UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

Christiano Freire Barbosa

MÉTODO PARA MEDIÇÃO DE TEMPOS EM OPERAÇÕES OPC UA NO ACESSO A DADOS DE TEMPO REAL DE UM SISTEMA DE AUTOMAÇÃO

Florianópolis
2018

Christiano Freire Barbosa

MÉTODO PARA MEDIÇÃO DE TEMPOS EM OPERAÇÕES OPC UA NO ACESSO A DADOS DE TEMPO REAL DE UM SISTEMA DE AUTOMAÇÃO

Dissertação submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas para a obtenção do Grau de Mestre em Engenharia de Automação e Sistemas.
Orientador: Prof. Rômulo Silva de Oliveira, Dr. Eng.

Florianópolis
2018

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Barbosa, Christiano Freire
Método para medição de tempos em operações opc ua no acesso a dados de tempo real de um sistema de automação / Christiano Freire Barbosa ; orientador, Rômulo Silva de Oliveira, 2018.
92 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós Graduação em Engenharia de Automação e Sistemas, Florianópolis, 2018.

Inclui referências.

1. Engenharia de Automação e Sistemas. 2. OPC UA. 3. Sistemas de tempo real. 4. Medição de tempos. I. Oliveira, Rômulo Silva de. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. III. Título.

MÉTODO PARA MEDIÇÃO DE TEMPOS EM OPERAÇÕES OPC UA NO ACESSO A DADOS DE TEMPO REAL DE UM SISTEMA DE AUTOMAÇÃO

Christiano Freire Barbosa

Esta Dissertação foi julgada adequada para obtenção do Título de "Mestre em Engenharia de Automação e Sistemas" e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Florianópolis, 16 de maio de 2018.

_____

Prof. Rômulo Silva de Oliveira, Dr.
Orientador

_____

Prof. Daniel Ferreira Coutinho, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia de Automação e Sistemas

Banca Examinadora:

_____

Prof. Rômulo Silva de Oliveira, Dr.
Presidente

_____

Profa. Patrícia Della Méa Plentz, Dra.
INE/UFSC

_____

Prof. Cristian Koliver, Dr.
INE/UFSC

Prof. Marcelo Ricardo Stemmer, Dr.
DAS/UFSC

Este trabalho é dedicado à minha família.

AGRADECIMENTOS

Agradeço primeiramente a Deus, pela minha vida.

Agradeço à minha esposa Juliene e aos meus filhos, Diogo e Beatriz, por serem a fonte que me motiva a enfrentar os desafios que na vida se apresentam.

Agradeço aos meus pais, Joil e Rosemary, e a minha irmã, Marcela por, mesmo distantes, serem a base da minha essência que se reflete em todo trabalho que realizo.

Agradeço ao professor Rômulo Silva de Oliveira, que orientou a execução deste trabalho, pela sua colaboração, dedicação e por suas sugestões imensamente construtivas.

À Universidade Federal de Santa Catarina, principalmente ao Programa de Pós-Graduação de Engenharia de Automação e Sistemas, por oferecer um mestrado com docentes qualificados.

E a todos familiares e amigos que, diretamente e indiretamente, colaboraram para que fosse possível a realização deste trabalho.

RESUMO

A modernização de um sistema de automação industrial pela utilização do padrão OPC UA na comunicação entre os seus dispositivos requer atenção no que diz respeito aos impactos deste padrão no desempenho de operações de acesso a dados de tempo real desse sistema. A tecnologia OPC UA introduz uma infraestrutura de software complexa às aplicações industriais, que pode impactar no desempenho geral da comunicação entre seus dispositivos. Medir os tempos envolvidos nessas operações é o caminho para avaliar a aplicabilidade da tecnologia OPC UA em sistemas com restrições temporais flexíveis. O objetivo deste trabalho é desenvolver um método que permita a medição de tempos envolvidos nas operações OPC UA no acesso a dados de tempo de real de uma variável de processo em um sistema de automação industrial, e aplicar o método desenvolvido em uma configuração de teste.

Palavras-chave: OPC UA. Sistemas de tempo real. Medição de tempos.

ABSTRACT

The modernization of an industrial automation system by the use of the OPC UA standard in the communication between its devices requires attention with the impacts caused by this standard on the real time data access operations performances of this system. OPC UA technology introduces a complex software infrastructure between industrial applications, which may impact the overall communication performance between its devices. Measuring the time involved in these operations is the way to evaluate the applicability of OPC UA technology in systems with soft time constraints. The aim of this work is to develop a method to measures time involved in the OPC UA operations, in the access to real time data in an industrial automation system, and to apply the method developed in a test configuration.

Keywords: OPC UA. Real-time systems. Time measurement.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

A&E	*Alarms and Events* – Alarmes e Eventos
API	*Application Programming Interface* – Interface de Programação de Aplicações
CLP	Controlador Lógico Programável
COM	*Component Object Model*
COTS	*Commercial Off-The-Shef*
DA	*Data Access* – Acesso de Dados
DCOM	*Distributed Component Object Model*
GPS	*Global Positioning System*
HDA	*Historical Data Access* – Acesso de Dados Históricos
HMI	*Human Machine Interface* – Interface Homem Máquina
IPqM	Instituto de Pesquisas da Marinha
MES	*Manufacturing Execution System*
NTP	*Network Time Protocol*
OLE	*Object Linking and Embedding*
OPC	*Open Platform Communications*
PTP	*Precision Time Protocol*
RPC	*Remote Procedure Call*
SAW	*Surface Acoustic Wave* – Onda Acústica de Superfície
SCADA	*Supervisory Control and Data Acquisitionn*
SCM	Sistema de Controle e Monitoramento
TAI	Tempo Atômico Internacional
UA	*Unified Architecture*
UTC	*Universal Time Coordinated* – Tempo Universal Coordenado
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

No Instituto de Pesquisas da Marinha (IPqM), organização militar da Marinha do Brasil sediada na cidade do Rio de Janeiro, são realizadas atividades de pesquisa científica, desenvolvimento tecnológico e prestação de serviços tecnológicos, associados a sistemas, equipamentos, componentes, materiais e técnicas, nas áreas de sistemas de armas, sensores, guerra eletrônica, guerra acústica, sistemas digitais e tecnologia de materiais. O propósito é contribuir para a independência tecnológica do Brasil, o desenvolvimento da base industrial de defesa e o fortalecimento do Poder Naval (IPQM, 2018).

O IPqM é responsável pelo desenvolvimento do Sistema de Controle e Monitoramento (SCM) instalado em alguns navios da Marinha do Brasil, dentre eles Corvetas e Navios Patrulha. Sua função é auxiliar na segurança física do navio e no monitoramento e controle de sua propulsão, monitorando e atuando sobre dispositivos como bombas, válvulas, ventiladores, exaustores, “flaps”, ar condicionado, sensores de fumaça, nível e temperatura, motores e turbinas.

O SCM é constituído por hardware COTS (*Commercial Off-The-Shef*) e software desenvolvido pelo IPqM. A comunicação entre o SCM e os Controladores Lógicos Programáveis (CLP) do sistema de automação e controle dos navios é possível pela utilização de drives proprietários dos fabricantes dos CLP. Tal situação torna a intercambiabilidade desses dispositivos por dispositivos de outros fabricantes uma tarefa não trivial ou até mesmo inviável, criando assim uma dependência tecnológica com o produto, prejudicando a manutenção do sistema ao longo de seu ciclo de vida, bem como uma possível modernização do mesmo.

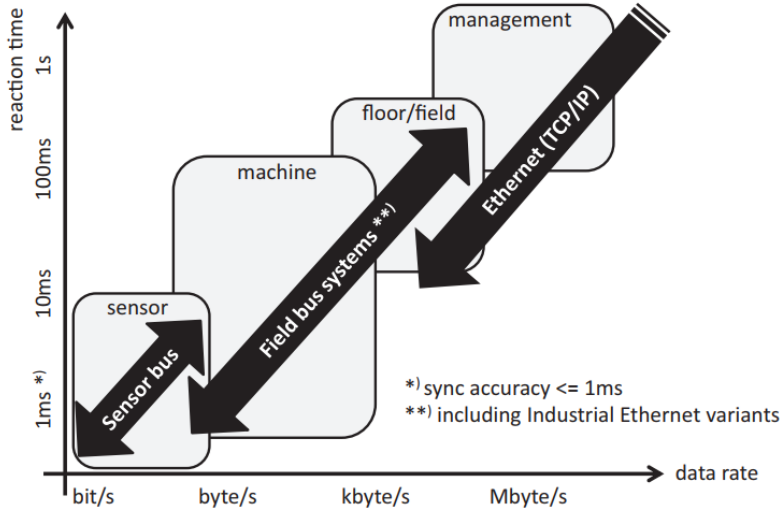
Como solução para o problema apresentado, já é de ampla utilização nas aplicações industriais a tecnologia OPC UA (*Open Platform Communications Unified Architecture*) que tem como proposta oferecer uma interface padrão de comunicação entre equipamentos de diferentes fabricantes, garantindo assim a interoperabilidade entre os mesmos.

Apesar dos benefícios que a utilização de uma especificação de comunicação padronizada agrega para um sistema de automação, a tecnologia OPC UA introduz uma infraestrutura de software complexa às aplicações industriais, que podem impactar no desempenho geral da comunicação entre os dispositivos do sistema (CAVALIERI, 2010 e 2013). No que diz respeito às aplicações de tempo real, ou seja, aplicações onde as tarefas devem ser realizadas dentro de prazos específicos, como ocorre nas aplicações de controle de processo, o impacto no desempenho

do sistema causado pela utilização da tecnologia OPC UA pode inviabilizar a sua aplicação.

O SCM, por exemplo, possui o modo de operação automático da propulsão, onde o próprio sistema realiza o controle da planta propulsora do navio seguindo leis de controle concebidas para minimizar o consumo de combustível do navio. Nesse modo de operação, um CLP, instalado no subsistema de controle e monitoramento da propulsão do SCM, é responsável pela lógica de controle, enquanto outro CLP, instalado próximo à máquina do navio, é responsável pela aquisição de dados e atuação sobre a planta. A troca de dados entre CLPs ocorre no nível de controle (ou de máquina) na hierarquia de comunicação de um sistema de automação industrial, apresentada na Figura 1. Nesse nível hierárquico o tempo de reação do sistema deve ser da ordem de 100 ms (MATHEUS e KÖNIGSEDER, 2017).

Figura 1 - Abordagem hierárquica para comunicação de fábrica



Fonte: Matheus e Königseder (2017)

O desempenho de uma comunicação OPC UA, neste caso, deve atender aos requisitos temporais do sistema. Nessa comunicação, o desempenho está relacionado ao atraso de *round-trip* de uma determinada operação. No caso de uma operação de leitura realizada por um cliente OPC UA, por exemplo, o atraso de *round-trip* pode ser definido como o tempo resposta total entre o instante em que uma requisição de leitura de uma ou mais variáveis é emitida pelo cliente, e o instante em que os

valores das leituras são entregues pelo servidor OPC UA. No contexto de tempo real, interessa medir o comportamento temporal dessas operações de forma que seja possível avaliar a utilização de servidores e clientes OPC UA em sistemas com restrições temporais.

Uma possível modernização do SCM, pela substituição do seu atual modelo de comunicação pela comunicação OPC UA, gera a necessidade de avaliar o desempenho dessa nova comunicação quanto ao cumprimento dos requisitos temporais do sistema. Essa necessidade é a motivação para o desenvolvimento de um método para medição de tempos em operações OPC UA.

1.1 OBJETIVO

O objetivo deste trabalho é desenvolver um método que permita a medição de tempos envolvidos nas operações OPC UA, no acesso a dados de tempo de real de uma variável de processo em um sistema de automação industrial. Essas medições devem permitir a análise do comportamento temporal da operação no contexto de requisitos temporais de aplicações de tempo real.

Para validar o método desenvolvido, ele será aplicado em uma comunicação OPC UA de teste, composto por um cliente e um servidor OPC UA interligados entre si por uma rede Ethernet. Este ambiente de teste reproduz, de forma simples e controlada, a classe de aplicações alvo deste trabalho.

1.2 ORGANIZAÇÃO

Este documento está organizado em seis capítulos, começando com o atual, o Capítulo 1, onde é apresentada uma visão geral do problema estudado, a motivação para o trabalho e os objetivos traçados.

No Capítulo 2 é apresentado o padrão OPC UA, abordando sua origem, aplicação e seus mecanismos para troca de informações entre cliente e servidor. As operações disponíveis na especificação da arquitetura para acesso a dados de tempo real de um processo de automação industrial, são apresentadas com foco nos parâmetros das mensagens trocadas entre os dispositivos, úteis para o desenvolvimento do método proposto e sua aplicação.

No Capítulo 3 são apresentados os conceitos de tempo real e a importância da sincronização de relógio nos sistemas de tempo real distribuídos e na aplicação do método desenvolvido.

No Capítulo 4 é apresentado o método para a medição de tempos relevantes na execução de operações OPC UA no acesso a dados de tempo real de uma planta de automação industrial.

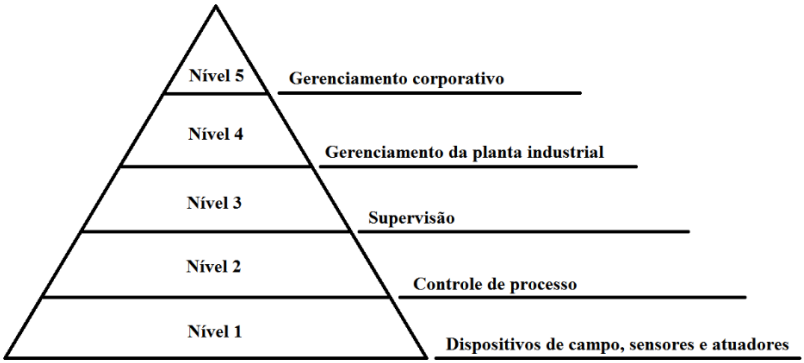
No Capítulo 5 são apresentados os resultados da aplicação do método desenvolvido sobre um ambiente de teste experimental, com a utilização de servidor OPC UA simulado, com variáveis de processo simuladas.

Finalmente, no Capítulo 6 é apresentada a conclusão.

2 A TECNOLOGIA OPC

Os dispositivos e equipamentos de um sistema de automação industrial e sua estrutura de comunicação, podem ser organizados dentro de cinco níveis interdependentes, de acordo com a sua função dentro do sistema (HOLLENDER, 2010), conforme ilustra a pirâmide da automação da Figura 2.

Figura 2 - Os 5 níveis da automação industrial – pirâmide da automação



Fonte: Hollender (2010)

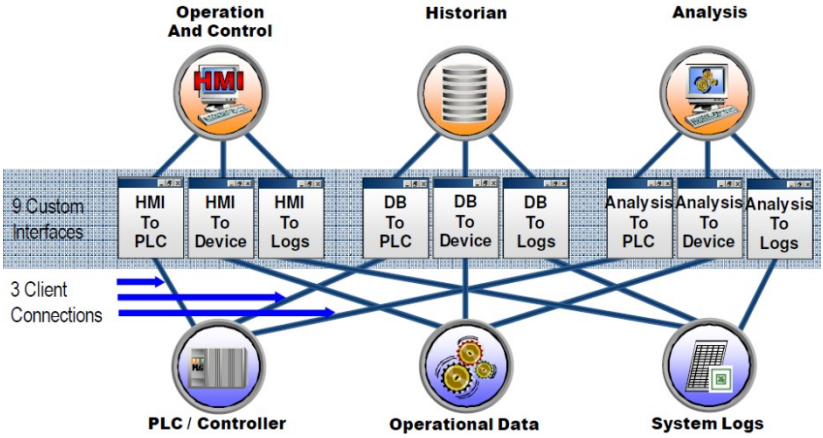
Em um cenário anterior ao surgimento da tecnologia OPC (*Open Platform Communications*), os fabricantes dos dispositivos do nível 3 (supervisão) precisavam implementar em suas aplicações os *drivers* necessários para a sua comunicação com os dispositivos do nível 2 (controle de processo) com quem se conectavam, como pode ser observado na Figura 3. Nessa situação, os custos de desenvolvimento e manutenção de aplicações SCADA (*Supervisory Control and Data Acquisition*), por exemplo, são maiores, e sua configuração mais trabalhosa (SHIMANUKI, 1999). Além disso, o uso de *drivers* proprietários torna o usuário dependente do mesmo fabricante.

O desenvolvimento e a evolução dos mecanismos para comunicação entre processos no sistema operacional Windows, da Microsoft, tornou possível o surgimento de um padrão de interface para a comunicação dos dispositivos de supervisão com os dispositivos de controle, reduzindo assim os esforços dos fabricantes com o desenvolvimento de *drivers* (MAHNKE, LEITNER e DAMM, 2009).

A primeira versão desse padrão de interface, atualmente denominado OPC Clássico, foi lançada em 1996 como resultado dos trabalhos exercidos pela força tarefa criada na época para esse propósito,

composta por fabricantes da indústria de automação. Anos mais tarde, a força tarefa passou a ser denominada OPC Foundation.

Figura 3 - Interligação de dispositivos de diferentes camadas da estrutura de automação industrial



Fonte: Material de apresentação da empresa MatrikonOPC (2017)

A versão clássica do OPC utiliza como base o padrão de interface COM (*Component Object Model*), desenvolvido pela Microsoft, e sua versão distribuída, DCOM (*Distributed Component Object Model*). Tais interfaces são a base do OLE (*Object Linking and Embedding*), que deram origem ao acrônimo OPC (*OLE for Process Control)*. Atualmente, o acrônimo OPC significa *Open Platform Communications* (OPC Foundation, 2018c).

O OPC clássico se divide em três especificações principais: OPC DA (*Data Access*), OPC HDA (*Historical Data Access*) e OPC A&E (*Alarms and Events*).

A interface OPC DA permite a leitura, a escrita e o monitoramento de variáveis contendo dados atuais do processo. É a interface OPC mais utilizada, sendo responsável por disponibilizar dados em tempo real provenientes dos dispositivos de controle aos dispositivos de supervisão.

Já interface OPC HDA provê o acesso a dados já armazenados do processo, enquanto a interface OPC A&E permite a recepção de notificações de eventos e notificações de alarmes, cuja função é informar ao cliente sobre a alteração de uma condição específica no processo.

Por se basear na tecnologia COM e DCOM, o OPC clássico é dependente de plataformas com o sistema operacional Windows. Além disso, a configuração de servidor OPC clássico para a comunicação com

um cliente remoto é trabalhosa, dado a complexidade de configuração da interface DCOM.

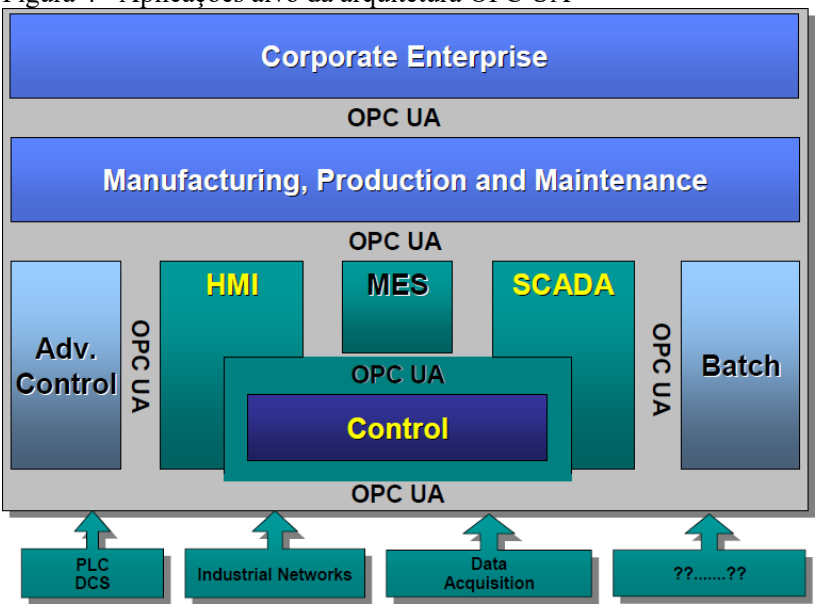
Em 2008, a OPC Foundation lançou o OPC UA (*OPC Unified Architecture*), uma arquitetura independente de plataforma, orientada a serviço, que integra todas as funcionalidades do OPC clássico (OPC Foundation, 2018b).

2.1 O PADRÃO DE COMUNICAÇÃO OPC UA

Baseado na linguagem XML (*eXtensible Markup Language*) e na tecnologia *Web Service*, o padrão OPC UA permite o transporte seguro e confiável de dados brutos e de informações pré-processadas através de uma plataforma interoperável, unificando ainda as funcionalidades de servidores e clientes OPC DA, OPC A&E e OPC HDA, para a troca horizontal e vertical de dados ao longo dos níveis da automação industrial (CAVALIERI e CUTULI, 2010).

Sua característica interoperável permite a sua aplicação em uma ampla variedade de plataformas, desde pequenos sistemas embarcados em dispositivos de campo, até aplicações de gerenciamento corporativo, que interagem entre si, conforme ilustra a Figura 4.

Figura 4 - Aplicações alvo da arquitetura OPC UA



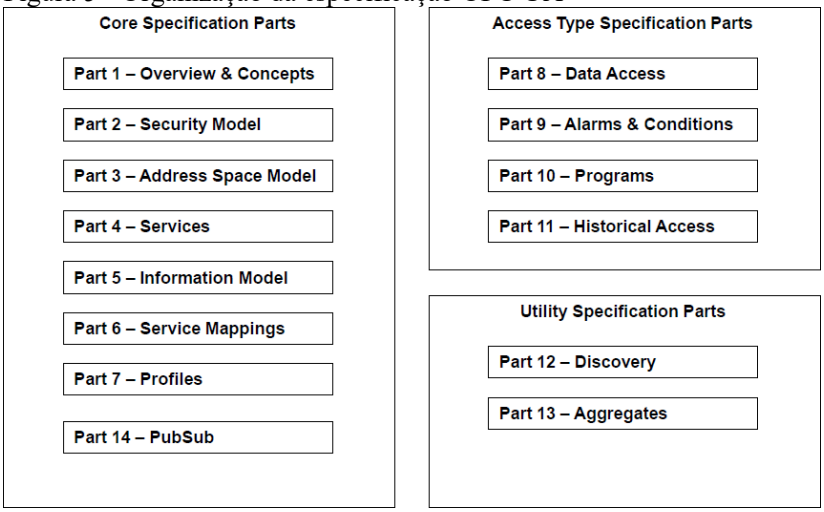
Fonte: OPC Foundation (2017a)

Os dados produzidos pelos diversos dispositivos em um sistema de automação industrial são traduzidos em um modelo de informação OPC UA, cujo acesso é possível por meio de serviços definidos na especificação da arquitetura.

A especificação da arquitetura é atualmente dividida em 14 partes, o que corresponde a 14 documentos que, juntos, definem as capacidades núcleo do OPC UA, os tipos de acesso, os mecanismos de descoberta e de agregação de dados do padrão (OPC Foundation, 2017a).

A Figura 5 apresenta a organização da especificação OPC UA.

Figura 5 - Organização da especificação OPC UA



Fonte: OPC Foundation (2017a)

Apesar de possuir uma especificação extensa, cujo conhecimento de seu todo é necessário para desenvolvedores de aplicações OPC UA, para o trabalho desenvolvido nessa dissertação de mestrado interessam apenas o conhecimento de alguns conceitos apresentados nas partes 1, 3 e 4 da especificação.

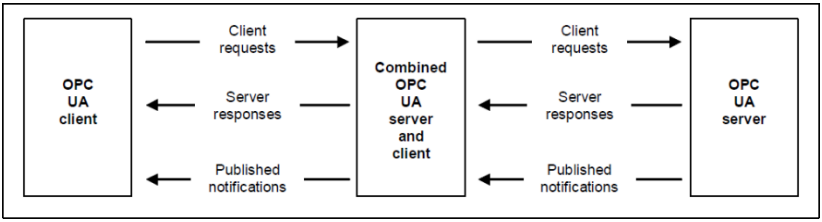
2.1.1 CLIENTE OPC UA

Na comunicação entre dispositivos que utilizam o padrão OPC UA, a interação entre os mesmos ocorre por meio de um elemento servidor, responsável por prover informações do processo, e de um elemento cliente, que consome essas informações. Cada cliente pode

interagir concorrentemente com um ou mais servidores, e cada servidor pode interagir concorrentemente com um ou mais clientes (OPC Foundation, 2017a).

A interação entre servidores e clientes na arquitetura do padrão OPC UA está ilustrada na Figura 6.

Figura 6 - Interação entre servidores e clientes na arquitetura do padrão OPC UA

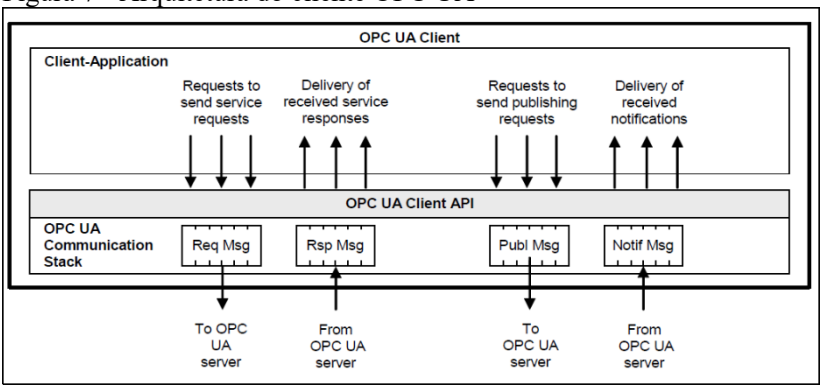


Fonte: OPC Foundation (2017a)

O cliente OPC UA possui uma configuração modular, compreendido por uma aplicação do cliente, onde são implementadas suas funções; pela API (*Application Programming Interface*) do cliente, que é a interface que isola o código da aplicação do cliente, da pilha de comunicação OPC UA; e pela pilha de comunicação OPC UA, responsável pela conversão das chamadas da API do cliente em mensagens e seu envio através de entidades de comunicação de camadas inferiores.

A Figura 7 apresenta a arquitetura do cliente OPC e a relação entre suas camadas.

Figura 7 - Arquitetura do cliente OPC UA



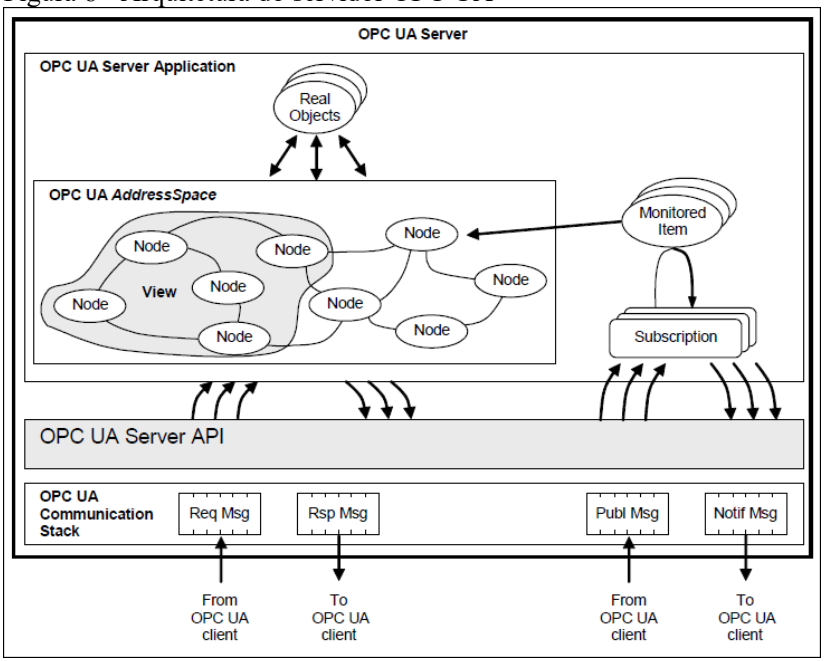
Fonte: OPC Foundation (2017a)

Como pode ser observado, o cliente obtém informações do servidor enviando requisições de serviço ou requisições de publicação. A dinâmica de funcionamento da aplicação OPC UA e os detalhes das informações trocadas são descritos na Parte 4 da especificação OPC UA (OPC Foundation, 2017c).

2.1.2 SERVIDOR OPC UA

Assim como ocorre com o cliente OPC UA, a arquitetura do servidor OPC UA é compreendida pela aplicação do servidor OPC UA, API do servidor e pilha de comunicação OPC UA, conforme mostra a Figura 8.

Figura 8 - Arquitetura do servidor OPC UA



Fonte: OPC Foundation (2017a)

A aplicação do servidor corresponde ao código que implementa as funções do servidor OPC UA, utilizando a API do servidor para enviar e receber mensagens dos clientes.

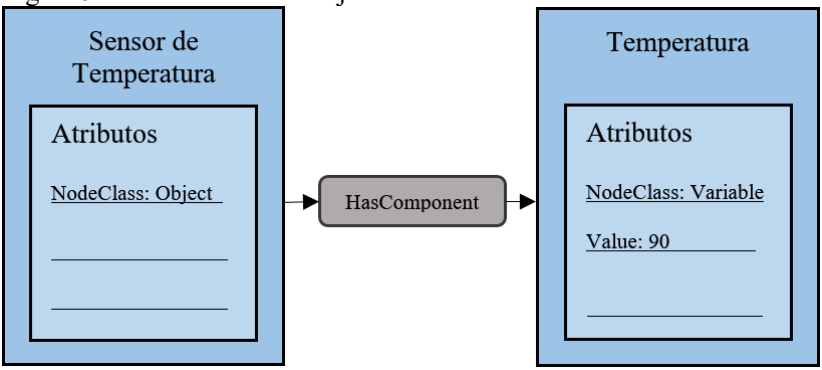
Os dispositivos físicos, apresentados na Figura 8 como *Real Objects*, são representados por conjuntos de nós (*nodes*) no espaço de endereçamento OPC UA (*AddressSpace OPC UA*) do servidor, e seus dados e informações estão contidos nos atributos (*Attributes*) desses nós.

O espaço de endereçamento é definido na Parte 1 da especificação da arquitetura OPC UA (OPC Foundation, 2017a) como uma coleção de informações que um servidor faz visível para seus clientes. Já um atributo define as características primitivas de um nó.

Pela utilização do padrão OPC UA, um sensor de temperatura pode ser representado no espaço de endereçamento do servidor por um nó da classe objeto, que tem como componente um nó da classe variável, onde a grandeza física é representada no atributo valor (*Value Attribute*).

A Figura 9 apresenta a arquitetura de um objeto OPC UA na forma como o mesmo é estruturado no espaço de endereçamento de um servidor OPC UA, tomando como exemplo um sensor de temperatura. O nó "Temperatura" é componente do nó "Sensor de Temperatura". Apenas nós da classe "variável" possuem o atributo valor que, para o exemplo do sensor de temperatura, esse valor é 90, representando a temperatura de 90ºC lida pelo sensor.

Figura 9 – Estrutura de um objeto OPC UA



Fonte: Autor (2018)

O acesso das informações disponíveis nos atributos dos nós é possível por meio de abstrações especificadas no padrão OPC UA, denominadas serviço.

2.1.3 SERVIÇOS

Na arquitetura do padrão OPC UA, os serviços são abstrações de Chamadas de Procedimento Remoto (*RPC – Remote Procedure Call*) implementadas nos servidores OPC UA, na pilha de comunicação, que podem ser chamados pelos clientes OPC UA. Toda comunicação entre aplicações OPC UA é baseada na troca de mensagens contendo parâmetros definidos na Parte 4 da especificação OPC UA (OPC Foundation, 2017c) para os serviços de cada operação.

Os serviços disponíveis no servidor OPC UA realizam funções que possibilitam ao cliente: encontrar o servidor na rede e então descobrir como se conectar ao mesmo; autenticar usuários e gerenciar sessões; adicionar, modificar e excluir nós no espaço de endereçamento; navegar no espaço de endereçamento; escrever e ler atributos dos nós, incluindo seus valores históricos; chamar métodos; e criar, modificar e excluir itens monitorados (*MonitoredItems*) usados para monitorar variáveis, para alteração de valores, e objetos, para identificação eventos; criar, modificar e excluir subscrições (*Subscriptions*), que enviam as notificações geradas pelos itens monitorados.

Os serviços de interesse desta dissertação são os serviços responsáveis pelas operações de leitura, escrita e monitoramento das variáveis de processo de um sistema de automação industrial. São os serviços responsáveis pela troca de dados e informações entre os dispositivos.

Para a leitura de dados em dispositivos remotos, o cliente OPC UA acessa o servidor por meio do serviço *Read*. Para a operação de escrita existe o serviço *Write* e para o monitoramento de variáveis é necessária a utilização dos serviços *CreateSubscription* e *CreateMonitoredItem*.

2.1.3.1 OPERAÇÃO DE LEITURA

O serviço *Read* tem a função de permitir que o cliente leia um ou mais atributos de um ou mais nós no espaço de endereçamento do servidor. No contexto desta dissertação, interessam os nós do espaço de endereçamento que representam as variáveis de processo de um sistema de automação.

Na execução de um serviço, o cliente envia para o servidor uma mensagem de requisição contendo os parâmetros de requisição para aquele serviço, definidos na Parte 4 da especificação da arquitetura OPC UA (OPC Foundation, 2017b). Para o serviço *Read*, esses parâmetros são os descritos no Quadro 1 exatamente como aparecem na norma. Alguns

parâmetros representam dados do tipo primitivo, tal como *Boolean* ou *String* (OPC Foundation, 2017b). Outros, são parâmetros compostos por dois ou mais parâmetros, que podem ser do tipo primitivo ou composto. A estrutura dos parâmetros compostos está apresentada na coluna "Estrutura do Parâmetro".

Quadro 1 - Parâmetros de requisição do Serviço *Read*

Serviço *Read* – Requisição	
Parâmetro	**Estrutura do Parâmetro**
requestHeader	authenticationToken
	timestamp
	requestHandle
	returnDiagnostics
	auditEntryID
	timeoutHint
	additionalHeader
maxAge	
timestampsToReturn	
nodesToRead	nodeId
	attributeId
	indexRange
	dataEncoding

Fonte: OPC Foundation (2017c)

Os parâmetros de requisição contêm as informações necessárias para que o servidor identifique a operação, realize a leitura do dado no dispositivo fonte e retorne uma mensagem de resposta ao cliente, contendo o valor lido na variável. O **requestHeader** é um parâmetro comum a todos os serviços disponíveis na arquitetura. Ele é responsável por transmitir informações úteis para a identificação da operação, registrar o instante em que a requisição é enviada e para transmitir informações de diagnóstico no nível da sessão da conexão OPC UA (OPC Foundation, 2017c).

Durante a operação de leitura, o servidor pode conter armazenado em memória mais do que um único valor para uma variável de processo a ser lida, com idades diferentes entre si. O valor a ser entregue ao cliente depende do valor atribuído ao parâmetro **maxAge** na mensagem de requisição do cliente. O valor de **maxAge** informa para o servidor a idade máxima, em milissegundos, que o valor lido deve ter. Esse parâmetro é do tipo *Duration*, definido pela especificação da arquitetura OPC UA

como um dado do tipo *Double*, que define um intervalo de tempo em milissegundos.

Para que o valor lido seja o valor mais recente possível, o parâmetro **maxAge** deve ser definido como 0. Nessa configuração, o servidor OPC UA deve tentar ler um novo valor da fonte de dados a cada operação de leitura.

O resultado da leitura é transmitido para o cliente no parâmetro **results** da mensagem de resposta do servidor. Esse parâmetro é definido na arquitetura OPC UA como sendo um parâmetro do tipo *DataValue*, cuja estrutura está apresentada no Quadro 2.

Quadro 2 - Parâmetros de resposta do serviço *Read*

Serviço *Read* - Resposta	
Parâmetro	**Estrutura do Parâmetro**
responseHeader	timestamp
	requestHandle
	serviceResult
	serviceDiagnostics
	stringTable
	additionalHeader
results	value
	statusCode
	sourceTimestamp
	sourcePicoSeconds
	serverTimestamp
	serverPicoSeconds
diagnosticInfos	namespaceUri
	symbolicId
	locale
	localizedText
	additionalInfo
	innerStatusCode
	innerDiagnosticInfo

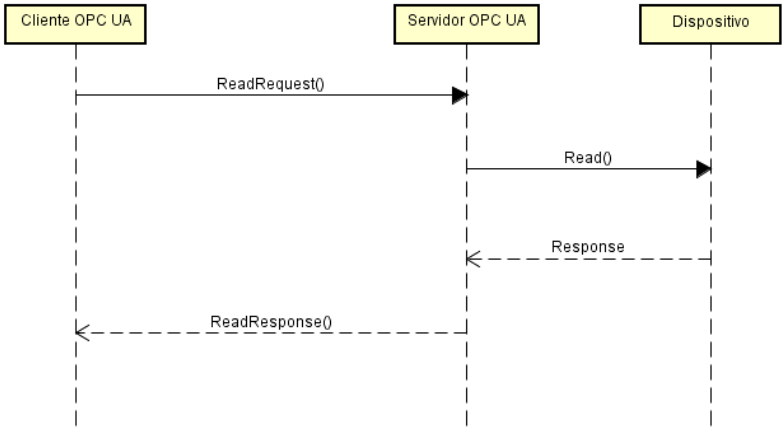
Fonte: OPC Foundation (2017c)

Se o serviço *Read* é requisitado pelo cliente para leitura do valor de um dispositivo de campo, por exemplo, o valor da leitura desse dispositivo será transmitido no parâmetro **value** da estrutura do parâmetro **results**, que pode assumir qualquer tipo de dado especificado na

arquitetura OPC UA, tais como *Byte*, *Double*, *Boolean*, *String*, *UtcTime*, dentro outros.

A dinâmica das mensagens trocadas entre o cliente e o servidor OPC UA na operação de leitura está representada na Figura 10.

Figura 10 - Dinâmica das mensagens trocadas na operação de leitura



Fonte: Autor (2018)

Durante a execução do serviço *Read*, alguns carimbos de tempo são registrados nos parâmetros das mensagens transmitidas entre cliente e servidor. Esses são os parâmetros de interesse no desenvolvimento e aplicação de um método que tem como propósito medir tempos.

O parâmetro **timestamp**, encontrado no cabeçalho da requisição (**requestHeader**), contém o data-hora do instante em que o cliente envia a requisição de leitura para o servidor. Esse parâmetro é utilizado apenas com o propósito de diagnóstico e *logging* na arquitetura (OPC Foundation, 2017c).

O parâmetro **timestamp** também está presente na mensagem de resposta do servidor. Nesse caso, o parâmetro registra o instante em que o valor lido é transmitido para o cliente.

O resultado da operação de leitura contido no parâmetro **results** transmite, além do valor lido (**value**), outros dois carimbos de tempo importantes para a aplicação do método: o **sourceTimestamp** e o **serverTimestamp** (Quadro 2). O parâmetro **sourceTimestamp** possui o data-hora do instante em que a variável tem o seu valor atribuído pela fonte do dado. O data-hora associado ao valor da variável não se altera, mesmo em aplicações onde há mais de um servidor no caminho entre a fonte do dado e o cliente.

O parâmetro **serverTimestamp** reflete o instante em que o servidor recebe o valor lido na fonte de dados. Na ocasião em que o valor do **maxAge** é 0, o **serverTimestamp** é atualizado em cada operação de leitura, mesmo quando o valor da variável não se altera.

Os parâmetros **sourcePicoSeconds** e **serverPicoSeconds** complementam os parâmetros **sourceTimestamp** e **serverTimestamp**, respectivamente. Esses parâmetros permitem que aplicações especifiquem carimbos de tempo com 10 picosegundos de resolução.

O **timestamp**, **sourceTimestamp**, e **serverTimestamp** são parâmetros especificados na arquitetura OPC UA como sendo do tipo *UtcTime*, usado para definir valores UTC (*Universal Time Coordinated*). Já os parâmetros **sourcePicoSeconds** e **serverPicoSeconds**, são do tipo *UInteger*, que define um inteiro sem sinal (OPC Foundation, 2017b).

2.1.3.2 OPERAÇÃO DE MONITORAMENTO

O monitoramento de uma variável de processo de um sistema de automação consiste em detectar as alterações do valor da variável monitorada no instante em que elas ocorrem. Na arquitetura OPC UA, essa tarefa é realizada por entidades criadas no servidor a partir do cliente, denominadas itens monitorados (*MonitoredItems*). Cada item monitorado está associado a um item a ser monitorado e a uma subscrição a ser utilizada para o envio de notificações para o cliente. O item monitorado pode ser qualquer atributo de um nó no espaço de endereçamento do servidor. No entanto, para a maioria das aplicações, esse atributo é o atributo valor de uma variável.

Quando um item monitorado detecta uma mudança no valor da variável, ou uma ocorrência de evento ou alarme, ele gera uma notificação que é enviada ao cliente por meio de uma subscrição (*Subscription*). Uma subscrição é uma extremidade de comunicação (*endpoint*) no servidor, que publica notificações para os clientes contendo informações sobre o dado alterado, evento ou alarme.

O serviço responsável por criar um *MonitoredItem* é o serviço *CreateMonitoredItem*. Já a criação de uma subscrição é realizada pelo serviço *CreateSubscription* (OPC Foundation, 2017c).

Na execução do serviço *CreateSubscription*, o cliente envia para o servidor os parâmetros descritos no Quadro 3.

Quadro 3 - Parâmetros de requisição do serviço *CreateSubscription*

Serviço CreateSubscription - Requisição	
Parâmetro	**Estrutura do Parâmetro**
requestHeader	Idem ao Quadro 1
requestedPublishingInterval	
requestedLifetimeCount	
requestedMaxKeepAliveCount	
maxNotificationPerPublish	
publishingEnable	
priority	

Fonte: OPC Foundation (2017c)

Como resposta à requisição do cliente, o servidor transmite uma mensagem de resposta contendo os parâmetros do Quadro 4.

Quadro 4 - Parâmetros de resposta do serviço *CreateSubscription*

Serviço CreateSubscription - Requisição	
Parâmetro	**Estrutura do Parâmetro**
responseHeader	Idem ao Quadro 2
subscriptionId	
revisedPublishingInterval	
revisedLifetimeCount	
revisedMaxKeepAliveCount	

Fonte: OPC Foundation (2017c)

A característica de uma subscrição que mais influencia no tempo decorrido entre o instante em que uma variável tem o seu valor alterado e o instante em que esta alteração é percebida no cliente é o intervalo de publicação (*publishing interval*). Esse parâmetro define a taxa cíclica na qual uma subscrição é executada, ou seja, a taxa máxima na qual as notificações são transmitidas para o cliente.

Uma notificação é uma estrutura de dados que descreve a ocorrência de alterações de uma variável ou de eventos, transferida para o cliente por meio de uma mensagem de notificação (*NotificationMessage*).

O intervalo de publicação de uma subscrição é definido na execução do serviço *CreateSubscription*. O cliente envia para o servidor o valor do intervalo de publicação desejado em milissegundos, por meio do parâmetro **requestedPublishingInterval** (Quadro 3), definido na especificação da arquitetura como sendo do tipo *Duration*. O servidor,

por sua vez, retorna o valor do intervalo de publicação que será aplicado na subscrição, por meio do parâmetro *revisedPublishingInterval* (Quadro 4), também do tipo *Duration*, em sua mensagem de resposta. Esse valor retornado poderá ser o mesmo informado pelo cliente, ou um valor revisado, visando atender as restrições do servidor.

Na execução do serviço *CreateMonitoredItem*, com uma subscrição já existente no servidor, o cliente envia uma mensagem de requisição contendo os parâmetros do Quadro 5.

Quadro 5 - Parâmetros de requisição do serviço *CreateMonitoredItem*

Serviço CreateMonitoredItem - Requisição	
Parâmetro	**Estrutura do Parâmetro**
requestHeader	Idem ao Quadro 1
subscriptionId	
timestampsToReturn	
itemsToCreate	itemToMonitor
	monitoringMode
	requestedParameters

Fonte: OPC Foundation (2017c)

O parâmetro **requestedParameters**, contido dentro da estrutura do parâmetro **itemsToCreate**, transmite os parâmetros com as informações que definem as características do *MonitoredItem*. Esses parâmetros estão apresentados no Quadro 6.

Quadro 6 - Detalhamento do parâmetro **requestedParameters**

requestedParameters	
Parâmetros	**Descrição**
clientHandle	Identificação do MonitoredItem
samplingInterval	Valor em milissegundos do intervalo de amostragem desejado
filter	Estrutura de dados contendo as condições em que o servidor deve gerar as notificações para o cliente
queueSize	Valor que indica o tamanho desejado para a fila de notificações
discardOldest	Valor booleano que especifica a política de descarte de notificações

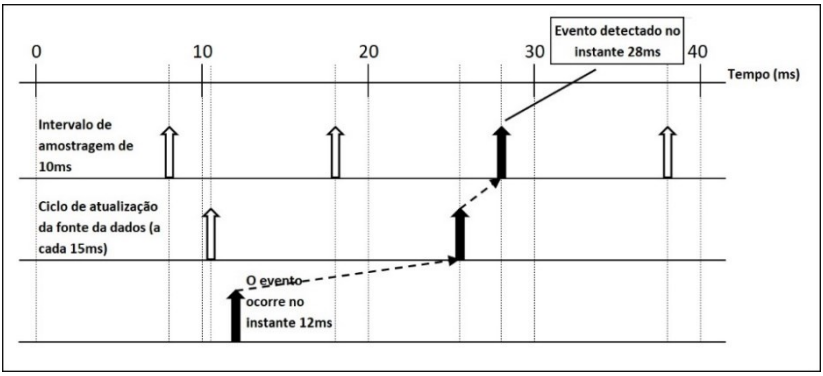
Fonte: OPC Foundation (2017c)

A velocidade com que servidor detecta uma alteração no valor da variável monitorada é influenciada pelo valor atribuído ao parâmetro

samplingInterval (intervalo de amostragem). Esse parâmetro define a taxa cíclica na qual o servidor faz o "melhor esforço" para amostrar o item em sua fonte de dados.

Um exemplo de como o intervalo de amostragem de um item monitorado pode influenciar o atraso na detecção de um evento pelo servidor, está apresentado na Figura 11.

Figura 11 - Atraso típico na detecção de um evento ou alteração de valor de uma variável



Fonte: OPC Foundation (2017c)

Nesse exemplo, um evento que ocorre no instante 12 ms só é detectado pelo servidor no instante 28 ms.

Para a definição do intervalo de amostragem de um *MonitoredItem*, o cliente informa para o servidor o valor desejado pelo parâmetro **samplingInterval** (Quadro 6). Após o processamento da mensagem do cliente, o servidor retorna os parâmetros contidos no Quadro 7.

Quadro 7 - Parâmetros de resposta do serviço CreateMonitoredItem

Serviço CreateMonitoredItem - Resposta	
Parâmetro	**Estrutura do Parâmetro**
responseHeader	Idem ao Quadro 2
results	statusCode
	monitoredItemId
	revisedSamplingInterval
	revisedQueueSize
	filterResult
diagnosticInfos	Idem ao Quadro 2

Fonte: OPC Foundation (2017c)

O intervalo de amostragem solicitado pelo cliente é confirmado ou ratificado pelo parâmetro **revisedSamplingInterval**. Este valor passa a valer como o intervalo de amostragem para o item monitorado criado.

Enquanto o **publishInterval** tem influência direta sobre o atraso de detecção no cliente, o **samplingInterval** tem influência direta sobre o atraso no servidor, que por sua vez influencia o atraso no sistema como um todo.

Outro elemento relevante em um item monitorado é a sua fila. Sua função é enfileirar as notificações geradas, para o seu posterior envio pela subscrição, a cada intervalo de publicação. O cliente informa o tamanho desejado da fila pelo parâmetro **queueSize** da mensagem de requisição do serviço *CreateMonitoredItem* (Quadro 6). Pelo parâmetro *revisedQueueSize* do Quadro 7, o servidor confirma ou ratifica o tamanho da fila.

Quando a fila do *MonitoredItem* está cheia, uma notificação é descartada conforme a política de descarte configurada no servidor, de acordo com o valor booleano atribuído ao parâmetro **discardOldest** (Quadro 6). Quando **discardOldest** é verdadeiro, a notificação mais antiga na fila é excluída para a inclusão de uma nova notificação. Quando **discardOldest** é falso, a notificação do fim da fila, consequentemente a mais recente, é excluída.

Se o interesse da aplicação é receber sempre o valor mais atualizado, então a fila deve ser configurada com o tamanho 1. Nesta situação, a política de descarte é ignorada. No entanto, se o interesse está em receber uma sequência contínua de notificações, sem perdas, o item monitorado deve ser criado com o tamanho de fila grande o suficiente para armazenar todas as notificações geradas entre dois ciclos de publicação consecutivos.

A cada ciclo de publicação, a subscrição envia todas as notificações enfileiradas para o cliente. Neste caso, as alterações nos valores da variável monitorada ocorridas em instantes distintos serão detectados no mesmo instante pelo cliente.

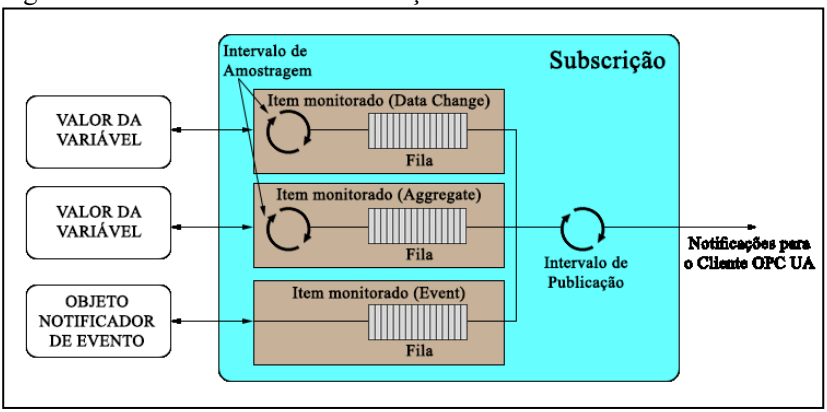
2.1.3.3 DINÂMICA DO MONITORAMENTO

Conforme a especificação da arquitetura OPC UA (OPC Foundation, 2017c), um item monitorado pode ser criado para três funções distintas: monitoramento do valor de uma variável no instante em que a alteração de seu valor ocorre (*data change*); monitoramento de um conjunto de valores de um intervalo de tempo específico, ou de um

cálculo específico sobre esse conjunto de valores (*aggregate*); e monitoramento de alarmes e eventos (*event*).

A Figura 12 apresenta a estrutura de cada um desses itens monitorados, bem como os elementos definidos na execução dos serviços *CreateSubscription* e *CreateMonitoredItem*.

Figura 12 - Estrutura de uma subscrição e seus itens monitorados

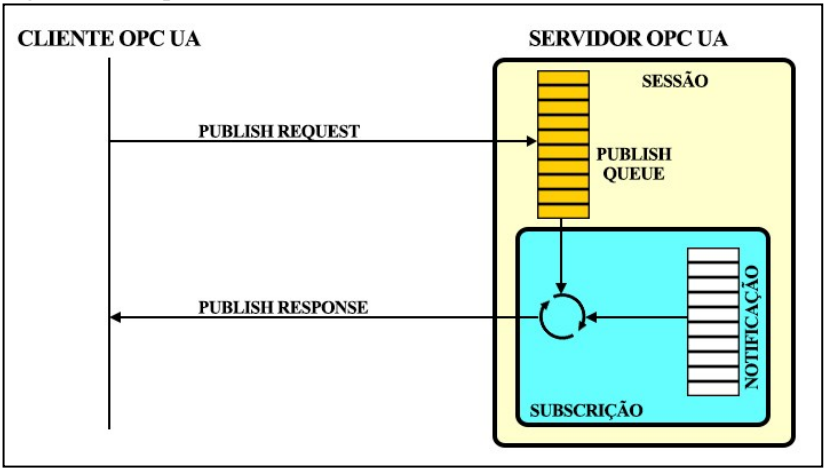


Fonte: Cavalieri (2013)

Cada item monitorado gera notificações que são armazenadas em sua fila até que a subscrição seja executada. A subscrição é executada a cada intervalo de publicação, quando tenta enviar as notificações enfileiradas nos itens monitorados. No entanto, essas notificações só são transmitidas na presença de uma requisição de publicação (*Publish request*) na fila da sessão da conexão. Quando não há requisições enfileiradas, o servidor entra no estado de espera de uma requisição. Assim que uma *Publish request* é recebida no servidor, este processa a requisição imediatamente, sem esperar pelo próximo ciclo de publicação da subscrição, enviando para o servidor uma resposta da publicação (*Publish response*) contendo o parâmetro mensagem de notificação (**notificationMessage**) com as notificações dos itens monitorados.

Uma *Puslish request* e uma *Publish response* compõem o serviço *Publish*, que tem a função de solicitar que o servidor retorne mensagens de notificação e de informar o reconhecimento das mensagens de notificações recebidas. A Figura 13 exemplifica a execução desse serviço.

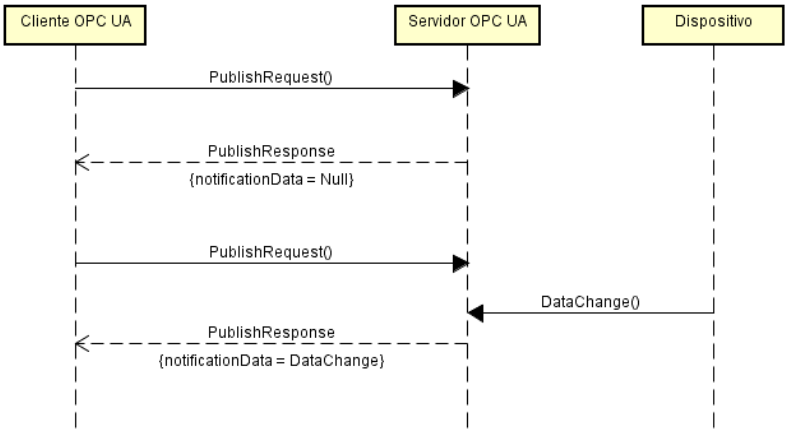
Figura 13 - Representação da execução do serviço *Publish*



Fonte: Cavalieri (2013)

A Figura 14 apresenta a dinâmica das mensagens trocadas entre um cliente e um servidor OPC UA durante a operação de monitoramento.

Figura 14 - Dinâmica das mensagens trocadas durante a operação de monitoramento



Fonte: Autor (2018)

2.1.3.4 CARIMBOS DE TEMPO

Medir o tempo de resposta da operação de monitoramento é relevante apenas para o monitoramento de dados de tempo real. Dentre as funções dos itens de monitoramento apresentados na Seção 2.1.3.3, o do tipo *data change* é o responsável por monitorar dados de tempo real.

As notificações geradas nos itens monitorados são transmitidas para o cliente por meio de uma *publish response* contendo uma **NotificationMessage**, cuja estrutura está representada no Quadro 8.

Quadro 8 - Estrutura de uma mensagem de notificação

NotificationMessage	
Parâmetros	**Descrição**
sequenceNumber	Número de sequência da mensagem de notificação
publishTime	Data-hora do instante em que a mensagem é transmitida para o cliente
notificationData	Estrutura de dados contendo uma ou mais notificações

Fonte: OPC Foundation (2017c)

Para um item monitorado do tipo *data change*, o parâmetro **notificationData** possui a estrutura apresentada no Quadro 9.

Quadro 9 - Estrutura do parâmetro **notificationData** para um item monitorado do tipo *data change*

notificationData – Data Change	
Parâmetro	**Estrutura do Parâmetro**
monitoredItems	clientHandle
	Value
diagnosticInfos	Idem ao Quadro 2

Fonte: OPC Foundation (2017c)

A noção do instante em que as alterações no valor da variável monitorada ocorrem é transmitida pelos parâmetros **sourceTimestamp**, **sourcePicoSeconds**, **serverTimestamp** e **serverPicoSeconds** contidos na estrutura do parâmetro **Value**. Esse parâmetro é do tipo *DataValue* e contém a mesma estrutura do parâmetro **results** da operação de leitura, no Quadro 2.

2.1.3.5 OPERAÇÃO DE ESCRITA

Na arquitetura OPC UA, a operação de escrita é realizada pelo serviço *Write*. Na execução desse serviço, o cliente envia para o servidor uma mensagem de requisição contendo os parâmetros descritos no Quadro 10.

Quadro 10 - Parâmetros de requisição do serviço *Write*

Serviço Write – Requisição	
Parâmetro	**Estrutura do Parâmetro**
requestHeader	Idem ao quadro 1
nodesToWrite	nodeId
	attributedId
	indexRange
	value

Fonte: OPC Foundation (2017c)

É possível realizar a escrita em vários nós no espaço de endereçamento do servidor com uma única solicitação do cliente por meio do parâmetro **nodesToWrite**.

O valor a ser escrito na variável da fonte de dados é transmitido no parâmetro **value**. O servidor não retorna uma mensagem de resposta até que o valor seja escrito ou que seja determinado que a escrita não é possível.

Após processar a mensagem de requisição e de escrever o valor na variável, o servidor retorna para o cliente uma mensagem de resposta contendo os parâmetros do Quadro 11.

Quadro 11 - Parâmetros de resposta do serviço *Write*

Serviço Write - Resposta	
Parâmetro	**Estrutura do Parâmetro**
responseHeader	Idem ao Quadro 2
results	
diagnosticInfos	

Fonte: OPC Foundation (2017c)

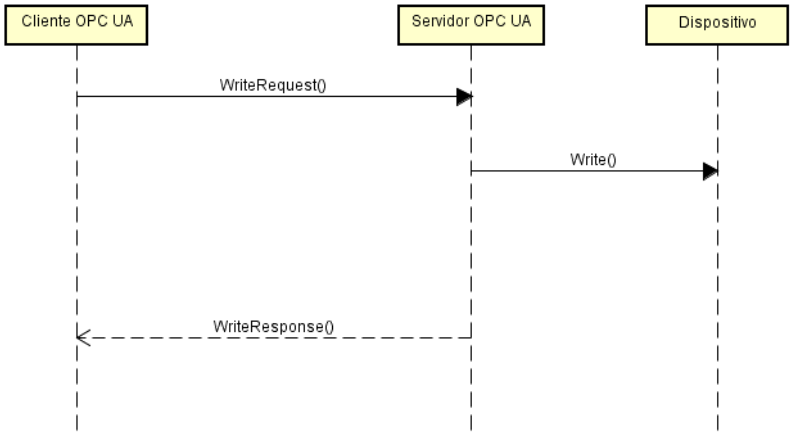
O parâmetro **results** transmite para o cliente o resultado da escrita para cada um dos nós solicitados, informando se a escrita foi bem-sucedida ou não.

Em sistemas cuja configuração contém um dispositivo intermediário entre o servidor e a fonte de dados, o parâmetro **results**

retorna um código de sucesso que indica que a escrita não foi verificada, uma vez que só é garantida a entrega da mensagem para o dispositivo intermediário. Nas ocasiões onde a escrita pode ser verificada, o servidor retorna uma indicação de sucesso incondicional (OPC Foundation, 2017c).

A Figura 15 apresenta a dinâmica das mensagens trocadas entre servidor e cliente durante a operação de escrita.

Figura 15 - Dinâmica das mensagens trocadas durante a operação de escrita



Fonte: Autor (2018)

O serviço *Write* faz o registro do instante em que a escrita tem início no cliente no parâmetro **timestamp** do cabeçalho da mensagem requisição.

Na mensagem de resposta do servidor, o parâmetro **timestamp** contida no cabeçalho indica o instante em que o servidor realiza o comando de escrita no nível inferior do sistema, não representando necessariamente o instante em que a escrita foi realizada na fonte de dados.

2.2 CONSIDERAÇÕES FINAIS

O método desenvolvido e apresentado no Capítulo 4 utiliza os parâmetros compartilhados entre clientes e servidores OPC UA, que carregam consigo informações de carimbos de tempo, úteis para medir o tempo de resposta de cada uma das operações de leitura, escrita e

monitoramento. O tempo de resposta, neste caso, é equivalente ao tempo decorrido entre o evento início da operação e o seu evento fim.

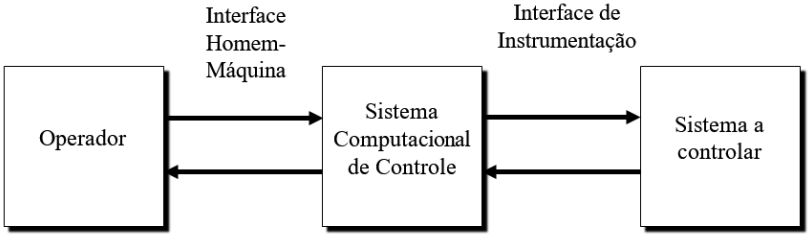
3 SISTEMAS DE TEMPO REAL

3.1 CONSIDERAÇÕES INICIAIS

Um sistema computacional de tempo real (*real-time computer system*) é um sistema computacional no qual o comportamento correto não depende apenas dos resultados lógicos de computação, mas também do instante físico no qual esses resultados são obtidos (KOPETZ, 2011). Um sistema maior, que engloba um sistema computacional de tempo real, é chamado de sistema de tempo real (*real-time system*).

Os requisitos temporais de um sistema de tempo real estão diretamente relacionados ao ambiente no qual o mesmo está inserido e com o qual interage. Para um sistema de controle, onde existe um sistema a controlar e um operador interagindo com um sistema computacional (Figura 16), as restrições temporais podem ser impostas tanto pelo comportamento dinâmico do sistema a controlar quanto pelas exigências dos serviços a serem oferecidos a um usuário humano ou computacional. Desta forma, o sistema computacional deve reagir aos estímulos do ambiente com o qual se relaciona, dentro de prazos específicos (*deadlines*) (FARINES, FRAGA e OLIVEIRA, 2000).

Figura 16 - Sistema de Tempo Real



Fonte: Farines, Fraga e Oliveira (2000)

Um sistema de tempo real pode, então, ser classificado em função das consequências das perdas desses prazos. Quando um sistema computacional oferece uma resposta após o prazo definido pelos requisitos temporais do ambiente e as consequências disto são desastrosas, esses prazos são definidos como rígidos (*hard deadline*) e o sistema é classificado como um sistema de tempo real rígido (*hard real-time system*). Se as consequências das perdas de prazos não são catastróficas, mas causam degradação no desempenho do sistema como um todo, esses prazos são definidos como flexíveis (*soft deadline*) e o

sistema é classificado como sistema de tempo real flexível (*soft real-time system*) (LIU, 2000; KOETZ, 2011).

É possível então observar que o problema de tempo real envolve determinar se um sistema é capaz ou não de realizar a sua função respeitando os limites temporais exigidos pela aplicação. Neste caso, busca-se a previsibilidade do sistema no domínio temporal, podendo ser obtida por meio de uma antecipação determinista ou probabilística do comportamento temporal do sistema (FARINES, FRAGA e OLIVEIRA, 2000).

No desenvolvimento de sistemas de tempo real, modelos abstratos são utilizados para validar propriedades não funcionais, tal como o comportamento temporal em análises de escalonabilidade (STIGGE et al., 2011). É muito comum a utilização de um modelo de tarefas, onde cada tarefa representa unidades de trabalho que concorrem por um ou mais recursos de um sistema (LIU, 2000). Um processador, uma rede de comunicação e um dispositivo de armazenamento são exemplos de recursos com os quais as tarefas concorrem pela utilização.

Para que uma determinada tarefa cumpra os requisitos de tempo real do sistema, é necessário que o tempo decorrido entre a sua chegada na fila de utilização do recurso e a sua completa execução seja menor ou igual ao prazo limite (*deadline*) daquela tarefa, estabelecido pelos requisitos temporais do sistema. O intervalo entre o instante de chegada da tarefa e o instante da sua conclusão é denominado tempo de resposta da tarefa.

Para aplicações com requisitos temporais rígidos (*hard real-time*), o sistema deve garantir que o tempo de resposta das tarefas nunca excederá os requisitos temporais ao longo de sua operação. Essa garantia deve ser validada por meio de um procedimento eficiente e correto, ou por meio de testes e simulações exaustivas, onde é possível determinar o tempo de reposta no pior caso das tarefas e verificar se os limites temporais serão sempre atendidos. Esta situação é diferente do que acontece com aplicações onde os requisitos temporais são flexíveis. Neste caso, os limites temporais são muitas vezes especificados em termos probabilísticos (LIU, 2000).

O desenvolvimento de um método de medição de tempos nas operações entre cliente e servidor OPC UA, proposto neste trabalho, funciona como ferramenta para a obtenção dos limites temporais probabilísticos do tempo de resposta dessas operações.

3.2 MEDIÇÃO DE TEMPO E SINCRONIZAÇÃO DE RELÓGIOS

A tarefa de medir e analisar dados temporais pode ser problemática em sistemas onde os dispositivos estão distribuídos ao longo de uma rede de comunicação. Um sistema distribuído tem como característica a falta de um relógio global (COULOURIS et al., 2013). No caso de uma rede industrial, cada dispositivo, tais como sensores, atuadores, controladores lógicos programáveis, computadores, workstations e HMI (*Human Machine Interface*), possuem seus próprios relógios físicos, que precisam ter suas referências de tempo sincronizadas para permitir a anotação correta das ocorrências de dados e eventos (*data logging*) nas tarefas de monitoramento e diagnóstico. No caso de aplicações OPC UA, o sincronismo deve garantir a anotação correta de *timestamps* anotados por clientes e servidores OPC UA.

A sincronização é fundamental também nas aplicações de tempo-real, onde diversos nós distribuídos na rede, tais como controladores, sensores e atuadores, precisam amostrar e controlar o sistema de uma maneira coordenada no tempo. A sincronização do tempo de referência através da rede é essencial no intuito de prover (FLAMMINI e FERRARI, 2010): a fusão dos dados de medidas obtidos de um sistema de aquisição de dados distribuído e de seus sensores; o processamento distribuído dos sinais que levam em consideração o tempo; e a coordenação de ações através de um conjunto distribuído de atuadores quando uma ação de controle deve ser tomada.

3.2.1 NOÇÃO DE TEMPO EM UM DISPOSITIVO ISOLADO

Os dispositivos conectados a uma rede de automação industrial possuem recursos de hardware e software que permitem que os mesmos estabeleçam uma referência de tempo para si. Muitos desses dispositivos são equipados com um relógio físico que, de uma maneira geral, é composto por um oscilador estabilizado de quartzo ou de onda acústica de superfície (SAW – *Surface Acoustic Wave*) e um contador que gera uma interrupção para o processador em intervalos de alguns milissegundos (*tick*) (MILLS, 2016).

A frequência do oscilador desses relógios pode variar de forma imprevisível, em função de diversos fatores físicos, tais como, variação na temperatura, envelhecimento do cristal oscilador e variação na tensão de alimentação desses relógios (ZHOU et al., 2008).

Isso significa que os relógios de computador se desviam de uma base de tempo e, mais importante, suas taxas de desvio diferem entre si.

A taxa de desvio do relógio (*drift*) se refere à quantidade relativa pela qual um relógio de computador difere de um relógio de referência perfeito (COULOURIS et al., 2013).

A taxa de desvio dos relógios não acarreta em grandes problemas para um dispositivo isolado, uma vez que para esse só importa a sua própria referência de tempo. No entanto, para dispositivos que se comunicam dentro de uma rede, a diferença de frequência entre os cristais faz com que cada relógio conte o tempo com diferentes velocidades.

Em relógios baseados em cristal de quartzo comum, a taxa de deriva (*drift rate*) é da ordem de 10^{-6} segundos a cada segundo, ou seja, aproximadamente 1 segundo a cada 11 dias e meio (COULOURIS et al., 2013). Essa diferença, que se acumula ao longo do tempo, pode ser inaceitável em muitas aplicações distribuídas.

3.2.2 SINCRONIZAÇÃO DE RELÓGIOS

Para que os relógios de dispositivos distintos de uma rede se mantenham sincronizados se faz necessário o ajuste constante destes em relação a uma referência comum, com frequência suficientemente necessária para evitar um desvio de tempo entre os relógios que seja superior ao desvio máximo permitido pela aplicação.

No que se propõe este trabalho, uma taxa de deriva de 10^{-6}, por exemplo, que equivale a 1 milissegundo a cada 1 minuto e meio, pode ser inaceitável se o tempo de resposta das operações que estamos medindo for da ordem de 1 milissegundo. Esse problema pode ser contornado ajustando os relógios do cliente e do servidor por uma mesma referência de tempo, periodicamente.

Diversos métodos foram desenvolvidos com a finalidade de possibilitar a sincronização de relógios em um sistema distribuído. Para a sincronização de um sistema em nível mundial se faz necessário uma referência de tempo que seja comum para todo o mundo. Essa referência é conhecida como Tempo Universal Coordenado (*Universal Time Coordinated - UTC*). O UTC é um padrão internacional para contagem de tempo, baseado no Tempo Atômico Internacional (TAI), e seus sinais são sincronizados e transmitidos regularmente de estações de rádio terrestre e satélites cobrindo muitas partes do planeta (COULOURIS et al., 2013).

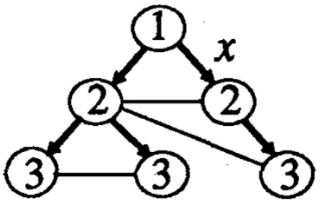
COULOURIS (2013) define sincronização externa como a sincronização dos relógios dos processos de um sistema distribuído com uma fonte externa de referência. Se os relógios são sincronizados com um grau de precisão conhecido em relação a outro relógio dentro do sistema,

mesmos que eles não estejam necessariamente sincronizados com uma fonte de tempo externa, então essa sincronização é interna. Neste trabalho, o interesse maior é a precisão da sincronização interna dos dispositivos do ambiente onde serão realizadas as medições temporais das operações OPC UA.

Para a sincronização desses dispositivos é necessário a utilização de um protocolo de sincronização de rede. O protocolo mais difundido atualmente é o NTP (*Network Time Protocol*), desenvolvido para distribuir informações de tempo na internet. No NTP, um ou mais servidores primários são sincronizados diretamente com uma fonte de tempo externa, como um GPS (*Global Positioning System*). Servidores de tempo secundários são sincronizados com os servidores primários e outros servidores na sub-rede de sincronização (MILLS, 1991), conforme hierarquia apresentada na Figura 17.

A sincronização utilizando o NTP apresenta precisão na ordem de dezenas de milissegundos na Internet e de um milissegundo em redes locais, dependendo das condições da rede (MILLS, 1995).

Figura 17 - Sub-rede de sincronização do protocolo NTP



Fonte: Mills (1991)

Para aplicações onde a precisão de um milissegundo não é o suficiente, a solução é utilizar o protocolo PTP (*Precision Time Protocol*), definido no padrão IEEE 1588-2008 (IEEE, 2008). O padrão foi originalmente concebido para a realização de testes em redes e para a sua aplicação na automação industrial (NEAGOE, CRISTEA e BANICA, 2006).

O protocolo PTP implementado em software pode atingir precisão da ordem de microssegundos (KANNISTO et al., 2005). Por esta razão a utilização do protocolo PTP é a melhor solução para a sincronização de relógios de dispositivos distribuídos em uma rede de comunicação onde há o interesse em realizar medidas temporais.

3.3 TRABALHOS RELACIONADOS

Na literatura é possível encontrar alguns poucos trabalhos relacionados em medir desempenho da arquitetura OPC UA. BRAUNE, HENNIG e HEGLER (2008) avaliam o desempenho da arquitetura OPC UA no contexto de seu mecanismo de segurança, medindo o tempo de computação de algoritmos de criptografia. Em POST, SEPPÄLÄ e KOIVISTO (2009), é discutido o desempenho da arquitetura OPC UA também em termos de segurança, avaliando o impacto desse desempenho na comunicação de dispositivos de campo.

CAVALIERI e CHIACCHIO (2013) tratam do desempenho da arquitetura OPC UA com foco nos principais mecanismos de troca de dados que podem influenciar a comunicação entre cliente e servidor em ambientes industriais. Neste caso, os resultados são apresentados em termos de tempos médios medidos, não havendo, assim, uma preocupação com o comportamento temporal da operação analisada. Além disso, para alcançar o objetivo proposto no trabalho, os autores utilizam um modelo de troca de dados entre cliente e servidor OPC UA, simulado na plataforma de simulação de rede OMNeT++. Nesta situação, a sincronização de relógio não é um problema para as medições realizadas.

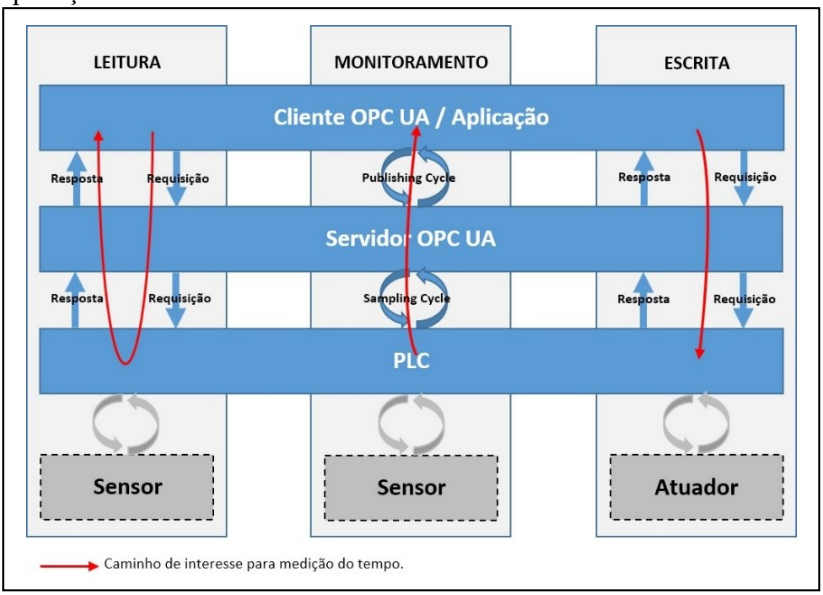
O método desenvolvido e apresentado nesta dissertação permite realizar medições dos tempos envolvidos nas operações de leitura, escrita e monitoramento de uma comunicação OPC UA real, de forma que o comportamento temporal da aplicação possa ser observado e analisado, considerando todos os fatores que possam influenciar nesses tempos. O Capítulo 4 apresenta a descrição do método desenvolvido.

4 MÉTODO PROPOSTO PARA MEDIÇÃO DE TEMPOS EM OPERAÇÕES OPC UA

Na arquitetura unificada OPC (OPC UA), o acesso à dados de tempo real dos dispositivos de chão de fábrica de um sistema de automação industrial é possível por meio das operações de leitura, escrita e monitoramento disponíveis nas especificações da tecnologia (OPC Foundation, 2017c).

O diagrama da Figura 18 apresenta as interações entre os elementos da tecnologia OPC UA e de um sistema de automação na execução dessas operações, bem como o caminho do fluxo de informações no qual há o interesse na medição de tempo.

Figura 18 - Operações da tecnologia OPC UA de interesse para a aplicação do método.



Fonte: Autor (2018)

Para a operação de leitura, o método permite medir o tempo decorrido entre o instante em que o cliente OPC UA dá início à operação e o instante em que este mesmo cliente recebe o valor lido como resposta do servidor OPC UA (*turnaround time*). Na operação de monitoramento, o método permite medir o tempo entre o instante em que uma variável monitorada tem o seu valor alterado na fonte de dados e o instante em que

essa alteração é percebida no cliente. Para a operação de escrita, o método permite medir o tempo decorrido entre o instante em que o cliente dá início a escrita e o instante em que o valor é efetivamente escrito na variável de processo do sistema de automação.

Na operação de leitura, o método permite medir também a idade do dado recebido, ou seja, a diferença entre o instante em que a informação do valor de uma variável chega no cliente e o instante em que esse valor foi atribuído à variável.

Para o desenvolvimento e aplicação do método, é necessário compreender detalhadamente os serviços OPC UA descritos na Seção 2.1.3.

4.1 DESCRIÇÃO DO MÉTODO

O método para medição do tempo de reposta das operações de leitura, escrita e monitoramento se utiliza dos carimbos de tempos disponíveis nos parâmetros dos serviços da arquitetura OPC UA para essas operações, descritos nos itens 2.1.3.1, 2.1.3.4, 2.1.3.5 e do registro dos instantes de chegada, no cliente, das mensagens de resposta do servidor.

Esse registro deve ser constante ao longo das operações, de forma a obter um conjunto de valores temporais que permitam a construção de um histograma, útil para a análise estatísticas desses dados.

4.1.1 MEDIÇÃO DE TEMPOS NA OPERAÇÃO DE LEITURA

Na leitura, são duas as medições de interesse: o *turnaround time* e a idade do dado.

4.1.1.1 *TURNAROUND TIME* NA OPERAÇÃO DE LEITURA

Na operação de leitura, o turnaround time TT pode ser obtido através de

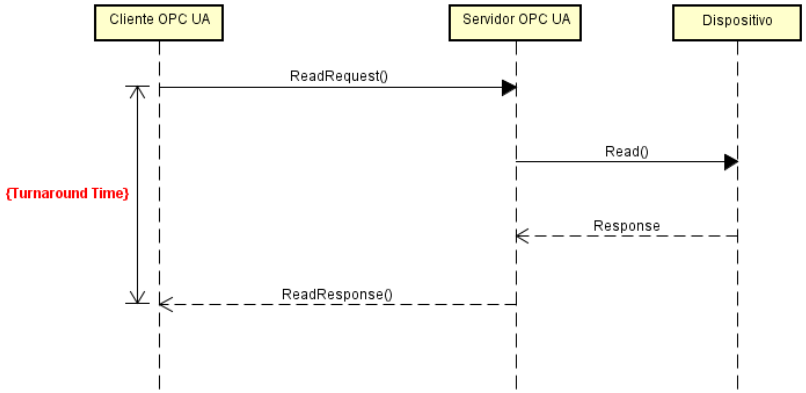
$$TT = t_{fim\_leitura} - t_{início\_leitura}, \tag{4.1}$$

onde $t_{fim\_leitura}$ é o instante em que a mensagem de resposta da operação de leitura é recebida no cliente e $t_{início\_leitura}$ é o instante em que a mensagem de requisição foi enviada pelo cliente, ambos no formato de data e hora UTC.

Para a obtenção dos valores de $t_{início\_leitura}$ e $t_{fim\_leitura}$, a aplicação cliente deve registrar em arquivo, respectivamente, o parâmetro **timestamp** contido no cabeçalho da mensagem de requisição (**requestHeader**) da operação de leitura, e o data-hora no instante em que a mensagem de resposta dessa mesma operação é recebida no cliente.

O intervalo correspondente ao *turnaround time* está representado na Figura 19.

Figura 19 - Representação do *turnaround time* na operação de leitura;



Fonte: Autor (2018)

4.1.1.2 IDADE DO DADO NA OPERAÇÃO DE LEITURA

Na operação de leitura, a idade do dado lido $I_{leitura}$ é equivalente à

$$I_{leitura} = t_{fim\_leitura} - t_{dado\_leitura}, \qquad (4.2)$$

onde $t_{dado\_leitura}$ é o data-hora no formato UTC do instante em que o valor lido foi atribuído pela fonte do dado.

O valor de $t_{dado\_leitura}$ corresponde ao valor registrado no parâmetro **sourceTimestamp** do parâmetro **results**, contido na mensagem de resposta do servidor na execução do serviço *Read*.

Para que o servidor retorne o valor do data-hora da fonte no parâmetro **sourceTimestamp**, é necessário que o cliente informe o valor SOURCE_0 ou BOTH_2 no parâmetro timestampToReturn em sua mensagem de requisição, conforme as opções de valores disponíveis no Quadro 12.

Quadro 12 - Detalhamento do parâmetro **timestampsToReturn**

timestampsToReturn	
Valores	**Descrição**
SOURCE_0	O servidor retorna o *timestamp* da fonte (*source timestamp*)
SERVER_1	O servidor retorna o *timestamp* do servidor (*server timestamp*)
BOTH_2	O servidor retorna ambos os *timestamps*, da fonte e do servidor
NEITHER_3	O servidor não retorna *timestamps*

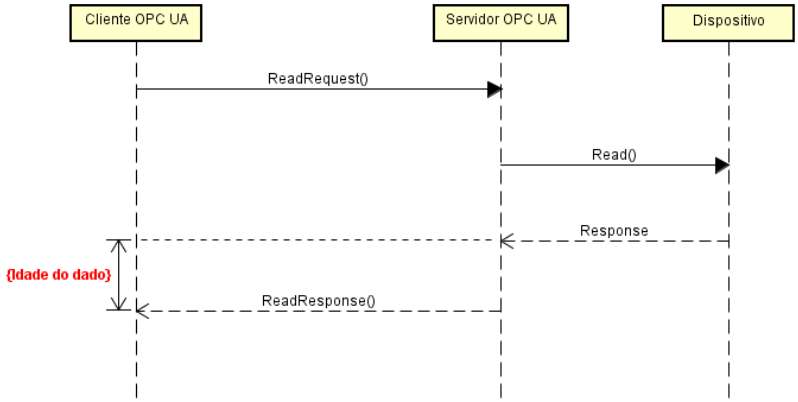
Fonte: OPC Foundation (2017c)

Para que que o valor lido seja o mais recente possível, é necessário que o parâmetro **maxAge** seja 0.

A aplicação cliente deve permitir o registro em arquivo do valor do parâmetro **sourceTimestamp** para cada operação de leitura realizada.

O intervalo correspondente à idade do dado está representado na Figura 20.

Figura 20 - Representação da idade do dado na operação de leitura



Fonte: Autor (2018)

4.1.2 MEDIÇÃO DE TEMPOS NA OPERAÇÃO DE ESCRITA

4.1.2.1 ATRASO ENTRE A REQUISIÇÃO E A ALTERAÇÃO NO VALOR DA VARIÁVEL

Na operação de escrita, o atraso da escrita $A_{escrita}$, compreendido pelo tempo decorrido entre o instante em que o cliente envia a mensagem de requisição para o serviço Write e o instante em que a variável a ser escrita tem o seu valor alterado.

O atraso da escrita pode ser obtido então por

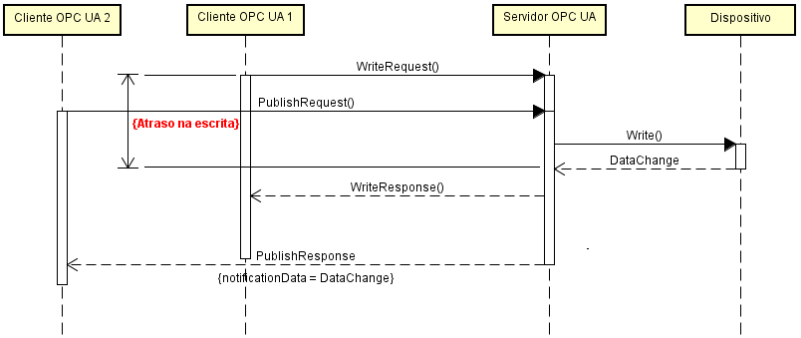
$$A_{escrita} = t_{fim\_escrita} - t_{início\_escrita}, \tag{4.3}$$

onde $t_{início\_escrita}$ é o instante em que o cliente inicia a operação de escrita e $t_{fim\_escrita}$ é o instante em que a escrita é efetivamente realizada no dispositivo.

Conforme já observado anteriormente, o parâmetro **timestamp** da mensagem de resposta do serviço *Write* não corresponde necessariamente ao instante em que a escrita é realizada na fonte de dados. Logo, para a determinação do valor de $t_{fim\_escrita}$, é necessário que o cliente realize o monitoramento da variável a ser escrita durante a operação. Quando a variável envolvida tem o seu valor alterado pela operação de escrita, um evento *data change* ocorre, gerando uma notificação para o cliente. Tal notificação tem a estrutura apresentada no Quadro 8, e carrega consigo o parâmetro **Value**, que contém o parâmetro **sourceTimestamp**. O parâmetro **sourceTimestamp** nesta situação representa o instante em que a variável teve o seu valor alterado pela operação de escrita. Assim, o valor de $t_{fim\_escrita}$ para a aplicação do método equivale ao valor registrado nesse parâmetro.

A dinâmica das mensagens trocadas nessa operação está apresentada na Figura 21.

Figura 21 - Representação do atraso na escrita



Fonte: Autor (2018)

4.1.3 MEDIÇÃO DE TEMPOS NA OPERAÇÃO DE MONITORAMENTO

4.1.3.1 ATRASO NA DETECÇÃO DE UM EVENTO *DATA CHANGE*

Na arquitetura OPC UA, um evento *data change* é um evento que tem como gatilho uma alteração no valor da variável de processo monitorada. Sempre que o valor dessa variável se altera, o servidor OPC UA envia uma mensagem de notificação para o cliente com o valor atualizado no parâmetro **Value**, Quadro 8.

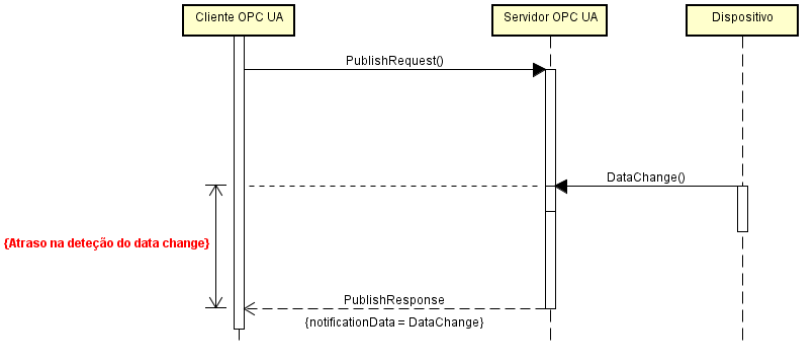
Na aplicação do método, o interesse está em medir o atraso entre o instante em que a variável tem o seu valor alterado na fonte de dados e o instante em que esse valor é recebido no cliente OPC UA. Logo, o atraso na detecção de um evento *data change* $A_{DataChange}$ é equivalente à

$$A_{DataChange} = t_{fim\_monitoramento} - t_{dado\_monitoramento}. \quad (4.4)$$

O valor de $t_{dado\_monitoramento}$ corresponde ao parâmetro **sourceTimestamp** contido no parâmetro **Value** da mensagem de notificação do Quadro 9.

O intervalo correspondente a um evento *data change* está representado na Figura 22.

Figura 22 - Representação do atraso na detecção de um *data change* na operação de monitoramento



Fonte: Autor (2018)

4.2 ETAPAS DO MÉTODO

Antes de realizar as medições, é importante que os dispositivos onde estão instalados o servidor e o cliente OPC UA estejam sincronizados. Nos testes realizados, foram obtidas medidas na ordem de grandeza de 1 milissegundo. Desta forma, é importante que o servidor e o cliente tenham seus relógios sincronizados com uma precisão menor do que a ordem de grandeza das medições.

Por utilizar informações disponíveis no próprio padrão OPC UA, o método pode ser aplicado ao longo da operação normal de uma comunicação OPC UA real. No período estipulado para o método, os parâmetros da arquitetura de interesse para cada operação, que correspondem aos parâmetros das equações (4.1), (4.2), (4.4) e (4.3), devem ser registrados em arquivo pelo cliente OPC UA.

As medições são realizadas levando em consideração a operação OPC UA aplicada e a variável de processo, ou nó no espaço de endereçamento do servidor OPC UA, envolvido. Desta forma, para uma operação de leitura, por exemplo, são registrados os parâmetros envolvidos na leitura de uma determinada variável de processo.

Para que seja possível observar o comportamento temporal da operação, é importante que as operações sejam realizadas em sequência, com uma frequência suficiente para que as influências da rede de comunicação e do sistema operacional possam ser observadas nas medições obtidas. Na aplicação do método apresentada nesta dissertação, as operações foram executadas a cada um segundo.

Não é definido um número mínimo ou máximo de operações a serem consideradas nas medições. No entanto, é importante que o número de operações seja suficientemente grande para que pontos de atrasos máximos sejam observados. Nesta dissertação, para a aplicação do método em um ambiente experimental, cada uma das operações de leitura, escrita e monitoramento foi executada 10.000 vezes.

Os parâmetros registrados em cada execução de cada operação OPC UA é aplicada nas equações (4.1), (4.2), (4.4) e (4.3), de forma que uma amostra de medições, de tamanho equivalente ao número de execuções, é obtido. A sequência a seguir apresenta as etapas da aplicação do método para a determinação do comportamento temporal do sistema.

1) Sincronizar os relógios do servidor e do cliente OPC UA continuamente. O *drift* dos relógios desses dispositivos deve ser medido ao longo da aplicação do método e registrado em arquivo;
2) Executar o servidor;
3) Executar o cliente e conectá-lo ao servidor;
4) Definir uma variável de processo a ser lida, monitorada ou escrita no espaço de endereçamento do servidor;
5) Executar a operação em teste sobre a variável de processo escolhida, pelo número de vezes equivalente ao tamanho da amostra de medições desejado;
6) Verificar o *drift* medido ao longo das execuções para validação dos parâmetros temporais registrados. O *drift* registrado deve ter ordem de grandeza menor do que a ordem de grandeza desses parâmetros;
7) Ao término das operações, se as condições do passo 6 não forem atendidas, deve-se descartar a amostra e voltar ao passo 5. Caso contrário, seguir para o passo 8;
8) Acessar o registro em arquivo da aplicação cliente e extrair os parâmetros úteis para o método; e
9) Aplicar os parâmetros correspondentes nas equações (4.1), (4.2), (4.3) e (4.4) para a determinação do comportamento temporal do *turnaround time* e da idade do dado, na operação de leitura, do atraso na escrita e do atraso na detecção evento *data change*, associado à variável escolhida.

Da amostra de medidas é possível obter o valor médio dos atrasos e o seu desvio padrão.

Pela relação entre o número da execução e o seu atraso medido, é possível gerar um gráfico que apresenta visualmente os atrasos medidos na amostra, podendo ser observados os seus valores de máximo.

Outro gráfico importante a ser gerado é o histograma. Com a amostra de medições realizadas, um histograma pode ser gerado para que a distribuição de frequência dos atrasos possa ser observada.

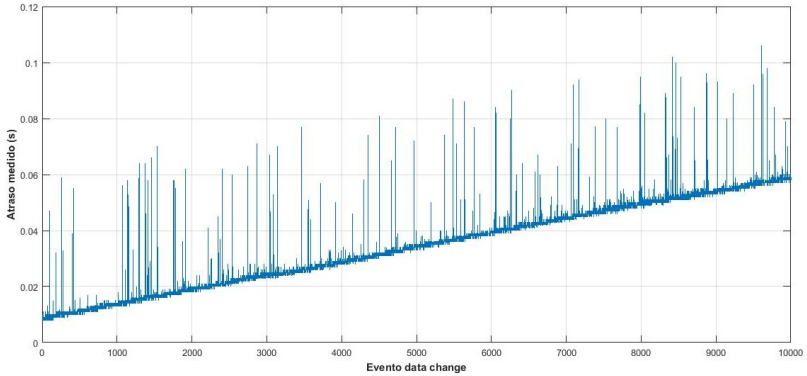
É importante determinar também se os atrasos de maior valor ocorrem em rajada ou se estão dispersos ao longo da amostra. Essa medida é realizada por meio do conceito de taxa de rajada de *spikes*, que será introduzido na aplicação do método, na Seção 5.3.1.1.

4.3 RESTRIÇÕES E CONDIÇÕES PARA APLICAÇÃO DO MÉTODO

Conforme observado, o método faz uso dos diversos parâmetros transmitidos por meio de mensagens entre clientes e servidores, na execução dos serviços envolvidos nas operações de leitura, escrita e monitoramento. Desta maneira, é necessário que o cliente possua o recurso de registrar em arquivo tais parâmetros.

Para que a aplicação do método seja viável, é necessário, ainda, que todos os componentes do sistema em teste estejam sincronizados em relação a uma mesma referência de tempo. Durante o desenvolvimento do método, verificou-se a necessidade de utilização do protocolo PTP para a sincronização do cliente e do servidor, em função dos atrasos medidos serem da ordem de unidades de milissegundos.

Figura 23 - Efeito do *drift* dos relógios na aplicação do método



Fonte: Autor (2018)

A não utilização de um protocolo de sincronização de relógios nos testes realizados gerou medidas inconsistentes, onde os atrasos medidos aumentavam linearmente ao longo do tempo, proporcionalmente ao *drift* entre os relógios.

A Figura 23 ilustra o efeito do *drift* dos relógios sobre os atrasos medidos.

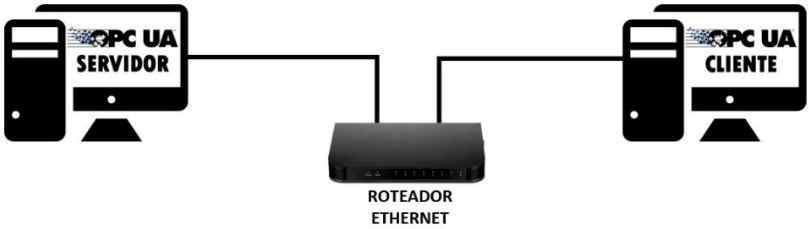
5 APLICAÇÃO DO MÉTODO

5.1 CONFIGURAÇÃO DO AMBIENTE DE TESTE

Para a aplicação do método descrito no capítulo 4, foi utilizado um ambiente de teste com a configuração apresentada na Figura 24, composto por dois computadores do tipo desktop, ambos com processadores multinúcleos (*multicore*) e sistema operacional Windows, conectados em uma mesma rede Ethernet com o auxílio de um roteador.

É importante ressaltar que o interesse aqui não é avaliar o sistema em teste, e sim a aplicação do método.

Figura 24 - Configuração do ambiente de teste



Fonte: Autor (2018)

No computador do servidor OPC UA foi instalado o simulador *Prosys OPC UA Simulation Server* (servidor Prosys), desenvolvido pela empresa Prosys, especializada no desenvolvimento de aplicações OPC e OPC UA, e de kits de desenvolvimento de aplicações OPC UA nas linguagens de programação Java, C, C++ e .NET. O servidor Prosys possui os mesmos recursos de um servidor OPC UA real, porém, os objetos OPC UA em seu espaço de endereçamento representam dispositivos simulados.

No computador cliente foram utilizadas como aplicações clientes o Matlab, da MathWorks, e o LabView, da National Instruments, com os seus respectivos kits de ferramenta OPC UA. Ao longo do desenvolvimento do método, foram utilizados os clientes UaExpert, da Unified Automation, Prosys OPC UA Client, da Prosys e o cliente OPC UA para demonstração disponibilizado pela OPC Foundation. No entanto, os *softwares* da MathWorks e da National Instruments foram utilizados em função de permitirem uma maior liberdade para implementar simulações e registrar em arquivo os parâmetros de

interesse. A utilização dos mesmos não traz prejuízos ao trabalho, tendo em vista que o objetivo aqui é avaliar a aplicação do método e não o sistema em teste.

O computador cliente se utiliza também do software de análise de protocolos de rede, Wireshark, para auxílio na compreensão das mensagens trocadas entre as aplicações cliente e servidor, e para o registro dos parâmetros dos serviços, utilizados no método, não registrados pelas aplicações cliente do Matlab e do LabView. Na aplicação do método em clientes e servidores OPC UA utilizados na indústria, esses dispositivos devem possuir o recurso de registrar em arquivo os parâmetros das mensagens trocadas, bem como o instante de chegada das mensagens de resposta do servidor, de modo que não haja dependência do software Wireshark para a aplicação do método.

Para aplicar o método de forma experimental, as aplicações clientes foram programadas para realizar as operações de leitura, escrita e monitoramento sequencialmente, de forma a gerar uma amostra de medições de tamanho satisfatório para a composição de histogramas e para uma análise adequada sobre o comportamento do tempo de resposta das operações.

5.2 PROCEDIMENTOS

Para a aplicação do método foram seguidas as etapas definidas na Seção 4.2 deste trabalho.

5.2.1 SINCRONIZAÇÃO DE RELÓGIOS

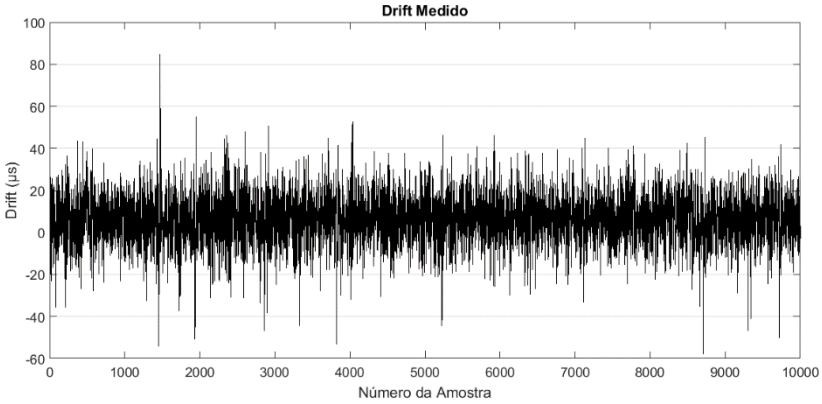
Durante a aplicação do método, realizar a sincronização dos relógios dos computadores cliente e servidor apenas no início do teste, mostrou não ser o suficiente. Em pouco tempo, a defasagem entre um relógio e outro era o suficiente para apresentar medições inconsistentes, semelhantes ao resultado apresentado na Figura 23. Pela análise dessas medições, foi possível medir uma taxa de deriva de aproximadamente 0,3 ms a cada minuto entre os relógios do cliente e do servidor. Esse valor é bem próximo da ordem de grandeza dos tempos medidos e dos *timestamps* registrados nos parâmetros das mensagens OPC, que é de 1ms.

Para contornar esse problema, o software de sincronização de tempo *Domain Time II* (DT2), da empresa *Greyware Automation Products*, foi utilizado. No computador servidor, o DT2 foi configurado para funcionar como um servidor PTP mestre, enquanto que no cliente, o

DT2 foi configurado para funcionar como um cliente PTP escravo, realizando a verificação de relógio com o DT2 mestre a cada minuto para a manutenção do sincronismo.

Ao longo de sua utilização, o DT2 instalado no cliente faz o registro em arquivo dos dados de sincronização. Desses dados é possível obter a defasagem (*drift*) entre os relógios do cliente e do servidor, medida a cada segundo. O gráfico da Figura 25 mostra o comportamento do *drift* no período das medições, apresentando um valor máximo de 85µs. Esse nível de precisão na sincronização é dez vezes menor do que a ordem de grandeza dos tempos registrados na aplicação do método, sendo considerada suficiente para garantir a confiabilidade das medições obtidas.

Figura 25 - *Drift* medido ao longo da aplicação do método



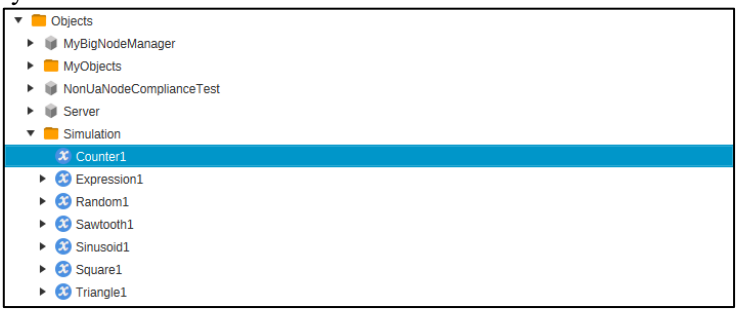
Fonte: Autor, 2018

5.2.2 OPERAÇÃO DE LEITURA

Para a aplicação do método na operação de leitura a aplicação cliente utilizada foi o LabView. O cliente foi programado de forma a realizar 10.000 operações de leitura em sequência, com 1 segundo de diferença entre uma operação e outra. O número de operações foi escolhido arbitrariamente, no entanto, com o cuidado de obter uma amostra de tamanho suficiente para que o comportamento temporal da operação fosse observado. Cada operação foi configurada para realizar a leitura do nó “Counter1”, disponível no espaço de endereçamento do servidor Prosys (Figura 26). O nó “Counter1” é um nó que simula um

contador de inteiros de 0 a 100, com frequência de incremento configurável pela aplicação. Para a aplicação do método na operação de leitura, o simulador foi configurado para incrementar o contador a cada 1 ms.

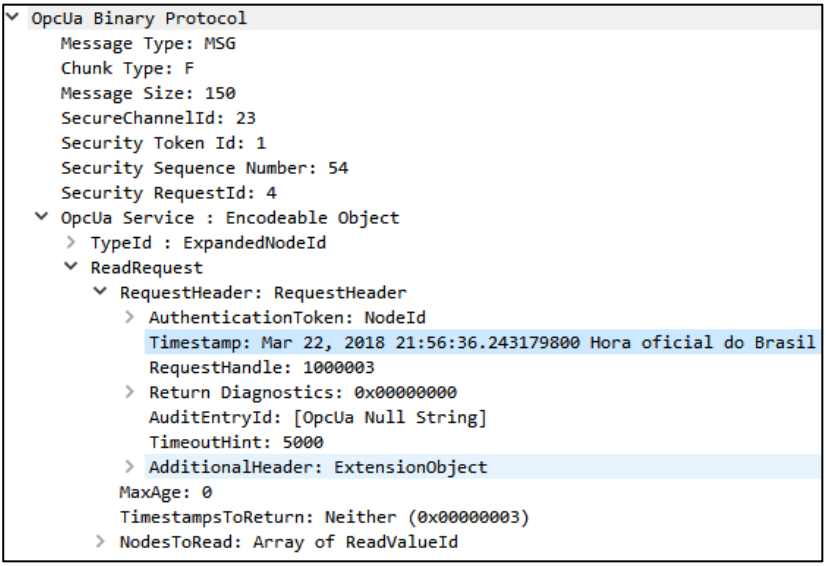
Figura 26 - Nó "*Counter1*" no espaço de endereçamento do servidor Prosys



Fonte: Tela do software Prosys OPC UA Simulation Server (2018)

A cada operação de leitura realizada, o servidor retorna para o cliente uma mensagem de resposta contendo a estrutura de parâmetros do Quadro 2, apresentado no Capítulo 4. O cliente LabView oferece recursos para registro apenas do parâmetro **sourceTimestamp** atribuído ao valor lido. Para o registro dos demais parâmetros, necessários para a aplicação das equações (4.1) e (4.2), foi utilizado o software Wireshark. O software foi configurado para fazer o registro do quadro Ethernet das mensagens trocadas entre os computadores cliente e servidor, de onde é possível obter informações detalhadas sobre o protocolo OPC UA, conforme mostra a Figura 27.

Figura 27 - Informações do protocolo OPC UA capturadas pelo Wireshark



Fonte: Tela do software Wireshark (2018)

A partir da análise das informações disponibilizadas pelo Wireshark, e do parâmetro **sourceTimestamp** registrado pelo LabView, é possível então obter os parâmetros $t_{fim\_leitura}$, $t_{início\_leitura}$ e $t_{dado\_leitura}$ das equações (4.1) e (4.2), e calcular, respectivamente, o *turnaround time* e a idade do dado para cada uma das 10.000 leituras.

5.2.3 OPERAÇÃO DE ESCRITA

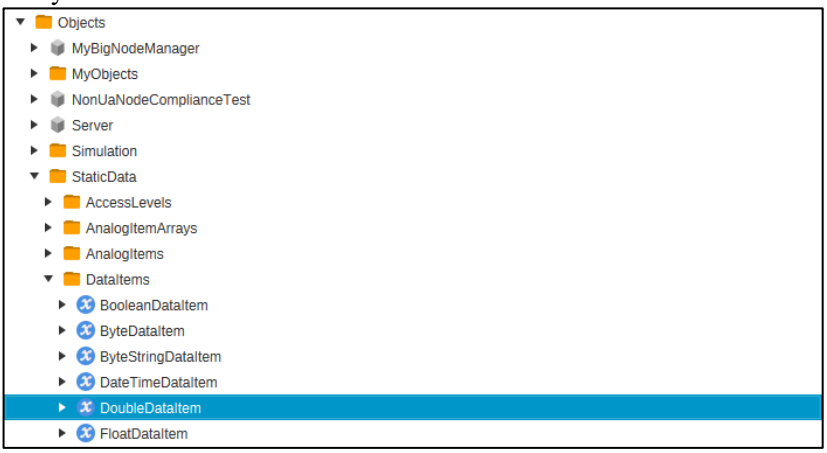
Para a aplicação do método na operação de escrita, o cliente utilizado foi o Matlab, onde foram utilizadas as funções disponíveis na OPC Toolbox para a conexão do cliente Matlab com o servidor e a escrita de valores no nó de destino no espaço de endereçamento desse servidor.

O servidor Prosys OPC UA contém, além de nós simulados, nós que armazenam dados estáticos em seu espaço de endereçamento. Para realizar a escrita, o nó **DoubleDataItem** foi escolhido (Figura 28). Esse nó armazena dados do tipo *double*.

O cliente Matlab foi programado para realizar 10.000 operações de escrita no nó **DoubleDataItem** a cada 1 minuto. Inicialmente, a proposta

era escrever no nó o valor correspondente aos segundos do data-hora no instante em que a operação é realizada no cliente. Essa informação seria utilizada para determinar o valor do parâmetro $t_{início\_escrita}$ da Equação (4.3. No entanto, optou-se por obter esse parâmetro pela informação disponibilizada na mensagem de requisição da operação de escrita.

Figura 28 - Nó **DoubleDataItem** no espaço de endereçamento do servidor Prosys



Fonte: Tela do software Wireshark (2018)

Desta forma, os parâmetros $t_{início\_escrita}$ e $t_{fim\_escrita}$ da Equação (4.3 foram obtidos por meio da análise das mensagens de requisição e resposta da operação de escrita, capturadas com o auxílio do software Wireshark. Dessas informações é possível calcular o atraso na operação de escrita para cada uma das 10.000 operações.

Conforme descrito na Seção 4.1.2.1 desta dissertação, o valor do parâmetro $t_{fim\_escrita}$ é obtido pelo parâmetro **sourceTimestamp** contido na notificação que o servidor gera para um item monitorado pelo cliente. Por esta razão, a aplicação do método para as operações de escrita e monitoramento foram realizadas paralelamente.

5.2.4 OPERAÇÃO DE MONITORAMENTO

Para a operação de monitoramento, o cliente utilizado foi o LabView, que foi programado para monitorar o nó **DoubleDataItem** no espaço de endereçamento do servidor Prosys.

O monitoramento do nó foi realizado paralelamente às operações de escrita do item anterior. A medida que a aplicação cliente do Matlab escreve no nó **DoubleDataItem**, o servidor Prosys gera mensagens de notificações para a aplicação cliente LabView, contendo o novo valor atribuído à variável.

A mensagem de notificação carrega consigo o **sourceTimestamp** associado ao valor alterado pelas operações de escrita da Seção 5.2.3. No instante em que o LabView detecta a notificação, o data-hora do instante é registrado, assim como o **sourceTimestamp** atribuído ao dado monitorado. Esses parâmetros correspondem aos parâmetros $t_{fim\_monitoramento}$ e $t_{dado\_monitoramento}$ da Equação (4.4. Ao término das medições, são obtidas 10.000 notificações de eventos *data change* no cliente LabView, referente às 10.000 operações de escrita realizadas pela aplicação cliente do Matlab.

5.3 RESULTADOS OBTIDOS

A execução dos procedimentos descritos na Seção 5.2 permite obter os valores dos parâmetros $t_{início\_leitura}$, $t_{fim\_leitura}$, $t_{dado\_leitura}$, $t_{início\_escrita}$, $t_{fim\_escrita}$, $t_{dado\_monitoramento}$ e $t_{fim\_monitoramento}$, das equações (4.1, (4.2, (4.3 e (4.4, para 10.000 operações de leitura, escrita e monitoramento realizadas entre o cliente e o servidor.

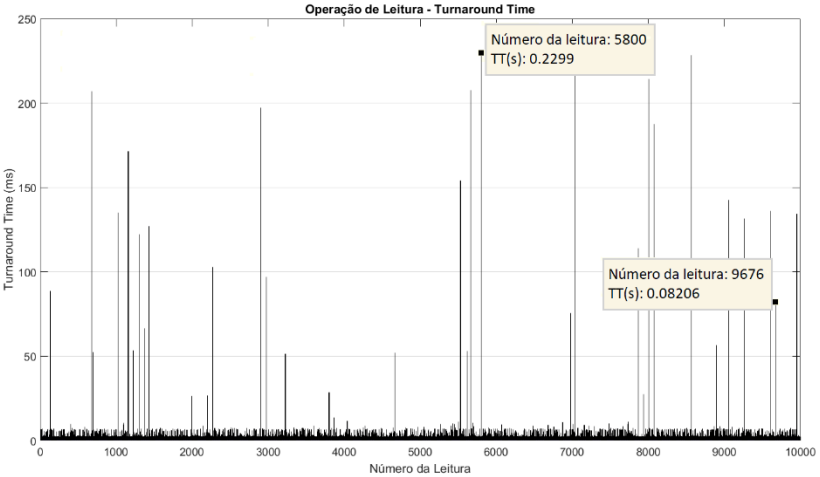
Dessas medições são extraídas informações importantes sobre o comportamento das operações em relação ao seu tempo de resposta ao longo do período de aplicação do método.

5.3.1 *TURNAROUND TIME* DA OPERAÇÃO DE LEITURA

5.3.1.1 RESULTADOS OBTIDOS

Pela aplicação do procedimento descrito na Seção 5.2.2, foi possível obter o resultado apresentado no gráfico da Figura 29, onde as operações de leitura estão distribuídas ao longo da abscissa, e suas respectivas medições de tempo, na ordenada.

Figura 29 - *Turnaround time* medido em 10.000 operações sequenciais de leitura



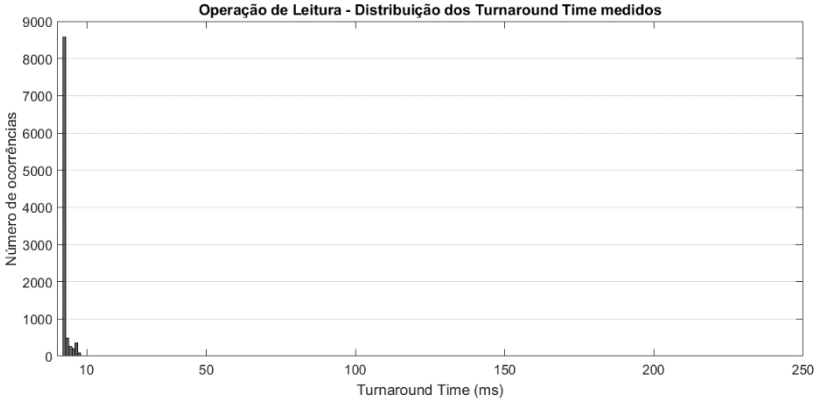
Fonte: Autor (2018)

É possível observar no gráfico medições onde o *turnaround time* é muito maior do que a média dos *turnaround time* medidos. Esses picos de atraso na comunicação, também conhecidos como *spikes*, podem ser causados por congestionamento na rede Ethernet, resultando em longas filas de pacotes de rede no roteador (SHALLWANI e KABAL, 2013). Além disso, os computadores pessoais, como os utilizados neste teste, por não terem um hardware e um sistema operacional construídos com o propósito de atenderem requisitos de tempo real, induzem latências não controláveis e não detectáveis ao sistema (STARKE e DE OLIVEIRA, 2011).

O *turnaround time* máximo medido é observado na leitura de número 5800, com um valor de aproximadamente 230 ms. Esse valor é bem superior ao valor médio do *turnaround time* ao longo das 10.000 operações, que é de 3 ms com um desvio padrão de 7 ms.

Apesar dos valores de pico apresentados no gráfico da Figura 29, 99,57% das operações realizadas possuem um *turnaround time* abaixo de 10 ms, o que justifica a forma do histograma da Figura 30.

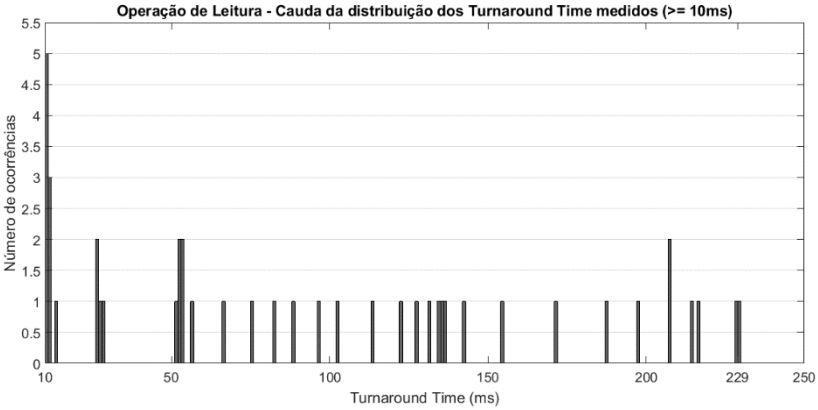
Figura 30 - Histograma dos *turnaround time* medidos



Fonte: Autor (2018)

Na Figura 31 é possível observar em detalhe a cauda do histograma da Figura 30. São poucas as ocorrências de *turnaround time* acima de 10 ms.

Figura 31 - Cauda do histograma dos *turnaround time* medidos



Fonte: Autor (2018)

Aplicações de tempo real com restrições temporais flexíveis suportam a perda de alguns prazos. No caso das medições apresentadas aqui, os *spikes* representam operações de leitura cujo *turnaround time* excede 10 ms, distribuídos na cauda do histograma. Em uma situação onde exceder esse limite signifique perder prazos, as perdas representariam apenas 0,47%. No entanto, para algumas aplicações,

perdas consecutivas de prazos podem não ser toleradas, o que gera a necessidade de analisar as medições realizadas considerando pequenas janelas de medições, avaliando a frequência de *spikes* nessas janelas. Neste trabalho, a frequência de *spikes* em uma janela será denominado taxa de rajada de *spikes*.

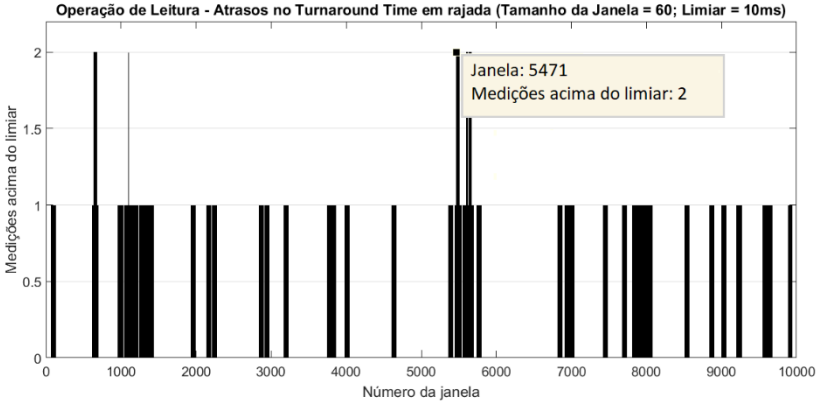
5.3.1.2 TAXA DE RAJADA DE *SPIKES*

Para determinar a taxa de rajadas de *spikes*, as medições são divididas em grupos menores (janela), onde é medido a ocorrência de *spikes*. Um tamanho de janela ($janela$) deve ser definido de forma que a primeira janela corresponda às medições de número 1 até $janela - 1$, a segunda janela corresponda às medições de número 2 até $janela - 2$, e assim sucessivamente, até a última janela correspondente às medições de número $(n - janela + 1)$ até n, onde n é o número total de medições realizadas.

Para as medições dos *turnararound time*, um *spike* foi definido como qualquer medição acima de 10 ms e o tamanho de janela equivalente a 60 medições. Em uma aplicação real, esses valores são obtidos em função dos requisitos temporais do sistema. Neste trabalho, o ambiente de teste não possui um requisito de tempo real definido. Por esta razão, os valores de 10ms para o limiar de um *spike* e de 60 medições para o tamanho de janela foram escolhidos arbitrariamente com um propósito ilustrativo. Apesar disto, esses valores são razoavelmente típicos em aplicações de tempo real com requisitos flexíveis, e úteis aqui para a validação do método.

Nessas condições, foi possível obter o gráfico da Figura 32, onde estão distribuídos na abscissa as janelas de medições e, na ordenada, as ocorrências de *spikes* nessas janelas.

Figura 32 - Ocorrências de *spikes* nas janelas de medições dos *turnaround time*



Fonte: Autor (2018)

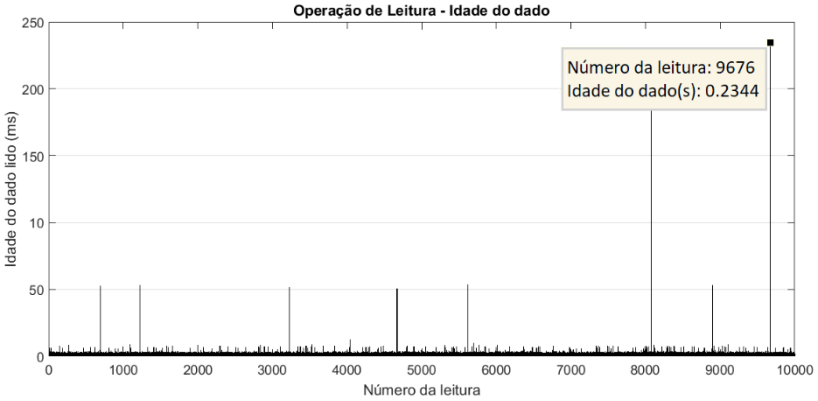
Nas medições dos *turnaround time*, é possível notar que existem janelas de medições onde duas medições apresentam um *turnaround time* maior do que o limite de 10 ms atribuído. Ou seja, a taxa de rajada nessas janelas é de $\frac{2}{60}$ *spikes*/medição, o equivalente a 3,33%.

5.3.2 IDADE DO DADO NA OPERAÇÃO DE LEITURA

5.3.2.1 RESULTADOS OBTIDOS

Aplicando o procedimento descrito na Seção 5.2.2, foi possível obter o seguinte resultado sobre as idades dos dados medidos, apresentado no gráfico da Figura 33. É possível observar na figura que, o maior valor para a idade do dado medido foi de, aproximadamente, 235 ms. O valor médio foi de 2 ms com desvio padrão de 3,5 ms.

Figura 33 - Idade do dado medido em 10.000 operações sequenciais de leitura

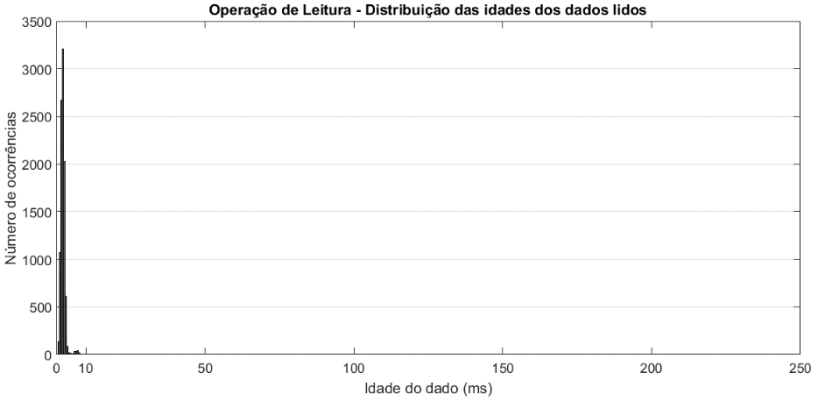


Fonte: Autor (2018)

O valor máximo da idade do dado medido ocorre na leitura de número 9676. O *turnaround time* medido na mesma operação (Figura 29) foi de 82 ms. Nesses termos, a medida de idade do dado de 235 ms parece inconsistente. No entanto, apesar do LabView transmitir a mensagem de requisição para a operação de leitura com valor 0 no parâmetro **maxAge**, o servidor realiza o melhor esforço para entregar a leitura mais recente recebida. Neste caso, o servidor não é capaz de entregar o valor mais recente, enviando para o cliente um valor de leitura anteriormente armazenado.

Um total de 9990 medições apresentaram idade do dado medido menor ou igual a 10 ms, o que representa 99,9% das medições. No histograma da Figura 34 é possível observar a distribuição das medidas concentradas abaixo de 10 ms no gráfico.

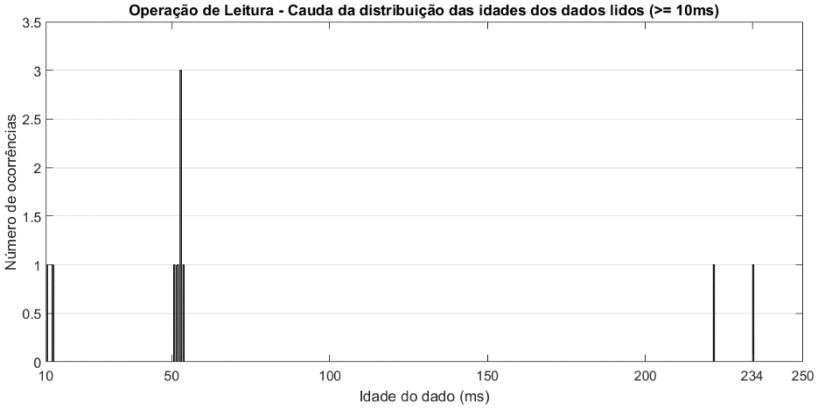
Figura 34 - Histograma das idades dos dados medidos



Fonte: Autor (2018)

As medidas com idade do dado acima de 10 ms representam apenas 0,1% do total de medições. Essas medidas estão concentradas na cauda do histograma e podem ser observadas na Figura 35.

Figura 35 - Cauda do histograma das idades dos dados medidos



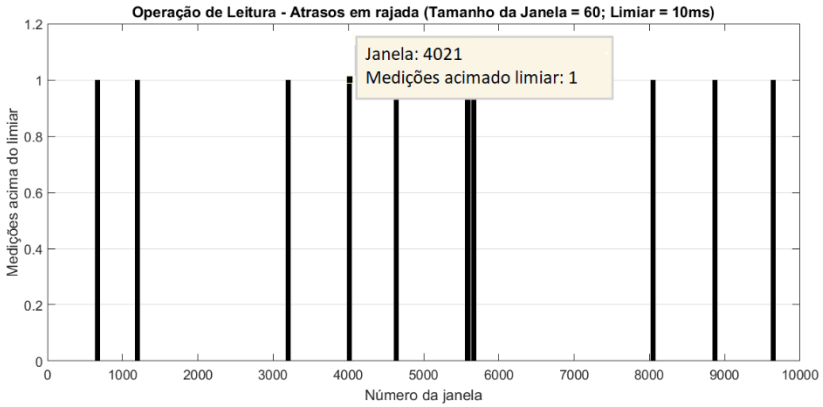
Fonte: Autor (2018)

5.3.2.2 TAXA DE RAJADA DE *SPIKES*

Aplicando o conceito de taxa de rajada de *spikes* apresentado na Seção 5.3.1.2, atribuindo um tamanho de janela de 60 medições e

considerando um *spike* qualquer medição acima de 10 ms, foi possível obter o gráfico da Figura 36.

Figura 36 - Ocorrência de *spikes* nas janelas de medições das idades dos dados
Fonte: Autor (2018)



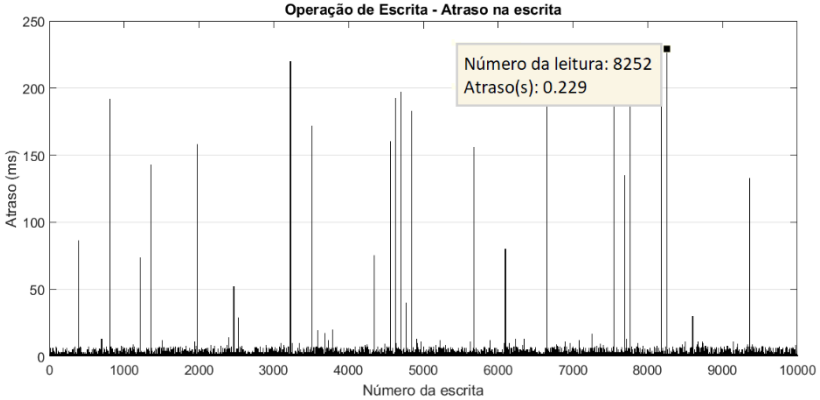
Dentre as janelas de medições definidas, a taxa de rajada no pior caso apresentado foi de $\frac{1}{60}$ *spikes*/medição, o equivalente a 1,67%.

5.3.3 ATRASO NA ESCRITA

5.3.3.1 RESULTADOS OBTIDOS

Aplicando o procedimento descrito na Seção 5.2.3, foi possível obter o seguinte resultado sobre o atraso na operação de escrita, apresentado no gráfico da Figura 37. É possível observar que o maior atraso medido foi de, aproximadamente, 229 ms. O atraso médio medido foi de 2 ms com desvio padrão de 8 ms.

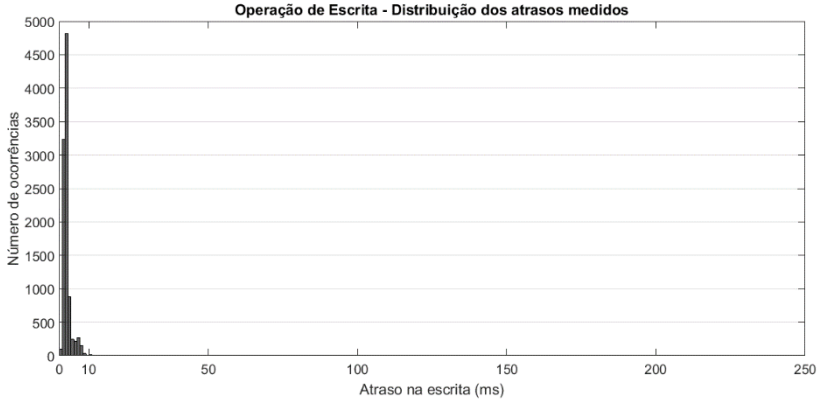
Figura 37 - Atraso na escrita medido em 10.000 operações sequenciais



Fonte: Autor (2018)

Das 10.000 operações de escrita realizadas, 99,41% apresentaram um atraso menor que 10 ms. O histograma da Figura 38 apresenta a distribuição dos atrasos medidos.

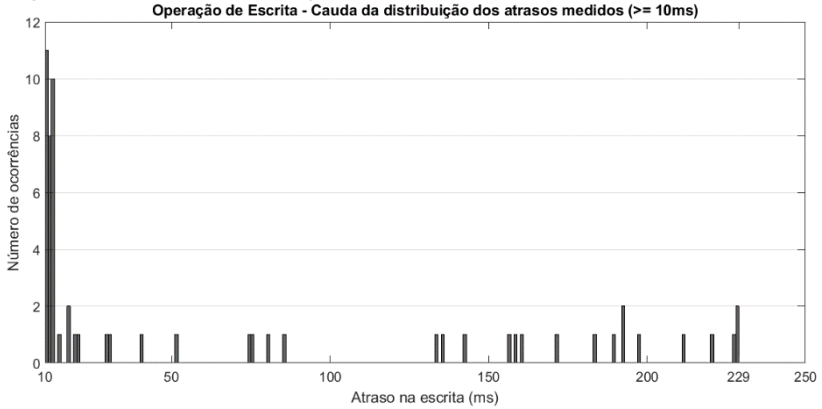
Figura 38 - Histograma dos atrasos de escrita medidos



Fonte: Autor (2018)

A distribuição das medidas com atrasos acima de 10 ms pode ser melhor observada na Figura 39, que detalha a cauda do histograma da Figura 38.

Figura 39 - Cauda do histograma dos atrasos de escrita medidos



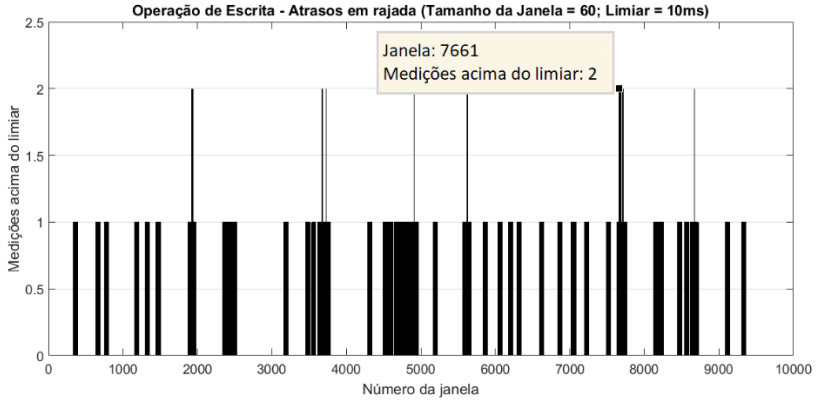
Fonte: Autor (2018)

Na operação de escrita, os atrasos acima de 10 ms representam 0,59% do total das medições.

5.3.3.2 TAXA DE RAJADA DE *SPIKES*

Aplicando o conceito de taxa de rajada de *spikes* nas medidas dos atrasos na operação de escrita, é possível obter o gráfico apresentado na Figura 40. O mesmo tamanho de janela e de limiar para os *spikes* dos seções anteriores foi aplicado aqui.

Figura 40 - Ocorrência de *spikes* nas janelas de medições de atrasos na escrita



Fonte: Autor (2018)

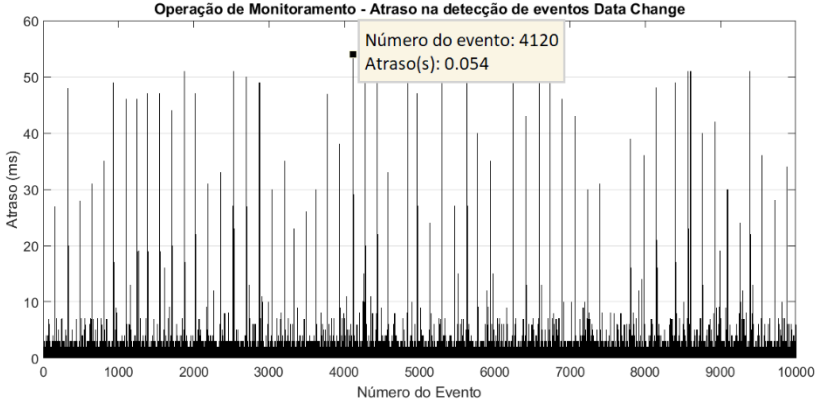
Em 93 janelas de medições, a taxa de rajada de *spikes* foi de $\frac{2}{60}$ *spikes*/medição, o equivalente a 3,33%.

5.3.4 ATRASO NO MONITORAMENTO

5.3.4.1 RESULTADOS OBTIDOS

Aplicando o procedimento descrito na Seção 5.2.4, foi possível medir os atrasos na detecção dos eventos *data change* de item monitorado, causados pelas 10.000 operações de escrita. O gráfico da Figura 41 apresenta o resultado dessas medições, onde é possível observar que o maior atraso medido ocorre na operação de número 4120, cujo valor de atraso é de 54 ms. O valor médio dos atrasos foi de 2 ms com desvio padrão de 3,5 ms.

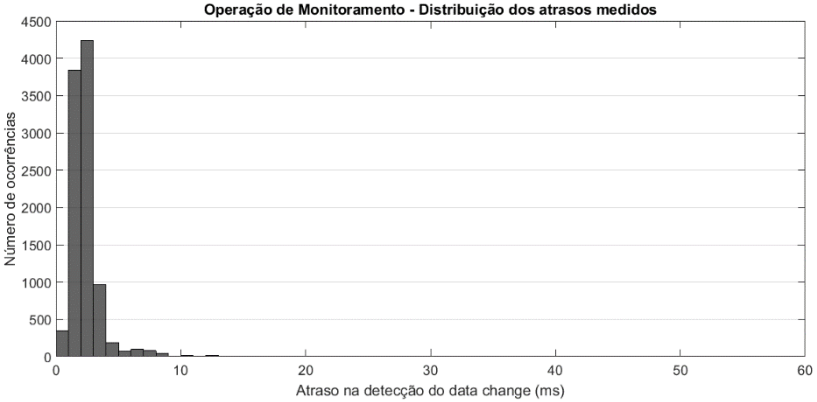
Figura 41 – Atraso na detecção de 10.000 eventos *data change*



Fonte: Autor (2018)

O histograma da Figura 42 permite observar que, assim como ocorre com as medições nas operações de leitura e escrita, os atrasos medidos se concentram abaixo de 10 ms. Nessa faixa estão concentradas 98,72% das medições.

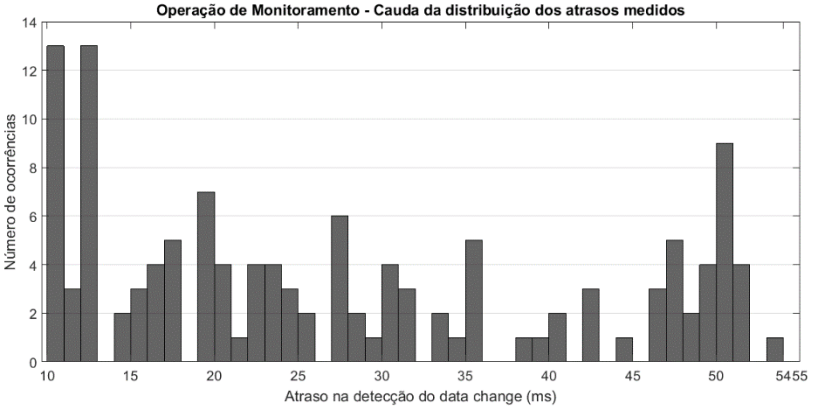
Figura 42 - Histograma dos atrasos na detecção de eventos *data change*



Fonte: Autor (2018)

As medidas cujo atraso é maior do que 10 ms estão concentradas na cauda do histograma, conforme pode ser observado no gráfico da Figura 43, e correspondem a 1,28% das medições.

Figura 43 - Cauda do histograma dos atrasos na detecção dos eventos *data change*



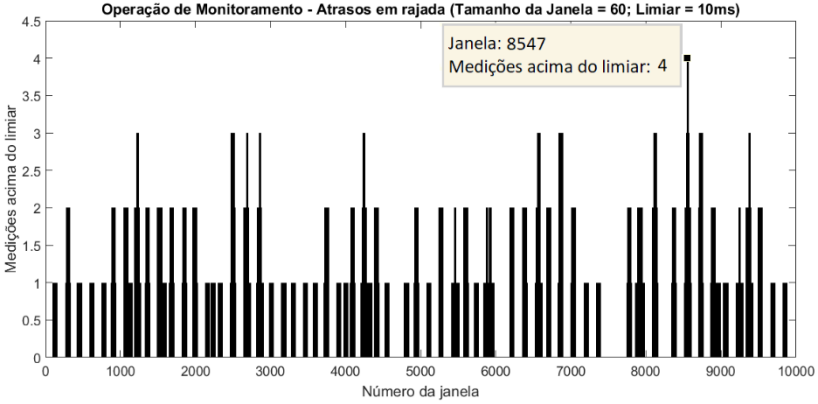
Fonte: Autor (2018)

5.3.4.2 TAXA DE RAJADA DOS *SPIKES*

O gráfico da Figura 44 apresenta o resultado da aplicação do conceito de taxa de rajada de *spikes* sobre as medições dos atrasos dos

eventos *data change* nas operações de monitoramento realizadas. O mesmo tamanho de janela e de limiar para os *spikes* dos itens anteriores foi aplicado aqui.

Figura 44 - Ocorrência de *spikes* nas janelas de medições dos atrasos na detecção de *data change*



Fonte: Autor (2018)

Na operação de monitoramento, 10 janelas de medições apresentaram uma taxa de rajada de *spikes* de $\frac{4}{60}$ *spikes*/medição, o equivalente a 6,67%.

5.4 CONSIDERAÇÕES FINAIS

Para a aplicação do método em um ambiente de automação real, é necessário que as aplicações clientes possuam o recurso de registrar em arquivo os parâmetros necessários. A utilização do software Wireshark para a captura dessas informações é limitada. Não é possível a sua utilização em conexões OPC UA seguras, tendo em vista que, neste caso, as informações estão criptografadas.

Para a utilização do método em aplicações de tempo real com restrições temporais flexíveis, o tempo máximo medido pode não ser o suficiente para validar o sistema, sendo necessário avaliar a taxa de rajada de *spikes* nas janelas de medições, de acordo com os requisitos temporais da aplicação. Neste trabalho, os parâmetros para a determinação desta taxa, tais como o tamanho da janela e o limiar de um *spike*, foram escolhidos arbitrariamente para avaliar a aplicação do método. Para a aplicação do método sobre um sistema real, os requisitos de temporais definirão o tamanho da janela e o limiar de *spike*.

É comum na indústria o uso de dispositivos que utilizam o Windows como sistema operacional de suporte para as aplicações cliente e servidor OPC UA. Obviamente, neste contexto, o atendimento de requisitos de tempo real rígidos (*hard real-time*) não é possível. No entanto, grande parte das aplicações de supervisão (SCADA) toleram atrasos eventuais, contanto que as rajadas de atrasos sejam limitadas. No caso da plataforma de teste, os resultados foram bons em termos probabilísticos.

O fato da arquitetura OPC UA ser independente da plataforma permite a sua utilização em *hardware* dedicado e em sistemas operacionais específicos para aplicações de tempo real. Nessas condições, os resultados obtidos podem ser ainda melhores, uma vez que podem diminuir as incidências de *spikes*.

Por utilizar informações contidas nas mensagens trocadas entre servidor e cliente OPC UA, o método pode ser aplicado durante a operação normal de um sistema. Neste caso, as medições têm a função de servir como ferramenta para auditoria e diagnóstico da aplicação no contexto de atendimento de requisitos temporais, uma vez que na operação normal de um sistema as operações OPC UA podem ser executadas esporadicamente.

6 CONCLUSÃO

A modernização de um sistema de automação industrial pela utilização do padrão OPC UA na comunicação entre os seus dispositivos, requer atenção no que diz respeito aos impactos deste padrão no desempenho de operações de acesso a dados de tempo real desse sistema. A tecnologia OPC UA introduz uma infraestrutura de software complexa às aplicações industriais, que pode impactar no desempenho geral da comunicação entre seus dispositivos.

O acesso a dados de tempo real é realizado por meio de serviços disponíveis na arquitetura OPC UA, que permitem que operações de leitura, escrita e monitoramento sejam realizadas nas variáveis de processo de um sistema de automação industrial.

Uma maneira eficaz de avaliar o desempenho dessa comunicação é medir os tempos envolvidos nessas operações e avaliar o seu comportamento temporal ao longo de várias operações sucessivas.

Pela utilização de parâmetros da comunicação OPC UA, utilizados como mecanismo para a troca de informações entre clientes e servidores na execução dos serviços, foi possível desenvolver um método de medição de tempos nas operações OPC UA. Medir tempo nessas operações traz a possibilidade de analisar o comportamento temporal de uma comunicação OPC UA, permitindo assim avaliar a sua aplicabilidade em sistema com requisitos temporais flexíveis (*soft real-time system*s).

Aplicar o método desenvolvido em uma comunicação OPC UA de teste, composto por um cliente e um servidor OPC UA interligados entre si por uma rede Ethernet, permitiu comprovar a eficácia do método. Nos resultados obtidos, é possível observar o comportamento temporal das operações ao longo da utilização da comunicação. A observação desse comportamento permite determinar possíveis perdas de prazos em aplicações com restrições temporais.

Ainda nos resultados obtidos, foi possível observar que alguns atrasos de maior intensidade (*spikes*) ocorrem esporadicamente na comunicação e, em algumas ocasiões, na forma de rajada. Para avaliar o impacto desse efeito, o conceito de taxa de rajada de *spikes* foi introduzido e uma avaliação nesses termos foi realizada sobre as medições.

Pela aplicação do método e pelos resultados obtidos, é possível concluir que o método desenvolvido é eficaz como ferramenta para a avaliação do comportamento temporal de uma comunicação OPC UA, desde que alguns requisitos sejam atendidos, tal como os dispositivos envolvidos terem os seus relógios sincronizados continuamente ao longo do período de aplicação do método. É necessário, ainda, que a aplicação

cliente OPC UA envolvida na medição seja capaz de registrar em arquivos as informações dos parâmetros trocados entre cliente e servidor na execução dos serviços das operações.

Ao longo do desenvolvimento deste trabalho, não foi possível aplicar o método sobre uma comunicação OPC UA de um sistema de automação real, tendo em vista que o objetivo e contribuição deste trabalho é o próprio método. A avaliação do método em uma aplicação real pode ser realizado em trabalhos futuros. Os servidores e clientes OPC UA podem ser instalados em diferentes plataformas com diferentes sistemas operacionais, o que torna interessante a aplicação do método desenvolvido em sistemas operacionais de tempo real, tais como o VxWorks, o QNX o Neutrino, o FreeRTOS, o LynxOS e o Linux com sua extensão RTAI (*Real Time Application Interface*). Neste sentido, pode ser desenvolvido um trabalho com o objetivo de comparar o desempenho da arquitetura OPC UA em diferentes sistemas operacionais.

Além de utilizar o método para medir o desempenho de diferentes sistemas operacionais, pode-se avaliar também a influência da rede ao aplicar o método sobre aplicações OPC UA utilizando diferentes protocolos de redes industriais, tais como, PROFINET, PROFIBUS e Modbus.

Neste ano de 2018, a OPC Foundation lançou a parte 14 da especificação da arquitetura OPC UA, que introduz um novo modelo de comunicação, cuja proposta é permitir a adoção da comunicação OPC UA nos níveis mais profundos do chão de fábrica, onde controladores, sensores e dispositivos embarcados tipicamente requerem baixa latência de comunicação em redes locais (OPC Foundation, 2018a). Esse novo modelo de comunicação permite a utilização do protocolo UDP (*User Data Protocol*) que, em conjunto com a utilização de dispositivos embarcados, pode tornar a comunicação mais determinista. Nesta situação, pode ser viável a utilização da Teoria de Valores Extremos (TVE) sobre as amostras de atrasos obtidas com a aplicação do método desenvolvido nesta dissertação, para a determinação de um limite superior probabilístico desse atraso. Um exemplo da aplicação de TVE na determinação do pior caso probabilístico de atrasos em redes de comunicação pode ser encontrado em MOURADIAN (2016).

REFERÊNCIAS

BRAUNE, Annerose; HENNIG, Stefan; HEGLER, Sebastian. Evaluation of OPC UA secure communication in web browser applications. In: **Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on**. IEEE, 2008. p. 1660-1665.

CAVALIERI, Salvatore; CHIACCHIO, Ferdinando. Analysis of OPC UA performances. **Computer Standards & Interfaces**, v. 36, n. 1, p. 165-177, 2013.

CAVALIERI, Salvatore; CHIACCHIO, Ferdinando. Analysis of OPC UA performances. **Computer Standards & Interfaces**, v. 36, n. 1, p. 165-177, 2013.

COULOURIS, George et al. **Sistemas Distribuídos-: Conceitos e Projeto**. Bookman Editora, 2013.

FARINES, Jean-Marie; FRAGA, Joni da Silva; OLIVEIRA, RS de. Sistemas de tempo real. **Escola de Computação**, v. 2000, p. 201, 2000.

FLAMMINI, Alessandra; FERRARI, Paolo. Clock synchronization of distributed, real-time, industrial data acquisition systems. In: **Data Acquisition**. InTech, 2010.

HOLLENDER, Martin. **Collaborative process automation systems**. ISA, 2010.

IEEE. Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, **IEEE Std. 1588-2008**, 2008.

IPQM – Instituto de Pesquisas da Marinha. **Missão**. Disponível em: <https://www.marinha.mil.br/ipqm/missao>. Acesso em: 14 fev 2018.

KANNISTO, Juha et al. Software and hardware prototypes of the IEEE 1588 precision time protocol on wireless LAN. In: **Local and Metropolitan Area Networks, 2005. LANMAN 2005. The 14th IEEE Workshop on**. IEEE, 2005. p. 6 pp.-6.

KOPETZ, Hermann. **Real-time systems: design principles for distributed embedded applications**. Springer Science & Business Media, 2011.

LIU, Jane WS. Real-time systems. 2000.

MAHNKE, Wolfgang; LEITNER, Stefan-Helmut; DAMM, Matthias. **OPC unified architecture**. Springer Science & Business Media, 2009.

MATHEUS, Kirsten; KÖNIGSEDER, Thomas. **Automotive ethernet**. Cambridge University Press, 2017.

MatrikonOPC. **About MatrikonOPC** Em: <https://www.matrikonopc.com/main/about.aspx>. Acessado em: 14 fev 2018.

MILLS, David L. **Computer network time synchronization: the network time protocol on earth and in space**. CRC Press, 2016.

MILLS, David L. Improved algorithms for synchronizing computer network clocks. **IEEE/ACM Transactions on networking**, v. 3, n. 3, p. 245-254, 1995.

MILLS, David L. Internet time synchronization: the network time protocol. **IEEE Transactions on communications**, v. 39, n. 10, p. 1482-1493, 1991.

MOURADIAN, Alexandre. Extreme value theory for the study of probabilistic worst case delays in wireless networks. **Ad Hoc Networks**, v. 48, p. 1-15, 2016.

NEAGOE, Teodor; CRISTEA, Valentin; BANICA, Logica. NTP versus PTP in com puter networks clock synchronization. In: **Industrial Electronics, 2006 IEEE International Symposium on**. IEEE, 2006. p. 317-362.

OPC Foundation – The Industrial Interoperability Standard. **OPC Foundation announces OPC UA PubSub release as important extension of OPC UA communication platform**. Disponível em:

<OPC Foundation – The Industrial Interoperability Standard>. Acessado em: 14 fev 2018.

OPC Foundation – The Industrial Interoperability Standard. **Unified Architecture**. Disponível em: <OPC Foundation – The Industrial Interoperability Standard>. Acessado em: 14 fev 2018.

OPC Foundation – The Industrial Interoperability Standard. **What is OPC?** Disponível em: <https://opcfoundation.org/about/what-is-opc/>. Acessado em: 14 fev 2018.

OPC Foundation, **OPC Unified Architecture Specification Part 1: Overview and Concepts**, Release 1.04 (2017)

OPC Foundation, **OPC Unified Architecture Specification Part 3: Address Space Model**, Release 1.04 (2017)

OPC Foundation, **OPC Unified Architecture Specification Part 4: Services**, Release 1.04 (2017)

POST, Olli; SEPPÄLÄ, Jari; KOIVISTO, Hannu. The Performance of OPC-UA Security Model at Field Device Level. In: **ICINCO-RA**. 2009. p. 337-341.

SHALLWANI, Aziz; KABAL, Peter. An adaptive playout algorithm with delay spike detection for real-time VoIP. In: **Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on**. IEEE, 2003. p. 997-1000.

SHIMANUKI, Yho. OLE for process control (OPC) for new industrial automation systems. In: **Systems, Man, and Cybernetics, 1999. IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on**. IEEE, 1999. p. 1048-1050.

STARKE, Renan A.; DE OLIVEIRA, Romulo S. Impact of the x86 System Management Mode in Real-Time Systems. In: **Computing System Engineering (SBESC), 2011 Brazilian Symposium on**. IEEE, 2011. p. 151-157.

STIGGE, Martin et al. The digraph real-time task model. In: **Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE**. IEEE, 2011. p. 71-80.

ZHOU, Hui et al. Frequency accuracy & stability dependencies of crystal oscillators. **Carleton University, Systems and Computer Engineering, Technical Report SCE-08-12**, 2008.