IMPLEMENTAÇÃO DE UMA ARQUITETURA *MIDDLEWARE* DDS

Albino Alves, Diogo Recharte, Ricardo Magno

I. RESUMO

O objectivo deste estudo foi a implementação de um *middleware* DDS com vista a replicar um ambiente RTDB já existente. Na procura de uma solução começou-se por utilizar o *framework* RTI Connext DDS, no entanto esta alternativa nõ se encontrava completa para os objectivos propostos e como tal uma diferente implementação de DDS foi abordada, o OpenDDS. Foi implementada uma estrutura de dados semelhante à já existente no projecto RTDB CAMBADA. No final foram efectuadas medições de tempo utilizando 3 meios de comunicação distintos, *Ethernet*, *Wi-Fi* e *LoopBack*.

II. DEFINIÇÃO DO PROBLEMA

Utilizar uma camada *middleware* baseada num sistema de Real Time Database (RTDB) apresenta, à priori, alguns inconvenientes, inerentes à ideia base de partilha de memória em todos os nós do grupo. Esta premissa leva, não só à necessidade de ter um espaço de memória alocado em cada nó com uma repetição de dados, como também garantir que todos os nós partilham os mesmos parâmetros do sistema, de modo a garantir a integridade na leitura, escrita e comunicação de dados. Um exemplo destes parâmetros é a *endianness* do sistema.

Relativamente à camada de *middleware*, é benéfico realizar um estudo com as diferentes tecnologias existentes de modo a ser possível compará-las no que toca à latência e complexidade de cada sistema implementado.

No caso específico deste projecto, a camada de *middleware* de RTDB foi diretamente comparada com uma implementação baseada em *Publisher/Subscriber*. Esta resolve os incovenientes mencionados previamente, visto ser baseada no conceito de *messaging pattern*, onde os *senders (publisher)* não transmitem a mensagem diretamente a cada *receiver (subscriber)* mas sim a uma entidade mediadora (*broker*). As mensagens a enviar são categorizadas em tópicos e o *broker* transmite-as apenas a quem subscreveu ao tópico em questão. Cabe então, a cada *subscriber* definir que tópicos subscrever de modo a garantir a receção das mensagens que lhe interessam.

Com esta ideia base garante-se uma abstração relativamente ao conhecimento dos participantes da rede. Contrariamente, a complexidade de uma possível implementação de tal sistema aumenta. Visto ser um protocolo baseado em troca de mensagens, como mencionado em cima, não necessita de espaço na memória destinado ao alocamento de todos os dados dos outros nós.

Posto isto, o objectivo máximo deste projecto foi uma replicação de um ambiente *middleware* RTDB numa camada *middleware* DDS de forma a obter alguns resultados relativos a esta última.

III. DESCRIÇÃO DA SOLUÇÃO

Inicialmente, estudou-se o uso de uma solução proprietária utilizando uma versão de avaliação da empresa RTI, tal não foi possível uma vez que esta não disponibiliza alguns ficheiros cruciais para a realização da comunicação, optou-se então por utilizar uma versão *opensource*, o OpenDDS.

Foram criadas estruturas de dados semelhantes às presentes em RTDB nomeadamente *Loc*, *Refboc* e *Vision* que representam os três tópicos necessários. *Loc* contém 2 variáveis do tipo *float*, x e y, que representam a posição do *robot*, *Refbox* apresenta a variável *refBoxCommand* do tipo *long* que é utilizada para comunicar comandos e, por fim, *Vision* contém *visionMatrix*, um vetor multidimensional de caracteres. A cada uma destas estruturas foi adicionado um indentificador do nodo assim como um *timestamp* absoluto que guarda o momento de escrita para as variáveis que antecede o envio dos dados.

```
#include "orbsvcs/TimeBase.idl"

typedef sequence <char> Row;
typedef sequence <Row> Matrix;

module RobotTracker
{
  #pragma DCPS_DATA_TYPE "RobotTracker::Loc"
  #pragma DCPS_DATA_KEY "RobotTracker::Loc node_id"

    struct Loc {
      string node_id;
        float x;
        float y;
      TimeBase::TimeT timestamp;
    };

  #pragma DCPS_DATA_TYPE "RobotTracker::Refbox"
  #pragma DCPS_DATA_KEY "RobotTracker::Refbox
      node_id"

    struct Refbox {
      string node_id;
      long refBoxCommand;
      TimeBase::TimeT timestamp;
    };

  #pragma DCPS_DATA_TYPE "RobotTracker::Vision"
  #pragma DCPS_DATA_KEY "RobotTracker::Vision
      node_id"

    struct Vision {
      string node_id;
        Matrix visionMatrix;
      TimeBase::TimeT timestamp;
    };
};
```

A arquitetura DDS impõe a criação de um domínio onde todos os participantes podem interagir com o *broker*. Desta forma, o primeiro passo foi a criação deste domínio. De seguida, foi criado um participante e associado a este o papel de publicador e de subscritor. Uma vez que o objetivo é que todos os nós da rede tenham informação sobre todos os restantes, é necessário que cada um seja simultâneamente publicador e subscritor de todos os tópicos. Seguidamente, foram registados os três tipos de tópicos e foram explicitamente criados cada um deles.Foi criado também um *listener* para cada um do tópicos e os respetivos *datareaders* e *datawriters*. O *listener* é a entidade responsável por ficar "à escuta" de notificações provenientes do *broker* referentes ao tópico em questão. Desta forma, quando o *listener* recebe algo do *broker* executa o *datareader* para processar a informação adequadamente. No contexto deste estudo, este processamento envolve guardar localmente os valores recebidos referentes aos restantes nós da rede e imprimir para um ficheiro os valores da latência para uma análise posterior.

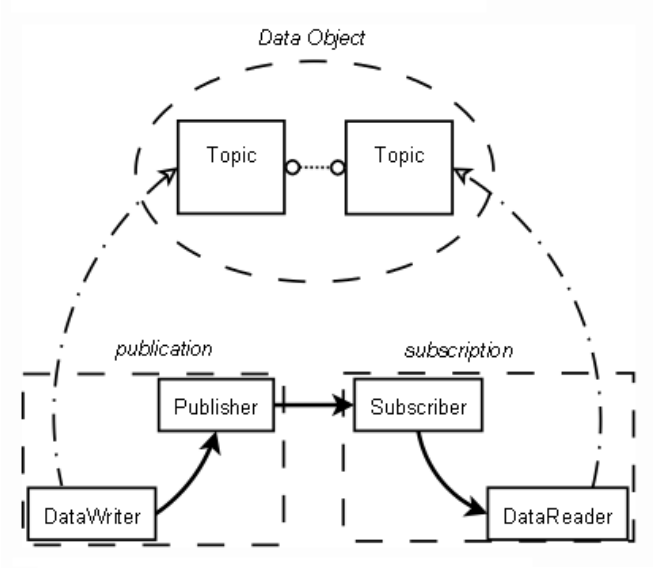

Fig. 1: DDS system overview

IV. RESULTADOS

Nesta secção são apresentados os resultados da latência envolvida na distribuição das informações dos nós pela rede. Esta latência é calculada pela diferença entre o *timestamp* obtido após o processamento dos dados no nó que recebe a notificação e o *timestamp* de quando o nó publicador alterou os valores dos seu dados. Para realizar a sincronização dos relógios dos dispositivos envolvidos foi utilizado o *chrony*.

A Fig. 2 apresenta o histograma das latências medidas utilizando dois processos na mesma máquina a comunicar através da interface localhost. Nesta é possível verificar que a transmissão do tópico *Vision* demora em média 4 a 5 vezes mais tempo do que *Loc* e *Refbox*. Isto era esperado dado que a transmissão de *Vision* envolve envio de uma matriz de 800x600 caracteres. A latência desta transmissão apresenta

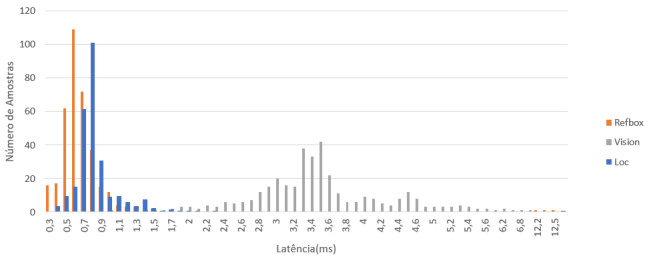

Fig. 2: Latência sobre *Localhost*

um desvio padrão de 0.67 ms para *Loc*, 1.09 ms para *Refbox* e 0.83 ms para *Vision*.

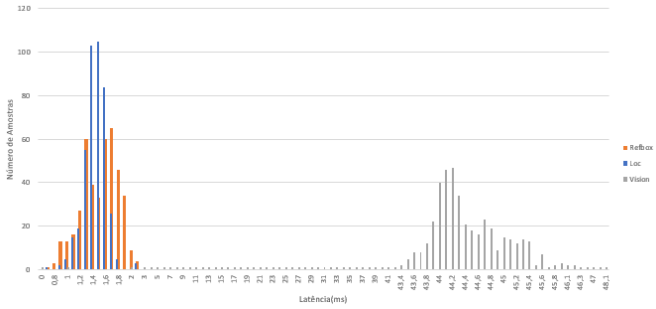

Fig. 3: Latência sobre *Ethernet*

Executando 2 processos em máquinas diferentes ligadas à mesma rede comunicando via *Ethernet* (Fig. 3)é possível ver que as latências médias aumentaram assim como a diferença entre o tópico *Vison* e os restantes. Pode-se também verificar que a distribuição dos tópicos mais pequenos se encontra mais compacta apresentando um desvio padrão de apenas 0.174 ms para *Loc* e 0.288 ms para *Refbox*. Já *Vision* apresentou um desvio padrão de 2.24 ms.

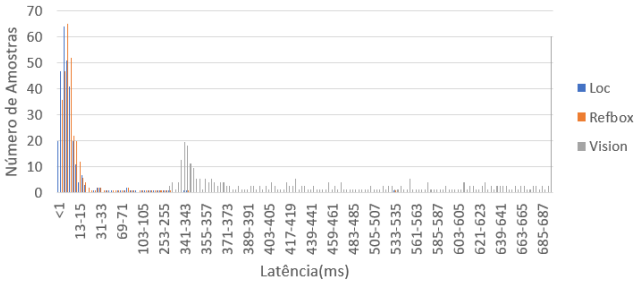

Fig. 4: Latência sobre *WiFi*

Utilizando *Wi-Fi* como meio de comunicação entre as duas máquinas é possível verificar latências médias muito maiores, 18.86 ms para *Loc*, 20.48 ms para *Refbox* e 532.71 ms para *Vision*. Era esperado que via *Wi-Fi* as latências fossem maiores, no entanto não foi antecipado uma diferença desta magnitude. O facto das medições terem sido obtidas num

ambiente com possivelmente muita congestão de tráfego, a rede da universidade, pode ter contribuído para estas diferenças. Não só os valores das latências foram muito elevados como foram também muito instáveis com um desvio padrão de 55.9 ms para *Loc*, 55.55 ms para *Refbox* e 220.61 ms para *Vision*.

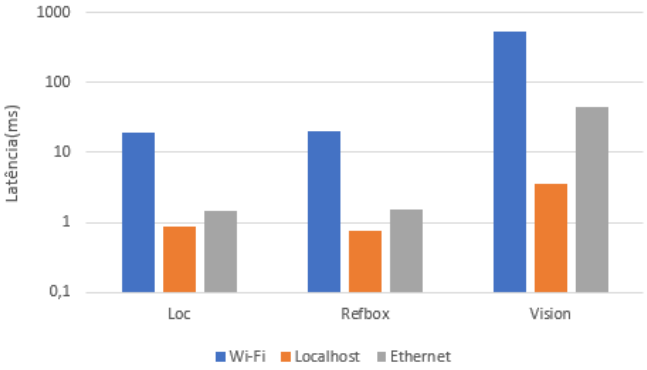

Fig. 5: Latência média

Juntando os valores médios das latências obtidas utilizando as diferentes interfaces transmitindo os diversos tópicos (Fig. 5) foi possível verificar que, como seria de esperar, a transmissão por *Localhost* apresenta a menor latência, seguida de *Ethernet* e, por fim, *Wi-Fi*. Este facto verifica-se independentemente do tópico e a diferença entre as interfaces torna-se mais aparente quanto maior o tópico, e.g. esta desigualdade é muito mais acentuada em *Vision* do que nos restantes tópicos.

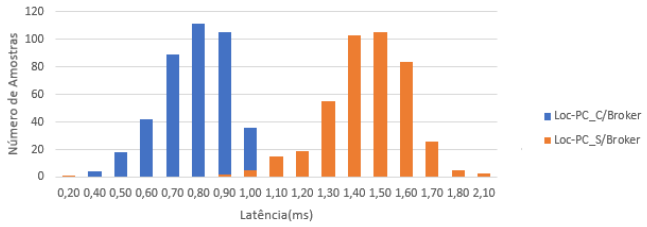

Fig. 6: Latência com ou sem *broker* na mesma máquina

Dada a necessidade desta arquitetura utilizar um *broker*, foi necessário uma das máquinas envolvidas nas medições executar este servidor. Foi notado que havia uma diferença dependendo se o publicador fosse executado na mesma máquina que o *broker* ou não. Na Fig. 6 apresenta-se o histograma de latências, a azul são representados os resultados obtidos quando o *publisher* e o *broker* partilhavam o mesmo computador e a laranja o caso em que o publicador foi executado numa máquina diferente enquanto que o *subscriber* correu em conjunto com o *broker*. É possível concluir então que a latência verificada no ato de publicar teve um maior impacto no panorama geral pois foi ao executar o *publisher* e o *broker* na mesma máquina que obtive-se melhores resultados.

V. ANÁLISE CRÍTICA

Após a familiarizaçao com ambas as arquiteturas de *middleware* abordadas neste relatório é de fácil conclusão que RTDB apresenta uma API mais simples e, por isso, torna a aplicação mais inteligível. No entanto, a maior abstração verificada em RTDB torna a camada de *middleware* menos flexível e, dessa forma, a adaptação desta a outras funcionalidades ou aplicações torna-se um processo mais intricado. Em oposição a este facto, o *middleware* OpenDDS torna muito fácil, com a sua arquitetura *Publisher/Subscriber*, a adaptação aos mais variados cenários e aplicações.

Uma análise interessante de fazer seria comparar os valores apresentados na seção anterior com valores equivalentes utilizando RTDB. No entanto, visto que a atual implementação apenas envia um valor de tempo de vida dos dados e depois é somado um valor estimado de 1 ms de *delay* da rede, obter tais medidas envolveria ter conhecimento de um *timestamp* absoluto o que implicaria uma restruturação enorme da implementação da memória partilhada. Porém, é possível especular que visto que os resultados foram obtidos utilizando ligações TCP e a implementação de RTDB utiliza uma comunicação UDP multicast, RTDB apresentaria latências menores. OpenDDS também disponibiliza uma interface que utiliza comunicações UDP, no entanto esta opção não foi testada.

No que toca a contribuição de cada elemento do grupo para o desenvolvimento final do projecto, foi uma contribuição muito semelhante, pois foram sempre presentes e activos todos os elementos do grupo participando estes em todas as fases de desenvolvimento, sendo assim cada elemento contribuiu com $\frac{1}{3}$ do esforo do trabalho global.

REFERENCES

[1] OpenDDS Version 3.12 Supported by Object Computing, Inc. (OCI)
[2] https://www.rti.com/labs
[3] http://opendds.org/
[4] https://bitbucket.org/fredericosantos/rtdb/wiki/Home