INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO RN
UNIDADE SEDE
DEPARTAMENTO ACADÊMICO DE TECNOLOGIA DA INFORMAÇÃO
CURSO SUPERIOR DE TECNOLOGIA EM DESENVOLVIMENTO DE SOFTWARE

JALERSON RAPOSO FERREIRA DE LIMA

XRTPS – EXTENSIBLE REAL-TIME PUBLISH SUBSCRIBE: UM MIDDLEWARE PARA TROCA DE DADOS EM REDES INDUSTRIAIS DE TEMPO REAL BASEADO EM XML UTILIZANDO MULTICAST

NATAL-RN
2007

JALERSON RAPOSO FERREIRA DE LIMA

XRTPS – EXTENSIBLE REAL-TIME PUBLISH SUBSCRIBE: UM MIDDLEWARE PARA TROCA DE DADOS EM REDES INDUSTRIAIS DE TEMPO REAL BASEADO EM XML UTILIZANDO MULTICAST

Monografia apresentada à Banca Examinadora do Trabalho de Conclusão do Curso de Tecnologia em Desenvolvimento de Software, em cumprimento às exigências legais como requisito parcial à obtenção do título de Tecnólogo em Desenvolvimento de Software.

Orientador: Doutor, José de Ribamar Silva Oliveira
Co-orientador: Mestre, Ricardo Alexsandro de Medeiros Valentim

NATAL-RN
2007

Dedico este trabalho aos meus pais por
terem me dado tão boa educação
e sempre me apoiado em tudo o que
sonhei em minha vida.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por ter me dado todas as condições necessárias para o desenvolvimento desse trabalho e ter me rodeado de pessoas maravilhosas as quais também me ajudaram nessa tarefa. Agradeço a minha família por sempre cuidar de tudo o que eu preciso, me deixando manter concentrado nos estudos sem precisar se preocupar com outros problemas. Agradeço aos meus orientadores, José de Ribamar e Ricardo Valentim, por terem sempre tão boa vontade, me guiado e estado sempre disponível quando precisei.

"O único lugar onde o sucesso vem antes do trabalho é no dicionário" (A. Einstein)

RESUMO

O padrão Ethernet é a principal tecnologia de interconexão de redes locais, porém ainda não foi estabelecido como padrão para redes industriais porque o seu método de acesso ao meio, o CSMA/CD, não atende aos requisitos temporais exigidos por esse tipo de rede. Esse trabalho tem como objetivo desenvolver um modelo de troca de dados para redes industriais, denominado *Extensible Real-Time Publish Subscribe*. Baseando suas mensagens em XML, esse modelo torna-se independente de plataforma. Além disso, é objetivo desse trabalho desenvolver um *middleware* para abstrair todos os detalhes de comunicação desse modelo.

Palavras-chave: RTPS. Redes industriais. Tempo real. Ethernet.

ABSTRACT

The Ethernet standard is the main technology of local internetwork, however not yet it was established as standard for industrial network because its method of access to the way, the CSMA/CD, does not take care of to the secular requirements demanded by this type of net. This work if considers to develop a model of exchange of data for industrial nets, called Extensible Real-Teams Publish Subscribe. Basing its messages in XML, this model one becomes independent of platform. Moreover, middleware is objective of this work to develop one to abstract all the details of communication of this model.

Key words: RTPS. *Industrial network*. *Real time*. Ethernet.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS

CEFET-RN	Centro Federal de Educação Tecnológica do Rio Grande do Norte
CLP	Controlador Lógico Programável
CORBA	Common Object Request Broker Architecture
COSMIC	Cooperating Smart Devices
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DCOM	Distributed Component Object Model
FIFO	First In First Out
GC	Grupo de Controle
H-BEB	High Priority Binary Exponential Backoff
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
LAN	Local Area Network
NDDS	Network Data Delivery Service
ORTE	Open Real-Time Ethernet
OSI	Open Systems Interconnection
PS	Publish-Subscribe
RFC	Request For Comments
RN	Rio Grande do Norte
RTI	Real-Time Innovations
RTPS	Real-Time Publish-Subscribe
RTT	Round-Trip Time
SC	Serviço de Comunicação
TCC	Trabalho de Conclusão de Curso
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VTPE	Virtual Token-Passing Ethernet
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XRTPS	Extensible Real-Time Publish-Subscribe

SUMÁRIO

1 INTRODUÇÃO

Um fator recorrente das redes de escritório é a busca por maior desempenho, ou seja, redes que executem suas tarefas no menor tempo possível. Porém, para redes industriais o desempenho médio não é o fator mais relevante, isso porque o desempenho de um sistema de tempo real está diretamente relacionado com a previsibilidade imposta pelo ambiente (STANKOVIC, 1988).

Enquanto as redes de escritório são orientadas a obter o melhor desempenho médio possível, as redes industriais são orientadas a requisitos temporais, dado que o fator de cumprimento das metas temporais é imperativo em muitas aplicações de tempo real, ou seja, o determinismo (FARINES ET. AL., 2000). A Figura 1 ilustra um exemplo hipotético que demonstra a importância dos requisitos temporais numa rede industrial.

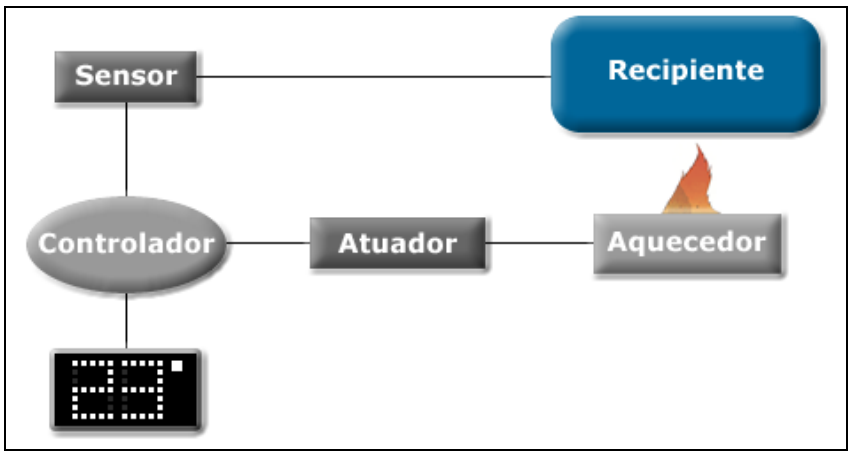

Figura 1 - Exemplo hipotético.
Fonte: Valentim (2006)

O sistema hipotético ilustrado na Figura 1 possui um recipiente que deve ser mantido a uma determinada temperatura, a variação demasiada para mais ou para menos poderá causar um desastre. A temperatura do recipiente é monitorada por um sensor, que captura essa grandeza entre um dado intervalo de tempo. Esse sensor envia os dados de temperatura para um *display* e para um controlador lógico programável (CLP), o primeiro servirá de interface humana, onde um operador poderá ver a temperatura atual do recipiente. O CLP é um dispositivo de processamento de dados, que neste caso lê a temperatura no sensor e de acordo com o valor obtido,

interage com um atuador para determinar a qual a incidência de calor deve ser propagado sobre o recipiente.

A necessidade dos requisitos de tempo real ocorre em função dos dados capturados pelo sensor, que devem chegar ao atuador em tempo hábil (não deve perder os *deadlines*), a fim de que ele possa garantir a temperatura mais adequada ao recipiente.

Existem diversos protocolos de comunicação de tempo real que usam o padrão Ethernet: NDDS (2002) (*Network Data Delivery Service*), que é uma implementação comercial do RTPS (2002) (*Real-Time Publish-Subscribe*). O ORTE (SMOLIK ET. AL., 2003) (*Open Real-Time Ethernet*) é uma alternativa de código-aberto ao NDDS. O PowerLink, desenvolvido pela companhia Bernecker&Rainer, é outra solução proprietária de protocolo de tempo real, este foi desenvolvido sobre a camada de enlace de dados (DOLEJS, 2004).

O uso de aplicações de tempo real é facilmente observado em diversas áreas, o que há de comum entre elas é que exigem maiores restrições de tempo do que outras. Alguns exemplos de aplicações de tempo real são: sistemas de defesa militar, sistemas de controle de plantas industriais (químicas e nucleares), controle de tráfego (aéreo e ferroviário), sistemas de monitoramento de pacientes em hospitais, sistemas embarcados em robôs e veículos (automóveis, aviões e sondas espaciais), videogames, aplicações de teleconferência e aplicações de multimídia em geral (FARINES ET. AL., 2000).

Neste contexto, o estudo sobre aplicações de tempo real segue em várias ramificações de pesquisas, como por exemplo: sistemas operacionais de tempo real, sistemas distribuídos de tempo real e bancos de dados de tempo real.

No contexto das aplicações distribuídas de tempo real existem pesquisas relacionadas ao mecanismo de controle de acesso ao meio, por exemplo: H-BEB (*High Priority Binary Exponential Backoff*) (MORAES, 2005), VTPE (*Virtual Token-Passing Ethernet*) (CARREIRO, 2005), FTT-Ethernet (PEDREIRAS, 2005), e estudos relacionados a métodos de troca de dados, como por exemplo, o PS (*Publish-Subscribe*) (THOMESSE, 2005) e o RTPS (*Real-Time Publish-Subscribe*) (DOLEJS, 2004).

O padrão Ethernet (IEEE 802.3, 1990) é uma tecnologia de interconexão para redes locais (LAN, *Local Area Network*) amplamente utilizada nas redes de escritório, esta tecnologia é bastante atraente para redes industriais pelo seu alto desempenho e,

em virtude de sua larga escala de produção, os produtos destinados à rede Ethernet são de baixo custo (BRITO ET. AL., 2004). Porém, o método de troca de dados CSMA/CD do padrão Ethernet não é determinístico, essa falta de determinismo do CSMA/CD dificulta a predição do tempo para a entrega dos pacotes, o que não pode ocorrer em sistemas de tempo real (VALENTIM, 2006).

O *Publish-Subscribe* é um método de troca de dados que permite o envio e/ou recebimento assíncrono de mensagens (ou eventos) entre máquinas (THOMESSE, 2005). Nesse modelo existem duas entidades:

- *Publishers* (publicadores) são responsáveis pelo envio de algum tipo de dado;
- *Subscribers* (subscritores) são nós de rede que receberão esses dados os quais tenham interesse em consumir.

Uma característica provida pelo modelo PS é o desacoplamento entre as entidades. Isso acontece porque a comunicação entre os nós da rede ocorre de forma anônima, (THOMESSE, 2004).

O método PS tem sido utilizado no contexto das aplicações de tempo real, porém é possível observar esses sistemas requerem mais funcionalidade do que o fornecido pela tradicional semântica *Publish-Subscribe*, já que este não contempla os requisitos temporais necessários. É neste ínterim, que o Real-Time Publish-Subscribe (RTPS) adiciona parâmetros e propriedades de sincronismo de tempo real (DOLEJS, 2004). Deste modo, provendo aos desenvolvedores de aplicações a possibilidade de controlar tipos diferentes de fluxos de dados, conseqüentemente alcançando os objetivos de desempenho e de confiabilidade impostas pelos requisitos das aplicações (DOLEJS, 2004).

Diante do exposto, este trabalho tem como objetivo elaborar um modelo de troca dados baseado no RTPS, porém, sendo mais flexível quanto à:

- Dependência de plataforma de desenvolvimento;
- Sistema operacional;
- Paradigma de programação.

O modelo de troca de dados proposto neste trabalho é o *Extensible Real-Time Publish-Subscribe* (XRTPS). O XRTPS é uma proposta mais simples, pois envolve uma semântica mais clara para descrever as mensagens, bem como, as propriedades pertinentes ao tempo real. A simplicidade e a interoperabilidade do XRTPS se

fundamentam no XML (*Extensible Markup Language*) (W3C, 2004) - por ser um padrão mundial, qualquer plataforma de desenvolvimento é capaz de manipulá-lo.

XRTPS define um conjunto *tags* as quais permitem construir mensagens para realizar a troca de mensagens entre *publishers* e *subscribers*. Assim, incorporando as facilidades da padronização do XML estabelecidas pela W3C. Além do XRTPS, este trabalho também desenvolverá um *middleware* que, fazendo uso desse modelo (XRTPS), irá abstrair os detalhes que envolvem os mecanismos de troca de dados baseados no RTPS.

1.1. Objetivos

Esta seção expõe os objetivos gerais e específicos desse trabalho.

1.1.1. Objetivo Geral

O objetivo geral deste trabalho é elaborar um método de troca de dados baseado no RTPS para redes industriais, o qual contemple os requisitos específicos deste tipo de rede, tais como determinismo no tempo de resposta e os requisitos críticos de tempo real. As mensagens desse modelo são formatadas em XML, que por ser um padrão mundial estabelecido pela W3C, é capaz de prover independência de plataforma e fácil integração.

É objetivo deste trabalho também desenvolver um *middleware* capaz de abstrair todos os detalhes de implementação para comunicação de rede utilizando o modelo XRTPS.

1.1.2. Objetivos Específicos

Os objetivos específicos deste projeto de pesquisa são:

- Pesquisar e analisar soluções similares;
- Elaborar o modelo de troca de dados;
- Especificar as mensagens de troca de dados;
- Modelar *middleware*;
- Desenvolver o *middleware* baseado no modelo;
- Realizar testes.

1.2. Metodologia

A metodologia adotada por esse trabalho se divide em duas vertentes: a teórica e a experimental. A primeira consiste em adquirir conhecimento necessário para desenvolver o objeto de estudo desse trabalho, a fim de que se possam atingir as metas desejadas. A partir dessa vertente, foi elaborada uma pesquisa para formar uma base de conhecimento sólida a fim de subsidiar o desenvolvimento efetivo do objeto de estudo.

A vertente experimental consiste em modelar e desenvolver o XRTPS.

1.3. Estrutura do trabalho

Este trabalho encontra-se dividido em mais três capítulos, aos quais são descritos a seguir:

O Capítulo 2 inicia com uma abordagem dos principais trabalhos relacionados ao objeto de estudo aqui proposto. Em seguida são apresentados mecanismos de comunicação e os principais métodos de troca de dados existentes. Por fim, o capítulo descreve os principais conceitos e aspectos acerca da computação de tempo real.

O Capítulo 3 apresenta o *Extensible Real-Time Publish-Subscribe*, que é o objeto de estudo desse trabalho. O capítulo descreve o mecanismo de comunicação do middleware, os serviços que ele disponibiliza, a configuração e, por fim, os resultados obtidos em experimentos.

O Capítulo 4 apresenta as conclusões e os possíveis trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são abordadas todas as pesquisas realizadas para formar uma base de conhecimento necessária para entendimento do objeto de estudo proposto por esse trabalho. Para tanto, foram subdivididas quatro seções:

2.1. O Estado da Arte: Essa seção descreve as principais em desenvolvimento atualmente que tem relação com este trabalho;

2.2. Métodos de transmissão de dados: Aborda alguns mecanismos de propagação de dados via rede, dando maior ênfase naquele que é utilizado neste trabalho;

2.3. Métodos de troca de dados: Descreve alguns métodos de troca de dados;

2.4. Computação de tempo real: Esta seção aborda os principais conceitos de uma aplicação de tempo real.

2.1. O Estado da Arte

A primeira das soluções existentes é o *Network Data Distribution Service* (NDDS), desenvolvido pela Real-Time Innovations (RTI). O NDDS é uma implementação comercial, portanto de código fechado, do Real-Time Publish-Subscribe (Dolejs, 2004), que desta forma, usa o modelo PS para comunicação entre publicadores e subscritores.

O NDDS é um *middleware* que, assim como qualquer outro, se situa entre a aplicação e o sistema operacional. É uma camada de software que fica no topo da pilha que simplifica a complexibilidade da comunicação de rede (NDDS, 2005). Dolejs (2003) ainda explica que o NDDS utiliza o protocolo UDP/IP, e é baseado na arquitetura publicador-subscritor, tendo em vista que o NDDS implementa o modelo RTPS. No entanto, o NDDS é uma solução proprietária e dependente da plataforma a qual ela foi desenvolvida.

Outra implementação do RTPS é o *Open Real-Time Ethernet* (ORTE), porém esta possui seu código aberto. O ORTE foi desenvolvido em C pela Universidade Tecnológica da República Tcheca em Praga, é um dos resultados do projeto OCERA (2003). Apesar de ser uma implementação de código aberto, o ORTE é dependente da plataforma de desenvolvimento, tendo em vista que foi desenvolvido em C.

Conforme descrito por Kaiser et. al. (2005), o COSMIC é um *middleware* que abstrai aspectos de comunicação de baixo nível seguindo os requisitos de tempo real,

ou seja, conforme explica Silva et. al. (2006), o COSMIC permite que o desenvolvedor da aplicação possa se ater às regras de negócio do sistema, deixando que o *middleware* se responsabilize pela comunicação de dados em tempo real.

Contudo, Silva et. al. (2006) afirma que o COSMIC não é capaz de priorizar as mensagens, deixando assim de atender a um dos requisitos de tempo real. Valentim (2006) ainda conclui que, por ser tratar de um *middleware* que utiliza o mecanismo de troca de dados do tipo *unicast*, o COSMIC ainda não atingiu um nível de maturidade tecnológica exigida pelas aplicações de tempo real da automação industrial.

2.2. Métodos de Transmissão de Dados

Nesta subseção são abordadas as principais características dos métodos de transmissão de dados *unicast*, *multicast* e *broadcast*.

O *unicast* é um método de transmissão de dados onde um nó transmissor da rede envia os dados para apenas um nó destinatário, dessa forma caracterizando uma conexão ponto-a-ponto. Para tanto, o nó transmissor simplesmente transmite os dados para o IP do nó destinatário (Tanenbaum, 2003).

O *broadcast*, por sua vez, é um método de transmissão de dados no qual um nó transmissor envia os dados para todos os nós da rede. Para realizar essa transmissão, ele envia os dados para o endereço IP de *broadcast* da rede, que sempre é o último endereço IP da rede (Tanenbaum, 2003).

O *broadcast* pode ser útil quando um nó precisa encontrar uma informação sem saber em qual nó da rede ele pode encontrá-la, ou quando um nó precisa enviar um dado para uma grande quantidade de nós (Mogul, 1984).

O *multicast* é a transmissão de um datagrama para um grupo de zero ou mais nós destinatários da rede, tal grupo é identificado por um único endereço IP. Por exemplo, alguns nós estão associados a um determinado endereço *multicast* (grupo). Quando um pacote é transmitido para esse endereço, o mesmo é entregue a todos os destinatários que estão associados ao grupo (Deering, 1985).

Algumas características importantes sobre o *multicast* devem ser levadas em consideração:

- A associação de um nó para um grupo é dinâmica, ou seja, os nós podem ingressar ou abandoná-lo a qualquer momento;
- Não há restrições quanto ao número de nós participantes;
- Não há restrições quanto à posição geográfica dos nós;

- Um nó pode participar de um ou mais grupos;
- Um nó não precisa participar de um grupo para enviar dados para ele;

2.3. Métodos de Troca de Dados

Outro ponto importante para as redes industriais são os métodos de troca de dados, pois estes também devem garantir os requisitos temporais exigidos pelas aplicações de automação industrial (Valentim, 2006). Nesta seção serão tratados os principais métodos: cliente-servidor (Thomesse, 2005), produtor-consumidor (Lopez, 2000), publicador-subscritor (Thomesse, 2005) e *Real-Time Publish-Subscribe* (Dolejs, 2004).

2.3.1. Cliente-Servidor

Nesse método há uma interação entre dois nós: o cliente e o servidor. O nó servidor provê algum serviço que será de interesse dos nós clientes. Ele ficará aguardando solicitações as quais serão respondidas por este serviço (Valentim, 2006).

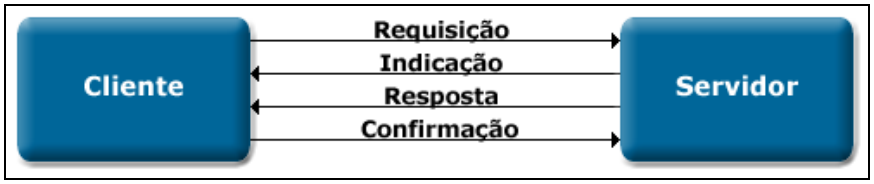

Figura 2 - Comunicação cliente-servidor
Fonte: Autoria própria

Segundo Thomesse (2005), e como ilustrado na Figura 2, o processo de comunicação entre os clientes e o servidor consiste em quatro etapas: requisição, indicação, resposta e confirmação.

- A requisição é a solicitação que o cliente faz ao servidor. Esta etapa inicia a conexão entre as duas entidades;
- A indicação é a resposta do servidor para o cliente que significa o recebimento da requisição;
- A resposta é o resultado da solicitação requisitada pelo cliente;
- A confirmação é a mensagem que indica o recebimento da resposta pelo cliente.

2.3.2. Produtor-Consumidor

Lopez (2000), explica que esse modelo usa um conjunto de *buffers* entre as estações da rede. Cada *buffer* corresponde a uma variável da aplicação e é identificado dentro do grupo da aplicação por um rótulo lógico. Cada *buffer* aguarda que seus dados sejam enviados via rede ou serem usados pela aplicação.

Nesse contexto, são observados três elementos interagindo entre si: o produtor, a rede e o consumidor.

- Produtor: Deposita o novo valor em um *buffer* de transmissão;
- Rede: Copia o conteúdo do *buffer* de transmissão do produtor para o *buffer* de recepção do consumidor;
- Consumidor: Captura o valor contido no seu *buffer* de recepção.

2.3.3. Publicador-Subscritor

Segundo Thomesse (2005), nesse tipo de comunicação há uma interação entre uma entidade denominada *publisher*, responsável pela publicação de dados, e um ou mais *subscribers*, que são entidades consumidoras. Esse modelo pode ser dividido em dois submodelos, o *pull* e o *push*.

- *Pull*: O *publisher* recebe um pedido de um gerente de publicação remoto para publicar, logo após envia sua resposta via *broadcast* ou *multicast* pela rede (Valentim, 2006);
- *Push*: Nesse caso, dois modelos podem ser usados: o confirmado, o qual o *subscriber* solicita uma resposta obrigatória ao *publisher*, e o não-confirmado, usado pelo *publisher* para distribuir a informação aos *subscribers*.

Castellote (2002) aponta algumas das vantagens do modelo *publish-subscribe* para aplicações de tempo real:

- PS é mais eficiente do que os objetos distribuídos, como DCOM e CORBA, pois os *subscribers* não precisam enviar mensagens de requisição aos *publishers*;
- O PS suporta conexões do tipo muitos-para-muitos e grupos de subscritores. Também serve perfeitamente para ambientes dinâmicos com tolerância a falhas que possuem grande quantidade de nós;

- Usado principalmente em protocolos não orientados a conexão, a fim de que não haja perda de desempenho com o tradicional *handshake* do TCP, e usa a tecnologia *multicast* para enviar dados a diversos *subscribers* simultaneamente.

2.3.4. Real-Time Publish-Subscribe

Castellote (2002) ainda acrescenta que as aplicações de tempo real necessitam de maiores funcionalidades do que as providas pela tradicional semântica do PS, tais como:

- Controle de tempo de entrega: As aplicações de tempo real precisam saber quando determinados dados serão entregues e por quanto tempo eles são válidos;
- Confiabilidade: Cada aplicação de tempo real precisa especificar quais são suas características de confiabilidade;
- Tolerância a falhas: A comunicação não deve possuir nenhum ponto de falha;
- Degradação seletiva: Cada canal de comunicação deve estar protegido dos outros, ou seja, o desempenho de um canal não deve ser afetado pelos outros.

O *Real-Time Publish-subscribe* (RTPS) surge nesse contexto para suprir as necessidades das aplicações de tempo real. Segundo Castellote (2002), o RTPS é um protocolo que foi desenvolvido para distribuição anônima de publicações, utilizando troca confiável (TCP) ou não-confiável (UDP) de mensagens para múltiplos *subscribers*.

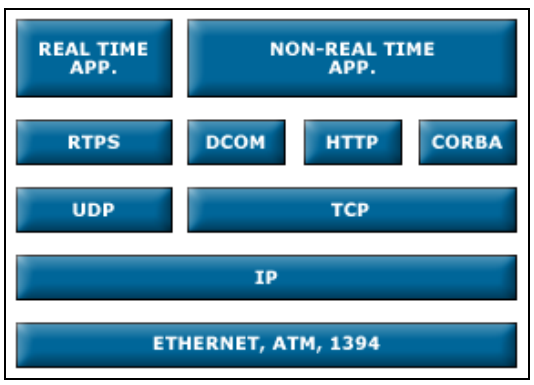

Figura 3 - Pilha de protocolos.
Fonte: Georgoudakis et. al. (2003)

Conforme é exibido na Figura 3, a arquitetura RTPS é executada sobre protocolo UDP (RFC 768 Postel, 1980), pelo fato do UDP ser muito mais rápido do que o TCP.

Kurose e Ross (2003) explicam que nas conexões UDP não há o tradicional *handshake* de três vias antes de começar a transferência de dados, como acontece no TCP. Assim, ele não introduz nenhum atraso para estabelecer a conexão. Outro aspecto que torna o UDP é que este protocolo tem um *overhead* menor em relação ao TCP, (Tanenbaum, 2003).

O TCP também possui um mecanismo de controle de congestionamento que limita o remetente quando um ou mais enlaces entre o remetente e os destinatários ficam excessivamente congestionados (Tanenbaum, 2003). Essa limitação pode causar significativo impacto sobre aplicações em tempo real, que podem tolerar alguma perda de pacotes, mas exigem uma taxa mínima de envio (Kurose e Ross, 2003).

Dolejs (2004) explica que o RTPS adiciona parâmetros e propriedades de sincronismo do produtor e do consumidor de modo que o desenvolvedor da aplicação possa controlar tipos diferentes de fluxos de dados e conseqüentemente alcançando os objetivos de desempenho e de confiabilidade da aplicação.

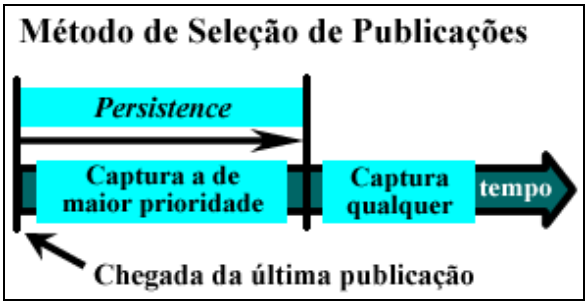

Figura 4 - Parâmetros de publicação.
Fonte: Dolejs (2004)

Entre os parâmetros de publicação (veja Figura 4), têm-se:

- Tópico (*topic*) e o tipo (*type*), que juntos identificam uma publicação específica;
- Prioridade (*strength*): É o peso relativo da publicação comparado com outras publicações do mesmo tópico e tipo. Mensagens com maior prioridade são publicadas primeiro;

- Persistência (*persistence*): Especifica por quanto tempo uma publicação é válida. Durante o decorrer do tempo de persistência, o consumidor processa (consome) a primeira publicação recebida.

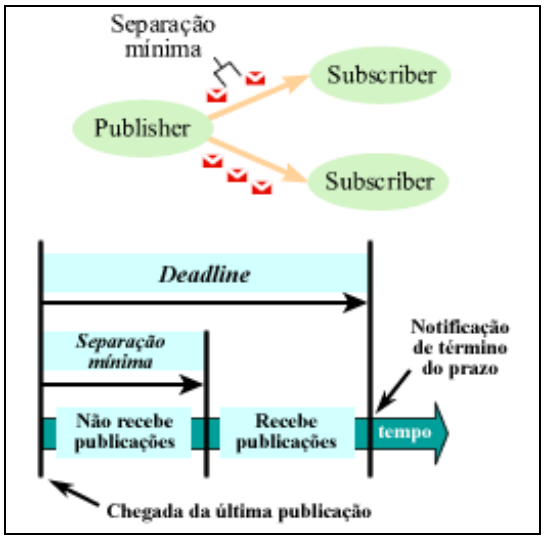

Figura 5 - Parâmetros de subscrição.
Fonte: Dolejs (2004)

Entre os parâmetros de subscrição (veja Figura 5), têm-se:

- Tópico (*topic*) e o tipo (*type*), que juntos identificam uma publicação específica;
- Tempo mínimo de separação (*minimum separation time*): Nenhuma nova publicação é aceita durante esse intervalo de tempo;
- Prazo (*deadline*): Especifica por quanto tempo uma publicação é válida. O usuário pode ajustar todos os parâmetros para se adequar às exigências de sua aplicação.

Segundo Georgoudakis et. al. (2003), o *publisher* não precisa saber quantos nós estão interessados em receber suas publicações, e nem seu local na rede, uma vez que os dados são enviados via *broadcast*. Vale salientar que cada publicação é identificada por um tópico e um tipo, essas informações serão usadas pelos *subscribers* para distinguir dados que lhe interessam daqueles que podem ser descartados.

2.4. Computação de Tempo Real

A computação de tempo real desempenha um papel vital na sociedade moderna, exemplos de aplicações para ela incluem o controle de experimentos laboratoriais, controle de motores e dispositivos de automóveis, sistemas de controle de comando, instalações nucleares, sistemas de controle de tráfego aéreo e robótica. E diferentemente da computação convencional, ela não apenas depende da corretude do resultado lógico, mas também o tempo em que esses resultados foram produzidos (Stankovic, 1988).

Para Stankovic (1988), a propriedade mais importante dos sistemas de tempo real não é o desempenho, e sim a previsibilidade. Segundo ele é um grande erro pensar que a computação de tempo real é equivalente a computação performática, ou seja, que ela busca diminuir o tempo médio de resposta de um conjunto de tarefas. Contudo, o objetivo da computação de tempo real é conhecer o sincronismo de cada tarefa. A computação rápida pode ajudar a concluir as tarefas num tempo menor, mas não garante que elas serão concluídas em tempo hábil.

Um sistema de tempo real é dito previsível quando, independente das variações que ocorram (de carga ou de falhas), o comportamento desse sistema pode ser predito, ou seja, antecipado, antes de sua execução. Para isso, é necessário definir um conjunto de hipóteses sobre o comportamento do ambiente ao qual o sistema irá residir, a fim de conferir ao sistema a capacidade de operar atendendo aos requisitos temporais mesmo na pior hipótese.

- A hipótese de carga: Determina a carga computacional máxima gerada pelo ambiente em um intervalo mínimo de tempo;
- A hipótese de falhas: Descreve as falhas que o sistema deve suportar para que continue atendendo aos requisitos funcionais e temporais.

Portanto, um sistema será dito previsível, se ele continuar a atender suas funcionalidades e aos requisitos temporais, mesmo operando sobre a pior hipótese possível (carga máxima e falhas). Mas a garantia de previsibilidade não depende apenas do atendimento dessas hipóteses, um conjunto de fatores ligados a arquitetura de hardware, de sistema operacional e de linguagens de programação também são importantes, pois eles também devem garantir previsibilidade (Farines et. al., 2000).

3 EXTENSIBLE REAL-TIME PUBLISH-SUBSCRIBE

Este capítulo tem como objetivo explicar sucintamente o objeto de estudo desse trabalho. O capítulo está organizado da seguinte forma:

3.1. Comunicação: descreve como funciona a comunicação entre os nós da rede;

3.2. Serviços do *middleware*: apresenta os dois serviços disponíveis pelo *middleware*: o Serviço de Eleição e o Serviço de Comunicação;

3.3. Configuração do *middleware*: descreve como configurar o *middleware*;

3.4. Resultados experimentais: apresenta os resultados obtidos em testes realizados com o XRTPS.

3.1. Comunicação

O modelo XRTPS baseia-se no método de troca de *Publish-Subscribe*, no qual participam nós publicadores (*publishers*), que são responsáveis por publicar dados para algum tópico, e nós subscritores (*subscribers*), que se inscrevem em um tópico a fim de consumir mensagens relacionadas aos tópicos os quais estão inscritos.

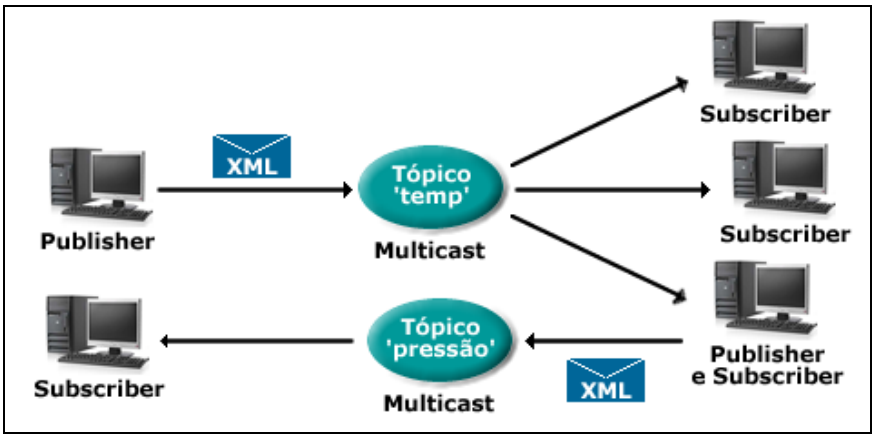

Figura 6 - Arquitetura do XRTPS.
Fonte: Autoria própria

Conforme ilustrado na Figura 6, o mecanismo de transmissão de dados é o *multicast*. Isso permite que cada tópico da aplicação esteja associado com um grupo *multicast*. Portanto, quando um nó ingressa em um determinado tópico, ele estará se associando a um grupo que está relacionado com o respectivo tópico. Uma

característica desta estratégia é que um mesmo nó pode ser publicador e subscritor simultaneamente.

O *middleware* de cada nó da rede possui um arquivo de configuração que associa os tópicos, aos quais ele está inscrito, aos respectivos grupos *multicast*. Portanto, quando um publicador deseja enviar um dado, ele sabe para qual grupo o mesmo deve ser endereçado. Da mesma forma, todos os *subscribers* saberão a quais grupos se associarem para receber as publicações desejadas.

O uso de *multicast* se justifica pelo ganho de desempenho, tendo em vista que somente uma conexão é aberta e apenas os subscritores interessados (inscritos no grupo *multicast*) receberão a publicação. Deste modo, otimizando o fator de utilização do meio físico.

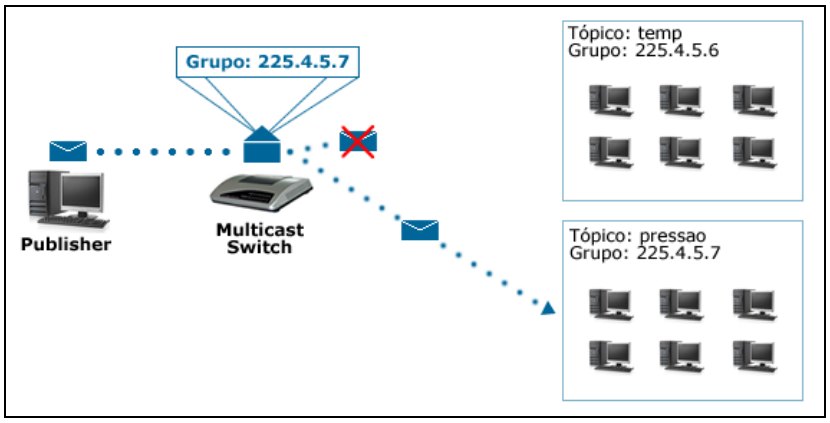

Figura 7 - Transmissão via *Multicast*.
Fonte: Autoria própria

Conforme ilustrado na Figura 7, caso os *switchs* da rede implementem *multicast*, eles filtrarão as mensagens antes que elas cheguem aos nós destinatários, diminuindo consideravelmente a carga de transmissão da rede (*overhead*). Isso é possível porque o *switch* é capaz de realizar um cálculo *hash* de acordo com os IP's do grupo *multicast* e do nó destinatário, determinando se o nó está inscrito no grupo ou não (Stevens et. al., 2005). Portanto, caso essa situação aconteça, os *switchs* da rede saberão quais mensagens cada nó deve receber.

Caso os *switchs* da rede não implementem *multicast*, todas as publicações chegarão aos nós destinatários, porém, a placa de rede (nível três) fará a filtragem das mensagens, permitindo passar apenas aquelas que estão endereçadas para o grupo *multicast,* o qual o nó faz parte. Deste modo, diminuindo o processamento na pilha de

protocolos (Modelo OSI), pois as mensagens não precisarão chegar às camadas superiores para verificar se elas realmente são de interesse do nó. A Figura 8 ilustra esse cenário.

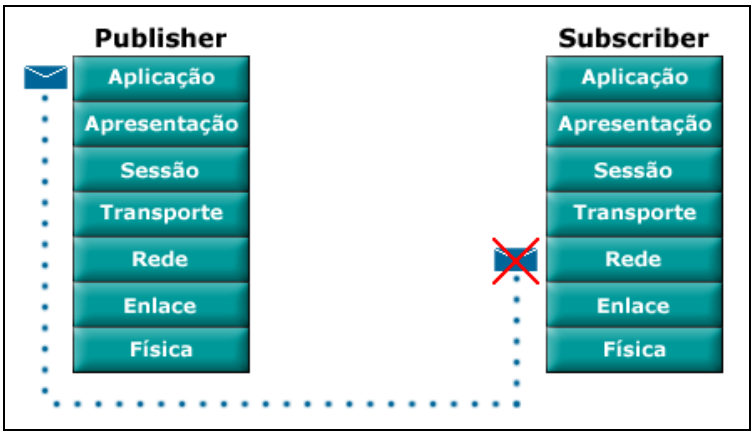

Figura 8 - Processamento na pilha de protocolo (modelo OSI).
Fonte: Autoria própria

3.1.1. Formato das Mensagens

Conforme ilustrado na Figura 6, as mensagens trocadas entre os nós da rede são formatadas utilizando a linguagem de marcação XML. A Figura 9 ilustra o XML Schema (2001) no qual define os dados e os tipos que compõem as *tags* das mensagens.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

	<xs:element name="message">
		<xs:complexType>
			<xs:element name="topic" type="xs:string" />
			<xs:element name="type" type="xs:string" />
			<xs:element name="priority" type="xs:integer" />
			<xs:element name="persistence" type="xs:integer" />
			<xs:element name="separation" type="xs:integer" />
			<xs:element name="deadline" type="xs:integer" />
			<xs:element name="body" type="xs:string" />
		</xs:complexType>
	</xs:element>

</xs:schema>
```

Figura 9 - XML Schema da mensagem
Fonte: Autoria própria

Toda mensagem inicia com a *tag* <message> e indica a abertura da mensagem (nó raiz). Em seguida são especificadas todas as outras *tags*:

- *Topic:* Dado do tipo texto que especifica o tópico da mensagem;
- *Type*: Dado do tipo texto que especifica o tipo da mensagem;
- *Priority*: Dado do tipo inteiro e representa a prioridade da mensagem;
- *Persistence*: Dado do tipo decimal que representa, em milissegundos, por quanto tempo uma mensagem pode esperar para ser publicada;
- *Deadline*: Dado do tipo decimal e indica, em milissegundos, por quanto tempo uma mensagem é válida. Esse tempo é somado ao tempo médio de envio das mensagens pela rede;
- *Body*: Dado do tipo texto e representa o corpo da mensagem.

A Figura 10 exemplifica uma mensagem do modelo XRTPS.

```
<?xml version='1.0'?>
<message>
      <topic>temperature</topic>
      <type>sensor</type>
      <priority>5</priority>
      <persistence>500</persistence>
      <separation>300</separation>
      <deadline>3000</deadline>
      <body>243</body>
</message>
```

Figura 10 - Exemplo de mensagem.
Fonte: Autoria própria

3.2. Serviços do Middleware

O XRTPS Middleware oferece serviços às camadas superiores, cada um deles com sua função específica. Existem dois serviços: o Serviço de Comunicação e o Serviço de Eleição, ambos são descritos abaixo.

3.2.1. Serviço de Comunicação

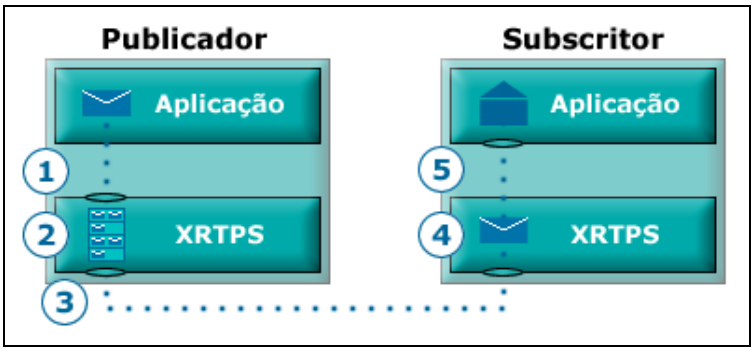

Figura 11 - Processo de comunicação do middleware.
Fonte: Autoria própria

A Figura 11 ilustra o funcionamento do Serviço de Comunicação.

1. O *middleware*, ao ser inicializado, carrega o Serviço de Comunicação (SC), o qual uma de suas primeiras tarefas é de abrir um soquete UDP para cada grupo ao qual ele está inscrito como publicador. Quando a aplicação (que está acima do *middleware*) quiser enviar uma publicação, ela deverá enviar para a porta correspondente do grupo. A inicialização do SC ainda envolve o ingresso num grupo *multicast* especial, chamado de Grupo dos Publicadores;
2. Ao receber uma mensagem da aplicação, o SC captura o *timestamp* atual e o associa à mensagem, esse dado será importante no momento do envio da publicação e na definição de sua prioridade. Feito isso, ela é armazenada numa fila, que é organizada a partir da prioridade determinada na mensagem, caso duas mensagens tenham a mesma prioridade, o *timestamp* de chegada será usado para definir quem será publicada primeiro (FIFO);
3. O SC ainda é responsável por enviar todas as publicações que estejam na fila de prioridade. Contudo, antes de enviar a publicação, é necessário verificar se o tempo de persistência foi expirado, caso tenha sido, a publicação é descartada.

Caso não, o *deadline* é subtraído pelo tempo que passou desde que a mensagem chegou ao publicador, e também do tempo médio de entrega das mensagens, afim de que, quando a publicação chegue ao subscritor, o valor do *deadline* seja exatamente o tempo que ele tem para consumi-la. Feito isso, a publicação é enviada para o grupo *multicast* correspondente, destinada para a Porta de Comunicação Padrão;

4. O SC também é responsável por abrir soquetes UDP para escutar a Porta de Comunicação Padrão, que é utilizada pelo *middleware* para troca de mensagens entre os nós;
5. Quando a publicação é recebida, ela é enviada imediatamente para a aplicação, para a Porta de Envio de Publicações, a qual a aplicação deve monitorar a fim de receber as mensagens.

3.2.2. Serviço de Eleição

Para o correto funcionamento do Serviço de Comunicação, é necessário que todos os nós saibam o Tempo Médio de Entrega das Publicações. Esse é o intervalo de tempo necessário para que uma determinada mensagem seja transferida de um publicador para um subscritor. Isso se faz necessário uma vez que os relógios dos nós da rede não estão sincronizados, então não seria possível um nó subscritor saber se determinada publicação seria válida (se não teria expirado o *deadline*).

O cálculo desse tempo é realizado por dois nós especiais, denominados nós Mestre e Escravo. Eles são definidos pela ordem de ingresso na rede, o primeiro nó a entrar na rede será o mestre, e o segundo será o escravo, os nós seguintes são chamados de nós regulares.

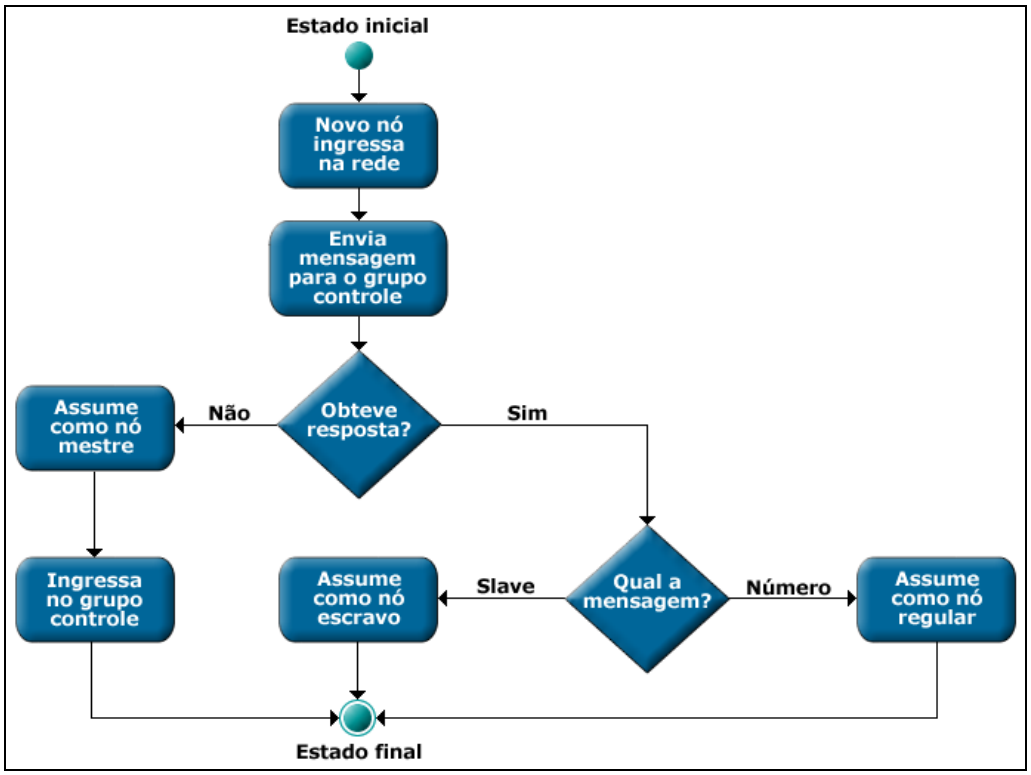

Figura 12 - Eleição dos nós mestre e escravo.
Fonte: Autoria própria

Conforme demonstrado no fluxograma da Figura 12, quando um nó ingressa na rede, ele envia uma mensagem de controle para um grupo *multicast* especial chamado Grupo de Controle (GC), caso ele não obtenha nenhuma resposta, o nó assumirá como sendo o mestre e ingressará no GC. Caso ele receba uma resposta, ou seja, a resposta do nó mestre, ele irá verificar a mensagem contida nela. Existem duas possíveis respostas, “slave” ou um valor numérico. Caso a resposta seja “slave”, o nó assumirá como sendo escravo e ingressará no GC. Caso a resposta seja um valor numérico, ele assumirá como nó regular da rede, não tendo a responsabilidade de realizar o cálculo de tempo médio para entrega das mensagens. Esse valor numérico é o Tempo Médio de Entrega das Publicações em milissegundos.

Esta estratégia reforça ainda mais o uso do *multicast,* pois contribui significativamente para otimizar o fator de utilização da rede, tendo em vista que sempre serão necessárias, no máximo, duas mensagens de controle para eleger um nó (uma requisição e uma resposta). Se fosse utilizado *broadcast*, o fator de utilização seria baixo, pois seriam necessárias N+1 mensagens para eleger um nó, onde N é o número de nós da rede (uma mensagem a mais que seria a resposta do mestre).

Por exemplo, se houvessem cem nós na rede, o uso do *broadcast* iria causar grande *overhead*, pois cento e uma mensagens de controle estariam trafegando (sem requisições e uma resposta). Com o mesmo número de nós, o uso do *multicast* diminuiria esse overhead em 98%, pois apenas duas mensagens de controle seriam necessárias. Um aspecto relevante que deve ser observado na transmissão *broadcast* é que, quanto maior a rede, maior o *overhead* que o *broadcast* causaria, enquanto o *multicast* sempre precisará de no máximo duas mensagens.

3.2.2.1. Cálculo e Publicação do Tempo Médio de Entrega das Mensagens

Para calcular o tempo médio de entrega das publicações, o nó mestre envia 100 mensagens UDP para o nó escravo, este irá respondê-las à medida que recebe. De posse do intervalo de tempo que cada mensagem levou para ser enviada do nó mestre ao nó escravo e voltar (RTT – *Round-Trip Time*), o nó mestre realiza o cálculo somando todos esses valores e dividindo pelo número de mensagens enviadas, depois divide por dois para obter apenas o tempo de ida (veja Equação 1).

$$Te = \frac{\left(\sum_{i=1}^{n} m \Big/ n \right)}{2}$$

Equação 1 - Cálculo do tempo médio de entrega das mensagens.
Fonte: Autoria própria

Onde,

Te é o tempo de entrega,

m é o valor do tempo de RTT e

n é o número de mensagens enviadas.

De posse do valor do tempo médio de entrega das mensagens, o nó mestre envia esse valor para o Grupo dos Publicadores, o qual fazem parte todos os nós publicadores da rede.

O processo de cálculo do tempo médio de entrega é repetido num intervalo de tempo definido no arquivo de configuração do *middleware*. Esta estratégia tem objetivo

de realizar ajustes periódicos no tempo médio de entrega, haja vista que este tempo pode mudar em função da carga na rede.

3.3. Configuração do Middleware

Todos os parâmetros do *middleware* devem ser definidos num arquivo de configuração. O caminho de onde se encontra esse arquivo deve ser informado como argumento para o *middleware*. Este irá procurar o arquivo no local informado e irá extrair todas as informações necessárias para seu funcionamento. A Figura 13 ilustra o XML Schema do arquivo de configuração.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="xrtps">
      <xs:complexType>
         <xs:attribute name="lenmsg" type="xs:positiveInteger" />
         <xs:attribute name="port" type="xs:positiveInteger" />
         <xs:attribute name="senport" type="xs:positiveInteger" />

         <xs:element name="control">
            <xs:complexType>
               <xs:element name="ip" type="xs:string" />
               <xs:element name="primaryport" type="xs:positiveInteger"
minOccurs="0" />
               <xs:element name="secondaryport" type="xs:positiveInteger" />
               <xs:element name="timeout" type="xs:positiveInteger"
minOccurs="0" />
               <xs:element name="delay" type="xs:positiveInteger"
minOccurs="0" />
            </xs:complexType>
         </xs:element>
         <xs:element name="publisher" minOccurs="0">
            <xs:complexType>
               <xs:element name="default">
                  <xs:complexType>
                     <xs:element name="ip" type="xs:string" />
                     <xs:element name="port" type="xs:positiveInteger" />
                  </xs:complexType>
               </xs:element>
               <xs:element name="group" maxOccurs="*">
                  <xs:complexType>
                     <xs:element name="ip" type="xs:string" />
                     <xs:element name="port" type="xs:positiveInteger" />
                     <xs:element name="topic" type="xs:string" />
                     <xs:element name="type" type="xs:string" />
                     <xs:element name="priority" type="xs:positiveInteger" />
                     <xs:element name="persistence" type="xs:positiveInteger"
/>
                     <xs:element name="deadline" type="xs:positiveInteger" />
                  </xs:complexType>
               </xs:element>
            </xs:complexType>
         </xs:element>
         <xs:element name="subscriber" minOccurs="0">
```

```
            <xs:complexType>
               <xs:element name="group" maxOccurs="*">
                  <xs:complexType>
                     <xs:element name="topic" type="xs:string" />
                     <xs:element name="ip" type="xs:string" />
                     <xs:element name="separation" type="xs:positiveInteger"
/>
                  </xs:complexType>
               </xs:element>
            </xs:complexType>
         </xs:element>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Figura 13 - XML Schema do arquivo de configuração
Fonte: Autoria própria

Todo arquivo de configuração deve começar com a *tag* raiz <xrtps>. Ela possui obrigatoriamente três atributos: *lenmsg*, *port* e *senport*. O primeiro atributo especifica o tamanho do *buffer* (um inteiro positivo, em bytes) que o *middleware* deve criar ao receber um pacote. O segundo atributo, *port* (tipo inteiro positivo), define a Porta de Comunicação Padrão, que será usada pelo *middleware* para envio e recebimento de publicações. O terceiro atributo (*senport*), do tipo inteiro positivo, especifica a Porta de Envio de Publicações, essa é a porta a qual o *middleware* irá mandar a publicação ao recebê-la de algum outro nó.

Em seguida, existe a *tag* <control> que define parâmetros do Grupo de Controle e é obrigatória para todos os nós da rede. A *tag* <ip> define o endereço IP do Grupo de Controle, ela é do tipo *string* e é a única *tag* obrigatória do elemento <control>. As *tags* <primartyport> e <secondaryport>, ambas do tipo inteiro positivo, definem as portas primárias e secundárias. A primeira será usada pelo *middleware* para cálculo do tempo de entrega entre os nós mestre e escravo. A segunda será usada para requisições e respostas de controle do serviço de eleição.

Ainda na *tag* <control>, existem as *tags* <timeout> e <delay>, ambas do tipo inteiro positivo. A primeira define o intervalo de tempo que um nó deve esperar pela resposta de controle do serviço de eleição, caso esse tempo expire, ele será eleito o nó mestre. A *tag* <delay> especifica o intervalo de tempo entre os cálculos de tempo médio de entrega das publicações.

Na seqüência do esquema existe a *tag* não-obrigatória <publisher>, que especifica os grupos aos quais o nó será inscrito como publicador. Caso essa *tag* esteja presente, será necessário configurar o Grupo dos Publicadores, que é o grupo

ao qual todo publicador se associa para receber os tempos médios de entrega das publicações. Essa configuração é feita na *tag* <default>, com os elementos <ip> (tipo *string*) e <port> (tipo inteiro positivo), o primeiro especifica o IP do grupo *multicast* dos publicadores, e o segundo define a porta a qual os publicadores devem monitorar.

Dentro da *tag* <publisher> deve existir pelo menos um grupo, definido pela tag <group>. Dentro dela, existem os seguintes parâmetros obrigatórios:

- <ip> - Especifica o endereço IP do grupo *multicast* ao qual o nó deverá se associar. Parâmetro do tipo *string*;
- <port> - Especifica a porta que o *middleware* deverá monitorar para receber dados da aplicação. Parâmetro do tipo inteiro positivo;
- <topic> - Define o nome do tópico ao qual o nó deverá se associar;
- <type> - Define o tipo do tópico;
- <priority> - Especifica a prioridade das publicações do grupo;
- <persistence> - Especifica o tempo de persistência das mensagens do grupo;
- <deadline> - Especifica por quanto tempo as mensagens do grupo são válidas.

Após a *tag* <publisher>, podem ser especificados os grupos aos quais o nó deverá se inscrever como subscritor. Para isso deve ser usada a *tag* <subscriber>, e dentro dela, a *tag* <group> para definir os grupos. A definição de um grupo subscritor envolve a configuração de apenas três parâmetros: tópico, IP e separação mínima. O tópico, definido pela *tag* <topic>, especifica o nome do tópico ao qual o nó ingressará como subscritor. O IP, definido pela *tag* <ip>, especifica o endereço IP do grupo *multicast* ao qual o nó deverá ingressar, e o tempo de separação mínima é definido pela *tag* <separation>. Segundo Dolejs (2004), o tempo mínimo de separação é um parâmetro de subscrição, portanto ele deve ser definido no grupo subscritor do arquivo de configuração.

3.4. Resultados Experimentais

Esta seção expõe alguns testes que foram realizados para avaliar o funcionamento do *middleware*. Os testes realizados não tiveram como objetivo fazer uma análise de desempenho, apenas colocar em prova a principal característica do XRTPS, a independência de plataforma e de sistema operacional.

3.4.1. Ambiente de Testes

Os testes foram realizados usando quatro máquinas conectadas via cabos de rede a um switch de 100 Mbps, conforme Figura 14. O Quadro 1 mostra como foi feita a distribuição das máquinas pelos grupos. Em todos os testes, as máquinas 01 e 03 foram eleitas como mestre e escravo, respectivamente.

	Grupo temperatura		Grupo pressão	
	Publicador	**Subscritor**	**Publicador**	**Subscritor**
Máquina 01	x		x	
Máquina 02	x			x
Máquina 03		x		x
Máquina 04		x		

Quadro 1 - Distribuição dos grupos.
Fonte: Autoria própria

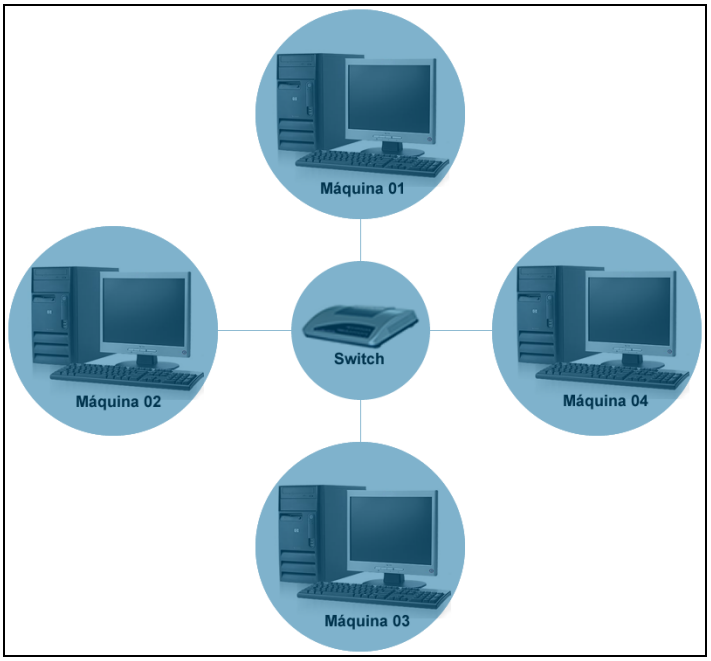

Figura 14 - Componentes de rede.
Fonte: Autoria própria

3.4.1.1. Configuração do ambiente

O switch era um D-Link DI-524 (100 Mbps) de cinco portas. O Quadro 2 expõe a configuração das máquinas usadas nos testes.

Nome	Sistema Operacional	Processador	Memória
Máquina 01	Ubuntu Linux 7.04	Intel Dual Core 1.6 GHz	504 MB
Máquina 02	Ubuntu Linux 7.04	Intel Celeron Mobile 1.6 GHz	439 MB
Máquina 03	Windows XP Professional SP2	Intel Pentium IV 2.8 GHz	512 MB
Máquina 04	Windows XP Professional SP2	AMD Semprom 2400+ 1.67 GHz	448 MB

Quadro 2 - Configuração das máquinas.
Fonte: Autoria própria

3.4.2. Cenários de Teste

Ao todo foram realizados dois testes, cada um com dez minutos de duração, ambos com a mesma configuração de *hardware* e de rede. Abaixo seguem alguns parâmetros que foram comuns em ambos os testes.

- O tamanho do *buffer* era de 512 bytes;
- A Porta de Comunicação Padrão era a 10101;
- A Porta de Envio de Publicações era a 30387;
- O IP do Grupo de Controle era o 225.4.5.6;
- As portas primária e secundária eram 25432 e 60000, respectivamente;
- O *timeout* e *delay* foram de 5 segundos e de 1 minuto, respectivamente;
- O Grupo dos Publicadores foi configurado para o grupo *multicast* 225.4.5.5, e a porta 5555;
- O tempo de separação mínima de ambos os grupos (temperatura e pressão) foi de 300 milissegundos.

O Quadro 3 mostra os parâmetros usados em ambos os grupos.

Tópico	Tipo	Endereço IP	Porta	Prioridade
temperatura	sensor	225.4.5.7	12345	5
pressao	sensor	225.4.5.8	12346	5

Quadro 3 - Parâmetros dos grupos
Fonte: Autoria própria

As únicas características que eram diferentes entre os testes era o intervalo de tempo que as aplicações enviavam as publicações, o *deadline* e o tempo de persistência dos grupos.

No primeiro teste, as aplicações das máquinas 01 e 02 (nós publicadores) foram configuradas para enviar dados a cada cinco segundos, portanto o *deadline* de todas as publicações era de cinco segundos, pois se mensagem não fosse consumida nesse tempo, deveria ser descartada, uma vez que já haveria uma nova publicação com um valor mais atualizado. O tempo de persistência das publicações foi configurado para dois segundos e meio.

No segundo teste, as aplicações foram configuradas para enviar publicações a cada três segundos, portanto o *deadline* também foi configurado para esse tempo, e o tempo de persistência foi de um segundo e meio.

3.4.3. Resultados

Os resultados do primeiro teste estão expostos no Quadro 4.

	Grupo temperatura		Grupo pressão	
	Enviadas	**Recebidas**	**Enviadas**	**Recebidas**
Máquina 01	103	-	102	-
Máquina 02	104	-	-	102
Máquina 03	-	207	-	102
Máquina 04	-	207	-	-

Quadro 4 - Resultados do primeiro teste
Fonte: Autoria própria

No primeiro teste, o total de publicações enviadas pelas máquinas 01 e 02 para o grupo temperatura foi de 207. As máquinas 03 e 04 receberam exatamente o mesmo número de mensagens, ou seja, apesar do modelo usar UDP (não orientado a conexão) para transmitir publicações, nenhuma delas foi perdida. No grupo pressão o

resultado foi o mesmo, as 102 publicações enviadas pela Máquina 01, foram recebidas pelas máquinas 02 e 03.

O Quadro 5 expõe os resultados obtidos no segundo teste.

	Grupo temperatura		Grupo pressão	
	Enviadas	**Recebidas**	**Enviadas**	**Recebidas**
Máquina 01	170	-	169	-
Máquina 02	103	-	-	169
Máquina 03	-	273	-	169
Máquina 04	-	273	-	-

Quadro 5 - Resultados do segundo teste
Fonte: Autoria própria

No segundo teste, o intervalo de tempo de envio de mensagens foi reduzido, dessa forma aumentando o número de mensagens que trafegam na rede. As máquinas 01 e 02 enviaram, ao total, 273 publicações para o grupo temperatura. Novamente os nós subscritores (máquinas 03 e 04) receberam todas elas, sem nenhuma perda. O resultado se repetiu mais uma vez para o grupo pressão, o qual foram enviadas 169 mensagens pela máquina 01, e todas foram recebidas pelos nós subscritores (máquinas 02 e 03).

3.4.3.1. Análise dos Resultados

A Figura 15 ilustra um gráfico que demonstra os tempos médios de entrega das publicações nos dois testes, bem como uma média entre os tempos médios.

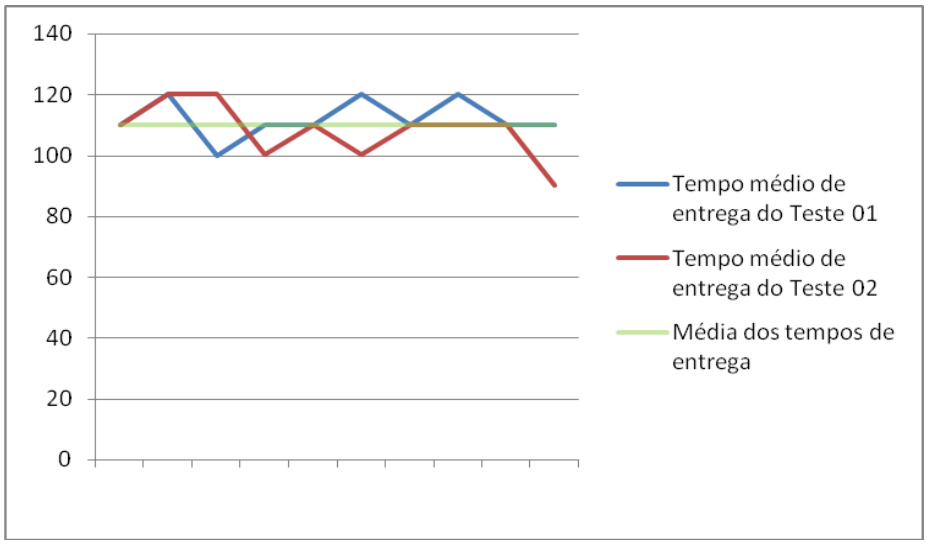

Figura 15 - Tempos médios de entrega dos testes em microssegundos.
Fonte: Autoria própria

A linha azul representa o tempo médio do primeiro teste, a linha vermelha representa o tempo médio do segundo teste, e a linha verde representa a média dos tempos médios.

O gráfico mostra que as linhas azul e vermelha estão muito próximas à linha verde, ou seja, os tempos médios de entrega estão próximos à média, portanto o desvio padrão é baixo. Isso indica que a previsibilidade da rede é alta, uma vez que é possível predizer um valor de tempo médio de entrega, pois este não varia muito. E segundo Stankovic (1988), a previsibilidade é a característica mais importante para a comunicação de tempo real.

4 CONSIDERAÇÕES FINAIS

4.1. Conclusão

O objeto de estudo desse trabalho permitiu desenvolver um *middleware* independente de plataforma que implementa um modelo de comunicação para redes de tempo real.

Ao final do trabalho, o *middleware* desenvolvido contribui para a comunicação entre nós num ambiente heterogêneo no que diz respeito a plataforma de desenvolvimento e ao sistema operacional.

4.2. Trabalhos Futuros

Como trabalhos futuros podem ser realizados testes de desempenho mais reais, tendo em vista que os testes realizados nesse trabalho usavam uma rede sem ruídos, pois o foco principal era avaliar o funcionamento do *middleware*.

Outra proposta de trabalho é implementar um mecanismo de cálculo de tempo médio de entrega das publicações mais inteligente, ou seja, que use como parâmetros informações mais precisas sobre a carga na rede, não se limitando a apenas calcular o RTT entre o mestre e escravo.

REFERÊNCIAS BIBLIOGRÁFICAS

Brito, A. E. M., Brasileiro, F. V., Leite C. E., Buriti, A. C. “Comunicação Ethernet em Tempo-Real para uma Rede de Microcontroladores”, Anais do XV Congresso Brasileiro de Automática (CBA 2004) – Brasil, setembro 2004.

Carreiro, F., Moraes, R., Fonseca, J. A e Vasques, F. “Real-Time Communication in Unconstrained Shared Ethernet Networks: The Virtual Token-Passing Approach”, submitted at Emerging Technologies and Factory Automation - ETFA, Catania, Italy, 2005.

Castellote, G., Bolton, P., “Distributed Real-Time Applications Now Have a Data Distribution Protocol”, Real-Time Innovations, Inc., February 2002.

Dolejs, O., Smolik, P., e Hanzalek Z. “On the Ethernet use for real-time publish-subscribe based applications”. In 5th IEEE International Workshop on Factory Communication Systems, Vienna, Austria, Sep. 2004.

Dolejs, O., Hanzalek, Z., “Simulation of Ethernet for Real-Time Applications”, submitted at IEEE International Conference on Industrial Technology – ICIT, Maribor, Slovenia, 2003.

Farines, J. M., Fraga, J. S., Oliveira, R. S., “Sistemas de Tempo Real”, IME-USP, São Paulo (SP), julho de 2000.

Georgoudakis, M., Kapsalis, V., Koubias, S., Papadopoulos, G., “Advancements, Trends and Real-Time Considerations in Industrial Ethernet Protocols”, IEEE 2003.

Kurose, J. F. e Ross, K. W. “Redes de Computadores e a Internet, Uma nova abordagem”. São Paulo: Addison Wesley, 2003.

Lopez, R. A., “Sistemas de Redes para Controle e Automação. Rede Industrial, Tecnologias de Controle, Meios de Transmissão, Modelo OSI, Redes Fieldbus industriais, Sistemas Residenciais e rede Ethernet”, Ed. Book Express, 2000.

Moraes, R.; Vasques, F., “Real-time traffic separation in shared Ethernet networks: simulation analysis of the h-BEB collision resolution algorithm”, Embedded and Real-Time Computing Systems and Applications, 2005. Proceedings. 11th IEEE International Conference on , vol., no.pp. 89- 92, 17-19 Aug. 2005.

NDDS. Real-Time Innovations, Inc., “NDDS Getting Started Guide”, Version 3.0, 2002.

NDDS. “Network Data Distribution Service, Real-Time Publish-Subscribe Network Middleware”. Real-Time Innovations (RTI), http://www.rti.com. 2005

OCERA. Czech Technical Univesity in Prague, Overall OpenSource project information containing ORTE, http://sourceforge.net/projects/ocera/, 2003.

Pedreiras, P., Almeida, L., Gai, P. and Giorgio, B. “FTT-Ethernet: A Flexible Real-Time communication Protocol That Supports Dynamic QoS Management on Ethernet-Based Systems”. IEEE Transactions on Industrial Informatics, Vol. 1, Nº. 3, August 2005.

RFC 768, Postel, J., “User Datagram Protocol”, Network Information Center, SRI International, Menlo Park, Calif., August 1980.

RFC 919, Mogul, J., “Broadcasting Internet Datagrams”, Computer Science Department, Stanford University, October 1984.

RFC 966, Deering, S. E., Cheriton, D. R., “Host Groups: A Multicast Extension to the Internet Protocol”, Stanford University, December 1985.

RTPS. Real-Time Innovations, Inc., “Real-Time Publish-Subscribe Wire Protocol Specification”, Protocol Version 1.0, Draft doc, version 1.17, http://www.rti.com/products/ndds/literature.html, 2002.

Smolik, P., Sebek, Z., Hanzalek, Z., “ORTE – Open Source Implementation of Real-Time Publish-Subscribe Protocol”, *2nd Intl Workshop on real-time LANs in the Internet age*, Polytechnic Institute of Porto, Portugal, 2003.

Stankovic, J. A., “Misconceptions about real-time computing: a serious problem for next-generation systems”. Dept. of Comput. & Inf. Sci., Massachusetts Univ., Amherst, MA.

Stevens, W. R., Fenner, B., Rudoff, A. M., “Programação de Rede UNIX: API para soquetes de rede”. 3rd. Ed., Bookman, 2005.

Tanenbaum, A. S. “Computer Networks”. 4rd. Ed., Prentice- Hall, 2003.

Thomesse, J.-P. “Fieldbus Technology in Industrial Automation”. Proceedings of The IEEE, Vol. 93, Nº. 6, June 2005.

Valentim, R. A. M. “Análise de Desempenho Experimental de Redes IEEE 802.3”. Dissertação de Mestrado. Laboratório de Engenharia de Computação e Automação da Universidade Federal do RN. 2006.

W3C. World Wide Web Consortium. Extensible Markup Language (XML). Disponível em: <http://www.w3.org/XML. 2004>. Acesso em: <04 de dezembro de 2006>.

XML Schema. World Wide Web Consortium (W3C). XML Schema. Disponível em: <http://www.w3.org/xml/schema. 2001>. Acesso em: <6 de maio de 2007>.