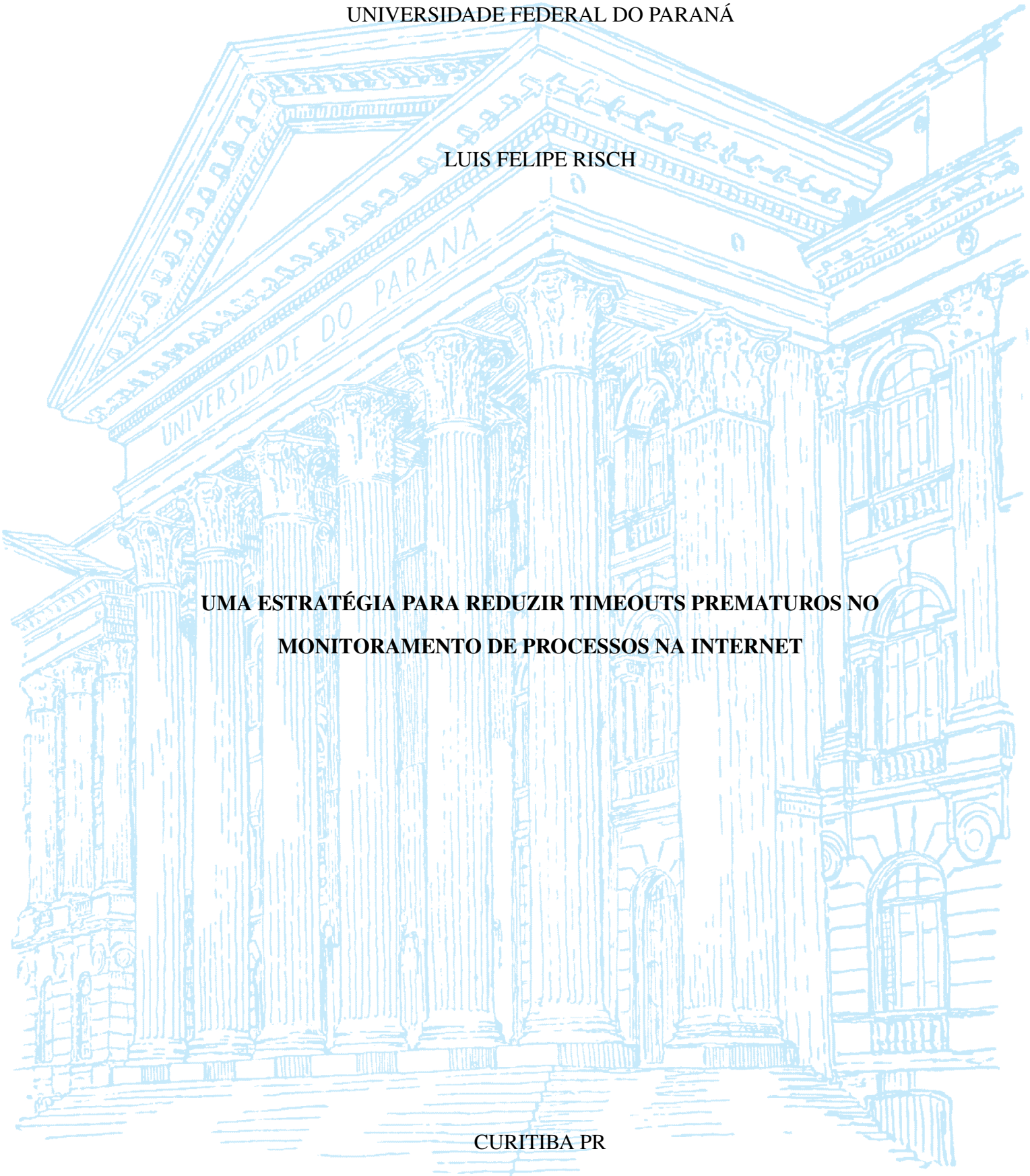
UNIVERSIDADE FEDERAL DO PARANÁ

LUIS FELIPE RISCH



UMA ESTRATÉGIA PARA REDUZIR TIMEOUTS PREMATUROS NO MONITORAMENTO DE PROCESSOS NA INTERNET

CURITIBA PR

2025

LUIS FELIPE RISCH

UMA ESTRATÉGIA PARA REDUZIR TIMEOUTS PREMATUROS NO MONITORAMENTO DE PROCESSOS NA INTERNET

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Computação*.

Orientador: Elias Procópio Duarte Jr..

CURITIBA PR

2025

AGRADECIMENTOS

Primeiramente, agradeço à minha família por todo o suporte incondicional, especialmente no que tange aos meus estudos. Não foram economizados esforços para me oferecer uma educação de qualidade e, sem esse pilar, a conclusão do bacharelado em Ciência da Computação não seria possível.

À Ana Paula Cardoso, minha namorada, agradeço pelo companheirismo e incentivo contínuo. Sou grato pela paciência ao ouvir meus desabafos sobre a faculdade e por estar ao meu lado nos momentos de estresse. Sem dúvida, esta jornada teria sido muito mais árdua sem a sua presença.

Aos meus amigos, obrigado por proporcionarem os momentos de descontração necessários para o meu equilíbrio durante o curso.

Agradeço também aos membros da banca examinadora: Fernando Kiotheka, Rogerio Correa Turchetti e Vinícius Fulber Garcia. As críticas, elogios e sugestões foram fundamentais para o refinamento deste trabalho.

Por fim, expresso minha gratidão ao professor Elias Duarte Jr. pela orientação oferecida ao longo deste ano. Sua contribuição foi de suma importância para o planejamento e a execução deste trabalho.

RESUMO

A confiabilidade é um requisito crítico para sistemas distribuídos modernos, onde a impossibilidade de distinguir deterministicamente entre processos falhos e lentos impõe o uso de mecanismos probabilísticos como o algoritmo de Van Jacobson (Jacobson, 1988), cuja sensibilidade a oscilações de rede frequentemente resulta em *timeouts* prematuros e, consequentemente, custos operacionais desnecessários. Para mitigar esse problema, este trabalho propõe e avalia a estratégia "Novo RTO", que integra a métrica de duração do erro (*Mistake Duration*) (Turchetti et al., 2016; Chen et al., 2002) para escalar o conservadorismo conforme a recorrência observada de *timeouts* prematuros. A avaliação experimental, utilizando *traces* reais de monitoramento de processos na Internet, demonstrou a superioridade da proposta frente ao algoritmo de Jacobson (RTO do TCP), alcançando, no melhor cenário, uma redução superior a 99,5% na quantidade de *timeouts* prematuros e maior agilidade na correção de falsas suspeitas, embora imponha um aumento no tempo de detecção de falhas legítimas em redes instáveis. O Novo RTO apresenta-se como uma solução robusta para sistemas que priorizam a redução de falsas suspeitas.

Palavras-chave: Cálculo de Timeout. Detectores de Falhas. Monitoramento de Processos. Sistemas Distribuídos. Internet.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

UDP	*User Datagram Protocol*
IP	*Internet Protocol*
TTL	*Time To Live*
UTC	*Coordinated Universal Time*
CSV	*Comma Separated Values*
AWS	*Amazon Web Services*
EC2	*Elastic Compute Cloud*
vCPU	*Virtual Central Processing Unit*
GiB	*Gibibyte*
RAM	*Random Access Memory*
LTS	*Long-Term Support*
QoS	*Quality of Service*
RTO	*Retransmission Timeout*
RTT	*Round-Trip Time*
TCP	*Transmission Control Protocol*

SUMÁRIO

1 INTRODUÇÃO

Com a expansão exponencial da Internet e de serviços geograficamente distribuídos, a capacidade de monitorar com precisão o estado operacional de processos remotos tornou-se uma tarefa fundamental e complexa (Duarte et al., 2010). Em ambientes assíncronos, como a Internet, onde não existem limites superiores garantidos para o tempo de transmissão de mensagens e execução de tarefas, a distinção determinística entre um processo falho (*crash*) e um processo que está simplesmente lento é impossível de ser realizada com certeza absoluta. Este fato foi irrevogavelmente estabelecido pelos resultados de impossibilidade de Fischer, Lynch e Paterson (FLP) (Fischer et al., 1985). Nesse trabalho, os autores demonstraram que em um sistema assíncrono sujeito a uma única falha, não é possível alcançar o consenso de forma determinística. Apesar dessas limitações teóricas, sistemas distribuídos reais precisam tomar decisões mesmo diante da incerteza. Por isso, recorrem a mecanismos baseados em temporização ou em estimativas probabilísticas para inferir o estado dos processos remotos. Esses mecanismos são fundamentais para operações como eleição de líder, remoção de nós de um cluster ou reconfiguração de réplicas (Huff et al., 2021) para manter consistência e disponibilidade. Foi nesse contexto que Chandra e Toueg (Chandra e Toueg, 1996) propuseram os detectores de falhas, que fornecem uma abstração prática para lidar com a impossibilidade demonstrada por Fischer, Lynch e Paterson (Fischer et al., 1985) e permitem construir sistemas tolerantes a falhas de maneira estruturada.

Esses detectores de falhas monitoram outros processos por meio da troca de mensagens (*heartbeats*): se a resposta chega dentro de um intervalo de tempo pré-estabelecido, o processo é considerado correto; caso contrário (*timeout*), passa a ser suspeito de ter falhado. Esse mecanismo é inerentemente sujeito a incertezas. Atrasos da rede, podem fazer um processo saudável aparentar estar falho. Sendo assim, os detectores de falhas são, por definição, não confiáveis. Eles podem tanto suspeitar de um processo correto (falsa suspeita), causado por um *timeout* prematuro, quanto demorar para identificar uma falha legítima. Para mitigar esses problemas, diferentes estratégias são usadas para ajustar o cálculo do valor do *timeout*, variando de métodos simples baseados em limites fixos até técnicas adaptativas que estimam o comportamento da rede ao longo do tempo. Assim, mesmo diante das restrições impostas pelo modelo assíncrono, os detectores de falhas oferecem uma abstração prática que permite construir sistemas tolerantes a falhas.

Dessa forma, é evidente que a precisão no cálculo do *timeout* exerce um papel fundamental na eficácia dos detectores de falhas. Como esses detectores baseiam suas decisões na chegada ou ausência de mensagens periódicas, um *timeout* mal dimensionado tem repercursão negativa no sistema distribuído. Valores agressivos tendem a gerar falsas suspeitas, enquanto *timeouts* muito conservadores aumentam o tempo necessário para identificar falhas legítimas. Assim, o algoritmo responsável por estimar esse limite temporal precisa capturar, de forma sensível, o comportamento dinâmico da rede, ajustando-se às variações de atraso e evitando decisões equivocadas. Na literatura, encontra-se diversas propostas para o cálculo do *timeout*: (Chen et al., 2002); (Nunes e Jansch-Porto, 2004); (Dixit e Casimiro, 2010); (Tomsic et al., 2015); (Bertier et al., 2003); (de Sá e de Araújo Macêdo, 2010); (Moraes e Duarte Jr, 2011). Mas, a proposta estabelecida como padrão para o cálculo de *timeouts* adaptativos foi o algoritmo de Van Jacobson (Jacobson, 1988), originalmente implementado no protocolo TCP (Postel, 1981).

Embora o algoritmo de Jacobson tenha se consolidado como o padrão de cálculo de *timeouts* adaptativos, observa-se, na prática, que diante de pequenas oscilações naturais da rede, resultam em uma taxa considerável de *timeouts* prematuros. Originalmente no TCP, um *timeout*

é interpretado como sinal de perda de pacote, forçando a redução da janela de congestionamento e, por consequência, a vazão da aplicação. Em sistemas distribuídos, um *timeout* gera uma falsa suspeita, podendo desencadear processos de tratamento desnecessários e onerosos.

Neste contexto, o presente trabalho propõe, implementa e avalia uma nova estratégia para o cálculo de *timeout*, denominada Novo RTO. Esta abordagem fundamenta-se na hipótese de que incorporar o tempo de erro médio ao cálculo do *timeout* de Jacobson, reduz a sensibilidade a pequenas oscilações da rede. O objetivo é reduzir a taxa de *timeouts* prematuros sem impor uma penalidade proibitiva nas métricas de QoS de detectores de falhas: tempo de correção de uma falsa suspeita (*Mistake Duration*) e tempo de detecção de uma falha legítima (*Detection Time*). A validação desta proposta baseia-se em experimentos realizados em ambiente de nuvem global (AWS), utilizando *traces* reais de comunicação entre continentes distintos (América do Norte, Europa e Ásia), comparando o desempenho da estratégia proposta com o algoritmo clássico de Jacobson e outras abordagens, como a *Tuning Phi* (Turchetti e Duarte Jr, 2018) e o *Timeout* Estimado, uma outra estratégia definida neste trabalho.

Os resultados experimentais mostram que o algoritmo de Jacobson não é a alternativa mais adequada para detectores de falhas quando o objetivo central é reduzir *timeouts* prematuros. O algoritmo Novo RTO surge como uma solução mais robusta, conseguindo eliminar mais de 99% das falsas suspeitas produzidas pelo método clássico, sem comprometer sua capacidade de corrigir rapidamente os erros cometidos. Além disso, o Novo RTO mantém desempenho competitivo na detecção ágil de falhas legítimas em cenários onde a rede se comporta de forma estável.

O restante deste trabalho está organizado da seguinte forma: O Capítulo 2 apresenta a fundamentação teórica, discutindo o problema da detecção de falhas em sistemas assíncronos e detalhando os algoritmos comparados com a estratégia proposta, como o método clássico de Jacobson, a *Tuning Phi* e o *Timeout* Estimado. O Capítulo 3 descreve a proposta central deste trabalho, o algoritmo Novo RTO, apresentando sua concepção e especificação. O Capítulo 4 detalha a metodologia e a arquitetura desenvolvida para a coleta dos *traces* reais de monitoramento de processos na Internet. O Capítulo 5 descreve o procedimento de avaliação experimental, especificando como as métricas avaliadas foram extraídas dos *traces* reais. O Capítulo 6 apresenta os resultados experimentais e a discussão, comparando quantitativamente e qualitativamente o algoritmo proposto frente às outras estratégias. Por fim, o Capítulo 7 apresenta as conclusões finais e aponta direções para trabalhos futuros.

2 CONCEITOS FUNDAMENTAIS

A confiabilidade em sistemas distribuídos depende intrinsecamente da capacidade de monitorar o estado dos processos participantes e reagir adequadamente a suspeitas de falhas. Este capítulo apresenta os conceitos fundamentais que norteiam essa tarefa, descrevendo as estratégias que fundamentam e precedem a proposta deste trabalho.

Inicialmente, discute-se o problema da detecção de falhas em sistemas sem garantias temporais, definindo o mecanismo de *timeout* como a ferramenta prática para o monitoramento de falhas. Ainda nesta primeira etapa, são apresentadas as métricas de qualidade de serviço (QoS): tempo de detecção (T_D), duração do erro (T_M) e tempo para recorrência do erro (T_{MR}).

Na sequência, o capítulo revisa as estratégias de cálculo de *timeout* que serão utilizadas como comparação ao algoritmo proposto. A Seção 2.2 detalha o algoritmo de Van Jacobson, estabelecido como a referência clássica para estimativas adaptativas. Posteriormente, examina-se a estratégia *Tuning Phi*, que introduz dinamismo ao cálculo de Jacobson através de análise de tendências.

Por fim, é apresentado o algoritmo *Timeout* Estimado, uma abordagem, também definida neste trabalho, que utiliza a predição direta de tendência como limite temporal, servindo como exemplo sobre os riscos da agressividade na detecção.

2.1 O PROBLEMA DA DETECÇÃO DE FALHAS EM SISTEMAS ASSÍNCRONOS

Um sistema distribuído consiste de um conjunto de processos que se comunicam e cooperam entre si (Cachin et al., 2011). Para a cooperação, existem modelos de diagnóstico baseados em testes (Duarte Jr et al., 2023) como alternativa à abordagem clássica. No entanto, optou-se neste trabalho pela estratégia clássica: troca de mensagens (*heartbeats*) sobre um canal de comunicação. Em um sistema síncrono, existem limites superiores conhecidos e garantidos tanto para o tempo de chegada das mensagens quanto para a execução de instruções por um processo (Lynch, 1996). Nesse cenário idealizado, a detecção de falhas torna-se trivial: se uma resposta esperada não chega dentro de um limite de tempo previamente estabelecido, é possível concluir com certeza que o processo falhou. No entanto, o cenário real aproxima-se de um sistema assíncrono, onde não há qualquer garantia temporal rígida: mensagens podem sofrer atrasos arbitrários, serem perdidas, duplicadas ou entregues fora de ordem, e processos podem pausar indefinidamente sua execução.

Para lidar com essa imprevisibilidade temporal, o mecanismo de *timeout* torna-se uma ferramenta prática essencial. Um *timeout* é um evento acionado por um temporizador que expira após um limite de tempo especificado; se nenhuma informação é recebida dentro desse intervalo, assume-se a ocorrência de uma falha e uma ação corretiva é executada (Kebarighotbi e Cassandras, 2011). Em redes de computadores, *timeouts* são fundamentais para garantir confiabilidade na comunicação ponto a ponto, permitindo, por exemplo, que pacotes perdidos sejam detectados e retransmitidos (Kesselman e Mansour, 2005). Entretanto, calcular um *timeout* preciso é um desafio significativo, pois o tempo de comunicação varia devido a atrasos de processamento, congestionamento e oscilação da latência. Por isso, abordagens adaptativas, como o algoritmo proposto por Jacobson, buscam ajustar continuamente o valor do *timeout* de acordo com o comportamento observado na rede.

É justamente sobre essa base de incerteza e temporização que Chandra e Toueg (Chandra e Toueg, 1996) construíram a abstração teórica dos detectores de falhas não confiáveis. Eles

demonstraram que, embora seja impossível resolver o consenso determinístico em sistemas totalmente assíncronos com falhas (Fischer et al., 1985), essa impossibilidade pode ser contornada ao se equipar o sistema com um oráculo de falhas. Isto é, um módulo que fornece, ainda que com erros, informações sobre o estado dos processos. Na literatura, encontram-se diversas estratégias de monitoramento baseadas nesse modelo, aplicadas aos mais variados contextos (Duarte Jr et al., 1997; Turchetti e Duarte, 2015; Turchetti et al., 2016; Duarte Jr et al., 2023; Stein et al., 2023).

Os detectores de falhas são caracterizados segundo duas propriedades fundamentais: a completude, que exige que processos falhos sejam eventualmente suspeitos; e a precisão, que refere-se à capacidade do detector de não suspeitar de processos que permanecem corretos. A completude é facilmente garantida através do uso de mecanismos de *timeout*: se um processo falha (*crash*), ele deixa de enviar *heartbeats*, fazendo com que qualquer *timeout* finito expire e gere uma suspeita. O verdadeiro desafio técnico reside na precisão, pois, em sistemas assíncronos, é impossível distinguir, com certeza absoluta, um processo lento de um processo falho. A complexidade em lidar com essas falsas suspeitas é tamanha que certas abordagens na literatura, como o (Stein et al., 2023), sugerem que o processo monitorado voluntariamente deixe o sistema distribuído ao descobrir que foi suspeitado, a fim de preservar a precisão.

Para avaliar a qualidade de um detector de falhas, a literatura utiliza métricas tradicionais definidas por Chen, Toueg e Aguilera (Chen et al., 2002), que quantificam a eficácia do mecanismo sob diferentes condições de rede. Duas dimensões são consideradas: a velocidade (tempo para suspeitar de uma falha real) e a exatidão (capacidade de evitar falsas suspeitas). Ambas são fortemente influenciadas pelo valor do *timeout*: um valor agressivo melhora a velocidade mas prejudica a exatidão, enquanto um valor conservador tem o efeito oposto. O equilíbrio entre esses fatores depende diretamente do algoritmo de cálculo de *timeout* adotado, motivando o desenvolvimento de técnicas adaptativas.

Para formalizar a mensuração dessas dimensões de qualidade, analisa-se as transições de estado do detector. Conforme ilustrado na Figura 2.1 (Turchetti e Duarte Jr, 2018), que apresenta a visão do processo monitor ao observar um processo que falha no instante t_4, definem-se as seguintes métricas primárias:

- Tempo de detecção (*Detection Time* - T_D): mede o intervalo decorrido entre o instante em que o processo monitorado falha e o momento em que o detector passa a suspeitar permanentemente dele ($T_D = t_5 - t_4$);
- Duração de um erro (*Mistake Duration* - T_M): representa o tempo que o processo monitor leva para corrigir uma falsa suspeita, percebendo que o processo monitorado ainda está ativo ($T_M = t_2 - t_1$);
- Tempo para recorrência do erro (*Mistake Recurrence Time* - T_{MR}): determina o intervalo de tempo entre dois erros consecutivos cometidos pelo detector, iniciando no momento da primeira suspeita errada até o início da próxima ($T_{MR} = t_3 - t_1$).

Neste trabalho, as métricas T_D e T_M são adotadas como critérios fundamentais para a avaliação empírica da estratégia proposta (Novo RTO), conforme detalhado no Capítulo 6. Estes indicadores fundamentam a avaliação comparativa entre a solução desenvolvida e três algoritmos: o Jacobson clássico, a *Tuning Phi* e o *Timeout* Estimado. Os detalhes de funcionamento desses três algoritmos de base são apresentados nas seções seguintes, enquanto o algoritmo proposto é discutido em profundidade no Capítulo 3.

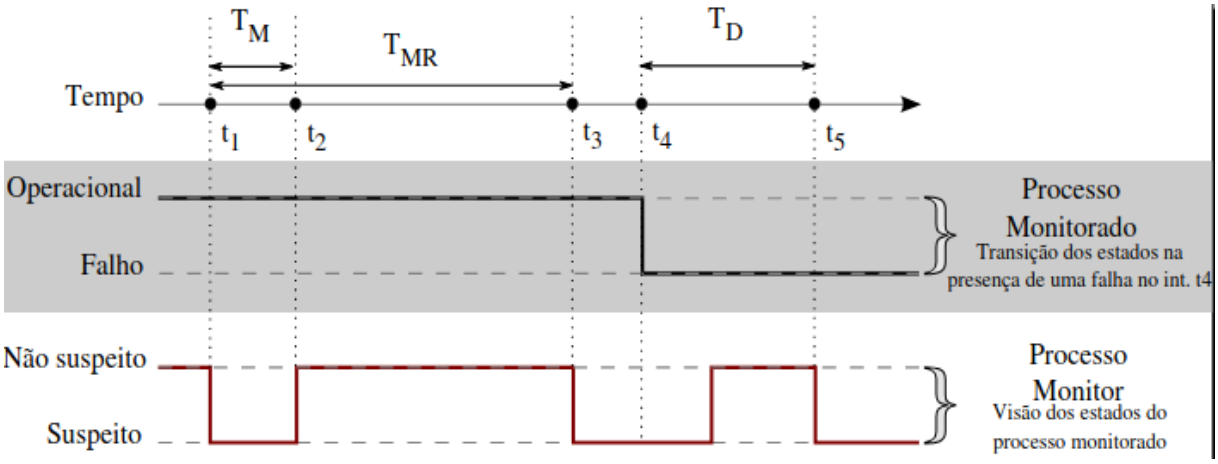


Figura 2.1: Métricas para avaliar a velocidade e a exatidão. Figura originalmente publicada em (Turchetti e Duarte Jr, 2018).

2.2 O ALGORITMO DE CÁLCULO DE *TIMEOUT* DE VAN JACOBSON

O algoritmo de cálculo de *timeout* proposto por Van Jacobson (Jacobson, 1988), originalmente desenvolvido para o protocolo TCP (Postel, 1981), tornou-se referência para estimativas adaptativas de tempo em redes. Sua principal contribuição é ajustar dinamicamente o valor do RTO com base no comportamento observado do RTT, reduzindo retransmissões desnecessárias. Embora tenha sido concebido para controle de congestionamento, o mesmo princípio pode ser aplicado de forma natural ao problema de detecção de falhas em sistemas distribuídos: em vez de indicar a necessidade de retransmitir pacotes, a expiração do *timeout* passa a indicar a possibilidade de que um processo monitorado tenha falhado.

No contexto de detecção de falhas, cada *heartbeat* recebido pelo detector indica que o processo monitorado continua correto. O detector, então, recalcula continuamente o próximo *timeout* com base nessas amostras, ajustando o limite temporal de suspeita.

O cálculo inicia pela obtenção do intervalo entre dois *heartbeats* consecutivos, representado na Equação 2.1. Essa diferença representa o tempo que a rede levou para entregar a mensagem mais recente:

$$intervalo\_heartbeat_{(k)} = heartbeat_{(k)} - heartbeat_{(k-1)} \tag{2.1}$$

Em seguida, o algoritmo mantém uma estimativa suavizada desse intervalo, combinando o valor histórico com a amostra mais recente. Essa média ponderada é atualizada conforme a Equação 2.2, onde γ (tipicamente definido como 0,1) é o peso dado para o novo intervalo:

$$intervalo\_medio_{(k+1)} = (1-\gamma) \cdot intervalo\_medio_{(k)} + \gamma \cdot intervalo\_heartbeat_{(k)} \tag{2.2}$$

Entretanto, a média por si só não captura a imprevisibilidade da rede. Por isso, Jacobson introduz uma medida de variação, que estima quão instáveis têm sido os intervalos observados. Ela é calculada pela diferença absoluta entre a nova amostra e a média estimada, suavizada de forma semelhante, conforme a Equação 2.3:

$$variacao_{(k+1)} = (1-\gamma) \cdot variacao_{(k)} + \gamma \cdot \left| intervalo\_heartbeat_{(k)} - intervalo\_medio_{(k+1)} \right| \tag{2.3}$$

Por fim, o algoritmo calcula o *timeout* propriamente dito. Como mostrado na Equação 2.4, o novo limite temporal é obtido somando-se o valor médio com múltiplos da variação. As constantes $\beta = 1$ e $\phi = 4$ definem o peso dado à média e à variação, respectivamente.

$$timeout\_heartbeat_{(k+1)} = \beta \cdot intervalo\_medio_{(k+1)} + \phi \cdot variacao_{(k+1)} \tag{2.4}$$

Vale destacar que o cálculo do *timeout* só pode ser iniciado a partir da chegada do segundo *heartbeat*. Isso ocorre, porque o algoritmo de Jacobson depende do intervalo entre mensagens sucessivas para computar tanto o intervalo médio quanto sua variação. Com apenas um *heartbeat* recebido, não há informação suficiente para calcular a diferença entre dois *heartbeats* nem para atualizar as métricas internas do algoritmo, já que todas as equações requerem, no mínimo, a diferença entre duas observações consecutivas. Assim, somente após a chegada do segundo *heartbeat* no detector, é possível calcular o primeiro intervalo, permitindo dar início ao processo adaptativo de atualização do *timeout*.

2.3 O ALGORITMO DE CÁLCULO DE *TIMEOUT TUNING PHI*

Embora o algoritmo de Jacobson seja amplamente utilizado devido à sua simplicidade e capacidade de adaptação, ele possui uma limitação inerente: a utilização da constante ϕ com valor fixo para o cálculo do *timeout*. A constante ϕ regula o peso dada para a variação, então alterá-la quando a rede tem variações cíclicas é proveitoso para aproximar o atraso medido no enlace da rede. Para endereçar essa questão, foi proposta a estratégia denominada *Tuning Phi* (Turchetti e Duarte Jr, 2018).

É fundamental ressaltar que a estratégia *Tuning Phi* preserva integralmente a estrutura algorítmica e as equações originais de Jacobson para o cálculo do *timeout*. A diferença reside exclusivamente na substituição do valor estático de ϕ por um valor dinâmico na Equação 2.4. O objetivo central da estratégia é ajustar esse valor de acordo com os padrões de comunicação observados. A premissa é que o cálculo do *timeout* deve refletir com maior fidelidade o comportamento variável do canal de comunicação.

Para determinar o valor ideal de ϕ a cada iteração, a estratégia baseia-se na análise de séries temporais. O algoritmo trata a sequência de tempos de comunicação das mensagens de monitoramento como uma série temporal, permitindo avaliar o comportamento atual do canal de comunicação para prever tendências futuras. A projeção do comportamento futuro da rede é modelada matematicamente pela equação de tendência linear apresentada na Equação 2.5:

$$T = l + \partial \cdot t \tag{2.5}$$

Nesta expressão, T representa o valor da tendência prevista (o tempo estimado para a próxima mensagem), t é o período futuro que se deseja estimar, l é o coeficiente linear e ∂ é o coeficiente angular.

Para obter a projeção T, é necessário calcular os coeficientes com base no histórico recente de mensagens. As Equações 2.6 e 2.7 detalham o cálculo do coeficiente linear (l) e do coeficiente angular (∂), respectivamente, sobre uma janela de observação.

$$l = \frac{n \cdot \sum_{i=1}^{n}(t_i \cdot y_i) - \sum_{i=1}^{n} t_i \cdot \sum_{i=1}^{n} y_i}{n \cdot \sum_{i=1}^{n}(t_i^2) - (\sum_{i=1}^{n} t_i)^2} \tag{2.6}$$

$$\partial = \frac{\sum_{i=1}^{n} y_i - l \cdot \sum_{i=1}^{n} t_i}{n} \tag{2.7}$$

Uma vez calculada a tendência T, o algoritmo precisa garantir que o novo *timeout* seja suficiente para cobrir essa estimativa de chegada, considerando também a margem de erro dada pela variação da rede. Essa exigência é formalizada na Equação 2.8, que estabelece que o *timeout* calculado deve ser sempre maior ou igual à soma da tendência prevista com a variação atual.

$$timeout\_heartbeat \geq T + variacao \tag{2.8}$$

Para encontrar o valor de ϕ que satisfaz essa condição, parte-se da equação original de cálculo de *timeout* de Jacobson, descrita na Expressão 2.4, e substitui-se $timeout\_heartbeat$ pela Inequação 2.8. Ao isolar a constante ϕ, obtém-se a Expressão 2.9. Onde $intervalo\_medio$ é dado pela Expressão 2.2 e a $variacao$ é dada pela Expressão 2.3.

$$\phi = \left\lceil \frac{(T + variacao) - intervalo\_medio}{variacao} \right\rceil \tag{2.9}$$

Finalmente, o valor inteiro obtido para ϕ através da Equação 2.9 é ajustado para permanecer dentro do intervalo pré-definido de $[1, 4]$ e, em seguida, inserido na estrutura original do algoritmo de Jacobson (Equação 2.4). Essa limitação é expressa formalmente pela Expressão 2.10:

$$\phi = \max(1, \min(\phi, 4)) \tag{2.10}$$

Dessa forma, a constante estática é substituída a cada iteração pelo valor do ϕ obtido, permitindo que o cálculo do *timeout* se adapte dinamicamente às tendências do canal, mantendo-se dentro de limites seguros de agressividade e conservadorismo.

2.4 O ALGORITMO DE CÁLCULO DE *TIMEOUT* ESTIMADO

O algoritmo de cálculo de *timeout* denominado *Timeout* Estimado apresenta-se como uma consequência direta da fundamentação matemática utilizada para a estratégia *Tuning Phi*. No entanto, esta abordagem diverge significativamente quanto à aplicação do resultado da predição. Enquanto a *Tuning Phi* utiliza a análise de séries temporais como uma ferramenta auxiliar para calibrar o ϕ da Equação 2.4, o algoritmo *Timeout* Estimado utiliza o valor da predição como o próprio limite temporal.

Nesta abordagem, a estrutura clássica de Jacobson é inteiramente descartada. O algoritmo fundamenta-se na premissa de que a análise de tendência linear é capaz de fornecer uma aproximação suficientemente precisa do comportamento futuro da rede.

O funcionamento do algoritmo consiste em aplicar as equações de tendência linear (as mesmas utilizadas internamente pela *Tuning Phi*) sobre o histórico recente de mensagens. O valor resultante dessa projeção, que representa a expectativa matemática para o instante de chegada do próximo *heartbeat*, é adotado como o novo valor de *timeout*.

Entretanto, a eliminação da margem de segurança impõe um risco à exatidão do detector. Ao adotar estritamente o valor projetado pela tendência, o algoritmo torna-se intolerante a qualquer desvio no tempo de chegada dos *heartbeats*. Consequentemente, pequenas oscilações naturais da rede são suficientes para provocar a expiração do temporizador. Essa política resulta em uma alta incidência de *timeouts* prematuros, degradando a confiabilidade do monitoramento ao gerar um excessivo número de falsas suspeitas, comportamento este que é corroborado pelos dados experimentais apresentados na Seção 6.1.

2.5 CONCLUSÃO

Este capítulo estabeleceu a fundamentação teórica necessária para o monitoramento de processos em sistemas distribuídos assíncronos. Definiu-se que, dada a incerteza temporal inerente a esses sistemas, a utilização de detectores de falhas baseados em *timeout*, se torna essencial para o monitoramento de falhas. Além disso, apresentou-se as métricas de qualidade, como o tempo detecção (*Detection Time*) e duração do erro (*Mistake Duration*), que são utilizadas para a avaliação dos detectores de falhas.

A apresentação dos trabalhos correlatos permitiu mapear as diferentes abordagens para o cálculo *timeout*. Foi descrito o algoritmo de Jacobson como a referência clássica, baseada em parâmetros estáticos, e a estratégia *Tuning Phi*, que propõe a adaptação dinâmica da constante ϕ via análise de tendência. Por fim, o algoritmo *Timeout* Estimado foi apresentado como um caso extremo de predição direta que, ao eliminar a margem de segurança, tende a degradar a precisão do detector gerando um número excessivo de falsas suspeitas.

Com as métricas de avaliação definidas e o funcionamento desses mecanismos detalhado, tem-se o cenário preparado para a proposição de uma nova abordagem. O próximo capítulo dedica-se à apresentação do algoritmo Novo RTO, detalhando sua concepção e especificação.

3 O ALGORITMO PROPOSTO: NOVO RTO

O presente capítulo dedica-se à apresentação detalhada do algoritmo Novo RTO, desenvolvido para mitigar a sensibilidade de detectores de falhas a oscilações naturais da rede. A discussão inicia-se na Seção 3.1, que estabelece a motivação e os objetivos centrais da estratégia. Na sequência, a Seção 3.2 formaliza a especificação, descrevendo as equações de atualização do erro médio e sua relação direta com a métrica de *Mistake Duration*, além de ilustrar o comportamento do algoritmo através de um comparativo passo a passo com o algoritmo de Jacobson sob um *trace* real. Por fim, o capítulo é concluído com uma síntese do algoritmo proposto, conectando-o com os resultados do Capítulo 6.

3.1 INTRODUÇÃO AO ALGORITMO PROPOSTO

O objetivo central do algoritmo Novo RTO é aumentar a tolerância do detector à oscilações naturais da rede. A proposta consiste em preservar a base estatística de Jacobson, mas incorporar uma camada adicional que reage especificamente à ocorrência de erros. Ao quantificar e memorizar os *timeouts* prematuros, o algoritmo busca evitar que pequenas variações no tempo de chegada dos *heartbeats* resultem em *timeouts* prematuros recorrentes, ajustando a margem de segurança de forma dinâmica e proporcional à instabilidade observada.

3.2 ESPECIFICAÇÃO E CÁLCULO DO ERRO

A estratégia Novo RTO calcula o intervalo de *timeout* da seguinte maneira. A cada *heartbeat* recebido, o algoritmo primeiramente realiza a atualização das estimativas padrão de $intervalo\_medio$ e $variacao$. Essas medidas são calculadas seguindo estritamente as definições originais de Jacobson, apresentadas anteriormente nas Equações 2.2 e 2.3. A inovação da proposta reside no tratamento dado aos eventos de falsa suspeita.

Sempre que um *heartbeat* é recebido após o temporizador já ter expirado (*timeout* prematuro), caracteriza-se uma falha na predição do detector. Nesse momento, o algoritmo calcula a diferença entre o instante em que o *heartbeat* foi recebido e o valor calculado para o *timeout*, denominada $erro\_atual$.

Para manter a coerência com a nomenclatura definida na Seção 2.2, onde $heartbeat_{(k)}$ representa o instante de chegada da mensagem atual, definimos $timeout\_heartbeat_{(k)}$ como o instante exato em que o *timeout* da iteração atual expirou. É importante notar que essa diferença corresponde exatamente à definição da métrica *Mistake Duration* (T_M), utilizada na avaliação de qualidade de serviço de detectores de falhas. Assim, o cálculo do erro atual é dado pela Equação 3.1:

$$erro\_atual_{(k)} = heartbeat_{(k)} - timeout\_heartbeat_{(k)} \tag{3.1}$$

Assume-se que a Equação 3.1 só é computada quando $heartbeat_{(k)}$ é maior que $timeout\_heartbeat_{(k)}$, ou seja, quando ocorre um *timeout* prematuro. Com base nesse cálculo, o algoritmo atualiza a estimativa suavizada do erro ($erro\_medio$). Ressalta-se que essa variável é inicializada com o valor 0, partindo da premissa de que o sistema inicia sua operação em um estado estável. A atualização utiliza a mesma lógica de média móvel ponderada empregada na Equação 2.2, contudo, ela é condicional: ocorre apenas quando há uma falsa suspeita. Caso o *heartbeat* chegue dentro do prazo, o valor do erro médio é mantido inalterado.

As regras de atualização para o $erro\_medio$ na iteração $k + 1$ são definidas da seguinte forma:

Caso 1: Houve falsa suspeita (*timeout* prematuro)

$$erro\_medio_{(k+1)} = (1 - \gamma) \cdot erro\_medio_{(k)} + \gamma \cdot erro\_atual_{(k)} \tag{3.2}$$

Caso 2: Não houve falsa suspeita (chegada dentro do prazo)

$$erro\_medio_{(k+1)} = erro\_medio_{(k)} \tag{3.3}$$

Onde γ representa o peso dado à nova amostra de erro. Utiliza-se o mesmo valor padrão proposto por Jacobson: 0,1.

Por fim, o algoritmo calcula o *timeout*. A fórmula final, apresentada na Equação 3.4, é composta pela soma da estimativa padrão de Jacobson (Equação 2.4) com o valor do erro médio acumulado:

$$timeout\_heartbeat_{(k+1)} = \beta \cdot intervalo\_medio_{(k+1)} + \phi \cdot variacao_{(k+1)} + erro\_medio_{(k+1)} \tag{3.4}$$

Dessa maneira, se o detector estiver cometendo erros frequentes ou de grande duração, o componente $erro\_medio$ crescerá, forçando o *timeout* a ser mais conservador e reduzindo a probabilidade de recorrência do erro nas próximas mensagens.

Para ilustrar o impacto prático dessa abordagem, a Tabela 3.1 apresenta uma comparação passo a passo entre os valores calculados pelo algoritmo clássico de Jacobson e o Novo RTO, sob um *trace* real. Observa-se especificamente a terceira linha, onde um aumento sutil no intervalo de chegada (100,03 ms) supera o *timeout* estimado na iteração anterior (aproximadamente 99,95 ms), causando uma falsa suspeita em ambos os algoritmos. Esse erro, aciona a atualização no componente $erro\_medio$, criando uma margem de segurança no Novo RTO.

A importância dessa margem torna-se evidente na quinta linha. O intervalo observado (100,02 ms) é novamente superior ao limite calculado pelo Jacobson (99,99 ms), o que resulta em uma nova falsa suspeita. No entanto, o Novo RTO, protegido pelo erro médio acumulado anteriormente, estabeleceu um timeout mais conservador (100,06 ms). Como o intervalo de chegada foi inferior a esse valor, o Novo RTO evitou corretamente a falsa suspeita.

Tabela 3.1: Comparação passo a passo entre o algoritmo de Jacobson e o Novo RTO. Legenda: H_k: Instante de chegada do *heartbeat* (ns UTC); I_k: Intervalo entre *heartbeats* consecutivos (ms); Med_{k+1}: Intervalo Médio (ms); Var_{k+1}: Variação média (ms); Err_{k+1}: Erro Médio (ms); $Timeout_{Jac}$: Limite calculado pelo algoritmo de Jacobson para a próxima espera (ms); $Timeout_{NovoRTO}$: Limite calculado pelo NovoRTO para a próxima espera (ms).

H_k	I_k	Med_{k+1}	Var_{k+1}	Err_{k+1}	$Timeout_{Jac}$	$Timeout_{NovoRTO}$
1760801425531704664	-	-	-	-	-	-
1760801425631659623	99,954959	99,954959	0	0	99,954959	99,954959
1760801425731690937	100,031314	99,9625945	0,00687195	0,076355	99,9900823	100,0664373
1760801425831658524	99,967587	99,96309375	0,00663408	0,076355	99,98963007	100,06598507
1760801425931682538	100,024014	99,969185775	0,060798897	0,076355	100,212381363	100,288736363
1760801426031690521	100,007983	99,973065497	0,058210758	0,076355	100,205908529	100,282263529

3.3 CONCLUSÃO

Neste capítulo, foi especificado o Novo RTO, desenvolvido para mitigar a sensibilidade do modelo clássico de Jacobson frente a oscilações naturais da rede. A base da nova estratégia

consiste da incorporação do *Mistake Duration* ao cálculo do *timeout*, sendo capaz de ajustar dinamicamente a margem de segurança do detector. A descrição passo a passo evidenciou que essa abordagem permite absorver oscilações que, tradicionalmente, resultariam em falsas suspeitas.

4 PROCEDIMENTO PARA GERAÇÃO DE *TRACES* REAIS

Este capítulo apresenta o procedimento aplicado para a geração dos *traces* reais de monitoramento de processos na Internet, utilizados para a avaliação da estratégia proposta. Inicialmente, é descrita a arquitetura cliente-servidor desenvolvida para gerar os *traces* e implementada na linguagem C, incluindo as responsabilidades específicas do cliente (processo monitorado) e do servidor (detector). Em seguida, é apresentado o ambiente experimental na nuvem (AWS) onde as medições foram executadas, especificando a topologia das instâncias EC2 e como se comunicam. Por fim, são descritos os seis conjuntos de dados resultantes, com amostras de cada um, e são fornecidos os *links* para o código-fonte e os próprios *traces*.

4.1 SOBRE O PROCEDIMENTO

Para a geração dos *traces* utilizados na avaliação da proposta deste trabalho, foi desenvolvida uma arquitetura cliente-servidor, composta por dois programas principais escritos em linguagem C: um cliente e um servidor. O objetivo central dessa arquitetura é enviar pacotes em intervalos regulares de um ponto (processo monitorado) a outro (detector) e registrar o exato momento da chegada da mensagem em seu destino.

4.1.1 O Cliente

O programa cliente foi projetado com o objetivo de simular o tráfego de dados. A sua principal responsabilidade consiste no envio de pacotes UDP (Postel, 1980) para um ou mais endereços de servidor, os quais são especificados no momento da inicialização do programa.

Para garantir a regularidade do tráfego, utiliza-se um temporizador parametrizável interno que dispara o envio de mensagens em intervalos de tempo fixos. Para este trabalho, o temporizador foi configurado com um intervalo de 100 milissegundos. Imediatamente antes da transmissão de cada pacote, duas ações são executadas: Registra-se o horário de envio em UTC; Incrementa-se o número de sequência para identificar o pacote.

Esses dois dados, o horário de envio e o número de sequência, formam o conteúdo da mensagem UDP enviada ao servidor.

A operação do cliente ocorre por um período pré-determinado e configurável. No caso específico deste trabalho, utilizou-se um dia de operação. A execução é finalizada automaticamente assim que se atinge o número máximo de pacotes a serem enviados, conforme estipulado nos arquivos de configuração.

4.1.2 O Servidor

O programa servidor (detector) atua como o receptor dos dados, sendo o principal responsável pela geração dos *traces*. Para a recepção, é realizada uma escuta ativa por pacotes UDP na porta especificada durante a inicialização do programa.

Quando um pacote é recebido, executa-se um processo de coleta de dados. Primeiramente, é registrado imediatamente o horário de chegada local em UTC. Em seguida, são extraídos os dados do conteúdo do pacote, que incluem o horário de envio e o número de sequência inseridos pelo cliente. Por fim, extraem-se metadados do pacote, como por exemplo o campo TTL do cabeçalho IP.

Ressalta-se que o sistema registra os horários de envio e recebimento sem assumir a premissa de relógios sincronizados. Visto que, os cálculos de *timeout*, conforme detalhado no Capítulo 5, utiliza-se estritamente o instante de chegada dos *heartbeats* ao detector (tempo local).

Para cada pacote recebido, executa-se a extração de dados detalhada acima e uma nova linha é escrita no arquivo de *log*. Cada linha segue um padrão CSV, onde os valores são separados por ponto e vírgula, contendo os seguintes campos: IP do cliente; Porta do cliente; Horário de envio do pacote UDP no cliente; Horário de chegada do pacote UDP no servidor; Número de sequência; Número de saltos.

Por fim, o processo de coleta é encerrado quando se detecta a chegada da mensagem final, identificada pelo número máximo de sequência. Nesse momento, todos os arquivos de log são fechados e a execução do servidor é finalizada.

4.1.3 Disponibilidade do Código Fonte

O código-fonte completo dos programas cliente e servidor está publicamente disponível.[1]

4.2 AMBIENTE EXPERIMENTAL E COLETA DE DADOS

Para a aplicação prática da metodologia descrita anteriormente e geração dos *traces* utilizados neste trabalho, foi estabelecido um ambiente experimental controlado na nuvem.

A plataforma utilizada foi a AWS, permitindo o provisionamento de recursos computacionais em diferentes regiões geográficas para executar a comunicação de rede em longa distância de forma realista. Foram utilizadas instâncias virtuais do tipo EC2.

Ao todo, foram provisionadas três instâncias do tipo *t3.micro*, possuindo duas vCPU e um GiB de memória RAM, todas executando o sistema operacional Ubuntu Server 24.04 LTS. As instâncias foram distribuídas estrategicamente em continentes distintos para analisar o comportamento da rede em rotas intercontinentais, sendo alocadas nas seguintes regiões:

- ***us-east-1*** (Norte da Virgínia, EUA);
- ***ap-northeast-1*** (Tóquio, Japão);
- ***eu-west-2*** (Londres, Reino Unido).

Nesta configuração, cada uma das três instâncias atuou simultaneamente como cliente e servidor para as outras duas. Por exemplo, a instância dos Estados Unidos (EUA) enviava pacotes para o Japão e Reino Unido, ao mesmo tempo em que recebia pacotes de ambas. Esta abordagem resultou na coleta de seis conjuntos de *traces*. Cada *trace* representa uma das rotas unidirecionais entre os pares de instâncias.

Os *traces* são detalhados nas subseções a seguir. Para fins de exemplificação e visualização da estrutura dos dados, será apresentada uma tabela contendo as dez primeiras entradas de cada *trace*. Em cada tabela constam: endereço IP e porta do cliente, instante de transmissão e recebimento (UTC), número de sequência do pacote e quantidade de saltos que percorreu na Internet.

[1] O repositório do projeto pode ser acessado em: `https://github.com/LuisFelipeRisch/ClientServerHeartbeats/tree/main/client_server`

4.2.1 *Trace* 1: EUA → Japão

Este conjunto de dados registra os pacotes UDP enviados da instância norte-americana para a instância asiática. A coleta foi realizada durante um período de aproximadamente 24 horas, com início às 15:30:23 (UTC) de 18 de outubro de 2025 e término às 15:30:23 (UTC) de 19 de outubro de 2025. Para fins de visualização de estrutura, a Tabela 4.1 apresenta uma pequena amostra deste conjunto de dados.

Tabela 4.1: Amostra dos 10 primeiros pacotes do *trace* EUA → Japão.

CLIENT_IP	CLIENT_PORT	CLIENT_SENT_AT_NS	SERVER_RECEIVED_AT_NS	SEQUENCE_NUMBER	HOPS
54.82.58.189	39411	1760801423567827440	1760801423639266651	0	7
54.82.58.189	39411	1760801423667810663	1760801423739148911	1	7
54.82.58.189	39411	1760801423767810534	1760801423839136430	2	7
54.82.58.189	39411	1760801423867809400	1760801423939163470	3	7
54.82.58.189	39411	1760801423967809693	1760801424039157890	4	7
54.82.58.189	39411	1760801424067809028	1760801424139132653	5	7
54.82.58.189	39411	1760801424167806711	1760801424239136373	6	7
54.82.58.189	39411	1760801424267801302	1760801424339133822	7	7
54.82.58.189	39411	1760801424367850594	1760801424439185833	8	7
54.82.58.189	39411	1760801424467794604	1760801424539124619	9	7

4.2.2 *Trace* 2: EUA → Reino Unido

Este conjunto de dados registra os pacotes UDP enviados da instância norte-americana para a instância europeia. A coleta foi realizada durante um período de aproximadamente 24 horas, com início às 15:30:23 (UTC) de 18 de outubro de 2025 e término às 15:30:23 (UTC) de 19 de outubro de 2025. Para fins de visualização de estrutura, a Tabela 4.2 apresenta uma pequena amostra deste conjunto de dados.

Tabela 4.2: Amostra dos 10 primeiros pacotes do *trace* EUA → Reino Unido.

CLIENT_IP	CLIENT_PORT	CLIENT_SENT_AT_NS	SERVER_RECEIVED_AT_NS	SEQUENCE_NUMBER	HOPS
54.82.58.189	39411	1760801423567827440	1760801423604799063	0	10
54.82.58.189	39411	1760801423667810663	1760801423704719497	1	10
54.82.58.189	39411	1760801423767810534	1760801423804713835	2	10
54.82.58.189	39411	1760801423867809400	1760801423904703790	3	10
54.82.58.189	39411	1760801423967809693	1760801424004747401	4	10
54.82.58.189	39411	1760801424067809028	1760801424104697769	5	10
54.82.58.189	39411	1760801424167806711	1760801424204704312	6	10
54.82.58.189	39411	1760801424267801302	1760801424304694051	7	10
54.82.58.189	39411	1760801424367850594	1760801424404753093	8	10
54.82.58.189	39411	1760801424467794604	1760801424504691598	9	10

4.2.3 *Trace* 3: Japão → EUA

Este conjunto de dados registra os pacotes UDP enviados da instância asiática para a instância norte-americana. A coleta foi realizada durante um período de aproximadamente 24 horas, com início às 15:30:24 (UTC) de 18 de outubro de 2025 e término às 15:30:24 (UTC) de 19 de outubro de 2025. Para fins de visualização de estrutura, a Tabela 4.3 apresenta uma pequena amostra deste conjunto de dados

4.2.4 *Trace* 4: Japão → Reino Unido

Este conjunto de dados registra os pacotes UDP enviados da instância asiática para a instância europeia. A coleta foi realizada durante um período de aproximadamente 24 horas,

Tabela 4.3: Amostra dos 10 primeiros pacotes do *trace* Japão → EUA.

CLIENT_IP	CLIENT_PORT	CLIENT_SENT_AT_NS	SERVER_RECEIVED_AT_NS	SEQUENCE_NUMBER	HOPS
18.179.37.35	47062	1760801424491013927	1760801424563314482	0	7
18.179.37.35	47062	1760801424591005384	1760801424663279518	1	7
18.179.37.35	47062	1760801424691005240	1760801424763381900	2	7
18.179.37.35	47062	1760801424791012765	1760801424863284913	3	7
18.179.37.35	47062	1760801424891005925	1760801424963274244	4	7
18.179.37.35	47062	1760801424991006991	1760801425063295094	5	7
18.179.37.35	47062	1760801425091004442	1760801425163296572	6	7
18.179.37.35	47062	1760801425191006500	1760801425263284999	7	7
18.179.37.35	47062	1760801425291004857	1760801425363277613	8	7
18.179.37.35	47062	1760801425391005585	1760801425463302457	9	7

com início às 15:30:24 (UTC) de 18 de outubro de 2025 e término às 15:30:24 (UTC) de 19 de outubro de 2025. Para fins de visualização de estrutura, a Tabela 4.4 apresenta uma pequena amostra deste conjunto de dados

Tabela 4.4: Amostra dos 10 primeiros pacotes do *trace* Japão → Reino Unido.

CLIENT_IP	CLIENT_PORT	CLIENT_SENT_AT_NS	SERVER_RECEIVED_AT_NS	SEQUENCE_NUMBER	HOPS
18.179.37.35	47062	1760801424491013927	1760801424594248368	0	10
18.179.37.35	47062	1760801424591005384	1760801424694275342	1	10
18.179.37.35	47062	1760801424691005240	1760801424794204580	2	10
18.179.37.35	47062	1760801424791012765	1760801424894207796	3	10
18.179.37.35	47062	1760801424891005925	1760801424994190891	4	10
18.179.37.35	47062	1760801424991006991	1760801425094198425	5	10
18.179.37.35	47062	1760801425091004442	1760801425194190934	6	10
18.179.37.35	47062	1760801425191006500	1760801425294191932	7	10
18.179.37.35	47062	1760801425291004857	1760801425394164995	8	10
18.179.37.35	47062	1760801425391005585	1760801425494134687	9	10

4.2.5 *Trace* 5: Reino Unido → EUA

Este conjunto de dados registra os pacotes UDP enviados da instância europeia para a instância norte-americana. A coleta foi realizada durante um período de aproximadamente 24 horas, com início às 15:30:25 (UTC) de 18 de outubro de 2025 e término às 15:30:25 (UTC) de 19 de outubro de 2025. Para fins de visualização de estrutura, a Tabela 4.5 apresenta uma pequena amostra deste conjunto de dados

Tabela 4.5: Amostra dos 10 primeiros pacotes do *trace* Reino Unido → EUA.

CLIENT_IP	CLIENT_PORT	CLIENT_SENT_AT_NS	SERVER_RECEIVED_AT_NS	SEQUENCE_NUMBER	HOPS
3.8.48.89	38843	1760801425493826965	1760801425531704664	0	10
3.8.48.89	38843	1760801425593811839	1760801425631659623	1	10
3.8.48.89	38843	1760801425693824172	1760801425731690937	2	10
3.8.48.89	38843	1760801425793808171	1760801425831658524	3	10
3.8.48.89	38843	1760801425893815360	1760801425931682538	4	10
3.8.48.89	38843	1760801425993816563	1760801426031690521	5	10
3.8.48.89	38843	1760801426093814797	1760801426131640861	6	10
3.8.48.89	38843	1760801426193811542	1760801426231664767	7	10
3.8.48.89	38843	1760801426293807386	1760801426331671094	8	10
3.8.48.89	38843	1760801426393816945	1760801426431674212	9	10

4.2.6 *Trace* 6: Reino Unido → Japão

Este conjunto de dados registra os pacotes UDP enviados da instância europeia para a instância asiática. A coleta foi realizada durante um período de aproximadamente 24 horas, com início às 15:30:25 (UTC) de 18 de outubro de 2025 e término às 15:30:25 (UTC) de 19 de

outubro de 2025. Para fins de visualização de estrutura, a Tabela 4.6 apresenta uma pequena amostra deste conjunto de dados

Tabela 4.6: Amostra dos 10 primeiros pacotes do *trace* Reino Unido → Japão.

CLIENT_IP	CLIENT_PORT	CLIENT_SENT_AT_NS	SERVER_RECEIVED_AT_NS	SEQUENCE_NUMBER	HOPS
3.8.48.89	38843	1760801425493826965	1760801425598926642	0	10
3.8.48.89	38843	1760801425593811839	1760801425698910515	1	10
3.8.48.89	38843	1760801425693824172	1760801425798953697	2	10
3.8.48.89	38843	1760801425793808171	1760801425898901141	3	10
3.8.48.89	38843	1760801425893815360	1760801425998922080	4	10
3.8.48.89	38843	1760801425993816563	1760801426098917111	5	10
3.8.48.89	38843	1760801426093814797	1760801426198911725	6	10
3.8.48.89	38843	1760801426193811542	1760801426298898320	7	10
3.8.48.89	38843	1760801426293807386	1760801426398900530	8	10
3.8.48.89	38843	1760801426393816945	1760801426498919407	9	10

4.2.7 Disponibilidade do Conjunto de Dados

O conjunto completo dos seis *traces* está publicamente disponível.[2]

4.3 CONCLUSÃO

Este capítulo detalhou o protocolo adotado para a geração dos *traces* reais utilizados na avaliação da estratégia proposta neste trabalho. A discussão abrangeu desde a concepção da arquitetura cliente-servidor até a execução da coleta em um ambiente de nuvem global (AWS). A execução desse procedimento resultou na obtenção de seis *traces* que fundamentam este trabalho. Concluída a etapa de aquisição, torna-se necessário o seu processamento para viabilizar a avaliação e discussão. Sendo assim, o próximo capítulo dedica-se a descrever o procedimento de extração das métricas necessárias para a validação do algoritmo.

[2]O repositório contendo os *traces* podem ser acessado em: `https://github.com/LuisFelipeRisch/ClientServerHeartbeats/tree/main/traces`

5 AVALIAÇÃO EXPERIMENTAL DA ESTRATÉGIA PROPOSTA

Este capítulo detalha o procedimento aplicado para avaliar o algoritmo proposto (Novo RTO) em comparação com as estratégias: Jacobson Clássico, *Tuning Phi* e *Timeout* Estimado. As estratégias utilizadas para comparação estão devidamente apresentadas e detalhadas no Capítulo 2, enquanto o algoritmo proposto está descrito no Capítulo 3. O objetivo deste capítulo é descrever os procedimentos algorítmicos utilizados para extrair as métricas fundamentais de Qualidade de Serviço (QoS) a partir dos *traces* coletados. A estrutura do capítulo está organizada da seguinte forma: as Seções 5.1 e 5.2 descrevem, respectivamente, os algoritmos implementados para o cálculo do *Mistake Duration* e do *Detection Time*. A Seção 5.3 esclarece como a contabilização de *timeouts* prematuros é derivada diretamente do processamento apresentado na Seção 5.1. Na Seção 5.4, são indicados os repositórios públicos contendo os dados processados resultantes. Por fim, o capítulo é concluído com uma síntese do processo de avaliação, preparando o terreno para a discussão dos resultados.

5.1 *MISTAKE DURATION*

O processo de extração da métrica *Mistake Duration* segue o fluxo de execução descrito a seguir:

1. **Leitura dos dados brutos:** o *script* é inicializado apontando para o diretório que contém os arquivos de *log*. Os arquivos são processados sequencialmente, lendo-se cada linha, que representa um pacote, individualmente;

2. **Extração de campos essenciais:** de cada linha lida, são extraídos apenas os dois campos necessários: o tempo de chegada no servidor, registrado em UTC, e o número de sequência da mensagem;

3. **Processamento dos dados:** o núcleo do *script* instancia quatro calculadoras de *timeout* independentes, uma para cada algoritmo avaliado. À medida que o *script* processa os pacotes, os tempos de chegada e sequência são fornecidos a todas as quatro calculadoras;

4. **Verificação do *timeout* e classificação de eventos:** para cada pacote i lido do *trace*, seu tempo de chegada registrado é comparado com o *timeout* calculado da iteração anterior ($i - 1$). A comparação resulta em duas classificações:

   - ***Hit* (Acerto):** se o pacote i chega antes do *timeout* calculado. O algoritmo previu corretamente a chegada;
   - ***Miss* (Erro):** se o pacote i chega depois do *timeout* calculado. O algoritmo gerou uma falsa suspeita, ocasionada pelo *timeout* prematuro.

5. **Cálculo do *Mistake Duration*:** quando, e somente quando, um evento de *Miss* ocorre, o *script* calcula o *Mistake Duration* em milissegundos. Este valor é a diferença exata entre o tempo de chegada real do pacote i e o *timeout* que falhou (calculado em $i - 1$);

6. **Atualização de estatísticas e cálculo do próximo *timeout*:** após a verificação e o eventual cálculo do *Mistake Duration*, as estatísticas internas de cada algoritmo são atualizadas com base nos dados do pacote i. Com base nestas estatísticas recém-atualizadas, é calculado o novo *timeout* para o pacote subsequente ($i + 1$);

7. **Agregação e exportação:** cada valor de *Mistake Duration* calculado é armazenado em uma lista de resultados específica para cada algoritmo. Ao final do processamento de todo o *trace*, os resultados são agregados e exportados para arquivos em formato CSV, sendo gerado um arquivo por algoritmo.

5.1.1 Disponibilidade do Código Fonte

O *script* implementado em Python, utilizado para processar os *traces* e gerar os arquivos CSV desta métrica, encontra-se publicamente disponível para consulta.[1]

5.2 *DETECTION TIME*

A segunda métrica fundamental para a avaliação comparativa é o *Detection Time* (tempo de detecção). O objetivo desta métrica é avaliar a capacidade de um algoritmo detectar uma falha real rapidamente. Para extrair esta métrica, foi utilizada uma variação do *script* apresentado na seção anterior. O processo de extração ocorre da seguinte forma:

1. **Inicialização com ponto de falha:** o *script* é iniciado recebendo dois argumentos: o diretório do *trace* bruto e um parâmetro que dita o número de sequência exato em que vai ocorrer a falha *crash* no processo monitorado;

2. **Processamento do *trace*:** é realizado de forma idêntica à extração do *Mistake Duration*. Os dados de tempo de chegada e número de sequência são lidos e fornecidos às quatro calculadoras;

3. **Cálculo e armazenamento do *Detection Time*:** o processamento continua até que o *script* encontra o pacote i, cujo número de sequência corresponde ao segundo parâmetro informado durante a inicialização do programa. Neste exato momento, os passos descritos abaixo são executados:

   - Os algoritmos primeiramente atualizam suas estatísticas internas utilizando os dados deste pacote;
   - Imediatamente após, cada algoritmo calcula o próximo *timeout* com base em seu estado interno recém-atualizado;
   - Calcula-se a diferença entre o *timeout* previsto (calculado no passo anterior) e o instante registrado para a chegada do pacote i. Esta diferença é exatamente o *Detection Time*, que é então armazenado.

4. **Interrupção e exportação:** imediatamente após o cálculo no ponto de falha, a execução do *script* é interrompida. O *script* então exporta um arquivo CSV para cada algoritmo, contendo apenas uma linha. Esta linha registra em qual número de sequência a falha ocorreu e o tempo que o algoritmo levou para detectar a falha permanentemente, valor este denominado *Detection Time*.

Este procedimento é, portanto, executado múltiplas vezes, escolhendo-se de forma aleatória o ponto de falha, permitindo uma análise comparativa da agilidade de detecção de cada algoritmo avaliado.

[1] O repositório do projeto pode ser acessado em: `https://github.com/LuisFelipeRisch/ClientServerHeartbeats/blob/main/time_mistake.py`

5.2.1 Disponibilidade do Código Fonte

O *script* implementado em Python, utilizado para esta finalidade, encontra-se publicamente disponível para consulta.[2]

5.3 QUANTIDADE DE *TIMEOUTS* PREMATUROS

Esta métrica não requer um procedimento de extração adicional, pois ela é um subproduto direto do cálculo do *Mistake Duration*, detalhado na Seção 5.1. Conforme descrito no passo 4, cada evento classificado como *Miss* representa um *timeout* prematuro. O cálculo do *Mistake Duration* (passo 5) é uma consequência direta, executado somente quando um evento *Miss* é detectado. Da mesma forma, o passo 7 armazena um novo registro na lista de resultados para cada *Miss* ocorrido.

Portanto, a quantidade de *timeouts* prematuros para um determinado algoritmo é obtida simplesmente pela contagem total de eventos *Miss* registrados durante o processamento. Na prática, este valor corresponde exatamente ao número total de linhas, ou registros, presentes no arquivo CSV gerado para o respectivo algoritmo.

5.4 DISPONIBILIDADE DOS *TRACES* PROCESSADOS

A execução dos procedimentos de extração descritos nas Seções 5.1 e 5.2 sob os seis *traces* descritos no Capítulo 4, resultou na geração dos conjuntos de dados processados. Estes dados representam as métricas extraídas para cada um dos quatro algoritmos avaliados. Os resultados estão organizados em formato CSV e serviram de base para a avaliação e discussão apresentado no capítulo seguinte. Estes arquivos CSV estão publicamente disponíveis. Os dados estão estruturados por diretório: cada diretório de um *trace* específico contém uma diretório `csvs` com os resultados correspondentes. [3]

5.5 CONCLUSÃO

Este capítulo detalhou o processo que transforma os *traces*, coletados conforme descrito no Capítulo 4, nos conjuntos de dados processados que fundamentam este trabalho. Foram descritos os procedimentos de extração das métricas: *Mistake Duration*, *Detection Time* e quantidade de *timeouts* prematuros.

Finalmente, foram disponibilizados os *links* tanto para os *scripts* utilizados para as extrações quanto para os arquivos CSV resultantes. Com as métricas devidamente extraídas e os dados preparados, o trabalho segue para o próximo capítulo, com os resultados da avaliação comparativa e à discussão dos resultados obtidos.

[2] O repositório do projeto pode ser acessado em: `https://github.com/LuisFelipeRisch/ClientServerHeartbeats/blob/main/time_detection.py`

[3] O repositório contendo os arquivos CSV podem ser acessados em: `https://github.com/LuisFelipeRisch/ClientServerHeartbeats/tree/main/traces`

6 RESULTADOS EXPERIMENTAIS E DISCUSSÃO

Este capítulo dedica-se à apresentação e análise crítica dos dados obtidos experimentalmente, com o objetivo de avaliar o algoritmo Novo RTO em comparação às outras estratégias. A discussão é estruturada sequencialmente em torno de três métricas fundamentais. Inicialmente, a Seção 6.1 quantifica a robustez dos algoritmos através da análise da ocorrência de *timeouts* prematuros. Em seguida, a Seção 6.2 aprofunda a análise qualitativa desses erros sob a ótica do *Mistake Duration*, avaliando a agilidade de correção dos erros cometidos. Posteriormente, a Seção 6.3 examina o *Detection Time* para mensurar o impacto das estratégias no tempo de detecção de falhas legítimas (*crashes*). Por fim, o capítulo sintetiza os resultados e as discussões para consolidar os *trade-offs* e a viabilidade da proposta.

6.1 AVALIAÇÃO DA QUANTIDADE DE *TIMEOUTS* PREMATUROS

A métrica primária para avaliar a eficácia de um algoritmo de cálculo de *timeout* é a sua capacidade de evitar falsas suspeitas, ocasionadas por *timeouts* prematuros. A Tabela 6.1 apresenta a contagem total de *timeouts* prematuros gerados por cada algoritmo em todos os *traces* analisados. A análise dessa tabela traz uma relevante disparidade entre os algoritmos. Essa diferença de magnitude nos resultados evidencia como cada abordagem lida de forma distinta com as incertezas da rede.

Tabela 6.1: Comparativo da quantidade total de *timeouts* prematuros por algoritmo e *trace*.

Trace	**Total de Pacotes**	**Jacobson**	**Novo RTO**	***Tuning Phi***	***Timeout* Estimado**
Tóquio → Reino Unido	863.990	24.452	206	61.754	404.422
Tóquio → EUA	863.991	20.132	129	65.758	405.020
Reino Unido → Tóquio	863.985	20.421	165	65.931	400.037
Reino Unido → EUA	864.000	18.961	305	71.026	433.644
EUA → Tóquio	863.989	18.194	85	69.966	407.964
EUA → Reino Unido	864.000	19.557	352	68.905	428.691

O algoritmo *Timeout* Estimado apresentou o pior desempenho. Seus resultados foram consistentemente os mais elevados, gerando o maior número de *timeouts* prematuros em todos os cenários. A sua taxa de erro variou entre 46,3% (Reino Unido → Tóquio) e 50,19% (Reino Unido → EUA), onde gerou falsas suspeitas para mais da metade de todos os pacotes. Isso inviabilizaria sua utilização prática, pois o sistema passaria a maior parte do tempo tentando tratar falhas que não existem. Por outro lado, o algoritmo *Tuning Phi* obteve valores inferiores ao *Timeout* Estimado, variando entre 7,14% (Tóquio → Reino Unido) e 8,22% (Reino Unido → EUA).

O algoritmo clássico de Jacobson, que serve como base de comparação, apresentou um desempenho significativamente melhor que os anteriores, com uma taxa de *timeouts* prematuros que variou entre 2,10% (EUA → Tóquio) e 2,83% (Tóquio → Reino Unido). Contudo, embora a porcentagem pareça baixa, em números absolutos isso representa cerca de 20.000 *timeouts* prematuros, o que ainda gera um ruído considerável e desperdício de recursos computacionais no tratamento dessas exceções.

O algoritmo proposto, Novo RTO, demonstrou superioridade nesta métrica em relação aos anteriores. Em todos os cenários, ele gerou uma fração minúscula de falsas suspeitas em comparação com os demais. No seu melhor cenário (EUA → Tóquio), disparou apenas 85 vezes de um total de 863.989 *heartbeats*, correspondendo aproximadamente 0,009% do total de

pacotes. No seu pior cenário (EUA → Reino Unido), gerou 352 *timeouts* prematuros de um total de 864.000 *heartbeats*, uma taxa de aproximadamente 0,0407%. No melhor cenário, o algoritmo proposto conseguiu reduzir em 99,53% de *timeouts* prematuros em relação ao Jacobson, provando ser extremamente robusto ao lidar com oscilações naturais da rede.

Essa redução drástica na quantidade de *timeouts* prematuros confirma que o Novo RTO cumpre seu objetivo em lidar com oscilações naturais da rede. No entanto, surge a questão se essa virtude do algoritmo não cobra um preço em outras métricas. É necessário investigar se, ao evitar tantos erros, o algoritmo não se tornou excessivamente lento para reagir ou corrigir suas suspeitas. Essa verificação de qualidade será o foco das próximas seções.

6.2 AVALIAÇÃO DO *MISTAKE DURATION*

O *Mistake Duration* quantifica o tempo para uma falsa suspeita ser corrigida. Isto é, mede a diferença entre o instante de chegada do *heartbeat* e o tempo de chegada esperado (*timeout*) para o mesmo *heartbeat* quando um *timeout* prematuro ocorre. Quanto menor for este valor, mais rapidamente o sistema percebe que cometeu um erro e retorna ao estado normal de operação, minimizando o impacto da falsa suspeita.

À primeira vista, os dados das Tabelas 6.2 e 6.3 sugerem que o algoritmo Novo RTO possui o pior desempenho, apresentando valores de *Mistake Duration* significativamente mais altos que os demais. Essa leitura inicial poderia levar à conclusão equivocada de que o algoritmo, apesar de errar menos, causa danos maiores quando erra, demorando muito para se recuperar.

Tabela 6.2: Comparativo da média do *Mistake Duration* (em ms).

Trace	**Jacobson**	**Novo RTO**	***TuningPhi***	**Estimado**
Tóquio → Reino Unido	0,34	6,94	0,20	0,07
Tóquio → EUA	0,23	9,85	0,12	0,05
Reino Unido → Tóquio	0,28	8,26	0,12	0,04
Reino Unido → EUA	0,13	1,25	0,06	0,03
EUA → Tóquio	0,22	14,57	0,09	0,04
EUA → Reino Unido	0,19	2,42	0,08	0,03

Tabela 6.3: Comparativo do desvio padrão do *Mistake Duration* (em ms).

Trace	**Jacobson**	**Novo RTO**	***TuningPhi***	**Estimado**
Tóquio → Reino Unido	6,43	69,26	4,08	1,63
Tóquio → EUA	6,39	78,77	3,56	1,47
Reino Unido → Tóquio	7,19	76,82	4,00	1,63
Reino Unido → EUA	0,51	2,29	0,28	0,12
EUA → Tóquio	7,48	107,88	3,81	1,58
EUA → Reino Unido	0,77	2,39	0,43	0,18

Contudo, esta interpretação inicial é precipitada. A média elevada do Novo RTO não é um indicador de desempenho inferior, mas sim uma consequência estatística direta de sua principal virtude: a redução de *timeouts* prematuros. Ao filtrar os casos triviais (oscilações naturais da rede), o que resta para análise no Novo RTO são os picos de latência da rede, distorcendo a comparação direta das médias aritméticas simples.

Conforme demonstrado na Tabela 6.1, algoritmos como o de Jacobson geraram uma quantidade considerável de *timeouts* prematuros. A grande maioria desses erros é trivial, causada por flutuações normais de baixa magnitude na rede. Por consequência, o acúmulo desses pequenos erros dilui o valor da média, fazendo com que o desempenho geral pareça ser melhor.

O Novo RTO, por outro lado, gerou uma quantidade muito menor de *timeouts* prematuros quando comparado aos outros algoritmos. Ele é projetado para ignorar o ruído de baixo impacto

e disparar apenas em picos de latência significativos. Por definição, esses eventos são *outliers*. Sendo assim, o conjunto de amostras do Novo RTO é composto quase que exclusivamente por eventos de alta latência, o que eleva naturalmente a sua média.

A Tabela 6.3 comprova esta afirmação. O desvio padrão do Novo RTO é drasticamente maior na maioria dos *traces*, porque seu pequeno conjunto de dados é majoritariamente composto por estes *outliers*. Isso não reflete instabilidade do algoritmo, mas sim a natureza dos poucos eventos que foram capazes de enganar o detector e causar um *timeout* prematuro.

Portanto, a média e o desvio padrão elevados do Novo RTO não são uma fraqueza, mas sim uma evidência de que o algoritmo funciona como projetado: ele ignora o ruído trivial e foca seu acionamento apenas nos picos de latência mais severos. Para validar se ele corrige esses erros de forma eficiente, é necessário comparar apenas os eventos comuns a todos os algoritmos.

A análise descrita acima das Tabelas 6.2 e 6.3 é corroborada pela inspeção visual das Figuras 6.1 a 6.6. Tais figuras plotam o *Mistake Duration* exclusivamente para *timeouts* prematuros que foram comuns aos quatro algoritmos. É fundamental citar que este conjunto de dados, embora represente um subconjunto das falhas dos outros algoritmos avaliados, constitui a totalidade das falhas ocorridas no Novo RTO.

A inspeção visual das figuras permite uma comparação justa do tempo de correção de cada algoritmo nos mesmos pontos de *timeout* prematuro. Como é evidenciado em todos os gráficos, os pontos do Novo RTO situam-se, predominantemente, em um patamar inferior aos demais. Isto é, o algoritmo proposto leva menos tempo para corrigir falhas, provando que, ele é mais ágil que os concorrentes.

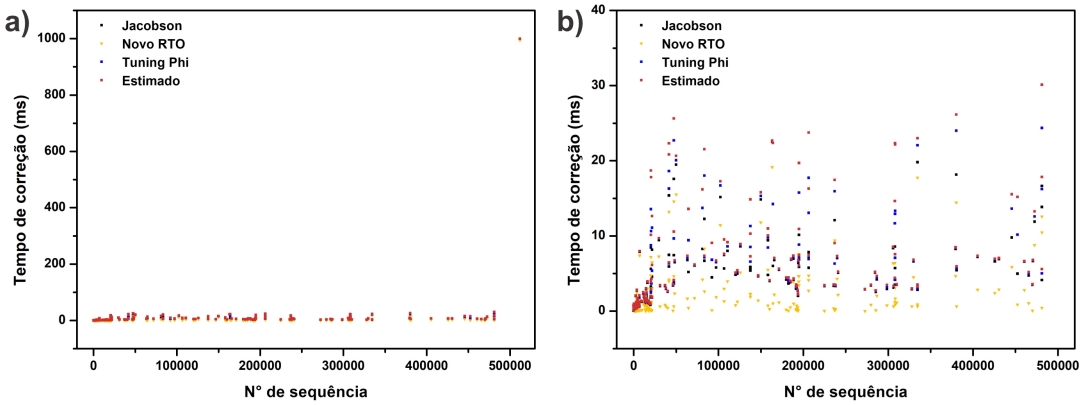


Figura 6.1: Tempo de correção (*Mistake Duration*) (ms) em função do número de sequência do pacote para *timeouts* prematuros comuns ao algoritmo Novo RTO e aos demais, obtidos do *trace* Tóquio → Reino Unido. (a) Visão geral dos dados. (b) Ampliação de uma seção de (a).

6.3 AVALIAÇÃO DO *DETECTION TIME*

Enquanto o *Mistake Duration* avalia a agilidade da correção de uma falsa suspeita, o *Detection Time* mensura a sua agilidade em identificar falhas legítimas (*crashes*). A análise desta métrica revela o custo associado à robustez demonstrada pelo algoritmo Novo RTO nas seções anteriores, mostrando o impacto no tempo necessário para confirmar a indisponibilidade de um processo monitorado.

Os resultados apresentados na Tabela 6.4 indicam que o algoritmo Novo RTO apresenta, em média, valores de *Detection Time* superiores aos algoritmos comparados. Este comportamento é esperado e representa um *trade-off*. É o preço pago pela sua virtude em evitar falsas suspeitas.

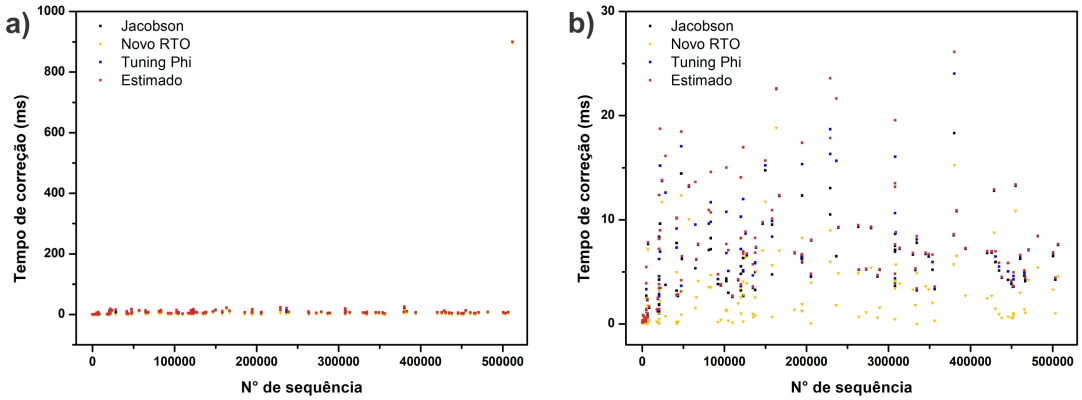


Figura 6.2: Tempo de correção (*Mistake Duration*) (ms) em função do número de sequência do pacote para *timeouts* prematuros comuns ao algoritmo Novo RTO e aos demais, obtidos do *trace* Tóquio → EUA. (a) Visão geral dos dados. (b) Ampliação de uma seção de (a).

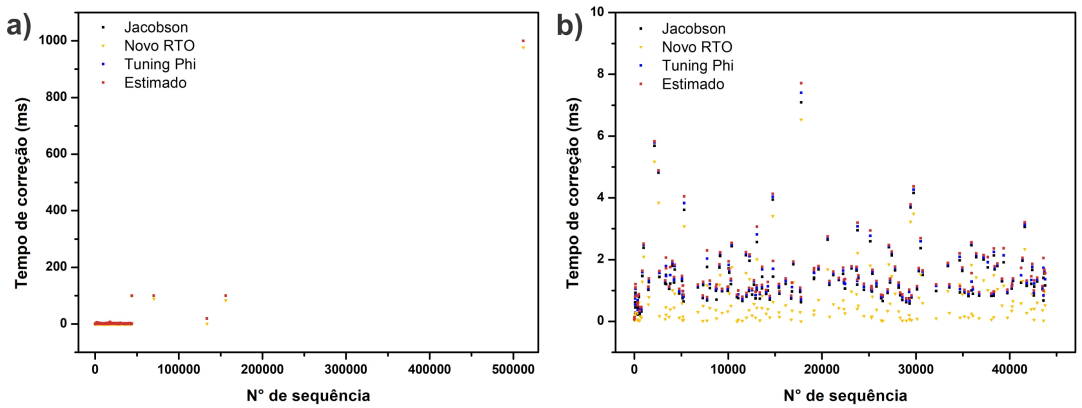


Figura 6.3: Tempo de correção (*Mistake Duration*) (ms) em função do número de sequência do pacote para *timeouts* prematuros comuns ao algoritmo Novo RTO e aos demais, obtidos do *trace* Reino Unido → Tóquio. (a) Visão geral dos dados. (b) Ampliação de uma seção de (a).

Aceita-se uma leve demora na confirmação de falhas *crash*, em troca da redução de *timeouts* prematuros e da diminuição do tempo para corrigir falsas suspeitas.

Para minimizar drasticamente a ocorrência de *timeouts* prematuros, o Novo RTO adota uma política de estimativa mais conservadora, resultando em um valor de *timeout* calculado ligeiramente maior que os demais. Consequentemente, quando ocorre uma falha real (*crash*), o algoritmo aguarda por um período maior antes de confirmar a suspeita, o que eleva a métrica do *Detection Time*.

Tabela 6.4: Comparativo da média do *Detection Time* (em ms).

Trace	**Jacobson**	**Novo RTO**	***TuningPhi***	**Estimado**
Tóquio → Reino Unido	100,44	134,12	100,21	100,00
Tóquio → EUA	100,09	153,99	100,04	100,00
Reino Unido → Tóquio	100,15	159,24	100,07	100,00
Reino Unido → EUA	100,12	102,80	100,05	100,00
EUA → Tóquio	100,12	146,77	100,05	100,00
EUA → Reino Unido	100,09	102,73	100,04	100,00

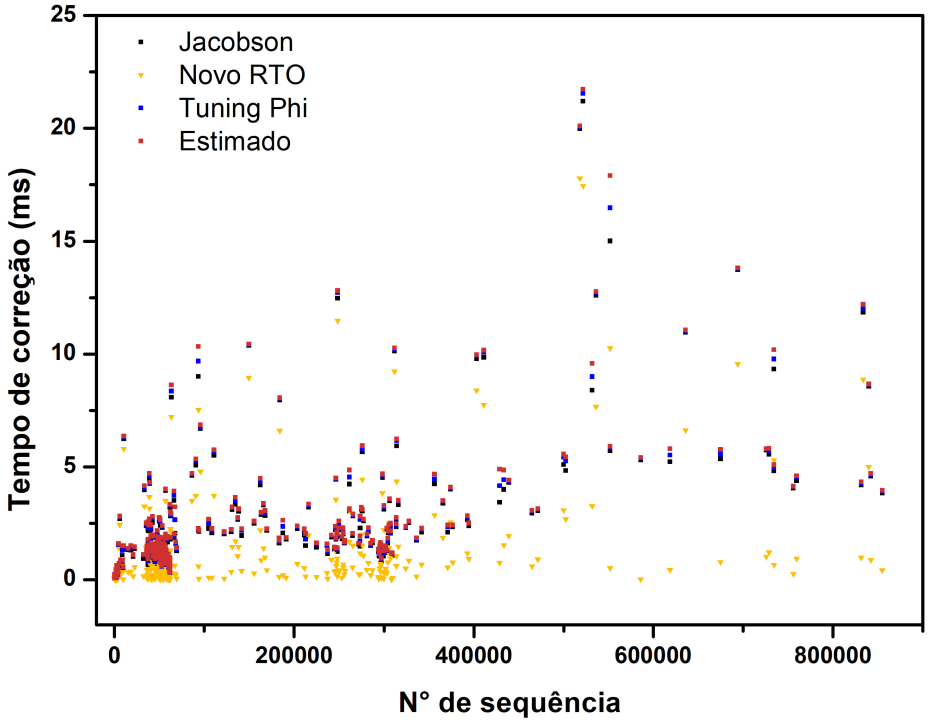


Figura 6.4: Tempo de correção (*Mistake Duration*) (ms) em função do número de sequência do pacote para *timeouts* prematuros comuns ao algoritmo Novo RTO e aos demais, obtidos do *trace* Reino Unido → EUA.

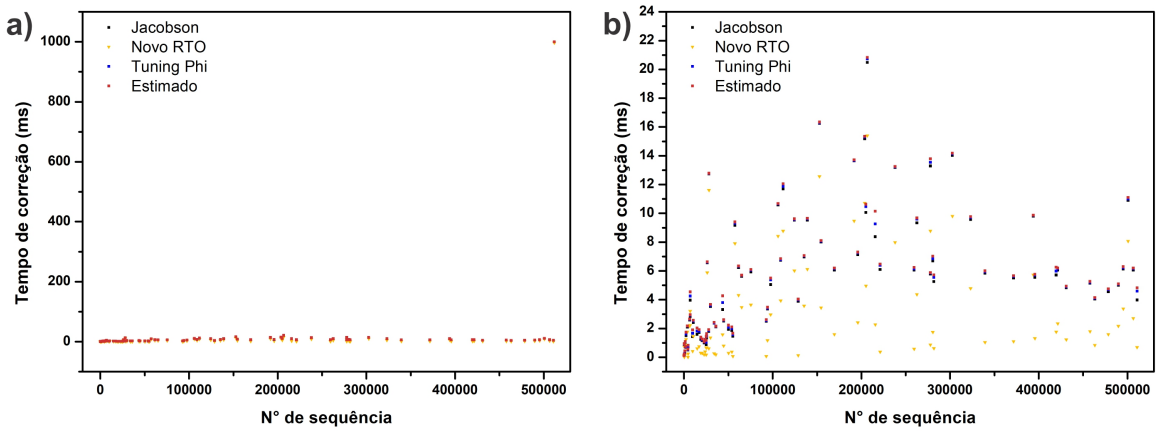


Figura 6.5: Tempo de correção (*Mistake Duration*) (ms) em função do número de sequência do pacote para *timeouts* prematuros comuns ao algoritmo Novo RTO e aos demais, obtidos do *trace* EUA → Tóquio. (a) Visão geral dos dados. (b) Ampliação de uma seção de (a).

No entanto, vale ressaltar a adaptabilidade do algoritmo frente às condições da rede. Ao observar os enlaces onde a rede é bem comportada, ou seja, apresenta baixa variância na latência (especificamente nos cenários *Reino Unido → EUA* e *EUA → Reino Unido*), a diferença de desempenho torna-se negligenciável a depender da aplicação. Nestes casos, a média do Novo RTO (102 ms) aproxima-se consideravelmente dos demais algoritmos (100 ms), mostrando que em redes estáveis, a penalidade é mínima.

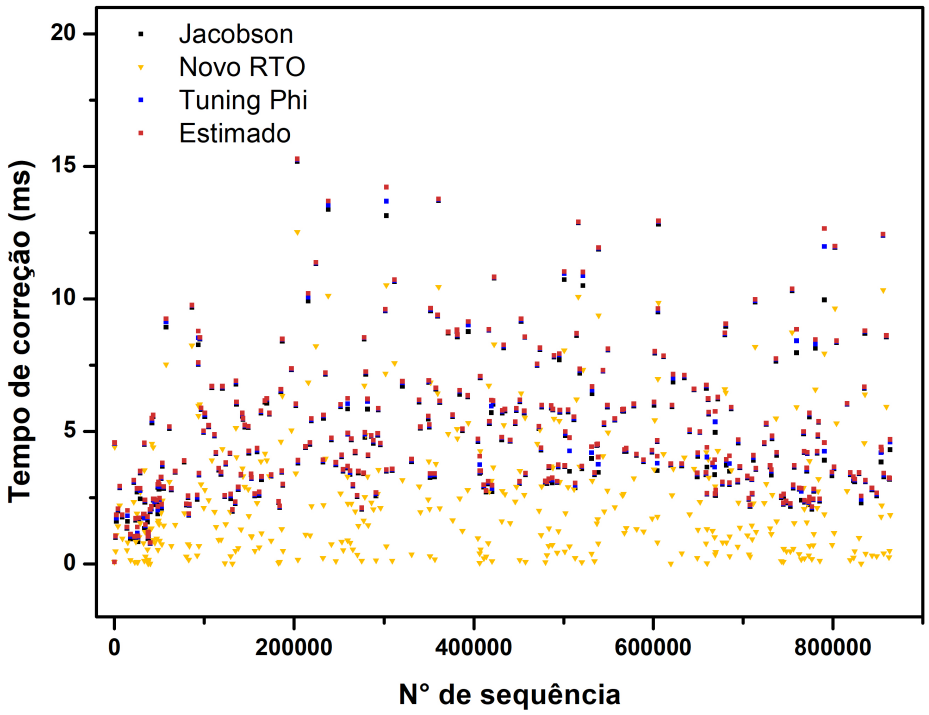


Figura 6.6: Tempo de correção (*Mistake Duration*) (ms) em função do número de sequência do pacote para *timeouts* prematuros comuns ao algoritmo Novo RTO e aos demais, obtidos do *trace* EUA → Reino Unido.

Isso demonstra que o Novo RTO não impõe uma penalidade fixa de atraso. Ele escala seu conservadorismo proporcionalmente à instabilidade da rede. Onde há estabilidade, o algoritmo entrega agilidade competitiva, ajustando suas margens de segurança quando a rede está volátil e propensa a atrasos que poderiam ser confundidos com falhas.

Tabela 6.5: Comparativo do desvio padrão do *Detection Time* (em ms).

Trace	**Jacobson**	**Novo RTO**	***TuningPhi***	**Estimado**
Tóquio → Reino Unido	1,688	47,278	0,817	0,002
Tóquio → EUA	0,0679	44,949	0,0348	0,002
Reino Unido → Tóquio	0,182	49,721	0,0913	0,002
Reino Unido → EUA	0,127	1,496	0,063	0,000
EUA → Tóquio	0,071	49,626	0,038	0,002
EUA → Reino Unido	0,038	0,716	0,022	0,000

A Tabela 6.5 reforça essa análise. O desvio padrão mais elevado nos enlaces que envolvem Tóquio reflete a capacidade dinâmica do algoritmo de ajustar seu limiar de correção de *timeouts* prematuros (*Mistake Duration*) para acomodar as grandes flutuações dessas rotas. O que, por sua vez, impacta negativamente no tempo de detecção (*Detection Time*) como foi discutido acima. Em contrapartida, nos enlaces estáveis (EUA/Reino Unido), o desvio padrão cai drasticamente (1,496 e 0,716), confirmando que o algoritmo converge para um comportamento estável quando o ambiente permite.

Em suma, o aumento no *Detection Time* não é um defeito, mas uma característica de segurança. O algoritmo sacrifica milissegundos na detecção de falhas para evitar os custos operacionais por falsas suspeitas frequentes. Essa troca é vantajosa em aplicações, onde a

estabilidade do sistema é preferível à detecção instantânea e excessivamente sensível de falhas legítimas.

As Figuras 6.7 a 6.12 apresentam o *Detection Time* em função do momento da falha *crash* do processo monitorado. Estes gráficos permitem observar a evolução do comportamento do algoritmo ao longo do tempo e como ele reage a eventos específicos de variação na rede durante a execução.

De modo geral, observa-se que os pontos do Novo RTO situam-se predominantemente acima dos demais, corroborando os dados das médias apresentados anteriormente. No entanto, uma análise minuciosa dos cenários de alta latência (Figuras 6.7, 6.8, 6.9 e 6.11) revela uma nuance crucial sobre o funcionamento do algoritmo que as médias sozinhas não conseguem capturar.

Nestes gráficos, nota-se que o Novo RTO inicia a execução com um *Detection Time* próximo aos demais algoritmos, indicando que, em condições normais, a penalidade de tempo é mínima. Contudo, observa-se um degrau acentuado em determinado momento da execução, onde o tempo de detecção sobe abruptamente, chegando a uma diferença de quase 100 ms em relação aos demais, e sustenta-se neste novo patamar até o fim do experimento.

Este fenômeno ocorre devido à chegada de um pacote com atraso excessivo (um pico de latência). O Novo RTO incorpora essa diferença ao seu cálculo de *timeout*. O algoritmo entende que a rede é capaz de produzir grandes atrasos e, preventivamente, eleva seu limiar de *timeout*. Portanto, ele sacrifica a velocidade de detecção para garantir que, caso aquele pico de latência se repita, não ocorra um *timeout* prematuro. Já nos cenários estáveis (Figuras 6.10 e 6.12), onde tais picos estão ausentes, o algoritmo mantém-se competitivo durante todo o experimento, apresentando apenas uma pequena diferença em relação aos demais.

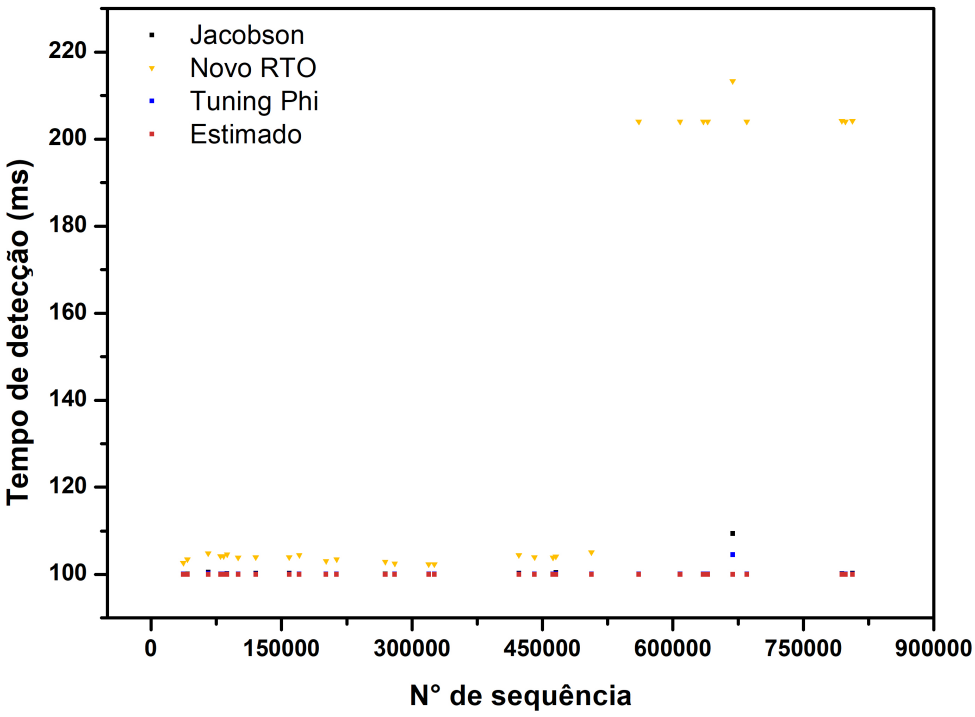


Figura 6.7: Tempo de detecção (*Detection Time*) (ms) em função do número de sequência do pacote definido para a falha *crash* do processo monitorado. Dados obtidos do *trace* Tóquio → Reino Unido.

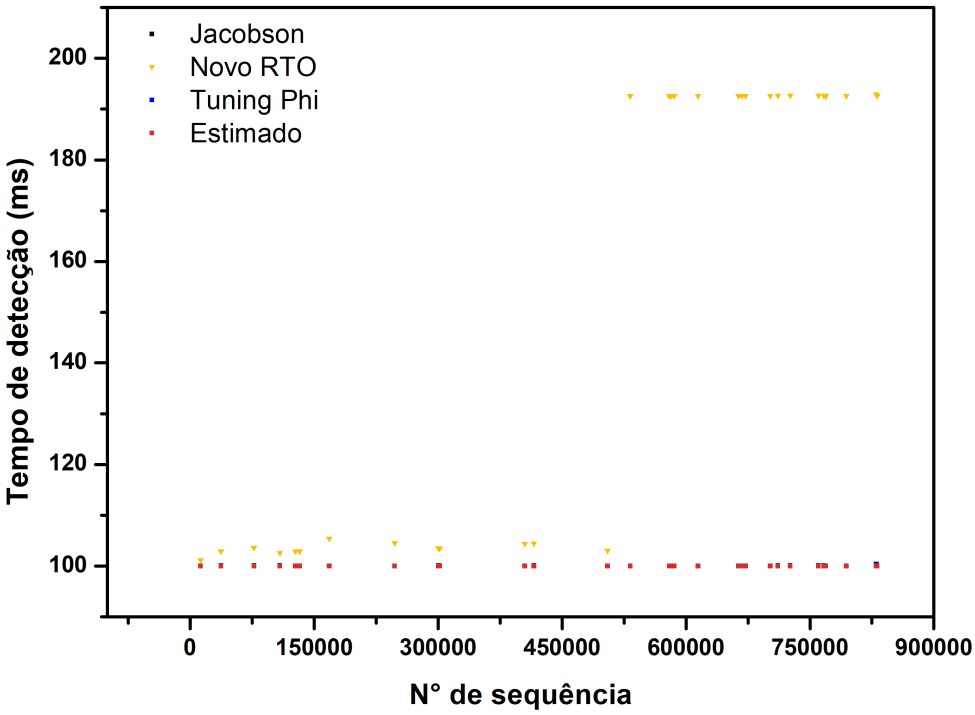


Figura 6.8: Tempo de detecção (*Detection Time*) (ms) em função do número de sequência do pacote definido para a falha *crash* do processo monitorado. Dados obtidos do *trace* Tóquio → EUA.

6.4 CONCLUSÃO

Os resultados experimentais apresentados e discutidos ao longo deste capítulo validam a proposta do algoritmo Novo RTO: ser flexível quanto às oscilações naturais da rede sem sacrificar as métricas de qualidade: *Mistake Duration* e *Detection Time*. A avaliação baseada em *traces* reais permitiu avaliar o comportamento do algoritmo em cenários de alta variabilidade e latência, comparando-o com outras estratégias. A análise conjunta das três métricas fundamentais (quantidade de *timeouts* prematuros, *Mistake Duration* e *Detection Time*), revela que a proposta cumpre com o objetivo.

No que tange à precisão, o Novo RTO demonstrou um desempenho excepcional, reduzindo a ocorrência de falsas suspeitas em até 99,53% em comparação ao algoritmo de Jacobson. Esta redução drástica não é apenas um ganho estatístico, mas traduz-se diretamente em economia de recursos computacionais e de rede, evitando que o sistema distribuído desperdice ciclos de processamento em ações de recuperação desnecessárias e, no contexto do TCP, impedindo a redução indevida da janela de congestionamento, o que penaliza a aplicação quanto a vazão de mensagens. Além disso, a análise aprofundada do *Mistake Duration* comprovou que, nos raros eventos em que o algoritmo falha, ele é capaz de corrigir sua suspeita e retomar a normalidade mais rapidamente que seus concorrentes.

A análise do *Detection Time* evidenciou o custo dessa robustez. Observou-se um aumento no tempo para detecção de falhas legítimas. Contudo, os dados revelaram que esse atraso não é estático, mas sim dinâmico e adaptativo: em redes estáveis, o algoritmo converge

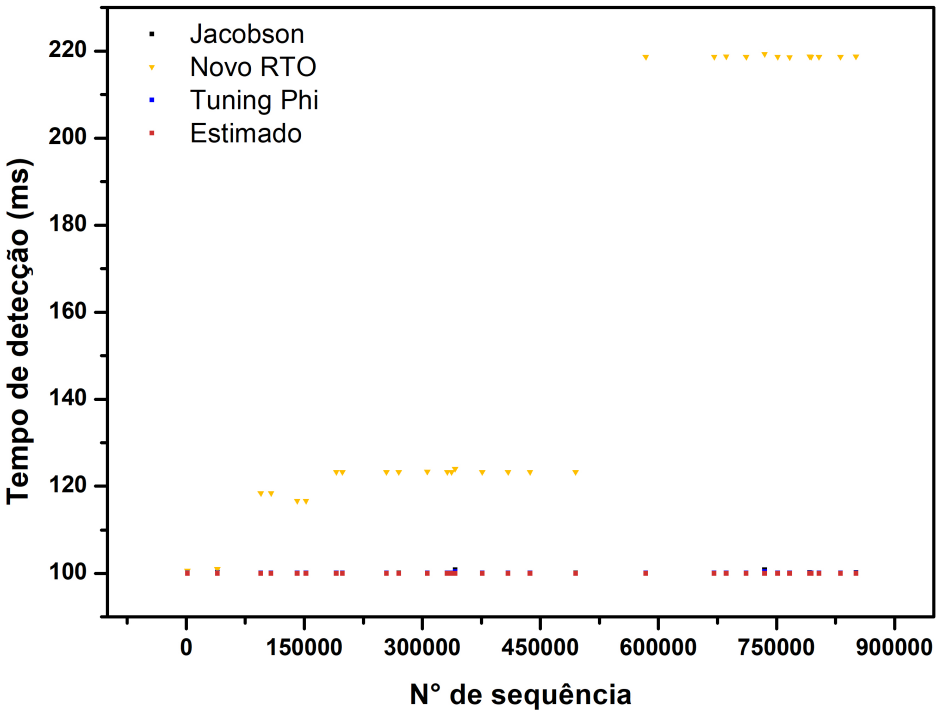


Figura 6.9: Tempo de detecção (*Detection Time*) (ms) em função do número de sequência do pacote definido para a falha *crash* do processo monitorado. Dados obtidos do *trace* Reino Unido → Tóquio.

para tempos de detecção competitivos, próximos aos demais; em redes instáveis, ele expande suas margens de segurança para evitar a recorrência de erros.

Portanto, o algoritmo Novo RTO provou ser a alternativa mais adequada para cenários em que se prioriza a redução de falsas suspeitas e a agilidade na correção de erros (*Mistake Duration*) quando ocorrem *timeouts* prematuros. Nos cenários, onde a detecção de falhas legítimas (*crashes*) seja um requisito essencial, o algoritmo proposto pode não ser a escolha ótima devido ao seu conservadorismo, porém ele demonstra-se plenamente competitivo em situações de estabilidade da rede. Assim, a proposta oferece um equilíbrio: sacrifica-se uma fração de tempo na detecção em troca da redução de *timeouts* prematuros e agilidade da correção de erros.

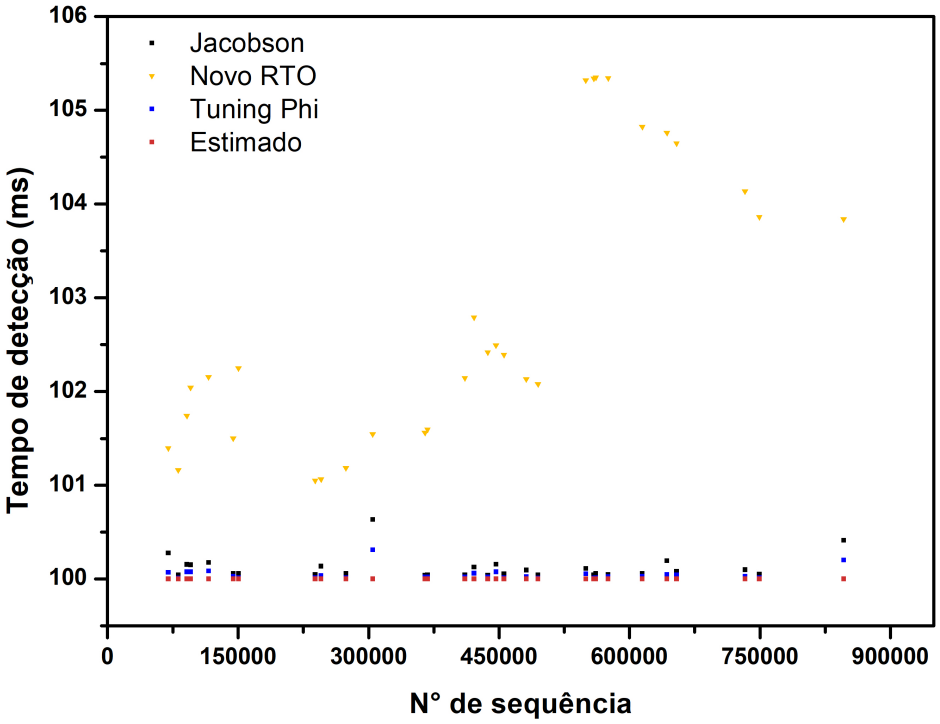


Figura 6.10: Tempo de detecção (*Detection Time*) (ms) em função do número de sequência do pacote definido para a falha *crash* do processo monitorado. Dados obtidos do *trace* Reino Unido → EUA.

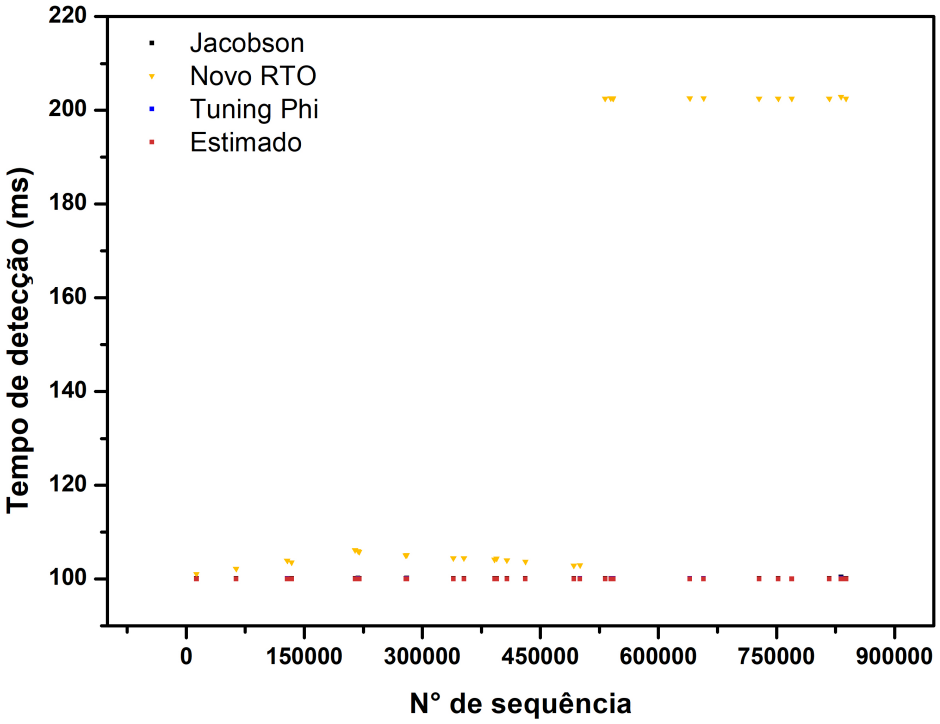


Figura 6.11: Tempo de detecção (*Detection Time*) (ms) em função do número de sequência do pacote definido para a falha *crash* do processo monitorado. Dados obtidos do *trace* EUA → Tóquio.

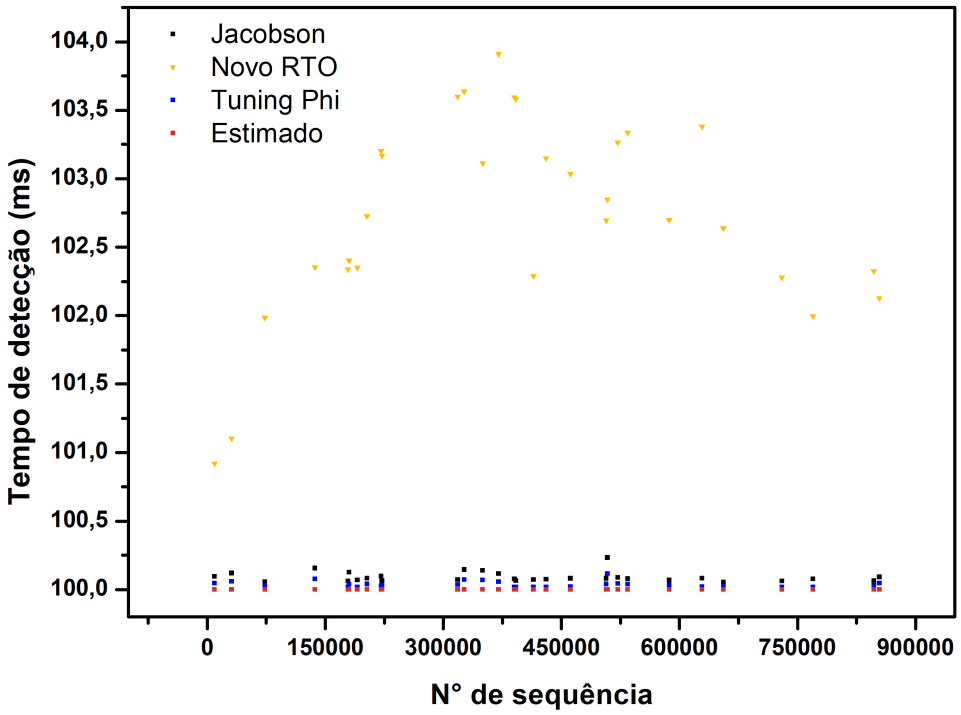


Figura 6.12: Tempo de detecção (*Detection Time*) (ms) em função do número de sequência do pacote definido para a falha *crash* do processo monitorado. Dados obtidos do *trace* EUA → Reino Unido.

7 CONCLUSÃO

Este trabalho abordou o desafio fundamental de garantir a precisão no monitoramento de processos em sistemas distribuídos assíncronos. A incerteza inerente a esse ambiente torna impossível a distinção entre processos falhos e processos lentos, resultando frequentemente em *timeouts* prematuros que degradam o desempenho do sistema. A proposta central deste trabalho foi o desenvolvimento e a avaliação do algoritmo Novo RTO. Diferente da abordagem clássica de Jacobson, que depende exclusivamente da média e da variação do tempo de chegada dos *heartbeats*, a nova estratégia incorpora a métrica de duração do erro (*Mistake Duration*) diretamente no cálculo do limite temporal. Ao criar uma margem de segurança dinâmica baseada no histórico de erros passados, o algoritmo busca filtrar as oscilações naturais da rede.

A avaliação experimental, realizada sob *traces* reais de monitoramento de processos na Internet, corroborou a eficácia da estratégia proposta. O resultado mais expressivo foi a redução drástica na quantidade de falsas suspeitas. Em cenários onde o algoritmo de Jacobson (do protocolo TCP) gerou mais de 20.000 *timeouts* prematuros, o Novo RTO registrou menos de 400 ocorrências, representando uma redução superior a 99,5% na taxa de alarmes falsos em relação ao Jacobson. Além disso, a análise do *Mistake Duration* revelou que, nas raras ocasiões em que o Novo RTO comete um erro, ele é capaz de corrigir sua estimativa e cessar as falsas suspeitas com maior agilidade que os demais algoritmos avaliados.

Esses ganhos em precisão, no entanto, apresentaram um *trade-off* em relação à velocidade de detecção. O Novo RTO apresentou um aumento no *Detection Time* em cenários de alta instabilidade, agindo de forma conservadora para evitar falsas suspeitas. Contudo, em enlaces estáveis, o algoritmo manteve-se competitivo, com uma diferença média muito reduzida em relação ao Jacobson, demonstrando que a penalidade de desempenho é adaptativa e proporcional à volatilidade do ambiente.

Como desdobramentos deste trabalho, sugerem-se os seguintes trabalhos futuros:

1. Aplicação no protocolo TCP: implementar a lógica do Novo RTO na camada de transporte, substituindo o algoritmo de Jacobson original no cálculo do RTO do TCP, a fim de avaliar seu impacto na redução de retransmissões espúrias e no controle de congestionamento;

2. Aprimorar o *Detection Time* da estratégia Novo RTO para redes instáveis: refinar o componente erro médio da expressão de cálculo de *timeout*, para tornar o *Detection Time* do Novo RTO mais competitivo em cenários de alta instabilidade;

3. Investigar o efeito do aumento do tempo de detecção na prática: após a detecção de uma falha real, há em geral uma ação de reconfiguração ou uma retransmissão. Por outro lado, após uma falsa suspeita, há redução da janela de controle de congestionamento e ações espúrias de reconfiguração, que não deveriam ter ocorrido. Acreditamos que o pequeno aumento do tempo de detecção tem efeito irrisório em sistemas reais e cargas reais, mas falta uma análise experimental e analítica para comprovar esta suspeita;

4. Aprimorar o algoritmo *Timeout* Estimado: explorar a incorporação de uma margem de segurança à estratégia. O objetivo é mitigar a agressividade excessiva da predição pura, adicionando um componente de tolerância para reduzir a taxa de *timeouts* prematuros, sem abandonar a ideia central baseada em séries temporais;

5. Investigar a remoção da limitação do ϕ na estratégia *Tuning Phi*: Existe a hipótese, ainda não comprovada experimentalmente, de que permitir que ϕ assuma valores livres (removendo a função $\max/\min$) possa tornar a estratégia mais competitiva.

Em suma, o Novo RTO consolidou-se como uma alternativa robusta para sistemas distribuídos que priorizam a estabilidade e a minimização de custos operacionais decorrentes de falsas suspeitas, oferecendo uma solução resiliente às incertezas da Internet.

REFERÊNCIAS

Bertier, M., Marin, O. e Sens, P. (2003). Performance analysis of a hierarchical failure detector. Em *Proceedings of the International Conference on Dependable Systems and Networks (DSN'2003)*, páginas 635–644.

Cachin, C., Guerraoui, R. e Rodrigues, L. (2011). *Introduction to reliable and secure distributed programming*. Springer Science & Business Media.

Chandra, T. D. e Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)*, 43(2):225–267.

Chen, W., Toueg, S. e Aguilera, M. K. (2002). On the quality of service of failure detectors. *IEEE Transactions on computers*, 51(5):561–580.

de Sá, A. S. e de Araújo Macêdo, R. J. (2010). Qos self-configuring failure detectors for distributed systems. Em *IFIP International Conference on Distributed Applications and Interoperable Systems*, páginas 126–140. Springer.

Dixit, M. e Casimiro, A. (2010). Adaptare-fd: A dependability-oriented adaptive failure detector. Em *2010 29th IEEE Symposium on Reliable Distributed Systems*, páginas 141–147. IEEE.

Duarte, E. P., Garrett, T., Bona, L. C., Carmo, R. e Züge, A. P. (2010). Finding stable cliques of planetlab nodes. Em *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, páginas 317–322. IEEE.

Duarte Jr, E. P., Nanya, T., Noguchi, S. e Mansfield, G. (1997). Non-broadcast network fault-monitoring based on system-level diagnosis. Em *International Symposium on Integrated Network Management*, páginas 597–609. Springer.

Duarte Jr, E. P., Rodrigues, L. A., Camargo, E. T. e Turchetti, R. C. (2023). The missing piece: a distributed system-level diagnosis model for the implementation of unreliable failure detectors. *Computing*, 105(12):2821–2845.

Fischer, M. J., Lynch, N. A. e Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382.

Huff, A., Hiltunen, M. e Duarte, E. P. (2021). Rft: Scalable and fault-tolerant microservices for the o-ran control plane. Em *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, páginas 402–409. IEEE.

Jacobson, V. (1988). Congestion avoidance and control. *ACM SIGCOMM computer communication review*, 18(4):314–329.

Kebarighotbi, A. e Cassandras, C. G. (2011). Timeout control in distributed systems using perturbation analysis. Em *2011 50th IEEE Conference on Decision and Control and European Control Conference*, páginas 5437–5442. IEEE.

Kesselman, A. e Mansour, Y. (2005). Optimizing tcp retransmission timeout. Em *International Conference on Networking*, páginas 133–140. Springer.

Lynch, N. A. (1996). *Distributed algorithms*. Elsevier.

Moraes, D. M. e Duarte Jr, E. P. (2011). A failure detection service for internet-based multi-as distributed systems. Em *2011 IEEE 17th International Conference on Parallel and Distributed Systems*, páginas 260–267. IEEE.

Nunes, R. C. e Jansch-Porto, I. (2004). Qos of timeout-based self-tuned failure detectors: the effects of the communication delay predictor and the safety margin. Em *International Conference on Dependable Systems and Networks, 2004*, páginas 753–761. IEEE.

Postel, J. (1980). User Datagram Protocol. RFC 768.

Postel, J. (1981). Transmission Control Protocol. RFC 793.

Stein, G., Rodrigues, L. A., Duarte Jr, E. P. e Arantes, L. (2023). Diamond-p-vcube: An eventually perfect hierarchical failure detector for asynchronous distributed systems. Em *Proceedings of the 12th Latin-American Symposium on Dependable and Secure Computing*, páginas 40–49.

Tomsic, A., Sens, P., Garcia, J., Arantes, L. e Sopena, J. (2015). 2w-fd: A failure detector algorithm with qos. Em *2015 IEEE International Parallel and Distributed Processing Symposium*, páginas 885–893. IEEE.

Turchetti, R. C. e Duarte, E. P. (2015). Implementation of failure detector based on network function virtualization. Em *2015 IEEE International Conference on Dependable Systems and Networks Workshops*, páginas 19–25. IEEE.

Turchetti, R. C. e Duarte Jr, E. P. (2018). Uma estratégia adaptativa para melhorar a precisao do timeout de detectores de falhas na internet. Em *Workshop de Testes e Tolerância a Falhas (WTF)*. SBC.

Turchetti, R. C., Duarte Jr, E. P., Arantes, L. e Sens, P. (2016). A qos-configurable failure detection service for internet applications. *Journal of Internet Services and Applications*, 7(1):9.