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

J. R. F. Lima[1,2], R. A. M. Valentim[2], B. G. Araújo[2], J. M. T. Lacerda[2], D. R. Carvalho[2] R. M. Medeiros[2,3]
[1]IFRN – Campus Pau dos Ferros / [2]UFRN – Depto Eng. Biomédica – [3]IFRN – Campus Mossoró
jalerson.lima@ifrn.edu.br - ricardo.valentim@ufrnet.br - jonymac@deb.ufrn.br - ronaldo.maia@ifrn.edu.br

Artigo submetido em maio/2011 e aceito em junho/2011

RESUMO

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. Diante do exposto, esse artigo apresenta um *middleware* desenvolvido em Java para troca de dados em redes industriais, denominado *Extensible Real-Time Publish Subscribe*. Java foi escolhida como a tecnologia de desenvolvimento a fim de garantir a independência de plataforma provida pela Máquina Virtual Java. Essa independência também é provida pelo próprio *middleware*, uma vez que suas mensagens são formatadas em XML, que por se tratar de um padrão mundial estabelecido pela W3C, pode ser manipulado por diferentes tecnologias.

PALAVRAS-CHAVE: RTPS, Redes Industriais, Tempo Real, Ethernet, Multicast.

XRTPS – EXTENSIBLE REAL-TIME PUBLISH-SUBSCRIBE: MULTICAST MIDDLEWARE TO DATA EXCHANGE BASED ON XML TO REAL TIME NETWORKS

ABSTRACT

A recurring factor of office networks is the search for greater performance, ie, networks that perform their tasks in the shortest possible time. However, for industrial networks the average performance is not the most important factor, which is why the performance of a real-time system is directly related to the predictability imposed by the environment. Given the above, this paper presents a middleware developed in Java for data exchange in industrial networks, called Extensible Real-Time Publish Subscribe. Java was chosen as the technology development to ensure platform independence provided by Java Virtual Machine. This independence is also provided by the middleware, since your messages are formatted in XML, which in the case of a global standard set by W3C, can be manipulated by different technologies.

KEY-WORDS: RTPS, Industrial Networks, Real Time, Ethernet, Multicast.

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

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).

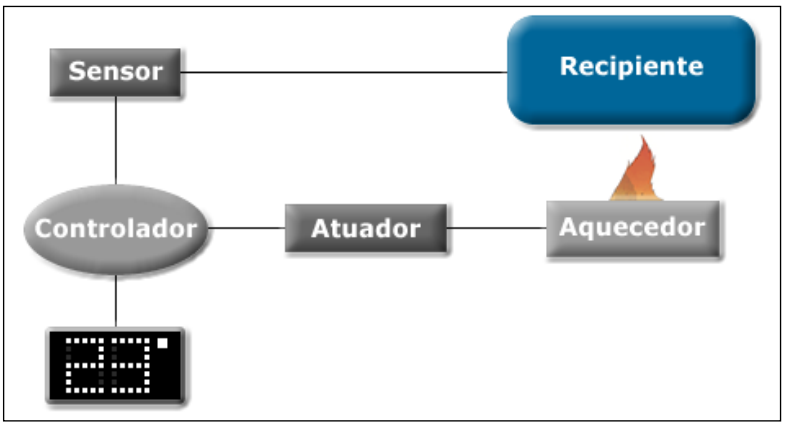

Figura 1 – Exemplo hipotético. Baseado em Valentim (2006)

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 e consumirão apenas os de interesse.

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, 2005).

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 *middleware* para troca dados baseado no RTPS, porém, sendo mais flexível quanto à plataforma de desenvolvimento, sistema operacional e paradigma de programação.

O *middleware* proposto nesse trabalho é o *Extensible Real-Time Publish-Subscribe* (XRTPS). A interoperabilidade do XRTPS se fundamenta no XML (*Extensible Markup Language*) (W3C, 2004), que 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.

FUNDAMENTAÇÃO TEÓRICA

Nesta sessão serão abordadas todas as pesquisas realizadas para formar uma base de conhecimento necessária para o entendimento do objeto de estudo proposto por esse trabalho.

MULTICAST

O método de transmissão de dados utilizado pelo XRTPS é o Multicast. O multicast consiste na 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 (RFC 966, 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.

REAL-TIME PUBLISH-SUBSCRIBE

O método de troca de dados utilizado pelo XRTPS é o *Real-Time Publish-Subscribe* (Dolejs, 2004), pois é um método baseado no *Publish-Subscribe*, agregando todas as vantagens providas por ele (Thomesse, 2005), e ainda acrescenta maiores funcionalidades necessárias para um ambiente de tempo real (Castellote, 2002), 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 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*.

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).

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.

Entre os parâmetros de publicação (veja Figura 2), 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.

Entre os parâmetros de subscrição (veja Figura 3), 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.

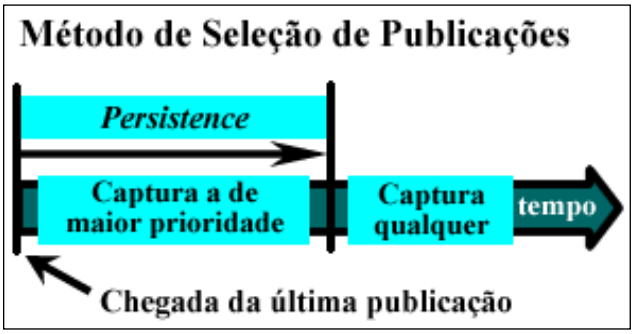

Figura 2 – Parâmetros de publicação. Fonte: Dolejs (2004)

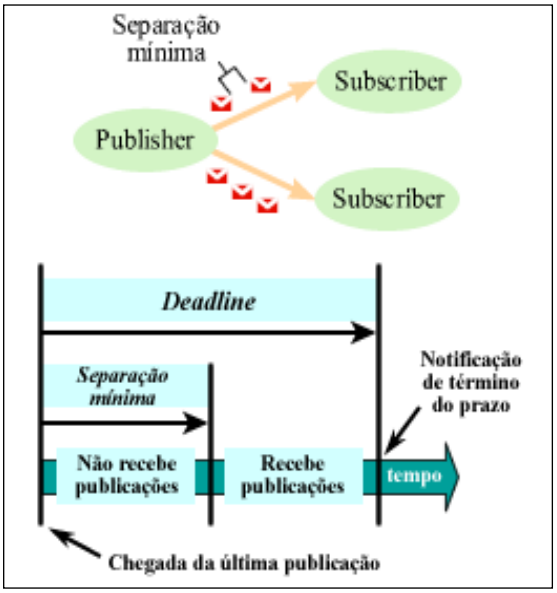

Figura 3 – Parâmetros de subscrição. Fonte: Dolejs (2004)

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.

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.

EXTENSIBLE REAL-TIME PUBLISH-SUBSCRIBE

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.

Conforme ilustrado na Figura 4, 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.

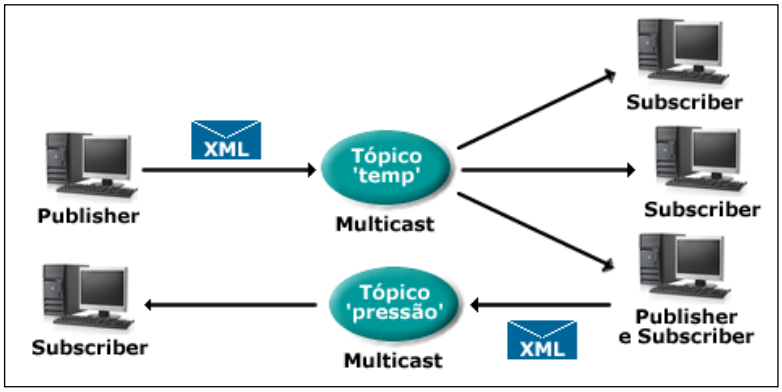

Figura 4 – Arquitetura do XRTPS

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.

Conforme ilustrado na Figura 5, 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.

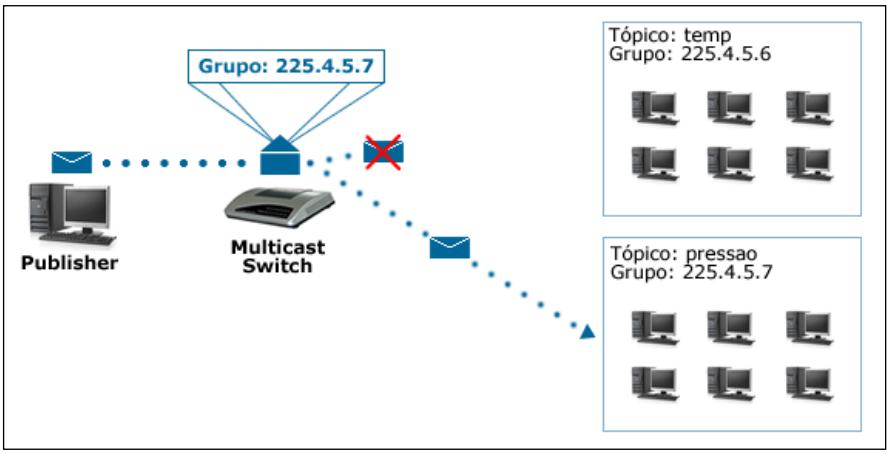

Figura 5 – Transmissão via Multicast

FORMATO DAS MENSAGENS

Conforme ilustrado na Figura 4, as mensagens trocadas entre os nós da rede são formatadas utilizando a linguagem de marcação XML. A Figura 6 ilustra o XML Schema (2001) no qual define os dados e os tipos que compõem as tags das mensagens.

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.

```
<?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 6 – XML Schema da mensagem

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.

A Figura 7 ilustra o funcionamento do Serviço de Comunicação.

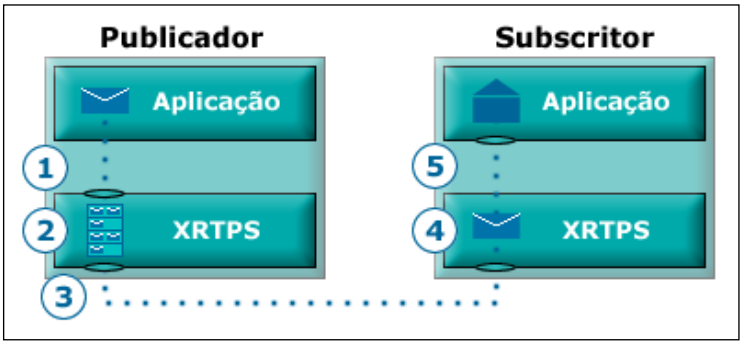

Figura 7 – Serviço de Comunicação

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.

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.

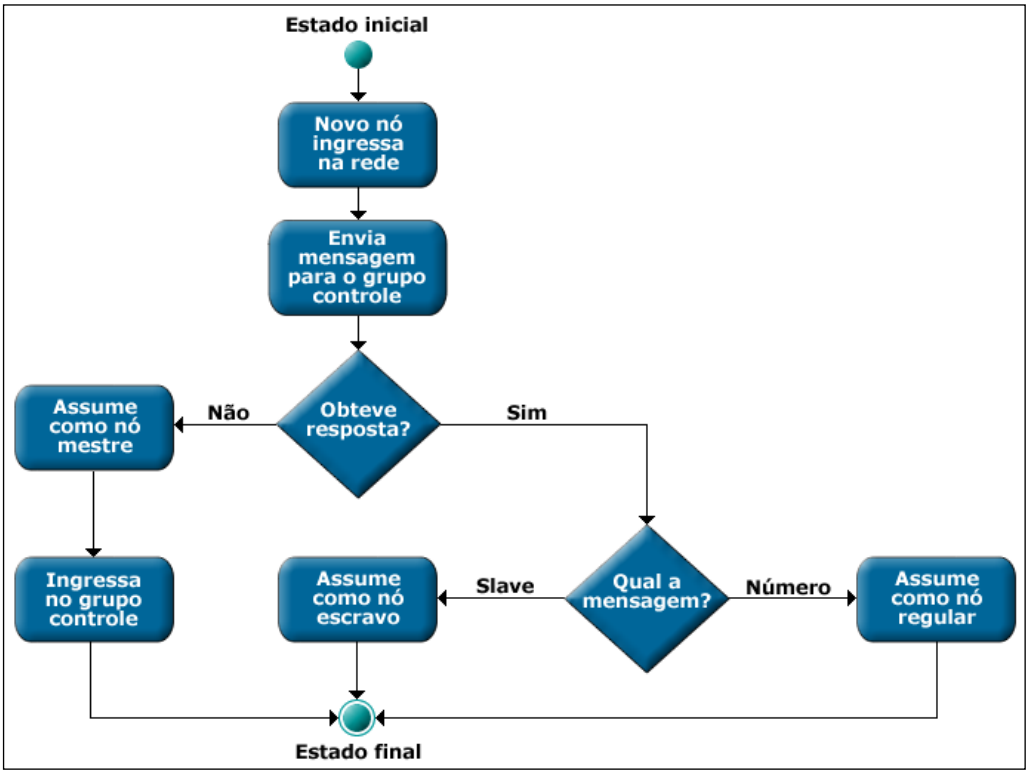

Figura 8 – Eleição dos nós mestre e escravo.

Conforme demonstrado no fluxograma da Figura 8, 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.

CÁLCULO E PUBLICAÇÃO DO TEMPO MÉDIO DE ENTREGA DAS PUBLICAÇÕES

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 Figura 8).

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

Figura 8 – Cálculo do Tempo Médio de Entrega das Publicações

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.

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 10 – XML Schema do arquivo de configuração.

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.

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.

AMBIENTE DE TESTES

Os testes foram realizados usando quatro máquinas conectadas via cabos de rede a um switch de 100 Mbps. Para simular um ambiente industrial, foram criados dois grupos fictícios chamados temperatura e pressão. A Máquina 01 (M1) foi inscrita nos dois grupos como publicadora. A Máquina 02 (M2) foi inscrita no grupo temperatura como publicadora e no grupo pressão como subscritora. A Máquina 03 (M3) foi inscrita em ambos os grupos como subscritora, e a Máquina 04 (M4) foi inscrita no grupo temperatura como subscritora. As máquinas M1 e M3 foram eleitas como mestre e escravo, respectivamente.

CONFIGURAÇÃO DO AMBIENTE

Para avaliar a independência de plataforma provida pelo middleware, as máquinas M1 e M2 usaram o sistema operacional Ubuntu Linux 7.04, as outras duas máquinas usaram o Windows XP Professional SP2.

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. As únicas diferenças entre os testes eram 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 M1 e M2 (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.

RESULTADOS

Os resultados do primeiro teste estão expostos na Tabela 1.

Tabela 1 – Resultados do primeiro teste

	Grupo temperatura		Grupo pressão	
	Enviadas	Recebidas	Enviadas	Recebidas
M1	103	-	102	-
M2	104	-	-	102
M3	-	207	-	102
M4	-	207	-	-

No primeiro teste, o total de publicações enviadas pelas máquinas M1 e M2 para o grupo temperatura foi de 207. As máquinas M3 e M4 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 1, foram recebidas pelas máquinas M2 e M3.

A Tabela 2 expõe os resultados do segundo teste.

Tabela 2 – Resultados do segundo teste

	Grupo temperatura	Grupo pressão

	Enviadas	Recebidas	Enviadas	Recebidas
M1	170	-	169	-
M2	103	-	-	169
M3	-	273	-	169
M4	-	273	-	-

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 M1 e M2 enviaram, ao total, 273 publicações para o grupo temperatura. Novamente os nós subscritores (M3 e M4) 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 M1, e todas foram recebidas pelos nós subscritores (M2 e M3).

ANÁLISE DOS RESULTADOS

A Figura 11 ilustra um gráfico que demonstra os tempos médios de entrega das publicações nos dois testes, bem como uma média entre os tempos médios.

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.

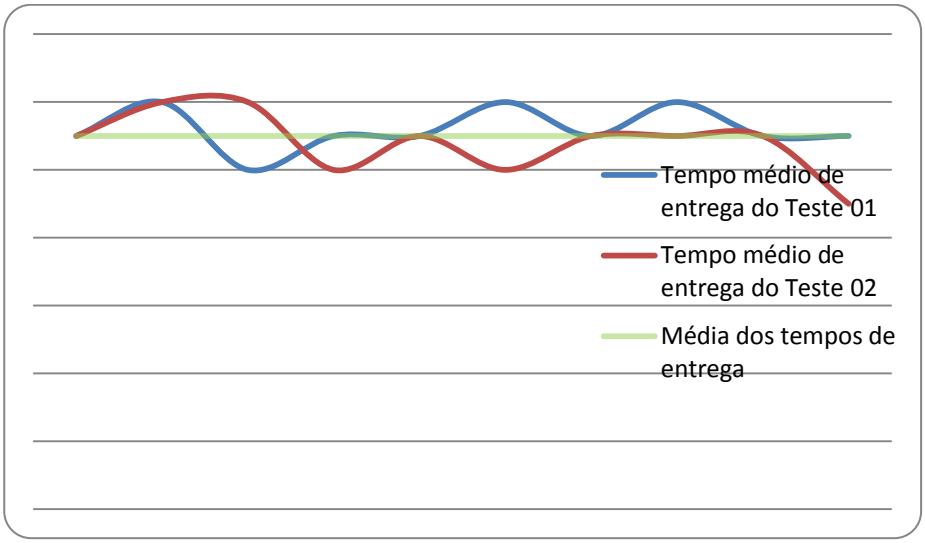

Figura 11 – Tempos médios de entrega dos testes (em microssegundos).

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.

REFERÊNCIAS

1. Stankovic, J. A., “Misconceptions about real-time computing: a serious problem for next-generation systems”. Dept. of Comput. & Inf. Sci., Massachusetts Univ., Amherst, MA.
2. Farines, J. M., Fraga, J. S., Oliveira, R. S., “Sistemas de Tempo Real”, IME-USP, São Paulo (SP), julho de 2000.
3. 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.
4. 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.
5. Thomesse, J.-P. “Fieldbus Technology in Industrial Automation”. Proceedings of The IEEE, Vol. 93, Nº. 6, June 2005.
6. 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.

7. W3C. World Wide Web Consortium. Extensible Markup Language (XML). Disponível em: <http://www.w3.org/XML>. Acesso em: <04 de dezembro de 2006>.
8. RFC 966, Deering, S. E., Cheriton, D. R., "Host Groups: A Multicast Extension to the Internet Protocol", Stanford University, December 1985.
9. Castellote, G., Bolton, P., "Distributed Real-Time Applications Now Have a Data Distribution Protocol", Real-Time Innovations, Inc., February 2002.
10. RFC 768, Postel, J., "User Datagram Protocol", Network Information Center, SRI International, Menlo Park, Calif., August 1980.
11. Kurose, J. F. e Ross, K. W. "Redes de Computadores e a Internet, Uma nova abordagem". São Paulo: Addison Wesley, 2003.
12. Georgoudakis, M., Kapsalis, V., Koubias, S., Papadopoulos, G., "Advancements, Trends and Real-Time Considerations in Industrial Ethernet Protocols", IEEE 2003.
13. Stevens, W. R., Fenner, B., Rudoff, A. M., "Programação de Rede UNIX: API para soquetes de rede". 3rd. Ed., Bookman, 2005.
14. XML Schema. World Wide Web Consortium (W3C). Disponível em: <http://www.w3.org/xml/schema>. Acesso em: 06/05/2007.
15. NDDS. Real-Time Innovations, Inc., "NDDS Getting Started Guide", Version 3.0, 2002.
16. NDDS. "Network Data Distribution Service, Real-Time Publish-Subscribe Network Middleware". Real-Time Innovations (RTI), http://www.rti.com. 2005
17. 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.
18. 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.
19. 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.
20. 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.