar o teto de vazão imposto pelo próprio *master*, evitando que a capacidade dos *workers* seja o fator limitante nesse cenário. O VW hospeda quinze endereços IP adicionais, configurados como *workers* do IPVS no *master*, e, embora fisicamente seja uma única máquina, sua capacidade elimina a possibilidade de que o *backend* seja o fator limitante do teste.

2.6.1 Ferramenta de Geração de Carga e Clientes

A geração de carga nos experimentos utiliza o k6 [1], uma ferramenta de teste de carga de código aberto, com scripts de teste escritos em JavaScript. O k6 foi escolhido por ser mais leve que ferramentas de teste de carga mais consolidadas, como o JMeter: enquanto o JMeter aloca uma thread da JVM para cada usuário virtual, o k6 é escrito em Go e utiliza *goroutines*, unidades de execução mais leves, permitindo gerar uma carga maior a partir dos mesmos clientes sem que a própria ferramenta de geração de carga se torne o gargalo do experimento. Além disso, seu modelo de "executores"(*executors*) permite controlar a taxa de chegada de requisições de forma independente do número de usuários virtuais (VUs) simultâneos, evitando a subestimação do efeito

[1] https://k6.io/

da saturação sobre a latência percebida pelo cliente. Usando essa ferramenta, a carga é gerada a partir de dois clientes Linux conectados ao mesmo *switch* do cluster, operando em paralelo.

2.6.2 Tolerância a falhas

Os experimentos de tolerância a falhas foram conduzidos com o cluster em sua configuração completa de três nós: um *master* e dois *workers*. Essa topologia permite observar tanto falhas do nó coordenador master, que exigem a eleição de um novo *master* e a reconfiguração do VIP, quanto falhas de um nó *worker*, que apenas removem um *backend* do IPVS sem afetar a coordenação do cluster.

Os cenários de *failover* foram avaliados mantendo uma carga constante de requisições sobre o VIP, gerada com o mesmo ambiente descrito na subseção anterior, com uma taxa alvo de 300 requisições por segundo, enquanto falhas eram induzidas manualmente em um dos nós. Essa taxa foi escolhida por ser alta o suficiente para produzir uma amostragem de requisições com precisão razoável durante o curto intervalo de indisponibilidade, mas baixa o suficiente para não interferir no resultado, ao introduzir saturação como fator de confusão na medição do impacto da falha. Cada requisição registra *timestamp* de início, *timestamp* de fim, e status de sucesso.

Foram avaliados quatro tipos de falha, aplicados separadamente ao *master* e a um *worker*, que se distinguem pela forma como o nó deixa de responder e pela existência ou não de aviso prévio aos demais:

- Desconexão controlada (*fail-stop*): o nó é encerrado via software, notificando explicitamente os pares antes de se desligar.

- Reconexão: um nó anteriormente ausente retorna ao cluster, podendo disparar uma nova eleição caso reingresse com identificador superior ao do *master* em exercício.

- Cabo desconectado (*fail-silent*): a interface Ethernet do nó é desligada fisicamente, sem qualquer aviso prévio aos demais, que precisam inferir a falha apenas pela ausência de *heartbeat*s.

- Desligamento abrupto (*fail-silent*): o nó é desligado diretamente na tomada, simulando uma falha de hardware sem qualquer possibilidade de notificação.

Para cada cenário, o *downtime* é calculado como o intervalo entre o *timestamp* da primeira requisição malsucedida e o *timestamp* da primeira requisição bem-sucedida subsequente, enquanto o número de requisições com erro nesse intervalo quantifica a extensão do impacto percebido pelo cliente durante o período de indisponibilidade.

2.6.3 Vazão máxima

O objetivo deste experimento é determinar a taxa máxima de requisições por segundo que o balanceador sustenta de forma estável antes de entrar em saturação, sob o modo de encaminhamento IPVS em *Direct Routing* utilizado pelo DFTLB. Para determinar com precisão a influência do *master* sobre esse resultado, a mesma metodologia de geração de carga é também aplicada acessando o VW diretamente, sem o IPVS ou o DFTLB à frente. Essa medição de referência permite atribuir ao *master*, por comparação, a parcela da vazão que é de fato limitada por ele.

A vazão é definida como o número de requisições HTTP concluídas com sucesso (resposta 200) por segundo, agregado entre os dois clientes geradores de carga. A carga é composta por requisições HTTP GET curtas ao VIP, com o corpo da resposta descartado no cliente, de modo a minimizar a sobrecarga de processamento no próprio gerador e manter o gargalo no caminho de rede e no balanceador.

Modelo de geração da carga. A forma com que o k6 gera as cargas, apresentado anteriormente, é essencial neste experimento por manter a taxa de requisições oferecida fixa, independentemente da latência observada. Em modelos que cada usuário virtual só emite a próxima requisição após receber a resposta da anterior, a saturação do balanceador reduziria automaticamente a taxa oferecida. Ja na ferramenta utilizada, a taxa oferecida é mantida mesmo sob latência alta, e a saturação passa a se manifestar como uma divergência entre a taxa oferecida e a taxa efetivamente atendida.

Cada execução segue um perfil de rampa de 10 segundos, elevando a vazão até a taxa-alvo do teste, seguida de um platô de 50 segundos de sustentação nessa taxa. A rampa funciona como aquecimento e serve para que sejam estabelecidas as conexões e populadas as tabelas internas do balanceador, e é descartada da janela de medição. A vazão reportada é calculada apenas sobre o platô, já em regime estacionário. O número de usuários virtuais pré-alocados foi dimensionado de acordo com a relação entre taxa de requisições, latência e o número necessários para sustentá-la medido experimentalmente.

Distribuição da carga entre clientes. Como nenhum dos dois clientes virtuais de teste, isoladamente, atinge as maiores taxas-alvo utilizadas neste experimento, a carga total é dividida entre eles, cada um executando uma instância independente do k6 até um pouco antes de seu teto prático individual, e assim a vazão reportada é a soma das taxas atendidas nos dois clientes.

Teto imposto pelos *workers*. Além de isolar o *master*, um segundo experimento busca caracterizar o teto de vazão imposto pela capacidade de processamento dos próprios *workers* TX2, executando a aplicação de teste descrita anteriormente. Esse experimento reutiliza a mesma metodologia de geração de carga, agora com o cluster completo, em duas configurações: dois nós (um *master* e um *worker*) e três nós (um *master* e dois *workers*). As taxas-alvo de 6 mil req/s, para dois nós, e 14 mil req/s, para três nós, foram definidas com base em testes preliminares realizados nessas mesmas configurações, de modo a garantir que a saturação ocorresse dentro da janela de medição.

Critério de vazão máxima. A vazão máxima é obtida aumentando em testes distintos a taxa alvo oferecida, e registrando a taxa efetivamente atendida em regime estacionário. O ponto de máximo é identificado onde a taxa atendida deixa de acompanhar a taxa oferecida, coincidindo com uma degradação da latência.

Medição e validação cruzada. A taxa atendida é extraída das métricas do próprio k6, agregadas sobre os dois clientes na janela do platô. Para confirmar que essa medição corresponde ao tráfego de fato encaminhado pelo balanceador, a vazão obtida no cliente é cruzada com métricas coletadas no lado do servidor, a partir de contadores de pacotes e bytes na interface de rede do *master* e estatísticas internas do IPVS sobre pacotes. Além disso, métricas de uso de CPU, memória e rede, no roteador, no *master* e no VW, foram coletadas ao longo de todos os testes de vazão, de modo a garantir que nenhum desses dois últimos equipamentos se tornasse um gargalo oculto do experimento. Por fim, o cruzamento desses dados permite confirmar a taxa medida e verificar qual está sendo o gargalo associado ao teto de vazão.

3 Resultados

Esta seção apresenta os resultados dos dois experimentos descritos na Metodologia: tolerância a falhas e vazão máxima. Em conjunto, eles avaliam se o modelo proposto de balanceador de

carga distribuído, sem mestre fixo, operando em TV Boxes é viável tanto do ponto de vista de disponibilidade quanto de desempenho.

3.1 Tolerância a Falhas

Os experimentos de tolerância a falhas confirmam que o cluster se recupera sozinho dos quatro tipos de falha considerados, redistribuindo o papel de coordenador quando necessário e restabelecendo o atendimento ao VIP. A Tabela 1 apresenta o *downtime* medido para cada tipo de falha, aplicada separadamente ao *master* e a um *worker*, sob uma carga constante de 300 requisições por segundo sobre o VIP.

Tabela 1: *downtime* medido por tipo de falha (ms)

Evento	**Falha do *master***	**Falha de *worker***
Desconexão controlada (*fail-stop*)	173	19
Reconexão	137	28
Cabo desconectado (*fail-silent*)	3693	3404
Desligamento abrupto (*fail-silent*)	3301	3311

Os resultados revelam dois grupos de tempo de recuperação, separados por ordens de grandeza distintos. Nos cenários *fail-stop* e de reconexão, em que o nó notifica os pares antes de sair ou o próprio protocolo de eleição reage à entrada de um novo nó, a recuperação é quase imediata, entre 137 ms e 173 ms para falhas do *master* e entre 19 ms e 28 ms para falhas de *worker*. Já nos cenários *fail-silent* (cabo desconectado e desligamento abrupto), em que os pares só percebem a ausência do nó pela falta de *heartbeat*s, o *downtime* sobe para a faixa de 3,3 a 3,7 segundos, independentemente de a falha ser do *master* ou do *worker*.

Além disso, apesar da recuperação do *master* ser mais cara que a do *worker*, já que exige eleição, apropriação do VIP via ARP gratuito e reconfiguração do IPVS, a diferença de *downtime* dos dois é baixa. Isso sugere que o custo da lógica de *failover* é pouco relevante diante do tempo de detecção, definido pela frequência de *heartbeat* configurado, com batimentos a cada 1 segundo e falha após 3 segundos consecutivos sem resposta.

Já entre os cenários *fail-stop*, o custo do *handover* do *master* é de 6 a 9 vezes maior que o do *worker*, consistente com a diferença de trabalho realizado. Ainda assim, mesmo o *handover* mais custoso permanece abaixo de 200 ms, bem menor que os tempos medidos para *fail-silent*, indicando novamente que o gargalo de recuperação do sistema está na detecção da falha, e não na resposta a ela. Na prática, o tempo de indisponibilidade em um cenário *fail-silent* é um parâmetro de projeto ajustável, e não uma limitação inerente à arquitetura.

3.2 Vazão Máxima

Os experimentos de vazão identificam um teto sustentado de aproximadamente 40 mil requisições por segundo (req/s) para o DFTLB. Taxas-alvo intermediárias, como 42 mil e 45 mil req/s, também foram testadas e já apresentavam sinais de degradação, com latência elevada e crescente ao longo do teste, e a taxa atendida ainda não ultrapassando os 40 mil. Nesses casos, no entanto, o colapso é mais gradual. Por isso, a taxa-alvo de 50 mil req/s é utilizada como exemplo ao longo desta seção, já que nela o colapso ocorre de forma mais direta e abrupta, facilitando a análise a seguir. Nesse cenário, a taxa efetivamente atendida colapsa para um patamar em torno de 36 mil req/s.

Isolando o *master* como gargalo. Para confirmar que esse teto é imposto pelo *master*, e não pelos clientes de carga ou pelo *backend*, o mesmo cenário foi repetido acessando o VW diretamente pelo seu IP real, sem o IPVS ou o DFTLB no caminho. Nessa configuração de controle, os clientes sustentam 50 mil req/s de forma limpa, o que confirma que os geradores de carga são capazes da taxa-alvo total e que o teto observado é uma limitação do *master*, e não do restante do ambiente de teste.

Picos de saturação da CPU. As Figuras 3 e 4 apresentam a vazão, a latência e o uso de CPU por núcleo do *master* ao longo dos testes a 40 mil e a 50 mil req/s.

Figura 3: vazão, latência e uso de CPU do *master* durante o teste a 40 mil req/s

Em ambos os níveis de carga, o núcleo responsável por atender a interrupção da interface de rede, já isolado na otimização descrita anteriormente, apresenta uma utilização um pouco mais alta que os demais. A utilização total de CPU é marcada por picos que se aproximam ou atingem 100% de utilização, intercalados com vales bem mais baixos, a cada ciclo de poucas centenas de milissegundos. A 50 mil req/s, esses picos de fato atingem 100% e permanecem nesse patamar por intervalos mais longos que a 40 mil, mas o padrão continua sendo de picos pontuais, e não de saturação contínua. Uma análise mais detalhada mostrou que o núcleo responsável pela interrupção de rede satura a 100% em 10,6% das janelas de 100 ms a 50 mil req/s, o dobro da proporção observada a 40 mil, enquanto sua utilização média ao longo de todo o teste, paradoxalmente, cai

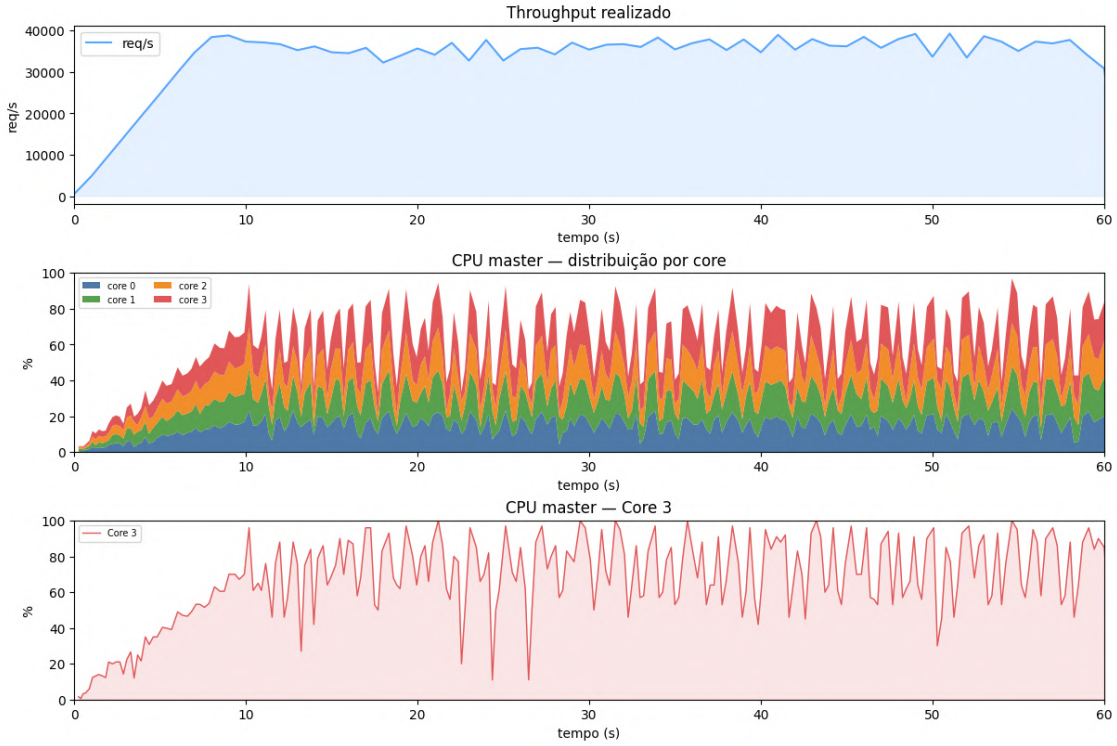

Figura 4: vazão e uso de CPU do *master* durante o teste a 50 mil req/s

de 42% para 33% entre esses dois níveis de carga. Ou seja, a saturação em rajadas fica encoberta quando se observa apenas a média. Além disso, em todos os testes realizados a memória utilizada pelas TV Box, seja na função de *master* ou *worker*, não passou de 15%, descartando esse recurso como um gargalo. Do lado dos clientes, o mesmo padrão é observado, já que eles utilizam menos CPU operando através do *master* do que operando direto contra o VW, o que descarta a hipótese de que a própria ferramenta de geração de carga estivesse saturada. Medições de CPU e memória no roteador e no VW ao longo dos mesmos testes também descartam esses dois equipamentos como gargalo, já que a CPU do roteador não ultrapassou 20%, com memória em torno de 15%, e a CPU do VW atingiu no máximo 26%, com memória em torno de 12%, ambos com folga suficiente para não serem o fator limitante do teste.

Esse comportamento intermitente é o que motiva a investigação mais aprofundada apresentada a seguir. Como a CPU não permanece continuamente saturada, o gargalo não é evidente e assim se torna necessário examinar separadamente a interface de rede, a fila de saída do *kernel*, as retransmissões TCP e a latência ponta a ponta para confirmar que a origem do colapso está, de fato, nesses picos. A Figura 5 adianta essa conclusão, mostrando a latência (tempo até o primeiro byte) durante e após o teste a 50 mil req/s, em escala logarítmica. Durante o platô de carga, a mediana já sobe para a casa de uma centena de milissegundos, com a cauda p95/p99 na casa de alguns segundos; ao final do teste, a latência salta para dezenas de segundos e permanece nesse patamar por um longo intervalo mesmo após o fim da oferta de carga, evidenciando que o *master* acumulou, durante os picos de saturação, um *backlog* de requisições que continuou sendo drenado muito depois de a carga ter cessado.

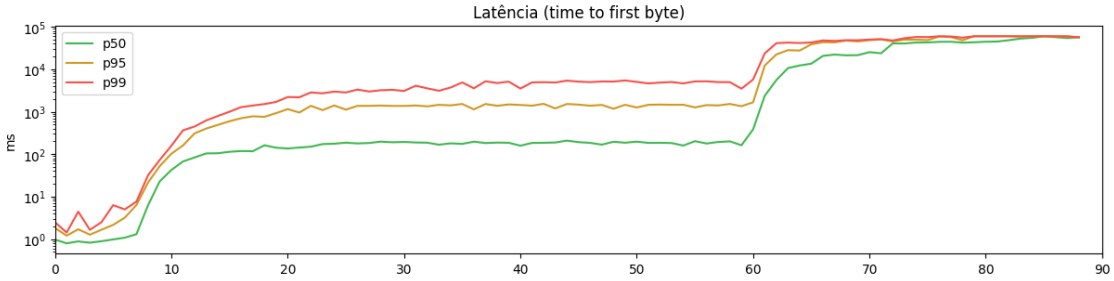

Figura 5: Latência durante e após o teste a 50 mil req/s

Descartando causas de hardware e de *kernel*. Uma investigação em camadas, cobrindo desde a interface de rede do *master* até a fila de saída do *kernel*, permitiu descartar uma série de hipóteses adicionais para esse teto. A Tabela 2 apresenta a utilização da interface de rede do *master* nos dois níveis de carga.

Tabela 2: Utilização da interface de rede do *master*, por taxa-alvo

	40k req/s	**50k req/s**
Quadros RX / TX	1.836.062 / 1.836.074	2.610.753 / 2.610.796
Descartes RX / TX	0 / 0	0 / 0
Ocupação do link	46% de 100 Mbps	57% de 100 Mbps

Os quadros recebidos (RX) e transmitidos (TX) praticamente se igualam nos dois níveis de carga, e nenhum descarte é observado em qualquer contador de erro (buffer FIFO, softnet, estatísticas do *driver* de rede). Ao mesmo tempo, ao passar de 40 para 50 mil req/s, a taxa de pacotes por segundo cresce cerca de 54%, bem mais que os 25% de aumento na taxa-alvo, enquanto o tamanho médio dos quadros encolhe (de 133 para 116 bytes). Isso é um sinal de que o tráfego excedente é composto majoritariamente de pacotes pequenos, como confirmações (ACKs) e retransmissões, e não de tráfego legítimo adicional. Mesmo assim, a ocupação do link nunca ultrapassa 57% da capacidade nominal de 100 Mbps da interface, descartando a banda como fator limitante. O *backlog* da fila de saída do *kernel* também permanece praticamente vazio nos dois níveis de carga, afastando o enfileiramento de saída como gargalo.

Retransmissão TCP. A Tabela 3 isola o efeito do *master* mantendo carga e clientes idênticos e alterando apenas o endereço de destino.

Tabela 3: Retransmissão TCP: acesso direto ao VW vs. via *master* (50k req/s)

Cenário	**Segmentos enviados**	**Retransmissões**	**Taxa**
Direto (cliente 1)	1.694.009	2.315	0,14%
Direto (cliente 2)	1.413.895	899	0,06%
Via *master* (cliente 1)	1.636.320	169.575	10,36%
Via *master* (cliente 2)	1.368.220	149.516	10,93%

A taxa de retransmissão salta de uma faixa desprezível (entre 0,06 e 0,14%) no acesso direto para mais de 10% quando o tráfego passa pelo *master*. A análise dos contadores internos do TCP mostra que a totalidade dessas retransmissões é motivada por estouro de temporizador (RTO), e mais da metade delas (56% e 46%, respectivamente, para os dois clientes) é uma retransmissão

espúria, ou seja, o receptor confirma já ter recebido o segmento original, indicando que o pacote apenas chegou atrasado, e não que foi perdido.

A Tabela 4 apresenta o RTT medido por VUs dedicados com baixa taxa de transmissão oferecida, em paralelo à carga principal. Isso garante que a latência reportada reflita as condições de rede impostas pela carga principal, sem que os próprios VUs dedicados contribuam de forma relevante para essa carga. Os valores apresentados consideram apenas os 60 segundos de duração do teste, desconsiderando as respostas que chegaram após seu encerramento, ao contrário da Figura 5, que estende a observação até os 90 segundos para capturar justamente esse período de drenagem do *backlog*.

Tabela 4: RTT dos VUs dedicados (10 req/s), por taxa-alvo

Carga	**p50**	**p95**	**p99**
40k req/s	0,8 ms	4,4 ms	5,4 ms
50k req/s	575,7 ms	1479,2 ms	2485,7 ms

Ao cruzar de 40 para 50 mil req/s, a mediana do RTT salta de 0,8 ms para 575,7 ms, um aumento de mais de 700 vezes, enquanto a cauda (p99) ultrapassa 2,4 segundos. Esse salto em latência ultrapassa o piso mínimo de retransmissão (RTO) padrão do Linux, de 200 ms. Com isso, uma fração dos segmentos passa a sofrer atrasos acima desse piso, suficiente para o TCP interpretar como perda e retransmitir, gerando mais tráfego, mais atraso, e realimentando o próprio colapso.

Em conjunto, os resultados sugerem mais fortemente que o teto de vazão é imposto pelo núcleo de CPU, responsável por atender a interrupção de rede do *master*, que chega ao seu limite de 100% de utilização, ainda que apenas em picos pontuais, e não de forma contínua, como visto nas Figuras 3 e 4, logo esse padrão de picos parece ser o suficiente para causar o colapso observado. Como a interface de rede do *master* possui uma única fila de recepção em hardware, drenada por esse único núcleo, quando a taxa de pacotes exige mais do núcleo do que ele consegue entregar durante a rajada, os pacotes já recebidos se acumulam à espera de processamento. Como o tráfego é apenas enfileirado, e não descartado, cada rajada de saturação se traduz em um pico de latência de cauda que, somado ao piso de retransmissão do TCP, dispara a espiral de retransmissão descrita anteriormente e o colapso visto na Figura 5. Os demais resultados reforçam essa leitura, já que a ausência de perda de pacote e de esgotamento de banda ou de fila de saída (Tabela 2), a taxa de retransmissão desprezível no acesso direto ao VW (Tabela 3) e o salto de latência dos VUs dedicados (Tabela 4) são consistentes com os picos de saturação da CPU de ingestão do *master*.

Por fim, esse limite não pode ser contornado por software, já que os mecanismos do sistema operacional que distribuem o processamento de pacotes entre os núcleos só atuam depois que o pacote já deixou a fila única de hardware, de modo que espalhar o processamento pelos demais núcleos não alivia o gargalo na entrada dos dados. Superar esse teto exigiria hardware com múltiplas filas de recepção ou um núcleo de ingestão mais rápido.

Teto imposto pelos *workers*. Além do teto imposto pelo *master*, um segundo experimento caracteriza o teto imposto pela capacidade de processamento dos próprios *workers* TX2, substituindo o VW pelos nós reais do cluster como *backend*. As Figuras 6 e 7 apresentam os resultados para duas configurações, um cluster de dois nós (um *master* e um *worker*), com meta de 6 mil req/s, e um cluster de três nós (um *master* e dois *workers*), com meta de 14 mil req/s.

Em ambos os casos, a vazão realizado se estabiliza bem abaixo da meta, em torno de 5,7 mil req/s com um *worker* e de 11,5 mil req/s com dois, coincidindo com a CPU do(s) *worker*(s) atingindo 100% de utilização já nos primeiros segundos do teste, enquanto o *master* permanece praticamente

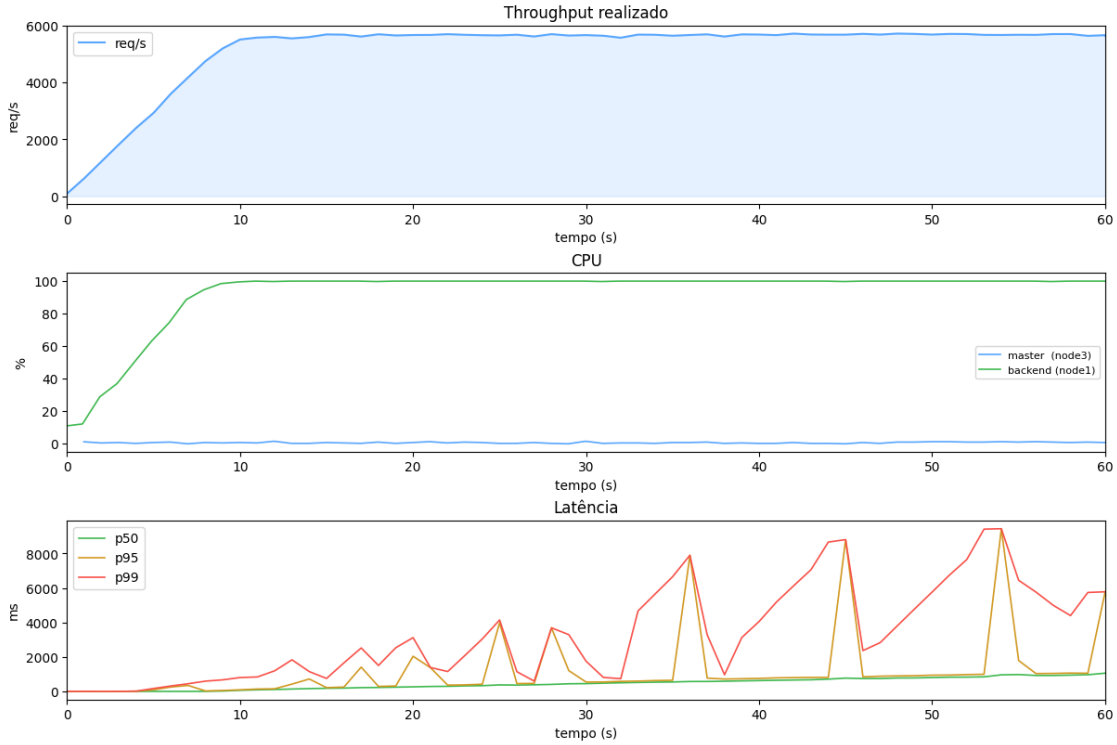

Figura 6: vazão, CPU e latência durante o teste com um *master* e um *worker* (meta de 6 mil req/s)

ocioso. Esse é um gargalo distinto do identificado no experimento anterior, em que o *master* satura por latência de cauda muito antes de a CPU do *backend* se tornar um fator limitante. Ou seja, em clusters pequenos como os testados, é a capacidade de processamento dos próprios *workers* que limita a vazão, e não o *master*. As duas figuras mostram ainda que, mantida a meta acima desse teto pelo restante do teste, a latência (p95 e p99) não se estabiliza em um patamar de colapso, como ocorre no *master*, mas cresce de forma progressiva e irregular ao longo do tempo, sugerindo que sustentar uma carga acima da capacidade do cluster degrada continuamente sua estabilidade.

Os dois pontos de saturação, aproximadamente 5,7 mil req/s para um *worker* e 11,5 mil req/s para dois, sugerem que o teto de vazão do sistema cresce de forma aproximadamente linear com o número de *workers* disponíveis ($5\,700 \times 2 \approx 11500$).

Além disso, o limite de vazão imposto pelo *master* acontece tanto quando o IPVS opera sozinho, quanto com o sistema completo da camada de coordenação, que inclui *heartbeat*s, eleição de líder e reconfiguração de papéis. Combinado com os resultados da subseção anterior, isso permite concluir que a tolerância a falhas foi obtida sem custo perceptível de desempenho em relação a uma configuração de balanceamento sem coordenação distribuída.

Estimativa do limite do cluster. Combinando os dois experimentos de vazão, é possível estimar o ponto em que o teto do *master* passaria a ser o fator limitante do cluster, em vez da capacidade dos *workers*. Como o teto por *worker* observado é de aproximadamente 5,7 mil req/s e o teto do *master* é de aproximadamente 40 mil req/s, a razão entre os dois ($40\,000 \div 5\,700 \approx 7$) sugere que o sistema, para esta carga de trabalho específica de servir um arquivo estático, escalaria de forma aproximadamente linear até um cluster de oito nós, um *master* e sete *workers*, ponto a partir do qual o *master* se tornaria o gargalo em vez dos *workers*.

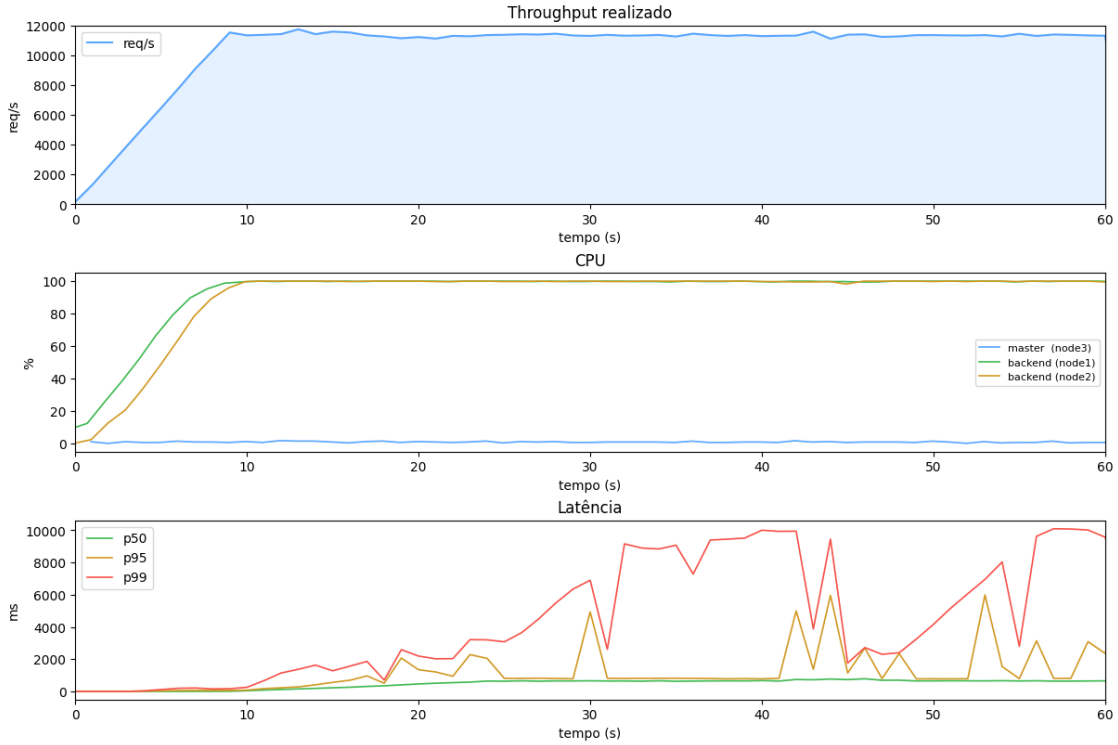

Figura 7: vazão, CPU e latência durante o teste com um *master* e dois *workers* (meta de 14 mil req/s)

Essa estimativa, no entanto, é específica da carga de trabalho testada. Em cenários nos quais os *workers* executam tarefas computacionalmente mais intensas do que apenas servir um arquivo estático, o teto individual de cada *worker* tende a ser menor que os 5,7 mil req/s observados aqui, o que desloca esse ponto de equilíbrio. Ou seja, mais *workers* caberiam no cluster antes que o teto do *master* de 40 mil req/s se tornasse o fator limitante. Com isso, o benefício de adicionar mais TV Boxes ao cluster é tanto maior quanto mais pesada for a carga de trabalho executada por elas.

4 Conclusão

Este trabalho investigou a viabilidade de utilizar TV Boxes apreendidas pela Receita Federal como nós de um balanceador de carga TCP distribuído, tolerante a falhas e sem mestre fixo, implementado sobre o subsistema IPVS do *kernel* Linux em modo *Direct Routing*, com eleição dinâmica de líder pelo algoritmo *Bully*. Os resultados obtidos confirmam que o sistema é viável, uma vez que o cluster de TV Boxes TX2 demonstrou ser capaz de detectar falhas e redistribuir o papel de coordenador de forma autônoma, sem intervenção externa, e de sustentar taxas de milhares de requisições por segundo sem que essa camada de coordenação introduzisse um custo mensurável de desempenho.

Os experimentos de tolerância a falhas mostraram que o cluster se recupera dos quatro tipos de falha considerados sem exigir um mestre fixo, com dois regimes bem definidos de recuperação. Falhas *fail-stop*, sejam elas a saída controlada de um nó ou o reingresso de um nó ausente, são resolvidas em menos de 200 ms, mesmo quando envolvem a eleição de um novo *master* e a reconfiguração do IPVS. Já falhas *fail-silent*, em que o nó deixa de responder sem aviso prévio, levam de 3,3 a 3,7 segundos para serem detectadas e resolvidas, tempo dominado pelo temporizador de

heartbeat configurado, e não pela complexidade do mecanismo de eleição em si. Esse resultado indica que o tempo de indisponibilidade do sistema é, em grande parte, um parâmetro de projeto ajustável, e não uma limitação inerente à arquitetura proposta.

Os experimentos de vazão, por sua vez, caracterizaram os limites de desempenho do cluster e tentaram isolar a origem desses limites. O *master*, operando como coordenador do IPVS, sustenta um teto de aproximadamente 40 mil requisições por segundo, imposto por uma limitação estrutural do hardware, uma única fila de recepção de rede drenada por um único núcleo de CPU, e não por escassez de banda, perda de pacotes ou saturação média de CPU. Esse teto se mostrou o mesmo tanto para o IPVS operando isoladamente quanto para o DFTLB completo, com toda a sua camada de coordenação distribuída. A tolerância a falhas obtida neste trabalho não tem, portanto, custo perceptível de vazão em relação a uma configuração de balanceamento sem coordenação distribuída.

Em clusters menores que o teto do *master*, no entanto, o fator limitante passa a ser a capacidade de processamento dos próprios *workers*. Os experimentos com dois e três nós mostraram que a vazão sustentado escala de forma aproximadamente linear com o número de *workers* disponíveis, cerca de 5,7 mil requisições por segundo por *worker*, para a carga de trabalho testada. Extrapolando essa relação, estima-se que um cluster de aproximadamente oito TV Boxes, um *master* e sete *workers*, seria necessário para que o teto do *master* se tornasse o fator limitante nessa carga de trabalho específica. Para cargas mais pesadas por requisição, esse ponto de equilíbrio se desloca em direção aos *workers* novamente, reforçando o valor de agregar mais unidades reaproveitadas ao cluster.

Em conjunto, esses resultados sustentam a proposta inicial deste trabalho de que TV Boxes reaproveitadas, originalmente destinadas ao descarte, podem ser reorganizadas em um cluster com balanceamento de carga funcional, resiliente a falhas de seus próprios nós, sem depender de um coordenador fixo e sem abrir mão de desempenho em relação a uma solução centralizada equivalente. Isso reforça o argumento de economia circular que motiva este trabalho [6], em que TV Boxes apreendidas por irregularidade comercial, que de outra forma seriam destruídas, demonstram capacidade computacional suficiente para desempenhar um papel de infraestrutura de rede real.

4.1 Limitações e Trabalhos Futuros

Os resultados apresentados possuem limitações que devem ser consideradas. Cabe uma análise ainda mais profunda do teto de vazão imposto pelo *master*, para confirmar se existe espaço para resultados ainda melhores. A estimativa de um cluster de oito nós como limite de escalabilidade é extrapolada a partir de resultados obtidos com no máximo três nós, e não foi validada empiricamente em clusters maiores. Por fim, todos os experimentos de vazão utilizaram uma carga de trabalho leve, a entrega de um arquivo estático via Nginx, de modo que o comportamento do sistema sob cargas computacionalmente mais intensas no *worker* permanece uma questão em aberto.

Trabalhos futuros podem endereçar essas limitações validando os resultados em clusters de maior escala e avaliando o comportamento do DFTLB sob cargas de trabalho de aplicação real, em vez da carga sintética utilizada neste trabalho.

Referências

[1] Agência Gov. “Combate à pirataria: Anatel retira mais de 1,3 milhão de produtos do mercado entre 2025 e 2026.” português, Agência Gov, acesso em 19 de jun. de 2026. endereço: `https://agenciagov.ebc.com.br/noticias/202604/anatel-retira-mais-de-1-3-milhao-de-produtos-irregulares-do-mercado-entre-2025-e-2026`.

[2] Ministério das Comunicações. "Anatel retira de circulação mais de 8 milhões de produtos de telecomunicações por falta de homologação, correspondendo a R$ 833,6 milhões." português, Ministério das Comunicações, acesso em 19 de jun. de 2026. endereço: https://www.gov.br/mcom/pt-br/noticias/2025/novembro/anatel-retira-de-circulacao-mais-de-8-milhoes-de-produtos-de-telecomunicacoes-por-falta-de-homologacao-correspondendo-a-r-833-6-milhoes.

[3] Agência Nacional de Telecomunicações. "Anatel lança painel com informações sobre bloqueios administrativos de TV Boxes ilegais." português, Agência Nacional de Telecomunicações, acesso em 19 de jun. de 2026. endereço: https://www.gov.br/anatel/pt-br/assuntos/noticias/anatel-lanca-painel-com-informacoes-sobre-bloqueios-administrativos-de-tv-boxes-ilegais.

[4] G. P. C. P. da Luz, G. M. Sato, L. F. G. Gonzalez e J. F. Borin, "Repurposing of TV boxes for a circular economy in smart cities applications," en, *Scientific Reports*, v. 15, n. 1, p. 22 638, 2025, ISSN: 2045-2322. DOI: 10.1038/s41598-025-97379-4.

[5] Secretaria Especial da Receita Federal do Brasil. "Receita Federal em Brasília destina 30 mil aparelhos de TV Box para serem transformados em minicomputadores." português, Receita Federal do Brasil, acesso em 19 de jun. de 2026. endereço: https://www.gov.br/receitafederal/pt-br/assuntos/noticias/2024/dezembro/receita-federal-em-brasilia-destina-30-mil-aparelhos-de-tv-box-para-serem-transformados-em-minicomputadores.

[6] T. S. Gaur, V. Yadav, S. Mittal e M. K. Sharma, "A systematic review on sustainable E-waste management: challenges, circular economy practices, and a conceptual framework," *Management of Environmental Quality: An International Journal*, v. 35, n. 4, pp. 858–884, dez. de 2023, ISSN: 1477-7835. DOI: 10.1108/MEQ-05-2023-0139.

[7] J. Switzer, G. Marcano, R. Kastner e P. Pannuto, "Junkyard Computing: Repurposing Discarded Smartphones to Minimize Carbon," em *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, sér. ASPLOS 2023, Association for Computing Machinery, jan. de 2023, pp. 400–412, ISBN: 978-1-4503-9916-6. DOI: 10.1145/3575693.3575710. endereço: https://dl.acm.org/doi/10.1145/3575693.3575710.

[8] C. Libert, "Low-cost edge computing using upcycled smartphones (in collaboration with Swarn)," eng, 2024. endereço: https://hdl.handle.net/2078.2/41570.

[9] A. A. d. O. Farias, "Construção de um cluster de baixo custo utilizando tv boxes descaracterizadas," en, fev. de 2026. endereço: https://bdm.ufpa.br/items/183a2b9b-d18a-4f34-8c74-ccd1e790c4d7.