# Usando Redes Neurais para Reconstruir Traços de Sessões de Usuários de Sistemas de Larga Escala

**Kayuã Oleques Paim**[1], **Rafael Duarte Beltran**[1],
**Rodrigo Brandão Mansilha**[1], **Weverton Cordeiro**[2]

[1] Universidade Federal do Pampa - UNIPAMPA

{kayuapaim.aluno, rafaelbeltran.aluno, mansilha}@unipampa.edu.br

[2]Universidade Federal do Rio Grande do Sul - UFRGS

weverton.cordeiro@inf.ufrgs.br

***Resumo.*** *A monitoração de presença* online *de entidades em sistemas distribuídos é fundamental para compreender o comportamento de tais sistemas e simular a dinâmica dos mesmos, entre outros. Em muitos sistemas, a presença* online *de entidades pode ser monitorada via amostragem – em intervalos regulares – das entidades atualmente* online*. Exemplos incluem usuários* online *em aplicações distribuídas, ou estações ativas na internet. A monitoração pode ser falha, e algumas entidades podem não aparecer como* online *em uma ou mais listas, comprometendo assim a acurácia dos dados coletados. Investigações anteriores aplicaram métodos estatísticos para identificar a ocorrência de tais falhas, e usaram limiares para corrigi-las. No presente artigo, propõe-se investigar a potencialidade de métodos de aprendizado de máquina para regenerar dados de monitoração coletados via amostragem. Em particular, avaliamos a potencialidade de se corrigir dados usando técnicas de aprendizado profundo, e mostramos que a acurácia, precisão e* recall *podem ser substancialmente melhorada em comparação com os métodos estatísticos existentes.*

***Abstract.*** *Monitoring online presence of entities in distributed systems is essential to understand the behavior of such systems and to simulate their dynamics, among others. In many systems, the online presence of entities can be sampled – at regular time intervals – of currently online entities. Examples include online users in distributed apps, or active nodes on the internet. The monitoring process may be flawed, though, and some entities may not appear as online in one or more lists, thus compromising the accuracy of the data collected. Previous investigations have applied statistical methods to identify the occurrence of such failures, and used thresholds to correct them. In this paper, we investigate the potential of machine learning methods to regenerate monitoring data collected via sampling. In particular, we assessed the potential for correcting data using deep learning, and showed that the accuracy, precision and recall can be substantially improved compared to existing statistical methods.*

## 1. Introdução

A monitoração da presença *online* de entidades (usuários, agentes, estações, etc.) em sistemas distribuídos dinâmicos e de larga escala é fundamental para compreender como tais sistemas se comportam, analisar sua dinâmica, principais propriedades e

limitações, e identificar oportunidades de melhoria. Exemplos de investigações baseadas na monitoração da presença *online* de entidades incluem monitoração de disponibilidade de conexão à internet na última milha para avaliar o impacto de ocorrências climáticas [Padmanabhan et al. 2019] e desastres ambientais como terremotos [Mayer et al. 2021] e incêndios [Anderson et al. 2020], e monitoração de quantidade de usuários *online* em aplicações distribuídas [Lareida et al. 2017, Cordeiro et al. 2021].

Uma estratégia tipicamente empregada para essa monitoração é amostrar, em intervalos de tempo regulares, o conjunto de entidades atualmente presentes no sistema alvo. No caso da monitoração de disponibilidade de conexão à internet na última milha para clientes residenciais, a amostragem é feita enviando requisições ICMP *echo request* via *ping* para um conjunto de endereços de um determinado domínio [Padmanabhan et al. 2019], em um dado instante de tempo. No caso de aplicações distribuídas como as baseadas no paradigma *peer-to-peer*, a amostragem pode ser feita via solicitação periódica de listas dos usuários atualmente *online* [Hoßfeld et al. 2011, Lareida et al. 2017].

Um dos problemas da estratégia baseada em amostragem para monitorar a presença *online* de entidades no sistema é que algumas entidades podem falhar em aparecer em uma ou mais amostragens, mesmo estando *online*. Por exemplo, algumas estações de usuários podem não responder requisições ICMP *echo request* devido a congestionamento na rede. No caso de usuários em aplicações distribuídas, o tamanho máximo determinado para as listas em cada amostragem limita a fração de usuários que pode ser observada em cada amostragem [Cordeiro et al. 2021]. Além disso, a própria coleta das amostras é desafiadora, por envolver sistemas distribuídos de monitoramentos em uma escala tipicamente proporcional à dos sistemas monitorados. Portanto, a coleta de dados de monitoração (ou traços) abrangentes, detalhados e precisos pode ser inviável, especialmente para uma rede de larga escala e/ou para um período de longa duração. Nesses contextos, a regeneração de conjuntos de dados de monitoração, através da identificação e correções de omissões geradas por falha durante a amostragem do sistema alvo, é considerado um problema chave. Esse problema tem sido abordado em diversos contextos, incluindo investigações relacionadas ao tráfego de rede [Xie et al. 2019], redes de sensores [Cheng et al. 2017], detecção de anomalias [Xie et al. 2018] e sessões de usuários em sistemas distribuídos de larga escala [Cordeiro et al. 2014, Cordeiro et al. 2021].

Recentemente, Cordeiro et al. [Cordeiro et al. 2021] aprofundaram investigações em um método estatístico baseado em processos de Bernoulli para regenerar traços de sessões de usuários em sistemas distribuídos de larga escala. Em resumo, o estudo concentra-se em uma técnica que (i) estima a probabilidade de que o sistema de monitoramento falhou em "capturar" cada usuário *online* em cada fatia de tempo do monitoramento e (ii) corrige o *dataset* caso a probabilidade de falha exceda um determinado limiar. Os autores mostraram que a técnica pode ser considerada promissora para corrigir traços de usuários em aplicações distribuídas baseadas no paradigma *peer-to-peer* com probabilidade de falha em grão grosso (*e.g.* probabilidade de falha entre 10% e 50%). Os autores também discutiram que a técnica foi incapaz de corrigir falhas em granularidade fina, isto é, cenários onde o *dataset* apresenta probabilidade de falha mais baixa ($< 10\%$).

O presente artigo avança o estado da arte ao investigar a viabilidade de se aplicar técnicas de Aprendizado Profundo para corrigir falhas em conjuntos de dados de monitoração de presença *online* de entidades em sistemas distribuídos. Aprendi-

zado profundo (*Deep Learning*) é um ramo da Aprendizagem de Máquina (*Machine Learning*, ML) com foco nas chamadas Redes Neurais Artificiais (RNAs) profundas [Boutaba et al. 2018]. Essas RNAs são compostas por diversas camadas de processamento que possuem capacidade para assimilar conhecimento em variados níveis de abstração [LeCun et al. 2015]. Aprendizado profundo tem permitido avanços significativos no processamento de dados em larga escala em diversos domínios. Neste contexto, o presente artigo faz as seguintes contribuições para o estado da arte:

- Uma análise da viabilidade de se empregar técnicas de aprendizado profundo para corrigir traços de monitoramento, obtidos via processo de amostragens regulares, da presença *online* de entidades em sistemas distribuídos em larga escala;
- Uma arquitetura de RNA para a correção de traços, com foco em monitoração de usuários *online* em sistemas distribuídos de larga escala baseados no paradigma *peer-to-peer*, e acurácia superior a de soluções baseadas em métodos estatísticos.

O protótipo implementado no escopo do artigo, bem como *scripts* auxiliares, traços de monitoração usados nos experimentos e os resultados obtidos, estão disponíveis sob domínio público para permitir a reprodutibilidade da pesquisa [Paim et al. 2021].

O restante do artigo está organizado como segue. A Seção 2 revisa o referencial teórico da pesquisa e discute os principais trabalhos relacionados. A Seção 3 apresenta uma proposta de arquitetura baseada em aprendizado profundo, incluindo uma topologia de Rede Neural Profunda, para corrigir traços de sessões de usuário em sistemas distribuídos. A Seção 4 apresenta os resultados obtidos com uma avaliação experimental baseada em traços reais, além de uma comparação de desempenho com o estado da arte. A Seção 5 fecha o artigo com considerações finais e perspectivas de pesquisa futura.

## 2. Referencial Teórico e Trabalhos Relacionados

As informações pertinentes ao processo de correção de traços no escopo do artigo podem ser classificadas entre "resultados do monitoramento" e "características (do sistema) de monitoramento". A seguir é explicada a primeira classe, usando a Figura 1 como base.

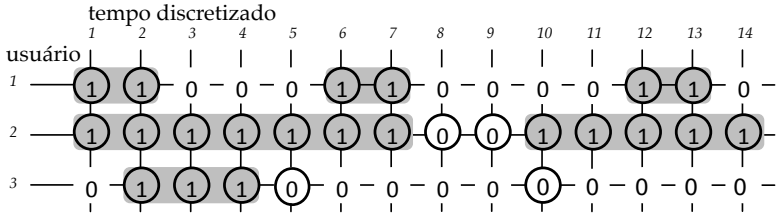



**Figura 1. Matriz de amostras $S = [s_{v,t}]$ indicando a presença (1) ou ausência (0) de usuários $v$ (linhas) *online* em amostras coletadas no instante $t$ (colunas)**

A Figura 1 ilustra um exemplo de **resultados do monitoramento**. Mais especificamente, ela representa um traço de monitoração de presença de usuários em amostras consecutivas capturadas ao longo do tempo, representado como uma matriz $S = [s_{v,t}]$. As linhas representam três usuários (rotulados como $1$, $2$, $3$) que foram observados pelo menos uma vez durante o monitoramento. As colunas representam as amostras capturadas em intervalos de tempo regulares durante o monitoramento (rotuladas como $1, 2 \ldots 14$). Cada célula $s_{v,t}$ da matriz $S$ contém um valor binário, onde $1$ ("positivo") indica que o

usuário $v$ estava presente (*online*) no sistema quando a amostra $t$ foi coletada; o valor $0$ ("nulo") indica o contrário. Conforme discutido anteriormente, um usuário pode constar como ausente na amostra mesmo estando, na realidade, *online* no sistema – tal pode ocorrer devido a falha durante a coleta de uma determinada amostra ou por limitação da técnica de monitoração empregada para coletar as amostras. Para fins de exemplificação desse fenômeno e seus impactos, círculos nas posições $s_{v,t}$ na Figura 1 indicam que o usuário $v$ efetivamente estava *online* no instante $t$. Por exemplo, o círculo em $s_{3,5}$ indica que o usuário $3$ efetivamente estava *online* no instante $5$, embora a amostra coletada não reporte que o mesmo estava (valor $s_{3,5}$ =0).

A partir do conjunto de amostras, posicionadas em sequência, é possível reconstruir a dinâmica de entrada e saída dos usuários no sistema. Uma sessão *online* de usuário corresponde a uma sequência de valores positivos. Na Figura 1, cada sessão é grifada em cinza. Isso significa afirmar que o usuário 2 teve duas sessões *online* no sistema, *de acordo com os resultados do monitoramento*: na primeira, ele entrou em $t = 1$ e saiu em $t = 7$ e, na segunda, entrou em $t = 10$ e saiu em $t = 14$. Quando o valor anotado na amostra corresponde à realidade (no exemplo ilustrado, posição $s_{v,t}$ é circulada e $s_{v,t} = 1$), não há falha a ser corrigida. Por outro lado, quando a amostra não corresponde à realidade (no exemplo, posição $s_{v,t}$ é circulada e $s_{v,t} = 0$), houve uma falha. Observe que as falhas impactam na quantidade e duração das sessões *online* dos usuários. No exemplo, o usuário $v_2$ efetivamente teve apenas uma sessão durante o monitoramento, com duração de 14 períodos. No entanto, o monitoramento indica que o usuário 2 esteve *online* duas vezes, a primeira sessão com duração de 7 períodos, e a segunda com duração de 5 períodos.

As **características de monitoramento**, por sua vez, incluem metadados acerca do sistema de monitoramento, por exemplo identificadores e propriedades dos monitores. De maneira geral, sistemas de monitoramento são projetados para alcançar quatro objetivos fundamentais [Mansilha et al. 2010]: abrangência, detalhamento, frequência e duração. A abrangência é relevante do ponto de vista estatístico, para que a amostra coletada represente adequadamente a população investigada. O detalhamento impacta no nível de estratificação permitido pelos dados – um estudo pode, por exemplo, demandar o total de *bytes* transferidos em geral, ou exigir o total de bytes transferidos por classe de aplicação. Quanto maior a frequência, maior a chance do monitoramento permitir a observação de fenômenos de curta duração. Quanto mais longo for o monitoramento, maior a chance de observar fenômenos recorrentes. Claramente, os custos e as probabilidades de falha de um sistema de monitoramento aumentam na medida em que se ampliam sua abrangência, detalhamento, frequência e duração. Para prover tolerância a falhas, uma abordagem típica [Junior et al. 2018] é empregar redundância de recursos, que no caso são os agentes de monitoramento. Por exemplo, [Zhang et al. 2011] propõem super-dimensionar a quantidade de monitores usados para "fotografar" o sistema alvo da monitoração.

A proposta do presente artigo pode ser considerada uma abordagem complementar às técnicas de prevenção e tolerância a falhas, pois atua *a posteriori* à monitoração, em contraste com as técnicas de prevenção e tolerância a falhas, que atuam *in loco*, durante a monitoração. Além disso, a técnica pode ser explorada para melhorar a qualidade dos traços de monitoramento de sistemas distribuídos de larga escala sem, no entanto, aumentar o custo necessário para obtê-los. É importante observar que essa é a única alternativa possível quando o monitoramento já ocorreu.

A literatura é rica em investigações recentes que visam corrigir traços de monitorações de larga escala [Xie et al. 2019, Cheng et al. 2017, Xie et al. 2018, Cordeiro et al. 2021], mostrando assim a relevância e atualidade do problema. No entanto, a literatura ainda não contempla investigações que abordam esse problema específico usando técnicas de aprendizagem profunda, apesar das potencialidades. Por exemplo, há trabalhos que visam aprimorar matrizes de tráfego [Roughan et al. 2003], cada um usando uma classe de rede neural profunda distinta: *Feedforward* [Zhou et al. 2016] e ConvNets [Emami et al. 2019].

## 3. Correção de Traços de Monitoração Usando Aprendizado Profundo

A seguir apresenta-se a solução proposta para o problema de correção de traços de monitoração, começando pela descrição de um modelo formal do sistema alvo (Subseção 3.1), seguido pela discussão da técnica proposta para a correção (Subseção 3.2) e apresentação da arquitetura de rede neural para a correção dos traços (Subseção 3.3).

### 3.1. Modelo

Seja um sistema $G = \langle V, B, E \rangle$, onde:

- $V$ é o conjunto de entidades do sistema $v \in V = \{1, 2, \ldots, |V|\}$, por exemplo usuários no caso de sistemas baseados no paradigma *peer-to-peer*;
- $B$ é o conjunto das entidades de *bootstrap* (isto é, entidades que servem como ponto de entrada dos usuários do sistema) $b \in B = \{1, 2, \ldots, |B|\}$, por exemplo rastreadores no caso de aplicações BitTorrent;
- $E$ é o conjunto de arestas $(i, j) \in (V \times B)$, que indicam a ciência de presença *online* de um nodo $i$ por um *bootstrap* $j$, por exemplo, tal como reportado por uma *peerlist* obtida de um rastreador BitTorrent.

Como o sistema apresenta uma dinâmica de entrada e saída de usuários ao longo do tempo, definimos uma versão dinâmica $G^t$, onde $t$ indica um determinado instante $t \in T = \{1, 2, \ldots |T|\}$ no qual amostras foram obtidas do sistema monitorado. Amostras são coletadas em intervalos de tempo $\Delta t$, expresso em unidades de tempo.

O traço de monitoramento de um sistema representado segundo o modelo acima pode ser dado por uma matriz $S = [s_{v,t}]$ de tamanho $|V| \times |T|$ (como exemplificado na Figura 1). Nessa matriz, $s_{vt} = 1$ caso o nodo $v \in V$ tenha sido visto pelo menos uma vez no instante $t \in T$, e $s_{vt} = 0$ caso contrário. Assumimos que o valor positivo sempre representa a presença do nodo no sistema. Na prática, tal pode não representar precisamente a realidade, pois um par pode continuar registrado no *bootstrap* após uma desconexão não graciosa. No escopo do presente trabalho, argumenta-se que essa situação tende a ter impacto pequeno em monitoramentos de longo prazo e, por isso, deixa-se sua investigação como trabalho futuro. É importante lembrar que o valor 0 pode significar tanto uma ausência do par $v$ no instante $t$, como uma falha do sistema de monitoramento em capturar o par $v$ na fatia de tempo $t$. A seguir, apresenta-se a técnica de aprendizado profundo para resolver essa ambiguidade e corrigir o traço caso necessário.

### 3.2. Proposta de Processo para Correção de Traços Usando RNAs

A seguir é apresentado um algoritmo para corrigir traços de monitoramento de sistemas distribuídos de larga escala usando RNAs. O Algoritmo 1 ilustra o processo em pseudocódigo. As entradas do algoritmo são a matriz $S = [s_{v,t}]$ representando o traço a ser

corrigido, um limiar de correção $\alpha$ ($\alpha \in [0, 1]$), e um tamanho de janela deslizante $|X|$. A saída é uma matriz $S' = [s_{v,t}]$ representando um traço $S$ corrigido.

**Algoritmo 1:** Correção de traço usando RNA

```
   Entrada: S = [s_{v,t}], α, |X|
   Saída: S' = [s'_{v,t}]
1  início
2      S' ← S ;
3      V ← conjunto de pares de S;
4      T ← conjunto de fatias de tempo de S ;
5      para v ∈ V faça
6          para t ∈ T faça
7              se s_{v,t} == 0 então
8                  p_{v,t} ← g(X^{v,t});
9                  se p_{v,t} >= α então
10                     s'_{v,t} ← 1;
11                 fim
12             fim
13         fim
14     fim
15     retorna S'
16 fim
```

Conforme Cordeiro et al. [Cordeiro et al. 2021], a probabilidade de uma determinada amostra nula $s_{v,t} = 0$ ser um falso negativo, para um usuário $v$, decresce com a quantidade de amostras nulas na sua vizinhança. Com base nessa premissa, em síntese, o algoritmo avalia o entorno de cada $s_{v,t} \in S$ e obtém a probabilidade, usando a rede neural, de que uma determinada amostra nula é um falso negativo – ou seja, o usuário estaria presente no sistema mas não apareceu na amostra. Caso a probabilidade seja superior ao limiar de correção $\alpha$, a amostra $s_{v,t}$ é *corrigida* – ou seja, o valor de $s_{v,t}$ é alterado para 1.

Para simplificar a notação e o treinamento do modelo, para cada $s_{v,t} \in S$ definimos um vetor $X^{v,t} = [x_k^{v,t}]$, $k \in \{t-c, t-c+1, \ldots, t-2, t-1, t, t+1, t+2, \ldots, t+c-1, t+c\}$, onde o centro do vetor $c = \lfloor |X|/2 \rfloor$, e cada elemento $x_k^{v,t} \in X^{v,t}$ corresponde a um elemento $x_{v,z} \in S$ usando a relação $k = z$. A função de predição de probabilidade de falha de uma janela $X^{v,t}$ pode ser formulada como:

$$p_{v,t} = g\big(s_{v,t-c}, s_{v,t-c+1}, \ldots, s_{t-2}, s_{t-1}, s_{v,t}, s_{v,t-1}, s_{v,t+2}, \ldots, s_{v,t+c-1}, s_{v,t+c}\big) \quad (1)$$

onde $g(\cdot)$ é uma função da janela centrada na amostra $s_{v,t}$ de interesse.

Primeiramente, o algoritmo cria uma cópia de $S$ para $S'$ e inicializa os conjuntos $V$ e $T$ (linhas 2-4). O algoritmo percorre o conjunto de amostra $s_{v,t} \in S$ referentes a cada usuário $v \in V$ (linha 5) e cada fatia de tempo $t \in T$ (linha 6). Caso o elemento indique uma presença (*i.e.*, amostra "positiva"), o algoritmo avança sem efetuar qualquer mudança em $S'$. É importante lembrar que o caso contrário (*i.e.*, amostra nula) pode significar a ausência do usuário ou uma falha do sistema de monitoramento. Nesse caso, o valor é avaliado considerando a janela ao seu entorno por uma RNA treinada previamente (linha 8). Essa RNA retorna a probabilidade $p_{v,t} \in \mathbb{R}$ tal que $0 \leq p_{v,t} \leq 1$ do valor nulo encontrado na posição central da janela corresponder a uma falha de monitoramento. Dado um limiar inferior de configuração $\alpha$, o valor da posição central da janela $X^{v,t}$ é modificado para 1 se $p_{v,t} \geq \alpha$; ou mantido como 0 se $p_{v,t} < \alpha$ (linhas 9 e 10). Em síntese, a correção é mais agressiva quanto menor for o valor de $\alpha$.

### 3.3. Arquitetura de RNA Proposta para Correção de Traços

A Figura 2 ilustra a topologia da RNA proposta, que segue uma arquitetura *Feedforward*. Nessa arquitetura, todas as conexões entre camadas seguem o sentido de uma camada de entrada para uma camada de saída [Haykin 1999]. Note-se que a camada de entrada (*InputLayer*) é uma janela de tamanho $|X|$ centrada em $s_{v,t}$. A camada de saída gera um valor referente à probabilidade $p$ usando uma função de ativação *sigmoid*. As camadas intermediárias (também conhecidas como camadas ocultas, do inglês *hidden layers*) permitem assimilar padrões de janelas e, posteriormente, corrigir traços adequadamente. As camadas intermediárias incluem uma séries de um ou mais pares de camadas *Dense* e *Dropout*, o quais chamamos de *arranjo*. Como será visto posteriormente, será avaliado o impacto da quantidade de arranjos na qualidade dos resultados. As camadas do tipo *Dense* implementam o padrão no qual as camadas são completamente conectadas. As camadas do tipo *Dropout* evitam sobreajuste [Srivastava et al. 2014].

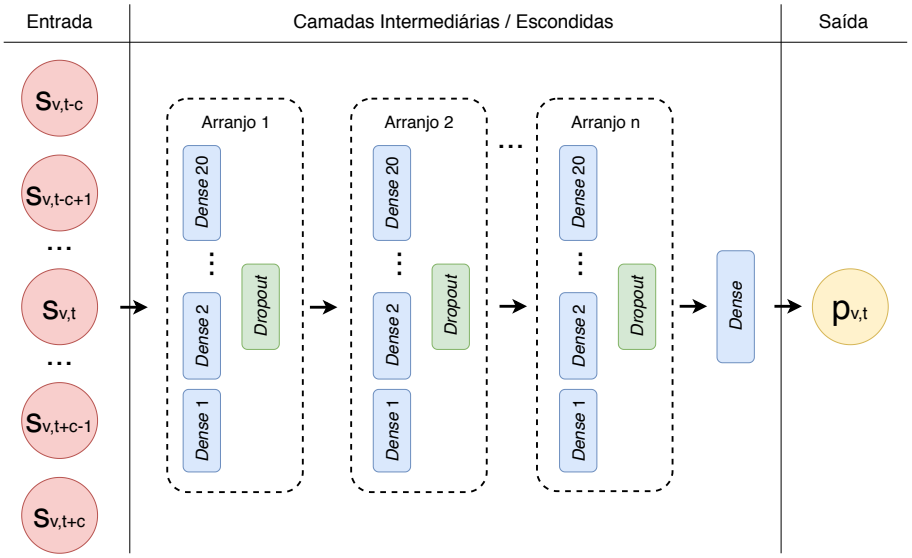



**Figura 2. Topologia da RNA**

A topologia proposta foi concebida essencialmente visando simplicidade, no que pode ser considerado o primeiro de potencialmente muitos passos em direção à correção de traços de sistemas distribuídos de larga escala usando técnicas de aprendizado profundo. Tanto a arquitetura *Feedforward* como a camada do tipo *Dense* podem ser considerados blocos básicos de construção de RNAs. Vislumbra-se como pesquisa futura explorar arquiteturas e combinações de camadas de RNAs mais sofisticadas.

## 4. Avaliação Experimental

Para aferir a viabilidade de se empregar RNAs no processo de regeneração de traços de monitoração de presença *online* de entidades em sistemas distribuídos, foi implementado um protótipo escrito em Python 3 do algoritmo e da RNA discutidas anteriormente. O protótipo, os *scripts* auxiliares, os traços usados no experimento e os resultados estão disponíveis no GitHub [Paim et al. 2021]. A implementação do algoritmo demandou alguns cuidados. Para facilitar o processamento, preencheu-se os espaços entre as múltiplas sessões de cada par com amostras nulas. Também realizou-se o preenchimento para permitir que todas as janelas a serem avaliadas estivessem completas, desde a primeira até a

última amostra. Especificamente, inseriu-se uma sequência de amostras nulas à esquerda (e à direita) da primeira (e última) amostra da primeira (e última) sessão de cada usuário.

A RNA foi implementada usando a API Keras do TensorFlow[1] (versão 2.4.1). Por exemplo, foram usadas as classes *Dense* e *Dropout* para implementar as respectivas camadas da arquitetura proposta. As instâncias da classe *Dense* dos arranjos foram parametrizadas com 20 unidades de neurônio por camada, nenhuma função de ativação e $bias$ verdadeiro. A última instância da classe *Dense* foi parametrizada com 1 unidade de neurônio, uma função de ativação *sigmoid*, e $bias$ verdadeiro. A classe *Dropout* foi parametrizada com taxa de queda de 20% na instância do primeiro arranjo, e 50% nas instâncias seguintes, como recomendado no artigo seminal sobre a camada *Dropout* [Srivastava et al. 2014]. Como método para otimização estocástica das funções objetivos, foi empregado o algoritmo Adam [Kingma and Ba 2015] no Keras. O algoritmo foi parametrizado seguindo sugestões do referido estudo: $\alpha_{adam} = 0,0001$, $\beta_1 = 0,9$ e $\beta_2 = 0,999$. Para facilitar a convergência do treinamento, em casos onde, para uma determinada janela de entrada, existam dois resultados diferentes possíveis, adotamos como resposta amostra nula, que é a opção mais conservadora. Futuramente, pretendemos explorar o impacto de outras configurações de parâmetros no processo de correção.

A técnica proposta foi avaliada em quatro etapas: análise de treinamento (Subseção 4.1), análise de sensibilidade paramétrica (Subseção 4.2), comparação com o estado da arte (Subseção 4.3) e estudo de caso (Subseção 4.4). A avaliação é baseada em alguns dos traços BitTorrent publicados por Cordeiro et al. [Cordeiro et al. 2021] e resumidos na Tabela 1. Os traços S1 e S2 foram escolhidos por contemplarem enxames BitTorrent pequenos ($|V| <$*peer list* enviado por rastreador), para os quais é possível afirmar que o traço é completo e, portanto, pode ser usado como *ground truth*. Na última etapa, analisou-se o resultado de um monitoramento de um enxame que possui tamanho médio maior que a capacidade de monitoramento. Para evitar condição de enviesamento no treinamento da RNA e aumentar o número de cenários avaliados, S1 e S2 foram divididos em quatro partes iguais. Em geral, usou-se a primeira parte de S2 para treinar a RNA e as quatro partes de S1 para avaliar os resultados gerados pela RNA.

**Tabela 1. Propriedades dos traços usados no estudo ($\Delta$ =15 minutos)**

| Arquivo | $\|T\|$ (Tempo) | # Pares $\|V\|$ | traço | % Arq. | # *Snapshots* $\|S\|$ |
|---|---|---|---|---|---|
| S1.bz2 | 7.196 (≈2,5 meses) | 69 | $E1_a$ | 25% | 131.577 |
| | | | $E1_b$ | 25% | 131.577 |
| | | | $E1_c$ | 25% | 131.577 |
| | | | $E1_d$ | 25% | 131.577 |
| S2.bz2 | 7.196 (≈2,5 meses) | 57 | $E2_a$ | 25% | 46.181 |
| S4.bz2 | 17.280 (≈6 meses) | 1.737 | $E3$ | 100% | 1.973.452 |

Para realizar análise de sensibilidade e impacto de sobreajuste, foram considerados diversos níveis de ruído nos traços. Foram escolhidos traços tidos como *ground truth* e neles injetou-se falhas seguindo uma distribuição uniforme com uma determinada probabilidade de falha injetada ($p_{fi}$). Por exemplo, para $p_{fi} = 1\%$, cada amostra tem 1% de chance de ser "danificada" (1 convertido em 0 para algum usuário). Dessa forma, no caso de uma falha ser injetada, para um determinado usuário, uma determinada amostra positiva é transformada em uma amostra nula. Em seguida, submeteu-se o traço com falha

[1] `https://www.tensorflow.org/`

injetada para ser corrigido pela técnica proposta, considerando um dado fator de correção $\alpha$. Por fim, comparamos o traço resultante com o traço original e o traço com falha para elaborar matriz de confusão e extrair os valores das métricas desempenho. As seguintes métricas, tradicionais em Aprendizagem de Máquina [Boutaba et al. 2018], foram consideradas: Acurácia, Precisão, *Recall* e *F1*.

A Tabela 2 resume os parâmetros e os respectivos valores utilizados na presente avaliação. Observe que nas etapas sobre sensibilidade paramétrica e potencial impacto de sobreajuste foram consideradas gamas de valores maiores que na comparação com algoritmo determinístico e no estudo de caso. Nessas etapas, foram realizadas 5 repetições, cada uma usando uma semente aleatória distinta para gerar falhas.

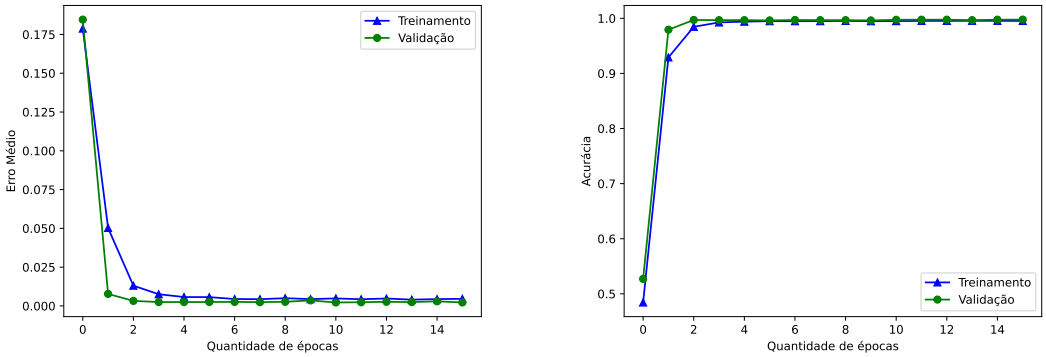
**Tabela 2. Parâmetros e valores considerados nas etapas desta avaliação**

| Etapa | Parâmetro | Valores |
|---|---|---|
| Todas | Tamanho da janela ($\|X\|$) | 11 |
| | Traço de treinamento | $E2_a$ |
| Sobreajuste | traço de avaliação | $E1_a$ |
| | Limiar inferior ($\alpha$) | 0,75 |
| | Quantidade de arranjos internos | 3 |
| | Probabilidade de falha injetada ($p_{fi}$) | 10% |
| | Épocas de treinamento | {1, 2, . . ., 15} |
| | Repetições | 1 |
| Sensibilidade | Traço a ser corrigido | $\{E1_a, E1_b, E1_c, SE_d\}$ |
| | Limiar inferior ($\alpha$) | {0,05; 0,75; 0,95} |
| | Quantidade de arranjos internos | {1; 3; 5} |
| | Probabilidade de falha injetada ($p_{fi}$) | 10% |
| | Épocas de treinamento | 10 |
| | Repetições | 5 |
| Comparação | Traço a ser corrigido | $\{E1_a, E1_b, E1_c, E2\}$ |
| | Limiar inferior ($\alpha$) | 0,75 |
| | Quantidade arranjos internos | 3 |
| | Probabilidade de falha injetada ($p_{fi}$) | {50%; 25%; 15%; 10%; 5%; 1%} |
| | Épocas de treinamento | 10 |
| | Repetições | 5 |
| Estudo de Caso | Traço a ser corrigido | $E3$ |
| | Limiar inferior ($\alpha$) | 0,75 |
| | Arranjos internos | 3 |
| | Épocas de treinamento | 10 |
| | Repetições | 1 |

## 4.1. Análise de Treinamento

Para evitar subajuste (*underfitting*) e sobreajuste (*overfitting*) no treinamento da RNA, foi avaliado o impacto da quantidade de épocas de treinamento na qualidade da resposta da RNA. Em síntese, o processo consiste em aplicar algoritmo de treinamento ao modelo de rede neural, e analisar o desempenho da RNA, comparando os resultados obtidos para as amostras de treinamento com os resultados obtidos para as amostras de testes.

As Figuras 3(a) e 3(b) mostram, respectivamente, a evolução do erro médio quadrático e da acurácia da RNA em função do número de épocas de treinamento. Visualmente, constata-se que as curvas tendem à estabilidade – o que indica a inexistência de subajuste e sobreajuste. Adicionalmente, constata-se que as curvas de treinamento e validação são próximas – indício adicional da inexistência de subajuste. Como resultado desta etapa, optou-se por adotar 10 épocas de treinamento no restante da avaliação.

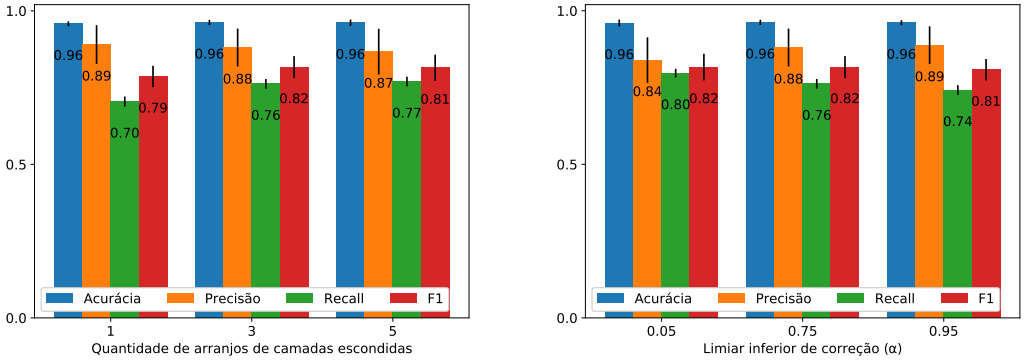



(a) Erro médio quadrático (b) Acurácia

**Figura 3. Impacto da quantidade de épocas no (a) erro médio quadrático e na (b) acurácia (quantidade de arranjos internos = 3)**

## 4.2. Análise de Sensibilidade

A Figura 4 apresenta os resultados da avaliação sobre a sensibilidade da técnica proposta em relação ao número de arranjos de camadas intermediárias (Figura 4(a)) e ao limiar inferior $\alpha$ de correção (Figura 4(b)). Em cada figura, para cada parâmetro variado, é apresentado um conjunto de barras – uma para cada métrica. Cada barra apresenta a média e o desvio padrão de cinco execuções.




(a) Limiar $\alpha = 0,75$ (b) Quantidade de arranjos $= 3$

**Figura 4. Sensibilidade da solução quanto aos arranjos (esq.) {1, 3, 5} e limiar $\alpha$ {0,05; 0,75; 0,95} (dir.) com probabilidade de falha injetada $p_{fi} = 10\%$**

Em geral, destacam-se duas observações principais a partir da Figura 4. Primeiro, podemos considerar que a sensibilidade do sistema em relação aos parâmetros e valores escolhidos é relativamente pequena. Uma hipótese é que a correção possível já esteja próxima de um limite superior. De todo modo, pretende-se explorar outros parâmetros e arquiteturas, incluindo tamanho da janela ($|X|$), em pesquisas futuras. Como esperado, a precisão aumenta conforme usamos um limiar mais conservador, ainda que minimamente; comportamento inverso pode ser observado para *recall*.

### 4.3. Comparação com o Estado da Arte

Para fins de comparação, o processo de correção foi realizado empregando a técnica baseada em análise estatística [Cordeiro et al. 2021]. O algoritmo foi configurado usando limiar superior de correção igual a 0,75. O valor foi escolhido por ter se mostrado a melhor opção entre os valores avaliados no referido estudo. É importante mencionar que o parâmetro alfa ($\alpha$) no referido artigo tem comportamento inverso ao parâmetro da técnica proposta no presente artigo – isto é, naquele caso, quanto maior o valor, menos conservadora é a técnica na correção das amostras que possam ter falhas.

A Figura 5 apresenta os resultados das métricas após correção para permitir a comparação da técnica baseada em aprendizado profundo apresentado neste trabalho e o método estatístico apresentada em trabalhos anteriores. Para cada probabilidade de falha injetada ($p_{fi}$=$\{$50%, 25%, 15%, 10%, 5% e 1%$\}$) são apresentadas 8 barras. Para cada uma das quatro métricas há um par de barras: uma para a técnica baseada em *machine learning* (ML), e outra para o método estatístico (sufixo DT). Cada barra apresenta o valor médio e o desvio padrão de cinco execuções. É importante observar que o algoritmo determinístico também sofre variação pois usamos sementes aleatórias distintas para gerar cada um dos traços com falha injetada usado em cada execução. Para fins de clareza, anotamos no gráfico apenas os valores das barras referentes à técnica proposta (ML).

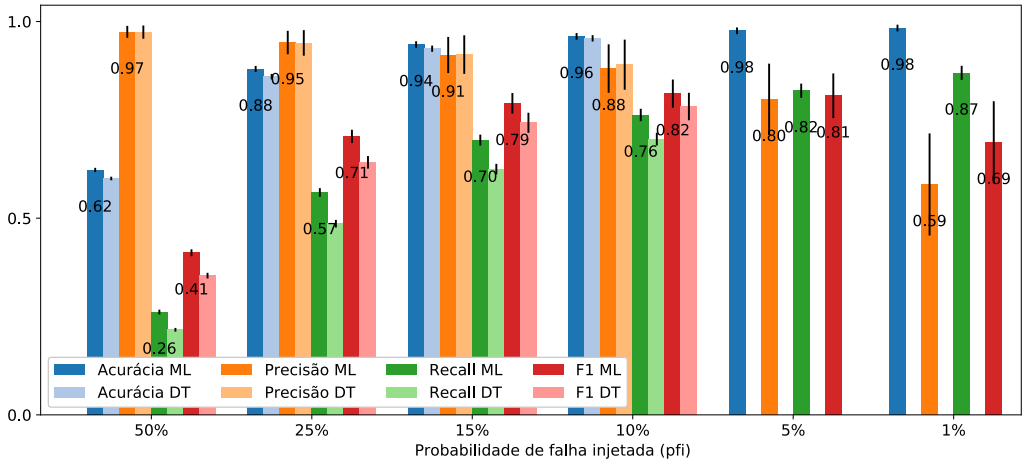



**Figura 5. Comparação da solução proposta (rótulos com sufixo ML) configurada com limiar inferior ($\alpha = 0,75$) e quantidade de arranjos ($a = 3$) fixos com técnica proposta anteriormente [Cordeiro et al. 2021] (DT), configurada com $\alpha = 0,75$, para diferentes probabilidades de falha injetada ($p_{fi}$)**

Destacam-se duas observações sobre a Figura 5. Primeiro, quando a probabilidade de falha injetada é igual ou superior à 10%, em quase todos os casos, a técnica proposta apresenta resultados superiores àqueles gerados pelo método estatístico. A exceção é o caso da probabilidade de falha injetada 10% e a métrica precisão, para a qual o método estatístico leva vantagem, ainda que relativamente pequena (menor que os respectivos desvios padrão). Esses resultados evidenciam a viabilidade técnica de se corrigir traços de sessões de usuários de sistemas distribuídos de larga escala usando aprendizado profundo.

Em segundo lugar, observa-se que, quando a probabilidade de falha injetada é menor que 10%, a técnica proposta baseada em aprendizado profundo é capaz de realizar correções enquanto que o método estatístico, não. Intuitivamente, a técnica baseada em

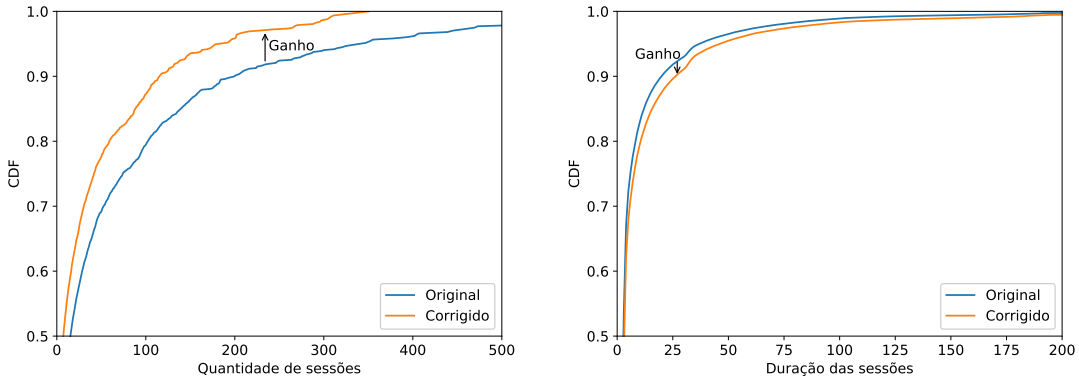
aprendizado profundo é mais adaptável que o método estatístico, o qual estima um valor de probabilidade de falha e corrige todo o traço usando o mesmo valor. Note-se que os resultados obtidos para a precisão em cenários com probabilidade de falha injetada $1\%$, embora possam parecer contra intuitivos, são plausíveis: a métrica apresentou uma redução em comparação com os outros casos devido à proporção dos acertos em um cenário com menos falhas e, portanto, mais desafiador do ponto de vista da precisão.

### 4.4. Estudo de Caso

Nesta etapa foi avaliado o impacto das correções em duas métricas de interesse sobre o traço de sessões de usuários de sistemas distribuídos, como em [Cordeiro et al. 2021]:

- **Quantidade de sessões**. Indica a quantidade de sequências de amostras positivas consecutivas observadas para um determinado usuário. No exemplo da Figura 1, o usuário 1 possui 3 sessões;
- **Duração das sessões**. Indica o tamanho de uma de determinada sequência de amostras positivas consecutivas para um determinado usuário. No exemplo da Figura 1, cada sessão do usuário 1 tem duração 2.

A Figura 6 apresenta a distribuição acumulada (CDF) para as duas métricas para os traços original ($S$) e corrigido ($S'$) usando a técnica proposta. Visualmente, observa-se o impacto da correção nas referidas métricas de sistema. Em trabalhos futuros, pretende-se analisar mais profundamente os impactos diretos e indiretos dessas métricas.




(a) (b)

**Figura 6. Impacto da correção com limiar inferior $\alpha = 0,75$ no traço $S3$ em termos de quantidade (esq.) e duração (dir.) de sessões**

## 5. Considerações Finais

O custo de se monitorar de forma eficaz e eficiente a presença *online* de entidades em sistemas distribuídos dinâmicos e de larga escala – tanto em termos de se garantir a ampla cobertura da monitoração e de minimizar a probabilidade de falhas durante a mesma – requer estratégias para melhorar a qualidade dos dados coletados após a monitoração, com vistas a remover falhas e ruídos ocorridos durante a monitoração. Embora rica em métodos estatísticos para correção de tais traços, as investigações ainda não tinham avaliado as potencialidades do uso de técnicas de aprendizado de máquina para esse fim.

Para preencher essa lacuna na literatura, o presente artigo propôs uma técnica baseada em aprendizado profundo para corrigir traços de sistemas distribuídos de larga escala. Resultados de uma avaliação com traços reais evidenciam as virtudes e limitações da proposta. Particularmente, mostrou-se que a técnica pode atingir resultados equivalentes aos do estado da arte em cenários com muitas falhas. Além disso, a técnica pode corrigir cenários com poucas falhas – nos quais as técnicas existentes são limitadas.

Os resultados promissores apresentados sugerem que a técnica proposta pode representar o primeiro de muitos passos em direção a técnicas mais sofisticadas de aprendizado profundo para correção de traços de sistemas de monitoramento distribuídos de larga escala. Nesse sentido, vislumbra-se as seguintes direções de pesquisa: (i) estender a avaliação da técnica, contemplando outros parâmetros e traços, para compreender melhor as suas limitações e potencialidades; (ii) explorar outras técnicas de redes neurais artificiais profundas, como *Long short-term memory* (LSTM), para aumentar a qualidade dos resultados; e (iii) explorar os cenários de adoção da técnica, potencialmente envolvendo ambientes mais desafiadores, como monitoramento de redes sensores e de *datacenter*.

## Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001. Este trabalho também recebeu apoio do Estado do RS, por intermédio da FAPERGS - Edital 10/2020.

## Referências

Anderson, S., Barford, C., and Barford, P. (2020). Five alarms: Assessing the vulnerability of us cellular communication infrastructure to wildfires. In *ACM Internet Measurement Conference*, IMC '20, page 162–175, New York, NY, USA. ACM.

Boutaba, R., Salahuddin, M. A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., and Caicedo, O. M. (2018). A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1):16.

Cheng, L., Niu, J., Kong, L., Luo, C., Gu, Y., He, W., and Das, S. K. (2017). Compressive sensing based data quality improvement for crowd-sensing applications. *Journal of Network and Computer Applications*, 77:123 – 134.

Cordeiro, W., Gaspary, L., Beltran, R., Paim, K., and Mansilha, R. (2021). Revisiting the coupon collector's problem to unveil users' online sessions in networked systems. *Peer-to-Peer Networking and Applications*.

Cordeiro, W., Mansilha, R. B., Santos, F. R., Gaspary, L. P., and Barcellos, M. P. (2014). Were you there? bridging the gap to unveil users' online sessions in networked, distributed systems. In *2014 Brazilian Symposium on Computer Networks and Distributed Systems*, pages 239–248.

Emami, M., Akbari, R., Javidan, R., and Zamani, A. (2019). A new approach for traffic matrix estimation in high load computer networks based on graph embedding and convolutional neural network. *Transactions on Emerging Telecommunications Technologies*, 30(6):e3604. e3604 ETT-18-0390.R2.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation*. Prentice Hall, Upper Saddle River, NJ. 2nd edition.

Hoßfeld, T., Lehrieder, F., Hock, D., Oechsner, S., Despotovic, Z., Kellerer, W., and Michel, M. (2011). Characterization of BitTorrent swarms and their distribution in the Internet. *Computer Networks*, 55(5):1197–1215.

Junior, N. A. A., Cordeiro, W. L. d. C., and Gaspary, L. P. (2018). Permitindo Maior Reprodutibilidade de Experimentos em Ambientes Distribuídos com Nodos de Baixa Confiabilidades. In *36º Simpósio Brasileiro de Redes de Computadores e de Sistemas Distribuídos (SBRC 2018)*, pages 1–14.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Lareida, A., Hoßfeld, T., and Stiller, B. (2017). The bittorrent peer collector problem. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 449–455. IEEE.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Mansilha, R. B., Mezzomo, A., Facchini, G., Gaspary, L. P., and Barcellos, M. P. (2010). Observando o universo bittorrent através de telescópios. In *28 Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, SBRC 2010, Porto Alegre, RS. SBC.

Mayer, J., Sahakian, V., Hooft, E., Toomey, D., and Durairajan, R. (2021). On the resilience of internet infrastructures in pacific northwest to earthquakes. In *Passive and Active Measurement*, pages 247–265, Cham. Springer International Publishing.

Padmanabhan, R., Schulman, A., Levin, D., and Spring, N. (2019). Residential links under the weather. In *ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 145–158, New York, NY, USA. ACM.

Paim, K. O., Beltran, R. D., Mansilha, R. B., and Cordeiro, W. (2021). GitHub - Correcting_Datasets_With_DL_SBRC21 repo. Available: `https://github.com/kayua/Correcting_Datasets_With_DL_SBRC21`.

Roughan, M., Thorup, M., and Zhang, Y. (2003). Traffic engineering with estimated traffic matrices. IMC '03, page 248–258, New York, NY, USA. ACM.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. 15(1):1929–1958.

Xie, K., Li, X., Wang, X., Xie, G., Wen, J., and Zhang, D. (2018). Graph based tensor recovery for accurate internet anomaly detection. In *IEEE INFOCOM 2018 - The 37th Annual IEEE Conference on Computer Communications*, pages 1502–1510.

Xie, K., Wang, X., Wang, X., Chen, Y., Xie, G., Ouyang, Y., Wen, J., Cao, J., and Zhang, D. (2019). Accurate recovery of missing network measurement data with localized tensor completion. *IEEE/ACM Transactions on Networking*, 27(6):2222–2235.

Zhang, C., Dhungel, P., Wu, D., and Ross, K. W. (2011). Unraveling the bittorrent ecosystem. *IEEE Transactions on Parallel and Distributed Systems*, 22(7):1164–1177.

Zhou, H., Tan, L., Zeng, Q., and Wu, C. (2016). Traffic matrix estimation: A neural network approach with extended input and expectation maximization iteration. *Journal of Network and Computer Applications*, 60:220 – 232.