Universidade de Aveiro
2012

Departamento de
Eletrónica, Telecomunicações e Informática

Luís Miguel Tomé Nóbrega

Suporte MSRP para *Hard* QdS *Switch*

Universidade de Aveiro
2012

Departamento de
Eletrónica, Telecomunicações e Informática

Luís Miguel Tomé Nóbrega

Suporte MSRP para *Hard* QdS *Switch*

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Professor Doutor Paulo Bacelar Reis Pedreiras do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro e do Professor Doutor Pedro Alexandre Sousa Gonçalves da Escola Superior de Tecnologia e Gestão de Águeda

Trabalho financiado por Fundos Nacionais através da FCT - Fundação para a Ciência e a Tecnologia no âmbito do projeto Serv-CPS PTDC/EAA-AUT/122362/2010

Dedico este trabalho à minha família, amigos e em especial à minha avó, que nos deixou no decorrer deste trabalho

o júri

presidente	**Prof. Doutor José Alberto Gouveia Fonseca** Professor Associado, Universidade de Aveiro
arguente	**Prof. Doutor Paulo José Lopes Machado Portugal** Professor Auxiliar, Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto
orientador	**Prof. Doutor Paulo Bacelar Reis Pedreiras** Professor Auxiliar, Universidade de Aveiro
coorientador	**Prof. Doutor Pedro Alexandre Sousa Gonçalves** Professor Adjunto, Universidade de Aveiro

agradecimentos

No percurso que agora termina pude contar com a colaboração e apoio de muitas pessoas, a quem quero agora manifestar o meu agradecimento.

Em primeiro lugar, quero agradecer ao meu orientador científico, Professor Doutor Paulo Bacelar Reis Pedreiras, pela prontidão e disponibilidade científica e académica que facultou sob a forma de críticas, sugestões e estímulos essenciais ao desenvolvimento deste projeto e ao meu enriquecimento intelectual. Pela amizade, apoio e confiança depositada, o meu muito obrigado.

Em segundo lugar, quero agradecer ao Professor Doutor Pedro Alexandre Sousa Gonçalves, pela forma irrepreensível como desempenhou o seu papel de coorientador, pelos conhecimentos científicos e técnicos transmitidos e pela amizade, apoio e motivação transmitida em todos os momentos do projeto.

À minha família e em especial aos meus pais, pelo apoio incondicional demonstrado em todos os instantes, bons e menos bons, do projeto que agora finda.

A todos os meus amigos que me apoiaram e nunca deixaram de acreditar nas minhas capacidades, mesmo nos momentos de maior desânimo.

Por fim, um especial agradecimento à minha avó que nos abandonou no decorrer deste projeto, mas que desde sempre confiou e me apoiou e a quem devo muito do que sou hoje.

Muito obrigado.

palavras-chave

Sistemas de Tempo-Real, Comunicações de Tempo-Real, *Ethernet, Switch HaRTES,* Protocolos de Sinalização, Qualidade de Serviço (QdS), Reserva de Recursos, *RSVP, SRP, MSRP*

resumo

A exigência de comunicações de Tempo-Real em processos industriais e sistemas embutidos distribuídos foi, durante muitos anos, satisfeita com recurso a redes de campo especializadas (comummente designadas por *fieldbuses*). Contudo, a crescente utilização de redes Ethernet, aliada a vantagens competitivas a elas inerentes como o preço e velocidade, levou a que se procurassem soluções que permitissem utilizar esta tecnologia em ambientes de Tempo-Real. Apesar de algumas limitações evidenciadas, como o indeterminismo temporal que o seu funcionamento apresenta, diversos protocolos surgiram nos últimos anos no sentido de adaptar o seu funcionamento aos requisitos das comunicações de Tempo-Real.

Contudo, os protocolos que foram sendo apresentados são maioritariamente estáticos e *off-line*, não possibilitando uma gestão dinâmica da Qualidade de Serviço. Assim, surgiu na Universidade Aveiro um *switch*, o *HaRTES* (*Hard Real-Time Ethernet Switch*), capaz de fornecer garantias de Tempo-Real com uma maior flexibilidade na gestão de recursos.

Adicionalmente, o fornecimento de garantias de Qualidade de Serviço pressupõe a existência de reserva de recursos nas estações de uma rede. Para que isso seja possível, foram propostos alguns protocolos de sinalização como o *Resource Reservation Protocol* (*RSVP*) e o *Stream Reservation Protocol* (*SRP*).

O *HaRTES*, não obstante o facto de garantir Qualidade de Serviço de uma forma dinâmica, não suporta nenhum protocolo normalizado no mercado. Consequentemente, surgiu a necessidade de desenvolver uma plataforma que permita ao referido *switch* suportar a reserva de recursos recorrendo a um desses protocolos.

Devido a um mais fácil mapeamento de parâmetros, foi escolhido o *SRP*. Este trabalho apresenta-o de uma forma detalhada, discute os blocos estruturais essenciais à sua implementação, detalha o seu funcionamento e pormenoriza as mensagens trocadas entres os diversos intervenientes.

A implementação de alguns dos blocos é discutida no âmbito desta dissertação e alguns testes funcionais descritos. Estes permitiram validar o trabalho desenvolvido, abrindo a oportunidade de se integrar por completo o protocolo de sinalização *SRP* no *switch HaRTES.*

keywords Real-Time systems, Real-Time communications, Switch *HaRTES,* Ethernet, Quality of Service (QoS), Signalling protocols, Resource Reservation, *RSVP, SRP, MSRP*

abstract

For several years, the hard demands of real-time communications in industrial processes and embedded systems, has been solve with the use of specialized fieldbuses. However, due to the increasing usage of Ethernet networks, together with its inherent competitive advantages like reduced price and fast velocities, a search for new solutions that allow the use of these networks in Real-Time environments began. Despite of its limitations, such as temporal indeterminism derived from its medium access control scheme, many protocols have been developed in the last few years with the objective of adapting its functionalities to the requirements of Real-Time communications.

Nevertheless, mostly of the protocols developed are static and based on pre-runtime analysis. As a consequence, they don't allow a dynamic management of the Quality of Service (*QoS*). Thus, it was constructed at Aveiro University a new modified switch, *HaRTES* (Hard Real-Time Ethernet Switch), capable of providing Real-Time guarantees with greater resource management flexibility.

Furthermore, providing guaranteed Quality of Service requires the existence of resource reservation along the nodes of a network. In order to make this possible some signalling protocols were proposed, such as the Resource Reservation Protocol (*RSVP*) and Stream Reservation Protocol (*SRP*).

Even if the *HaRTES* switch guarantees Quality of Service in a dynamic way, it doesn't support any protocol standard. Therefore, the need of developing a new platform that will enable the switch to perform resource reservations using standard protocols emerged.

Due to the easier mapping of parameters, the *SRP* was chosen to be that platform. This work describes it in detail. It discusses the structural blocks crucial for its implementation, describes the protocol operation and details the messages exchanged between the different nodes in a network.

The implementation of some blocks is discussed and some tests are performed allowing the validation of the work developed. The results of this work open a window opportunity for the total integration of the *SRP* protocol in the *HaRTES* switch.

Índice

Índice de Figuras

Índice de Tabelas

Lista de Acrónimos

ADFX	*Avionics Full Duplex Switched Ethernet*
AVB	*Audio/Video Bridging*
CAN	*Controller Area Network*
COTS	*Commercial-Of-The-Shelf*
CPU	*Central Processing Unit*
CSMA/CD	*Carrier Sense Multiple Access / Collision Detected*
DM	*Deadline Monotonic*
DRP	*Dynamic Sender-Initiated Reservation Protocol*
DS	*Deferrable Server*
EC	*Elementary Cycle*
EDF	*Earliest Deadline First*
FCFS	*First Come First Served*
FCS	*Frame Check Sequence*
FTT	*Flexible Time Triggered*
FTT-SE	*Flexible Time Triggered - Switched Ethernet*
GARP	*Generic Attribute Registration Protocol*
HaRTES	*Hard Real-Time Ethernet Switch*
IETF	*Internet Engineering Task Force*
IP	*Internet Protocol*
IRT	*Isochronous Real-Time*
LAN	*Local Area Network*
LEC	*Length of Elementary Cycle*
LSF	*Least Slack First*
lsw	*Length of the synchronous window*
LTM	*Length of Trigger Message*
MAC	*Media Access Control*
MAD	*MRP Attribute Declaration*
MAP	*MRP Attribute Propagation*
MMRP	*Multiple MAC Registration Protocol*
MRPDU	*Multiple Registration Protocol Data Unit*
MSRP	*Multiple Stream Reservation Protocol*
MSRPDU	*Multiple Stream Reservation Protocol Data Unit*
MTU	*Maximum Transmission Unit*
MVRP	*Multiple VLAN Registration Protocol*
OPWA	*One Pass With Advertising*
OSI	*Open Systems Interconnection*

PCP	*Priority Code Point*
PDU	*Protocol Data Unit*
PS	*Polling Server*
QdS	*Qualidade de Serviço*
RM	*Rate Monotonic*
RSVP	*Resource Reservation Protocol*
SOF	*Start of Frame*
SRP	*Stream Reservation Protocol*
SS	*Sporadic Server*
ST	*Stream Protocol*
TAT	*Turn Around Time*
TM	*Trigger Message*
VLAN	*Virtual Local Area Network*

1 Introdução

O passado recente viveu momentos de grande crescimento e inovação na área da eletrónica. Tal resultou num aumento exponencial do número de equipamentos eletrónicos, miniaturização dos mesmos, crescente aumento da capacidade computacional e redução de custos. Isto conduziu a um aumento do número de sistemas embutidos e das suas capacidades. A necessidade de lidar com aplicações cada vez mais complexas levou à adoção de soluções compostas por diversas nós, interligados por redes de dados, o que levou à definição de um novo conceito, o de sistemas embutidos distribuídos. Estes caracterizam-se pela existência de uma relação de simbiose entre as diferentes estações de modo a atingir um objetivo comum. A existência de diferentes estações que mantêm uma interação constante leva a que a comunicação entre elas tenha que ser feita de forma eficiente, segura e garantida.

Os sistemas embutidos distribuídos podem ser encontrados, no dia-a-dia, numa vasta lista de áreas, tais como automação, processos industriais, aeronáutica e indústria automóvel. Muitas destas aplicações exigem garantias temporais nos seus processos. Os sistemas que apresentam restrições temporais são denominados Sistemas de Tempo-Real e exigem às suas redes de comunicação iguais garantias, ou seja, impõem às redes de comunicação exigentes níveis de Qualidade de Serviço (QdS).

Durante muitos anos a tecnologia dominante neste tipo de comunicações foi o uso de redes de campo especializadas, denominadas vulgarmente como *fieldbuses*, que garantem o cumprimento dos requisitos temporais das aplicações de Tempo-Real.

A *Ethernet* tem vindo a emergir na área das comunicações Tempo-Real, devido ao facto de ser, atualmente, uma tecnologia amplamente conhecida e desenvolvida, apresentar baixos custos de implementação e manutenção, permitir elevado débito de dados e, portanto, um melhor desempenho. Existe ainda uma grande massificação nas redes de dados comummente usadas. Por conseguinte, e apesar de inicialmente esta não apresentar as características mais adequadas a aplicações de Tempo-Real, nomeadamente e principalmente devido à sua imprevisibilidade e indeterminismo no acesso ao meio, foram sendo desenvolvidas algumas soluções que a permitiram adaptar-se às necessidades das comunicações de Tempo-Real. A grande fonte de indeterminismo da *Ethernet* tradicional encontra-se na forma como esta permite o acesso ao meio por parte dos seus participantes para transmissão de dados. O *Carrier Sense Multiple Access / Collision Detected* (*CSMA/CD*) pode levar a que, para que uma mensagem seja transmitida, múltiplas

tentativas sejam feitas, levando a que haja um tempo de resposta elevado e não limitado, ou seja, um grande indeterminismo associado.

O aparecimento da tecnologia *switched-Ethernet* permitiu oferecer múltiplos domínios de colisão (ao contrário da *shared-Ethernet)*, diminuindo, portanto, o indeterminismo devido ao *CSMA/CD* já que as diferentes portas de um *switch* não se encontram diretamente ligadas.

O sistema de transmissão de mensagens num *switch* é baseado em filas e portanto, caso uma porta se encontre ocupada, a mensagem aguarda em fila até poder ser transmitida. Isto pode não ser aceitável para aplicações de Tempo-Real que exijam garantias temporais rigorosas. Assim, muitos *switches* passaram a disponibilizar múltiplas filas com diferentes níveis de prioridade. Contudo, este número é limitado a 8 e insuficiente dadas as necessidades de Tempo-Real. Adicionalmente, os *switches* mais recentes, apesar de serem suficientemente rápidos a processar as mensagens recebidas nas portas de entrada, não adicionando, portanto, atraso suplementar à transmissão dos dados, podem não ser capazes de ser igualmente rápidos no processamento de mensagens nas portas de saída. Tal pode levar a situações de sobrecarga e consequentemente aumento de atraso de transmissão ou até perda de mensagens.

Para controlar as deficiências dos *switches* tradicionais, diversas soluções surgiram no mercado. Algumas são baseadas em *switches* comuns e recorrem a modificações nas estações terminais enquanto outras são baseadas em *switches* modificados. Entre as várias soluções apresentadas destacaram-se o *FTT-SE* [1] e o *Ethernet Powerlink* [2] para a primeira categoria, e o *EtheReal* [3], *EDF switched Ethernet* [4], *TTEthernet*, *Profinet-IRT* [5] ou *AFDX* [6] para a segunda categoria. Estas, apesar de oferecerem garantias temporais tal como o requerido pelos Sistemas de Tempo-Real, são soluções essencialmente estáticas e baseadas em análises feitas antes da execução do sistema, ou seja, em soluções que não são capazes de dinamicamente adaptarem o seu funcionamento às necessidades instantâneas de Qualidade de Serviço requeridas pela aplicação.

Para colmatar estas limitações, foi desenvolvido na Universidade de Aveiro, um *switch* modificado baseado no paradigma *Flexible Time Triggered (FTT)*, capaz de fornecer garantias de *QdS*, maximizando em cada instante, os recursos disponíveis na rede.

No entanto, a reserva de recursos, essencial para obtenção de garantias de QdS numa rede, é garantida no *switch* com recurso a comunicações específicas e desenvolvidas localmente para o efeito. Não suporta, portanto, qualquer norma universal, o que lhe dificulta a entrada no mercado.

Este mercado pode ir desde a indústria automóvel à aeronáutica. No mercado automóvel, o mais atraído, devido ao interesse em agrupar as diferentes redes de dados existentes num carro (desde redes multimédia às redes que interligam a mecânica do carro) numa única rede global, capaz de oferecer as mesmas garantias, qualidade e segurança, e ao mesmo tempo capaz de reduzir recursos e consequentemente custos.

Para responder a estas necessidades, surgiu o grupo *Audio/Video Bridging* [7] o qual adicionou diversas normas *IEEE* de forma a permitir às redes *Ethernet* obter baixas latências na transmissão de dados. Estas normas foram normalizadas no *IEEE AVB Standard* e apresentam mecanismos que, apesar de ainda estarem em desenvolvimento, apresentam-se como soluções emergentes e praticáveis.

Um desses mecanismos traduz como deve ser feita a reserva de recursos numa rede de forma a garantir a Qualidade de Serviço requerida por uma aplicação. Este mecanismo é denominado de *Stream Reservation Protocol (SRP)* e está normalizado na norma *IEEE 802.1Qat [8]*.

Além deste mecanismo para a sinalização de reserva de recursos, outros já tinham sido desenvolvidos ao longo dos últimos anos com o mesmo intuito. Entre esses destaca-se o RSVP, um mecanismo de sinalização que permite às estações terminais anunciarem as características do fluxo de dados que pretendem transmitir, permitindo aos restantes elementos da rede reservarem recursos de acordo com as necessidades da aplicação.

No âmbito desta dissertação, pretende-se estudar as características destes dois protocolos de sinalização de reserva de recursos, *RSVP* e *SRP*, comparar com os requisitos funcionais do *switch HaRTES* e avaliar a possibilidade de integração destes no *HaRTES*.

Adicionalmente, pretende-se os analisar requisitos do *SRP*, arquitetura, blocos funcionais e mensagens trocadas, e implementar alguns destes blocos.

1.1 Organização do documento

No capítulo 1 é feita uma contextualização do tema à volta do qual a dissertação se insere, quais as motivações que levaram ao desenvolvimento da mesma e quais os objetivos do trabalho.

Com o capítulo 2 pretende-se instruir o leitor com a informação básica e essencial à compreensão do resto do documento. Numa primeira fase abordam-se alguns dos conceitos básicos de Sistemas de Tempo-Real, as suas características e requisitos, para

numa segunda fase se introduzirem as comunicações de Tempo-Real e a *Ethernet*. Aqui, apresentam-se alguns protocolos capazes de adequar a *Ethernet* às necessidades de Tempo-Real.

A introdução aos mecanismos de reserva de recursos é feita no capítulo 3. O *RSVP* e *SRP* são descritos, mais concretamente as suas características, funcionamento e mensagens que usam. Por fim é estudada e comparada a interoperabilidade de ambos com o *switch HaRTES*.

O capítulo 4 é dedicado à implementação de alguns dos blocos e mensagens funcionais do *SRP.*

Os testes efetuados para validação do trabalho são apresentados no capítulo 5 bem como os resultados obtidos e sua discussão.

Por fim, no capítulo 6 são apresentadas as conclusões deste trabalho e deixadas algumas sugestões para trabalho futuro.

2 Conceitos básicos: Sistemas de Tempo-Real e *Ethernet* de Tempo-Real

2.1 Sistemas de Tempo-Real

Nos dias de hoje seria impensável imaginar o nosso dia-a-dia sem a existência dos mais variados sistemas computacionais. Dos mais simples aos mais complexos, estes acompanham-nos em todos os aspetos do dia a dia e contribuem para uma melhor qualidade de vida de toda a sociedade. Muitos destes sistemas mantém uma interação, muitas vezes complexa, com o meio envolvente e, por conseguinte, apresentam características especiais e distintivas.

Destes, existem aqueles que devem reagir a eventos do meio envolvente dentro de determinados intervalos de tempo, aos quais se dá o nome de Sistemas de Tempo-Real. Existem inúmeros exemplos de aplicações de Tempo-Real, que vão desde o controlo de complexos processos de produção, aquisição de dados e monitorização, automação e robótica, até sistemas de aviação, sistemas de comutação de linhas ferroviárias e missões espaciais [9].

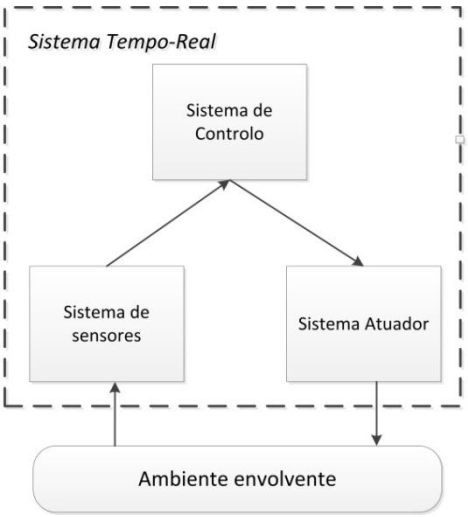

Figura 2.1: Modelo genérico de um Sistema de Tempo-Real (baseado em [9])

A Figura 2.1 ilustra um modelo simplificado da arquitetura de um Sistema de Tempo-Real. O sistema de sensores permite captar informação relevante do meio envolvente (ex. temperatura, humidade, movimento), que é tratada pelo sistema de controlo no sen-

tido de fazer reagir corretamente o sistema atuador com vista a atingir um determinado objetivo.

Contudo, em Sistemas de Tempo-Real não interessa só que estes respondam corretamente a qualquer estímulo do meio envolvente mas, também, o instante de tempo em o fazem. Qualquer atraso na resposta do sistema pode resultar em informação sem qualquer utilidade ou até em falhas catastróficas que podem por em causa vidas humanas ou causar danos patrimoniais elevados.

De acordo com as consequências do não cumprimento das restrições temporais, os Sistemas de Tempo-Real podem ser classificados como *Hard Real-Time,* quando as consequências são catastróficas, isto é, quando envolvem perda de material ou põem em perigo vidas humanas, e *Soft Real-Time* quando há degradação do desempenho sem que as consequências sejam tão catastróficas.

Os Sistemas de Tempo-Real, de forma a suportar aplicações críticas de Tempo-Real, têm, como foi referido, propriedades distintivas dos restantes sistemas computacionais. Segundo *Butazzo* [9] destacam-se:

- **Pontualidade e correção lógica:** os resultados das computações devem ser logicamente corretos e cumprir as restrições temporais;
- **Cenários de pior caso:** os sistemas devem ser desenhados tendo em conta o caso de carga máxima, ou seja, devem ser capazes de responder corretamente em situações limites;
- **Tolerante a falhas:** o sistema deve ser capaz de suportar falhas pontuais;
- **Previsibilidade**: o sistema deve ser capaz de prever as consequências de determinada ação de forma a evitar resultados inesperados e permitir ao sistema procurar soluções alternativas.

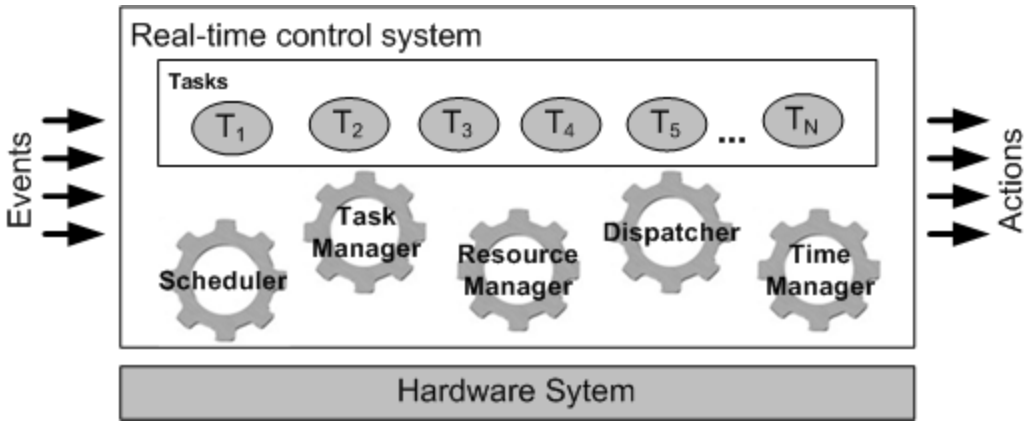

Figura 2.2: Arquitetura genérica de um Sistema Operativo de Tempo-Real [10]

A arquitetura de um Sistema Operativo de Tempo-Real é retratada, genericamente, na Figura 2.2. Nela é possível identificar diferentes componentes que, em conjunto, permitem ao sistema responder com correção lógica e temporal. As tarefas (*tasks)* são a unidade básica de um Sistema de Tempo-Real e podem ser definidas como sendo uma "Sequência de ativações (instâncias ou *jobs*), cada uma composta por um conjunto de instruções que, na ausência de outras atividades, é executada pelo CPU sem interrupção" [11]. O gestor de tarefas *(task manager)* é responsável pela criação, atualização e término das tarefas. No entanto, quando existe um conjunto de tarefas concorrentes, o número de processadores pode não ser suficiente para executar paralelamente todas as tarefas e portanto é necessário um escalonamento. Este escalonamento é feito por um escalonador (*scheduler*) e o conjunto de regras que determina a ordem de execução das tarefas é denominado algoritmo de escalonamento.

Existem diversos algoritmos de escalonamento, dos quais alguns dos mais relevantes serão abordados mais à frente neste texto. A tarefa selecionada pelo escalonador para execução é posta em execução pelo *dispatcher.* Como vulgarmente existe a partilha de recursos entre as diferentes tarefas é necessário haver uma entidade que controle e monitorize essa partilha. Essa entidade é denominada de gestor de recursos (*resource manager)* e garante que o acesso aos recursos partilhados se efetua com exclusividade.

Nos Sistemas de Tempo-Real existe ainda uma entidade responsável pela ativação de tarefas periódicas, medição de intervalos de tempo e verificação do cumprimento ou não das restrições temporais das tarefas em execução.

2.1.1 Tarefas

Sendo as tarefas unidades essenciais e centrais num Sistema de Tempo-Real, torna-se imprescindível o seu estudo e caracterização. Cada uma das tarefas implementa uma função específica no sistema e de acordo com a sua periodicidade podem ser classificadas como periódicas, esporádicas e aperiódicas.

As tarefas periódicas são caracterizadas por um período (T) bem definido correspondente ao tempo entre duas ativações consecutivas de instâncias da mesma tarefa. Já uma tarefa esporádica é caracterizada por um tempo mínimo entre ativações consecutivas (*mit),* enquanto numa tarefa aperiódica os instantes de ativação não são regulares e portanto apenas possíveis de caracterizar de forma probabilística.

Para que o escalonamento de tarefas seja corretamente efetuado e para que o comportamento do sistema seja previsível, é necessário conhecer algumas características

essenciais das tarefas que compõem determinado sistema. Assim, segundo *Butazzo* [9] podemos encontrar as seguintes características (Figura 2.3):

- **a_i**: instante de ativação da instância i;
- **C_i:** tempo máximo de execução;
- **T_i:** período (no caso de tarefas periódicas);
- **Φ_i:** fase relativa da primeira instância (no caso de tarefas periódicas);
- **mit_i:** tempo mínimo entre ativações consecutivas (*minimum interarrival time)* no caso de tarefas esporádicas;
- **s_i:** instante no qual a tarefa inicia a sua execução;
- **f_i:** instante no qual a tarefa termina a sua execução;
- **$c_i(t)$**: tempo de execução residual da instância i no instante t.

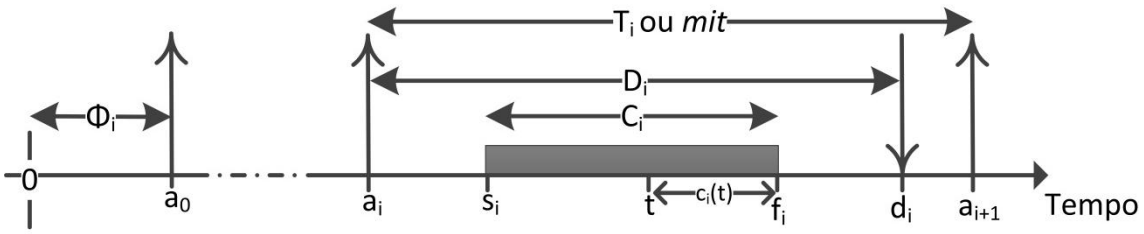

Figura 2.3: Caracterização de uma tarefa [11]

As tarefas podem apresentar diferentes tipos de restrições: temporais, de precedência e exclusão mútua em recursos partilhados. As restrições de precedência, tal como o nome indica, impõem condições na ordem de execução das tarefas. Em algumas aplicações determinadas tarefas apenas podem ser executadas após o término de outras e, portanto, tal deverá ser tomado em conta no escalonamento. A existência de recursos que podem ser partilhados por diversas tarefas mas que não permitem acesso simultâneo, problema denominado de exclusão mútua em recursos partilhados, representa mais uma restrição no escalonamento de tarefas.

As restrições temporais, que se apresentam como o grande elemento distintivo entre os Sistemas de Tempo-Real e os restantes, estabelecem limites temporais e vêm, usualmente, sob a forma de *deadlines*. O valor da *deadline* identifica o instante de tempo antes do qual a tarefa deve terminar a sua execução. Caso o instante de tempo seja relativo ao instante de ativação da tarefa, diz-se *deadline* relativa (d_r). Caso o instante de tempo seja em relação ao tempo zero, a *deadline* diz-se absoluta (D_r).

Não obstante o facto de uma tarefa, τ_i, poder ser completamente caracterizada através de C_i, Φ_i,T_i e D_r no caso de uma tarefa periódica e através de C_i, mit_i e D_r no caso de

uma tarefa esporádica, outros parâmetros podem ser usados para caracterizar uma tarefa:

- **R_i: tempo de resposta,** isto é, tempo que decorre entre o instante em que a tarefa é ativa e o instante de tempo que a tarefa termina ($R_i=f_i - a_i$);
- **L_i: atraso ou *lateness***, isto é, o atraso entre o término da tarefa e a sua *deadline* absoluta ($L_i=f_i-d_i$);
- **E_i: tempo de excesso ou *tardiness***, representa o tempo no qual a tarefa se mantem ativa após a sua *deadline* ($E_i = \max(0,L_i)$);
- **X_i: folga ou *laxity*,** é o atraso máximo que o instante inicial de uma tarefa pode sofrer sem comprometer o cumprimento da sua *deadline (*$X_i=d_i-a_i-C_i$);

2.1.2 Algoritmos de escalonamento

Depois da apresentação e breve descrição dos parâmetros tipicamente usados para caracterizar uma tarefa, a subsecção que agora se inicia descreverá, ainda que superficialmente, alguns dos mais conhecidos algoritmos de escalonamento, tanto para tarefas periódicas como para tarefas aperiódicas.

Um escalonador permite, como já foi referido, ordenar as diferentes tarefas prontas a executar. A natureza da solução do problema de escalonamento define diferentes tipos de algoritmos de escalonamento. Esta ordenação é particularmente importante em Sistemas de Tempo-Real já que o escalonamento deve permitir a todas as tarefas cumprir todas as suas restrições (temporais, de precedência e de recursos).

Um escalonamento diz-se praticável caso cumpra todas as restrições impostas pelas diferentes tarefas e um conjunto de tarefas diz-se escalonável caso exista no mínimo um escalonamento que seja praticável.

Existem diferentes categorias de algoritmos de escalonamento, consoante se baseiem em parâmetros fixos ou dinâmicos. Os parâmetros que se mantenham inalterados denominam-se estáticos e os que se alterem durante a execução da tarefa, são chamados de dinâmicos. Caso o escalonamento seja feito antes do início do sistema, o escalonamento é denominado *off-line*. Já no caso em que o escalonamento é avaliado e determinado durante a execução do sistema, está-se perante um escalonamento *on-line*.

O escalonamento pode também permitir preempção ou não. Caso o permita, o escalonamento é avaliado assumindo que a execução dos processos pode ser interrompida temporariamente sem afetar o correto funcionamento do sistema.

2.1.2.1 Escalonamento de tarefas periódicas

No que se refere ao escalonamento *on-line* de tarefas periódicas podem identificar-se, essencialmente, cinco diferentes algoritmos: *Rate-Monotonic* (*RM*), *Deadline-Monotonic* (*DM*), *Earliest Deadline First* (*EDF*), *Least Slack First* (*LSF*) e *First Come First Served* (*FCFS*). Destes, o critério *RM* e *DM* dizem respeito a algoritmos com prioridades fixas enquanto os restantes usam prioridades dinâmicas.

No caso do *RM,* as prioridades são atribuídas de acordo com o seu período, sendo a maior prioridade conferida ao processo com menor período. A verificação da escalonabilidade é efetuada assumindo que as *deadlines* das tarefas são iguais aos seus períodos[1], assunção esta nem sempre verificada em aplicações de Tempo-Real. Já o *DM* define a prioridade das tarefas como sendo inversamente proporcional às suas *deadlines* relativas e o teste de escalonabilidade é estendido a tarefas com D<T.

Quando o algoritmo usado é o *EDF*, a ordenação das tarefas é feita de acordo com distância temporal para a *deadline*. Assim, a próxima tarefa a ser executada (a tarefa com maior prioridade) pelo processador é aquela que em determinado instante (ex. ativação ou término de uma instância) apresentar menor *deadline* absoluta.

Já no caso do *LSF*, a tarefa com maior prioridade é aquela que apresentar menor tempo de folga (*slack)* enquanto no *FCFS* a tarefa com maior prioridade é aquela que se encontra há mais tempo à espera de ser executada.

2.1.2.2 Escalonamento de tarefas aperiódicas

Os algoritmos de escalonamento previamente apresentados e associados a tarefas periódicas podem também ser estendidos ao caso de tarefas esporádicas, considerando-se o cenário de pior caso, em que as tarefas ativam-se a cada *mit*, tendo assim um comportamento semelhante às tarefas periódicas. Já quando são tarefas aperiódicas, estas podem interferir de uma forma não controlada no escalonamento das restantes tarefas, o que não é aceitável. Na prática, a grande maioria dos sistemas de Tempo-Real lidam com conjuntos de tarefas heterogéneo, podendo estas ser periódicas, esporádicas ou aperiódicas. Adicionalmente, as tarefas periódicas estão frequentemente associadas a aplicações de controlo com restrições temporais rígidas (do tipo *hard*) enquanto as tarefas aperiódicas aparecem associadas a restrições do tipo *hard*, *soft* ou até *non-real*. Consequentemente, o sistema deve ser capaz de escalonar as tarefas de modo a dar garan-

[1] Os testes de escalonabilidade, suas condições e resultados estão fora do âmbito desta dissertação, sendo apenas relevante a identificação e exposição dos principais algoritmos de escalonamento. Para mais detalhes consultar Buttazzo, G.C., *Hard Real-Time Computing Systems : Predictable Scheduling Algorithms and Applications*, 2005, Nueva York, EUA : Springer.

tias de resposta de pior caso às tarefas de tipo *hard* e ao mesmo tempo garantir o melhor tempo de resposta possível (*best-effort*) às restantes tarefas do tipo *soft* ou *non-real*.

Uma forma simples de conciliar ambos os tipos de tarefas, periódicas e aperiódicas, consiste em executar as tarefas aperiódicas em plano de fundo (*background*). Neste caso é dada máxima prioridade às tarefas periódicas, sendo que as tarefas aperiódicas apenas são executadas quando não há nenhuma tarefa periódica pronta a executar. Este tipo de escalonamento, apesar de adequado para tarefas periódicas críticas, tem, em caso de um elevado número de tarefas periódicas, um desempenho muito fraco para tarefas aperiódicas de Tempo-Real. Assim, é apenas adequado para casos em que tarefas aperiódicas do sistema não apresentem restrições temporais ou haja um pequeno número de tarefas periódicas.

Para colmatar a deficiência apresentada pelo escalonamento em *background* e assim melhorar o tempo de resposta das tarefas aperiódicas, introduziu-se o conceito de servidor. Um servidor é, neste caso, uma pseudo-tarefa periódica destinada unicamente à execução de tarefas aperiódicas. Estes servidores são, tal como uma tarefa periódica, caracterizados por um período (T_s) e uma capacidade (C_s) e, como tal, o escalonamento pode ser feito através dos mesmos algoritmos usados para tarefas periódicas. A prioridade atribuída aos servidores pode ser escolhida de acordo com as necessidades das diferentes tarefas periódicas e aperiódicas. O escalonamento das diferentes tarefas periódicas (incluindo servidores) não influencia a ordenação dos diferentes pedidos de tarefas aperiódicas, podendo esta ordenação ser feita segundo um critério pré-definido pelo sistema.

Existem diferentes implementações de servidores aperiódicos, podendo estes ser divididos em dois grandes grupos: servidores de prioridades fixas e servidores de prioridades dinâmicas. No que se refere a servidores de prioridades fixas, destacam-se os servidores de sondagem (*polling*), os servidores adiáveis (*deferrable*) e os servidores esporádicos (*sporadic*), que serão apresentados seguidamente. Já o estudo dos servidores de prioridades dinâmicas está fora do âmbito desta dissertação e como tal não serão descritos[2].

Servidor de sondagem (*Polling server – PS)*

Um servidor de sondagem é, tal como todos os restantes servidores, caraterizado por um período (T_s) e uma capacidade (C_s). No início de cada ciclo periódico, o servidor verifica se há alguma tarefa aperiódica na fila para execução. Caso exista, essa tarefa é

[2] Para mais detalhes neste tema aconselha-se bibliografia adicional como Buttazzo, G.C., *Hard Real-Time Computing Systems : Predictable Scheduling Algorithms and Applications*, 2005, Nueva York, EUA : Springer.

executada até esgotar a capacidade do servidor ou, caso a capacidade do servidor seja suficiente para execução da tarefa, até ao término da mesma.

A capacidade do servidor é reposta no início de cada período de *polling*. No entanto, caso a fila se encontre vazia, o servidor suspende de imediato a sua execução e só volta a estar ativo no próximo período. Este tempo é usado para execução das tarefas periódicas. Daqui se pode concluir que este tipo de servidor beneficia o tempo de resposta de tarefas periódicas. A Figura 2.4 descreve o escalonamento segundo o critério *RM* de um sistema composto por um servidor de sondagem (T_s=2.5 e C_s=0.5) e duas tarefas periódicas (T_1= 3, C_1=1, T_2= 10 e C_2=4).

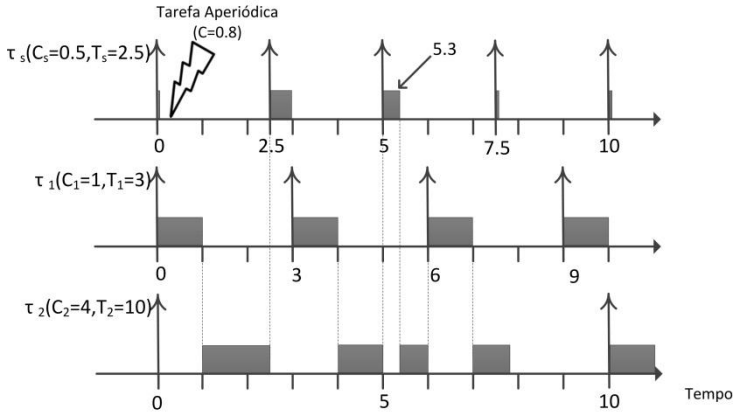

Figura 2.4: Exemplo de um servidor de sondagem com escalonamento *RM* (adaptado a partir de [12])

Servidor adiável (*Deferrable server – DS)*

O uso de servidores aperiódicos pode resultar em tempos de resposta demasiado elevados para tarefas aperiódicas. Quando um pedido aperiódico surge após o escalonamento do servidor, este só poderá ser atendido no próximo ciclo do servidor. Isto acontece mesmo nos casos em que a capacidade do servidor desse ciclo ainda não tenha sido esgotada, o que resulta numa baixa eficiência e elevados tempos de resposta.

Os servidores adiáveis permitem colmatar essa deficiência com um custo na escalonabilidade das tarefas periódicas. A grande diferença a nível de funcionamento do servidor de sondagem e do servidor adiável reflete-se no facto de nos servidores adiáveis a sua capacidade ser preservada até ao fim do seu período ou até ser totalmente consumida.

Todavia torna-se possível o aumento temporário de carga do servidor sobre a tarefa de menor prioridade, pelo que a escalonabilidade desta é prejudicada.

A Figura 2.5 ilustra um exemplo de um servidor adiável.

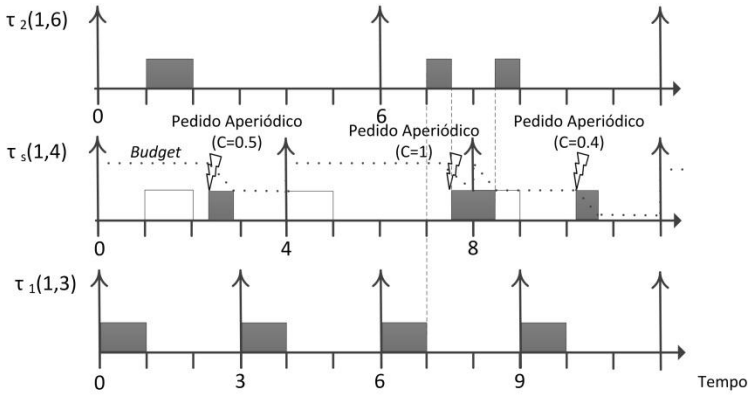

Figura 2.5: Exemplo de um servidor adiável com escalonamento *RM* (adaptado a partir de [11])

Servidor esporádico (*Sporadic server – SS)*

Um servidor esporádico permite agregar o melhor do servidor de sondagem e do servidor adiável, permitindo, nomeadamente responder a pedidos de tarefas aperiódicas em qualquer instante sem que a escalonabilidade das tarefas periódicas seja prejudicada. Neste caso a capacidade do servidor é reposta dependendo do instante de tempo em que foi efetivamente consumida. Isto é, ao invés do caso dos servidores de sondagem e adiáveis em que a capacidade do servidor era resposta periodicamente no início do período do servidor, neste caso o momento de reposição da capacidade é agendado no instante em que a capacidade é consumida. Este tipo de servidores apresenta, contudo, a desvantagem de serem mais complexos e difíceis de implementar. A Figura 2.6 ilustra um exemplo de funcionamento de um servidor esporádico.

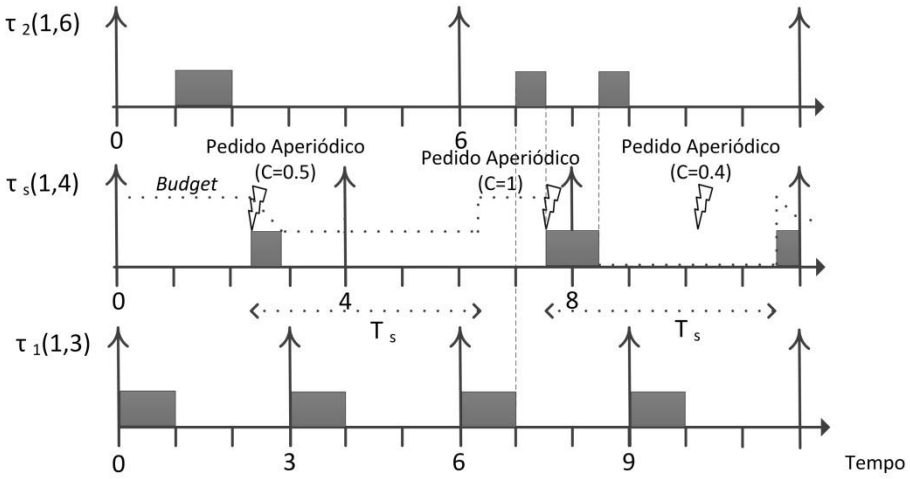

Figura 2.6: Exemplo de um servidor esporádico com escalonamento *RM* (adaptado a partir de [11])

2.2 Comunicações Tempo-Real e *Ethernet*

O contínuo desenvolvimento tecnológico e o aumento da exigência das aplicações de Tempo-Real, levou a que os sistemas se tornassem distribuídos. Por conseguinte, o cumprimento de *deadlines* passou a depender também das redes de dados. Para este tipo de sistemas, a condição mais relevante passou a ser o cumprimento das restrições temporais das aplicações e a largura de banda fornecida às aplicações.

A necessidade de existência de um alto nível de previsibilidade e pré-determinismo nas comunicações de Tempo-Real manteve afastada, até muito recentemente, a *Ethernet* como solução viável [13]. Este tipo de comunicações era, e ainda é em alguns casos, asseguradas através de sistemas *fieldbuses*.

Diversos standards foram desenvolvidos neste âmbito, entre os quais se podem referir *Interbus [14], Profibus [15] ou CAN [16],* sendo que nunca foram capazes de acompanhar a evolução da *Ethernet* em termos de custos, capacidade de transporte e simplicidade.

Estas vantagens competitivas, aliadas ao facto de ser a tecnologia dominadora em redes locais e consequente presença na grande maioria dos dispositivos computacionais, levou a que se procurasse adaptar a tecnologia *Ethernet* às características e requisitos das comunicações de Tempo-Real.

Nas subsecções seguintes, após um breve abordagem à tecnologia *Ethernet*, introduzir-se-ão alguns protocolos de Tempo-Real sobre *Ethernet* até hoje desenvolvidos no sentido de adequar o comportamento de uma rede *Ethernet* às necessidades das comunicações de Tempo-Real.

2.2.1 *Ethernet*

A *Ethernet* é uma tecnologia de comunicação para redes locais (*LAN's*) que interliga computadores ou outros dispositivos computacionais que residem no mesmo prédio ou numa área delimitada. É baseada no envio de pacotes, atua a nível da camada física e da camada de ligação de dados do modelo *OSI* [17, 18] e foi padronizada como *IEEE* 802.3 [18].

Apesar de ter sido apresentada pela primeira vez em 1976 por *Robert Metcalfe*, esta não se manteve imutável até aos dias de hoje. Ao longo de décadas a sua evolução permitiu passar de velocidades de 10 megabits por segundo para 100 gigabits por segundo.

Este incremento não só de velocidade mas também de robustez e fiabilidade deveu-se sobretudo ao tipo de topologia usada para implementar a rede. Esta topologia define a forma como os diferentes nós da rede se ligam e acedem ao meio.

Inicialmente implementada com uma topologia em barramento e anel, rapidamente provou a sua ineficiência devido ao facto de uma simples falha num dos nós resultar numa falha total da rede. Adicionalmente, é requerido um mecanismo de controlo de acesso ao meio de forma a evitar colisões, nomeadamente, e no caso da *Ethernet, Carrier Sense Multiple Access / Collision Detection (CSMA/CD) [19]*. Este mecanismo apresenta uma eficiência pobre já que requer a retransmissão dos pacotes no caso de colisão o que pode resultar em atrasos imprevisíveis, significativos e proibitivos para sistemas de Tempo-Real.

Este tipo de mecanismos limitava a capacidade de débito da *Ethernet.* Consequentemente, e de forma a evitar as deficiências evidenciadas pelas topologias referidas, passou a usar-se uma topologia em estrela. Esta topologia é caracterizada pela existência de um nó central, ao qual os restantes nós se ligam, permitindo, assim, adicionar e remover nós sem afetar o funcionamento da restante rede.

Os primeiros nós centrais a serem usados nas redes *Ethernet* foram os denominados *hubs*. Estes dispositivos de funcionamento simples, apenas reencaminham os pacotes recebidos para todas as restantes portas. Assim, e devido ao referido método de funcionamento, é mantido um único domínio de colisão, resultando num baixo débito de dados na rede.

Os *switches/bridges,* ao contrário dos *hubs,* permitiram obter múltiplos domínios de colisão, nomeadamente um por cada porta pois estas não se encontram diretamente ligadas. Este modelo permitiu não só aumentar significativamente o débito de dados, mas também obter um maior isolamento de tráfego e por conseguinte aumentar o determinismo em redes *Ethernet.* Este fator conduziu a um ganho de importância deste tipo de redes na área de Tempo-Real e, como tal, conduziu ao desenvolvimento de soluções que permitissem adaptar a tecnologia *Ethernet* às necessidades das comunicações de Tempo-Real.

Antes de se descrever mais detalhadamente o funcionamento da *switched-Ethernet,* torna-se pertinente, no âmbito do trabalho desenvolvido nesta dissertação, descrever o conteúdo de um pacote *Ethernet.*

A Figura 2.7 ilustra os diferentes campos de um pacote *Ethernet II* também conhecido como *DIX Ethernet*. Os oitos primeiros bytes que englobam o campo *preamble* e o campo *Start Of Frame (SOF)* são usados para sincronização da transmissão. Os dois

campos seguintes, constituídos cada um por seis *bytes*, contêm, respetivamente, os endereços MAC do dispositivo físico destinatário e remetente. Por outro lado, o campo *Ethernet Type* identifica de que forma a informação útil (*payload)* está codificada no respetivo campo, isto é, identifica o protocolo da camada superior usado para transportar a informação útil do pacote. O campo *data* transporta a informação útil do pacote, podendo ter entre 46 a 1500 *bytes.* Por fim, o campo *Frame Check Sequence (FCS)* é usado para deteção de possíveis erros.

Figura 2.7: Trama de um pacote de dados *Ethernet II [10]*

2.2.2 *Switched Ethernet*

O número de redes *Switched Ethernet* passou, nos últimos anos, a ser a principal tecnologia usada para comunicações *Ethernet* devido às suas vantagens a nível de débito, isolamento de tráfego e determinismo. Assim, é importante perceber de que forma este modelo permite obter as referidas vantagens.

O ponto fulcral do seu funcionamento está no facto de permitir obter domínios privados de colisão em cada porta, algo que não acontecia com os *hubs.* Deste modo, o pacote recebido numa porta é transmitido apenas para a porta que está ligada ao respetivo destinatário do pacote. Para isto ser possível, os *switches* possuem um mecanismo próprio de reencaminhamento.

Este mecanismo consiste essencialmente numa tabela de reencaminhamento existente internamente a cada *switch* e que guarda as associações entre os endereços *MAC* dos nós da rede e as portas dos *switch* às quais estes estão ligados. Por conseguinte, quando um pacote é recebido numa porta, caso o endereço destino do pacote esteja na tabela, o pacote é apenas reencaminhado para a respetiva porta associada. Caso contrário, o pacote é enviado em *broadcast* para todas as portas exceto a que recebeu o pacote.

A tabela pode ser construída estaticamente pelo utilizador ou dinamicamente preenchida pelo próprio *switch.* Neste último caso, sempre que um pacote é recebido numa porta, o endereço do remetente da mensagem é guardado automaticamente juntamente com a identificação da porta.

Contudo, e apesar da *Switched Ethernet* permitir maiores velocidades de transmissão, apresenta ainda algumas deficiências, não sendo suficiente para dar garantias de Tempo-Real [20]. Para se perceberem quais, repare-se, através da Figura 2.8, na arquitetura típica de um *switch.*

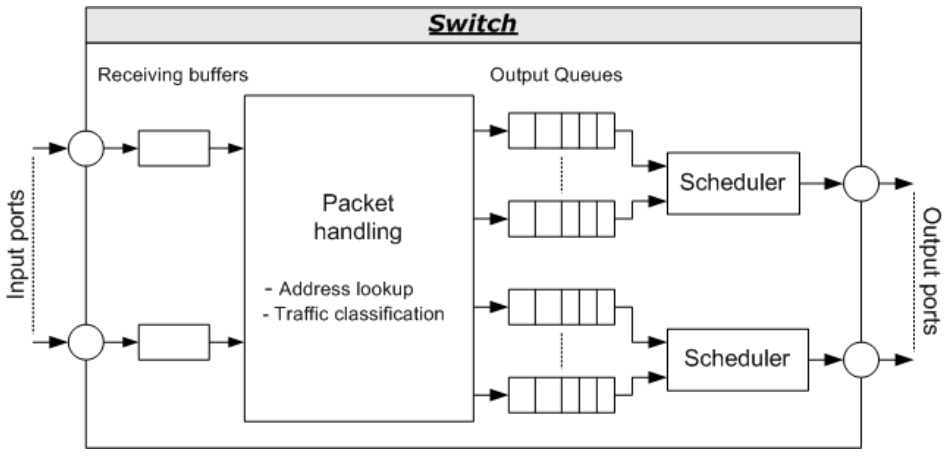

Figura 2.8: Arquitetura genérica de um *switch [1]*.

Quando uma mensagem é recebida numa porta do *switch,* é colocada num *buffer* de receção, sendo posteriormente analisado o campo do endereço destino por forma a ser transmitida para o *buffer* da porta de saída conectada ao destinatário da mensagem.

Nos casos em que a porta está ocupada a transmitir uma mensagem, a recém chegada mensagem fica em fila à espera de oportunidade de transmissão. De modo a evitar o bloqueio de mensagens de maior prioridade por parte de mensagens de menor prioridade, alguns *switches* disponibilizam diferentes filas com diferentes níveis de prioridade. No entanto, esse número é limitado a 8, sendo insuficiente para o correto funcionamento de sistemas de Tempo-Real.

Os *switches* atualmente disponíveis no mercado conseguem ser suficientemente rápidos a despachar as mensagens recebidas nas portas de entrada e, portanto, estas não provocam atrasos na entrega de mensagens. Porém, se um grande número de mensagens com o mesmo destinatário chegar num curto espaço de tempo, podem ser geradas filas nos *buffers* de saída. Em casos extremos, estas filas podem resultar em situações de sobrecarga e consequente perda de mensagens.

Em suma, apesar da tecnologia *Switched Ethernet* não ser suficiente para dar garantias de Tempo-Real, as suas características proporcionaram a oportunidade do surgimen-

to de implementações mais eficientes para comunicações *Ethernet* de Tempo-Real. Algumas dessas implementações serão apresentadas de seguida.

2.2.3 Protocolos Tempo-Real sobre *Ethernet*

No sentido de melhorar o comportamento da *Ethernet* em comunicações de Tempo-Real, algumas técnicas e protocolos foram desenvolvidos. As soluções desenvolvidas podem dividir-se em dois grandes grupos. Um grupo onde se utilizam *Switches Ethernet* disponibilizados no mercado, denominados *Comercial-Of-The-Shelf (COTS) Ethernet Switches,* em que as alterações são efetuadas sobretudo nos nós da rede; um outro grupo em que as soluções são baseadas em *switches* modificados.

Neste segundo grupo as soluções apresentadas baseiam-se essencialmente na introdução no *switch* de capacidade de controlo de tráfego e escalonamento. Entre as diversas soluções desenvolvidas destacam-se *EDF Switched Ethernet [4]* que implementa um escalonamento *EDF* e confere controlo de admissão *on-line*, *EtheReal [3]* em que os serviços são implementados apenas nos *switches* e acedidos pelos nós terminais através do recurso a librarias específicas, e *Profinet-IRT [5],* especialmente desenvolvida para automação industrial que implementa ciclos de curta dimensão temporal englobando *slots* temporais para tráfego de Tempo-Real e tráfego de não Tempo-Real.

Já no que diz respeito às soluções baseadas em *switches COTS*, evidenciam-se protocolos como o ETHERNET Powerlink [2] e o *Flexible Time Triggered – Switched Ethernet* (*FTT-SE) [1].* Estes caracterizam-se como tendo uma arquitetura *Master-Slave,* onde o *Master* controla toda a transmissão de dados em cada ciclo definido em cada protocolo.

Flexible Time Triggered – Switched Ethernet (FTT-SE)

Dos protocolos apresentados, o *FTT-SE* evidencia características adequadas e favoráveis ao seu uso como suporte para comunicações de Tempo-Real. Estas resultam do paradigma *FTT* e de acordo com Pedreiras e Almeida [21] o *FTT-SE* permite: controlo temporal do tráfego, admissão de controlo *on-line*, gestão dinâmica de *QdS*, suporte de diferentes tipos de escalonamento do tráfego síncrono e suporte de diferentes tipos de tráfego (real e não real).

Este protocolo segue uma arquitetura *Multi-Slave* sendo que o nó master está conectado a uma das portas e usa sinais de controlo de forma a garantir oportunidade de

transmissão a todas as mensagens mantendo a integridade do sistema, isto é, o *master* é responsável por controlar o acesso ao meio e executar o escalonamento das tarefas.

A transmissão de dados está organizada em ciclos, denominados ciclos elementares (*EC*), onde se identificam três janelas distintas de tempo (Figura 2.9). No início do ciclo o *Master* envia uma mensagem de controlo para todos os *slaves* (*Trigger Message – TM*) permitindo sincronizar toda a rede e transportando a informação sobre o escalonamento desse ciclo elementar. Essa mensagem é recebida e descodificada por todos os *slaves* que assim ficam a saber se e quando podem transmitir as suas mensagens dentro do respetivo ciclo elementar. Após a mensagem *TM*, existe um certo período, denominado *Turn Around Time (TAT)* associado ao tempo de descodificação e processamento da mensagem *TM*.

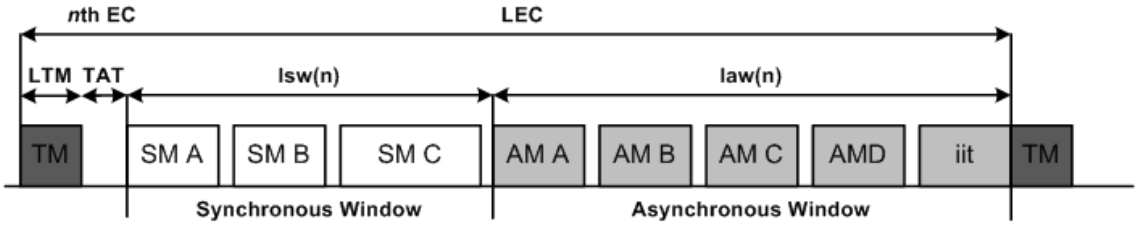

Figura 2.9: Ciclo elementar (*EC)* do *FTT-SE [10]*

Posteriormente seguem-se duas janelas de tempo destinadas à transmissão de tráfego síncrono e tráfego assíncrono por esta ordem. O escalonamento da janela síncrona do ciclo é transportado pela mensagem *TM*. Já no caso da janela assíncrona, sempre que um *slave* deseje enviar uma mensagem assíncrona, necessita de informar o *master* através do envio de uma mensagem paralela à mensagem *TM*. O tráfego de não Tempo-Real é considerado um caso especial de tráfego não assíncrono, sendo transmitido na janela de tempo assíncrona e com a mais baixa prioridade.

A duração do ciclo elementar (*LEC*), da mensagem de controlo (*LTM*) e o *TAT* são parâmetros fixos do sistema, sendo que a duração da janela síncrona e assíncrona variam de acordo com a quantidade de tráfego existente. Porém, e de modo a garantir um tempo mínimo para a janela assíncrona, é definido uma duração máxima para a janela síncrona (*LSW*).

Este protocolo apresenta, contudo, algumas limitações estruturais. Para que a integridade do sistema não seja posta em causa, todos os nós devem possuir um módulo capaz de garantir que o escalonamento definido em cada *EC* é cumprido por todos os nós. Logo, a existência na rede de algum nó que não cumpra este requisito pode com-

prometer o correto funcionamento de todo o sistema, particularmente o cumprimento das garantias temporais.

Para combater esta lacuna, Santos [10] propôs a introdução do master dentro do *switch*, atribuindo ao *switch* a capacidade de controlar o tráfego. Esta nova solução deu origem a um *switch* modificado denominado *HaRTES (Hard Real Time Ethernet Switch)* que além de manter os atributos do protocolo FTT-SE permite obter uma maior simplicidade de gestão de tráfego assíncrono, uma maior integridade do sistema, a transmissão de mensagens *TM* com maior precisão e a inclusão de nós sem suporte *FTT* sem que todo o serviço Tempo-Real seja posto em causa.

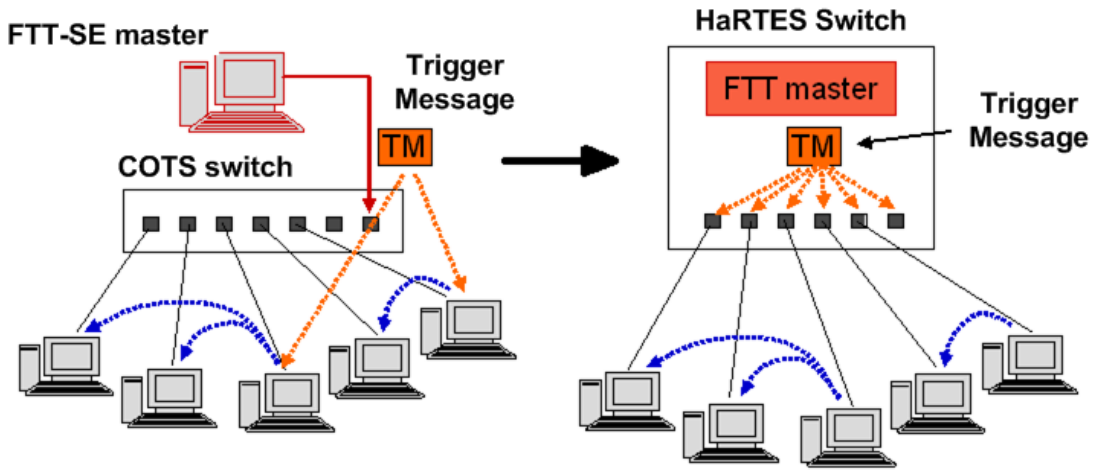

Figura 2.10: *Esquema de rede usando FTT-SE e HaRTES Switch [10]*

Além disso, o *HaRTES* apresenta melhorias significativas no suporte a tráfego assíncrono de Tempo-Real, permitindo uma reconfiguração dinâmica do sistema de acordo com as necessidades das aplicações. Tal foi possível com recurso a uma estrutura de servidores desenvolvida e implementada no *HaRTES* exclusivamente dedicada à gestão da janela assíncrona do ciclo elementar do paradigma *FTT.*

Esta estrutura está organizada hierarquicamente e que pode ser interpretada como uma árvore, em que as *streams* representam o extremo final desta convenientemente suportadas por servidores. Este modelo permite dividir multiplamente a largura de banda destinada à janela assíncrona. A cada ponto da hierarquia, isto é, a cada servidor, está associada uma porção de largura de banda que pode ser partilhada por outros pontos conectados a um nível inferior.

Obtém-se, assim, uma hierarquia multinível em que cada nó pai controla o acesso à largura de banda que tem disponível pelos seus filhos, aumentando a flexibilidade e robustez da janela assíncrona do paradigma *FTT*.

3 Mecanismos de reserva de recursos

No capítulo anterior introduziu-se o conceito de sistemas Tempo-Real, o seu modelo e características. Sendo sistemas cujas aplicações exigem o cumprimento rigoroso dos seus requisitos temporais, muitas vezes para evitar consequências catastróficas, requerem das redes de dados que interligam os diferentes intervenientes do sistema garantias de QdS na transmissão de dados.

A massificação da utilização da tecnologia *Ethernet* como tecnologia de transporte trouxe enormes ganhos em termos de custos de produção de equipamentos, de ferramentas de gestão e *know-how.* Tais ganhos fizeram com que a tecnologia, apesar de não fornecer por si garantias de QdS, fosse considerada para o transporte de informação de aplicações de Tempo-Real. Assim, desenvolveram-se diversas soluções que lhe permitiram responder (ou pelo menos aproximar) às necessidades dos sistemas de Tempo-Real. Alguns exemplos foram apresentados, desde protocolos até ao desenvolvimento de *hardware* específico, como o exemplo do *switch HaRTES* desenvolvido na Universidade de Aveiro.

Apesar de o protocolo *IP* incluir um campo *Type of Service* para permitir diferenciação de tráfego, na utilização típica, as redes IP fornecem apenas um serviço *Best-effort* o que, como se viu, não é de todo suficiente para aplicações de Tempo-Real. Assim, definiram-se novos serviços complementares que sejam capazes de fornecer garantias de QdS. Segundo *Zhang et al*. [22] para que uma arquitetura de rede seja capaz de dar diferentes garantias de QdS às suas aplicações deve apresentar cinco componentes essenciais:

- **Especificação do fluxo**: as aplicações devem ser capazes de descrever as características do seu fluxo e dos seus requisitos à rede de forma que esta possa especificar a Qualidade de Serviço que tem de atribuir a esse fluxo;
- **Encaminhamento**: a rede deve saber como transportar os pacotes entre a fonte e o destino;
- **Reserva de recursos**: para que a rede possa fornecer à aplicação determinado *QdS*, é necessário reservar recursos da rede como memória e largura de banda nos nós terminais, *routers, switches* ou *bridges*;
- **Controlo de admissão**: os recursos de uma rede são finitos. Como tal, deve haver uma entidade capaz de verificar que pedidos de reserva podem ou não ser aceites;

- **Escalonamento de pacotes**: a rede deve saber qual o próximo pacote a transferir. Esta decisão é questão central na arquitetura já que determina a QdS que a rede pode fornecer.

Como forma de dotar as redes *IP* de garantias de QdS, foram desenvolvidos ao longo do tempo mecanismos de diferenciação de tráfego, de controlo de admissão e de sinalização e reserva de recursos. Diferentes protocolos e algoritmos foram desenvolvidos e propostos sendo todos eles baseados em mecanismos de sinalização. Estes mecanismos podem manter um funcionamento em *hard* e *soft state [23].*

Dentro do primeiro grupo destacou-se o *ST-II [24]* enquanto no segundo se podem referir o *RSVP* [25] e o *DRP* [26]. Adicionalmente a estes surgiram algumas versões computacionalmente mais leves e menos complexas tais como o *Ticket* [27], *YESSIR* [28] ou *Boomerang* [29]. Alguns algoritmos foram também apresentados em *Firoiu et al*. [30] e *Almesberger et al*. [31]. Mais recentemente, foi desenvolvido e normalizado na norma 802.1Qat o *Stream Reservation Protocol (SRP) [8].*

Das diferentes opções referidas, o *RSVP* e *SRP* revelam ter as características mais adequadas aos requisitos do *switch HaRTES,* e portanto serão estudados e apresentados de seguida no sentido de se perceber qual se apresenta mais adequado a ser usada como tecnologia de sinalização para o *switch.*

3.1 *Resource Reservation Protocol (RSVP)*

A arquitetura de Serviços integrados (*IntServ*) foi desenvolvida pelo grupo *Internet Engineering Task Force (IETF) [32]* com o objetivo de permitir fornecer diferentes níveis de controlo de serviço às aplicações. Para isso, e segundo *Wroclawski* [33], duas características são necessárias:

- Os elementos da rede usados pela aplicação devem suportar mecanismos de controlo de Qualidade de Serviço;
- Um método de a aplicação comunicar à rede os seus requisitos;

O primeiro ponto é alcançado através de serviços de *controlo* QdS tais como *Controlled-Load service* e *Guaranteed service*, enquanto o segundo é fornecido normalmente por protocolos de reserva de recursos, nomeadamente o protocolo *RSVP*.

O protocolo de reserva de recursos (*RSVP*) é um dos mais conhecidos [33] e garante *QdS* através de mecanismos próprios de controlo. A Figura 3.1 representa os blocos funcionais existentes num sistema que suporte *RSVP*, onde se podem distinguir os componentes definidos como essenciais na seção anterior.

Os blocos intitulados *classifier* e *packet scheduler* definem e permitem obter o nível de *QdS* pretendido. Já os elementos *admission control* e *policy control* apresentam uma função de decisão. Nomeadamente, após a receção de um pedido de reserva, o primeiro verifica se o nó em questão tem recursos suficientes para servir o pedido enquanto o segundo verifica se a aplicação que efetuou o pedido, tem as permissões necessárias. O protocolo de *routing* permite obter o caminho que os dados devem seguir para chegarem ao seu destino. Por fim o *RSVP process* é responsável pela sinalização da reserva de recurso, isto é, responsável pela implementação do protocolo.

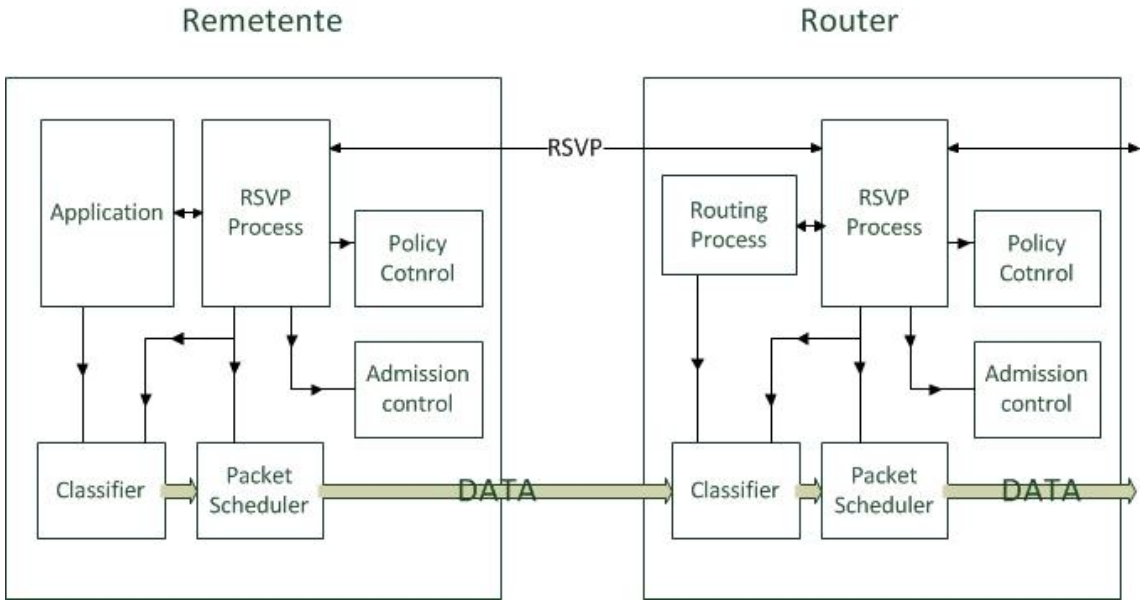

Figura 3.1: Blocos funcionais de um remente e um *router* intermédio que suportem *RSVP* (baseado em Zhang et al. [25]**)**

Nas subsecções seguintes descreve-se o funcionamento básico do protocolo, as mensagens usadas para efetivar o serviço, assim como os parâmetros utilizados no processo de reserva de recursos.

3.1.1 Características e descrição básica de funcionamento

O protocolo *RSVP* opera sobre *IP* inserindo-se no lugar dos protocolos de transporte, não sendo, contudo, responsável pelo transporte dos dados. Apresenta apenas uma função de controlo [25]. É orientado ao recetor, isto é, o processo de reserva (nível de recur-

sos a reservar, inicialização e manutenção da reserva ativa) é operado pelo recetor e não pelo transmissor como na maioria dos protocolos de sinalização.

Os *routers* intermédios mantêm um funcionamento em *soft-state [34]*, isto é, os recetores têm a responsabilidade de periodicamente atualizar a reserva para que esta não expire. Caso tal não aconteça dentro de um período de tempo definido a reserva é perdida. Este modelo permite ter um controlo de sobrecargas assim como mantém o funcionamento do sistema dinâmico no sentido em que, caso o caminho entre nós seja modificado, as reservas sejam também atualizadas de acordo com essas mudanças. Por outro lado, dado que a reserva de recursos é destinada a uma entidade (a que os reservou) e não a pacotes específicos, o protocolo permite selecionar os pacotes que podem usufruir dessa reserva de recursos. Estes permitem definir de que forma as reservas de diferentes recetores no mesmo grupo *multicast* são agrupadas na rede. O *RSVP* define três tipos de reservas: *no-filter, fixed-filter e dynamic-filter* e são descritos em *Zhang et al*. [25].

Descrição básica de funcionamento

Quando um dispositivo remetente pretende transmitir dados, envia uma mensagem[3] (*Path message*) ao recetor com as especificações do fluxo de dados que espera transmitir (*Sender Tspec)*. Quando um *router* intermédio que suporte RSVP recebe a mensagem, verifica se já existe alguma informação acerca daquele fluxo de dados. Se existir, essa informação é atualizada. Se não é criado um novo estado associado a esse fluxo. O *router* obtém informação acerca do caminho que os dados devem seguir através de um protocolo de *routing* e encaminha os dados na direção do(s) respetivo(s) destinatário(s).

Quando um destinatário recebe um pedido de transmissão, este usa os parâmetros recebidos para formular um pedido de reserva que é codificado numa mensagem protocolar (*Resv message)* e que segue o caminho inverso à mensagem recebida. Se algum nó intermédio da rede rejeitar essa reserva por falta de recursos, uma mensagem de erro (*ResvErr)* é retornada e a reserva é descartada. Caso todos os nós disponham de recursos suficientes, a reserva é estabelecida e o recetor é responsável por periodicamente enviar mensagens que mantenham a reserva ativa. Caso um dos nós terminais queira encerrar a conexão, pode enviar uma mensagem específica (*PathTear* ou *ResvTear*) para o efeito.

Numa primeira fase, o protocolo *RSVP* seguia um modelo "*one pass"* [25, 35], isto é os pedidos fluíam dos recetores em direção aos remetentes, podendo ser aceites ou re-

[3] As mensagens envolvidas numa reserva usando RSVP são descritas na secção seguinte.

jeitados em cada nó do caminho que percorriam. Este modelo não permitia no entanto fazer uma previsão do resultado de cada pedido de reserva. Assim, surgiu um modelo melhorado denominado "*one pass with advertising" (OPWA),* em que mensagens de controlo (*Adspec*) fluem em direção aos recetores com informação que possibilita aos mesmos prever *os* recursos de rede disponíveis ponta a ponta, permitindo a formulação de reservas adequadas.

3.1.2 Mensagens

Qualquer que seja o serviço usado, *Guaranteed* ou *Controlled-Load service,* o protocolo define um conjunto de mensagens (Figura 3.2) que são transmitidas entre remetentes e os destinatários de forma a garantir o serviço requisitado.

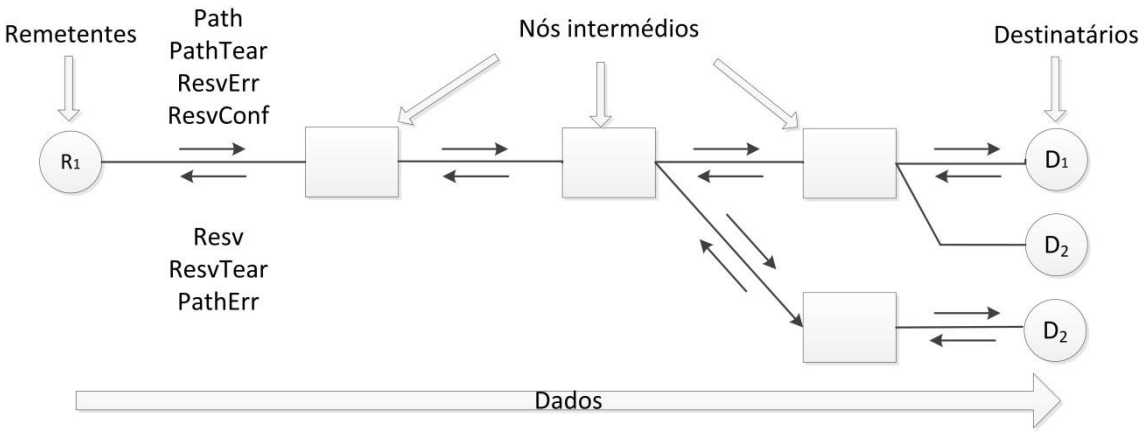

Figura 3.2 : Mensagens transmitidas quando se pretende efetuar uma reserva de recursos (baseado em *White* [36]).

3.1.2.1 Mensagens *Path*

As mensagens *Path* contêm informação acerca do ramo imediatamente anterior, juntamente com as estruturas *Sender Template*, *Sender Tspec* e *Adspec.*

A estrutura *Sender Template* permite identificar os rementes dos pacotes e contém uma descrição do formato dos pacotes que o remetente irá enviar. A estrutura *Sender Tspec* é gerada pelos remetentes de dados, inclui informação acerca das características do tráfego gerado (*TSpec*), flui inalterado em direção aos recetores e contém parâmetros referentes a um *token bucket*, nomeadamente:

- *p = peak rate of flow (bytes/s)*
- *b = bucket depth (bytes)*

- *r = token bucket rate (bytes/s)*
- *m = minimum policed unit (bytes)*
- *M = maximum datagram size (bytes)*

Os valores de *r* e *b* são definidos de forma a descrever o tráfico gerado, enquanto o valor de *p* é definido como o pico de taxa de geração de dados (se conhecido), a taxa da linha de interface, ou definido como infinito se nenhuma das anteriores informações for disponibilizada. O valor de *m* e *M* é definido como o tamanho do menor e o maior pacote gerado, respetivamente.

A estrutura *Adspec* é usada quando se usa um modelo de reserva *OPWA* em que informação adicional flui em direção aos recetores notificando-os acerca dos serviços disponíveis de ponta a ponta. Esta informação permite aos recetores formular níveis de reserva adequados aos recursos disponíveis. Três zonas podem ser identificadas: um cabeçalho, uma zona de parâmetros globais e uma ou mais zonas para *Controlled-Load service* e *Guaranteed service.* A omissão de qualquer um dos últimos campos referidos é uma mera indicação da não existência desse tipo de serviço. Na porção dos parâmetros globais é atualizada informação acerca de: a latência mínima do caminho, isto é, a latência existente de ponta a ponta na ausência de filas de espera, a largura de banda do caminho (a mínima caso haja múltiplos caminhos), o bit global de quebra, o número de ramos percorridos e a unidade de transmissão máxima (*PathMTU) [36].* Estes campos são atualizados em cada nó da rede de acordo com os recursos disponíveis nesse mesmo nó. Relativamente aos fragmentos específicos de cada serviço, estes dispõem de informação necessária e específica de cada serviço, bem como uma sinalização caso algum *router* não suporte o serviço em questão.

3.1.2.2 Mensagens *Resv*

Uma reserva *Resv* é constituída por um par *Flowspec-Filterspec.* O *flowspec* é gerado pelos recetores, flui em direção aos remetentes e transporta o pedido de reserva. Os parâmetros da reserva são transportados pelo *Receiver Tspec e Rspec,* sendo que no primeiro os parâmetros são idênticos ao caso do *Sender Tspec* (r,b,p,m e M), enquanto no segundo surgem mais dois parâmetros:

- R = bandwidth (bytes/s)

- S = slack term (ms)

Os valores de *r*, *b* e *p* são usados, tal como no caso do *Sender Tspec,* para descrever o tráfego sobre o qual o recetor efetuará a reserva. O valor de R e S são escolhidos

de forma a obter as garantias requisitadas pelo serviço, como será descrito na secção 3.1.3..

O *filterspec,* simultaneamente com um identificador de sessão, distingue os pacotes que irão receber o serviço requisitado.

3.1.2.3 Mensagens *Teardown*

Apesar de não ser obrigatório o envio de uma mensagem específica a sinalizar o fim de uma reserva (devido ao funcionamento em *soft-state)*, é recomendado que se use. Neste sentido, distinguem-se dois tipos de mensagem consoante esta flua em direção aos recetores (*PathTear)* ou aos remetentes (*ResvTear)*. Estas mensagens podem ser inicializadas tanto por recetores, remetentes ou nós intermédios e são transmitidas nó a nó sem qualquer tipo de atraso.

3.1.2.4 Mensagens de erro

O *RSVP* define dois tipos de mensagens de erro: *PathErr* e *ResvErr.* As primeiras notificam o respetivo remetente que um erro ocorreu em certo nó da rede na leitura de uma mensagem *path.* As segundas notificam os recetores que a tentativa de reserva num dos nós da rede falhou. A análise deste tipo de mensagens é bastante complexo, já que, em certos nós da rede há a fusão de pedidos de reservas distintos e consequentemente é difícil distinguir o nó responsável pela falha dos recursos. Assim, a mensagem de erro tem de ser enviada a todos os possíveis responsáveis.

3.1.2.5 Mensagens de confirmação

Um recetor que deseje incluir um pedido de confirmação de que uma reserva foi efetuada com sucesso, pode incluir na mensagem *Resv*, um pedido de confirmação. No entanto, as mensagens de confirmação podem não ser válidas em todos os nós do caminho, mais concretamente, há a possibilidade de o recetor receber uma mensagem de confirmação falsa. Isto acontece devido ao agrupamento de reservas distintas em nós intermédios da rede. Contudo, este problema é facilmente detetável pois neste tipo de situação uma mensagem *ResvConf* é sucedida de uma mensagem *ResvErr.*

3.1.3 *Guaranteed Service*

Este tipo de serviço garante às aplicações uma largura de banda suficiente de modo a limitar o atraso de entrega das suas mensagens.

O fluxo de dados representativo de um pedido de reserva segue um modelo fluídico. É constituído por parâmetros de um *token bucket* (r,b) (Figura 3.3), uma taxa de transferência máxima que uma aplicação pode transmitir (p) e o tamanho mínimo e máximo dos pacotes a transmitir (m e M). Estes, são usados para computar dois valores que descrevem o nível de reserva (R e S) e que devem permitir responder aos requisitos da aplicação. De uma forma prática, o valor de R pode ser visto como a largura de banda disponibilizada à aplicação para transmitir o fluxo de dados especificado pelos parâmetros *r, b* e *p.*

No entanto, o funcionamento do *router* não segue de forma exata o modelo fluídico. Por isso, são consideradas duas componentes de erro: C e D. Estas duas componentes de erro são transportadas pelas mensagens *Adspec* no fragmento específico deste serviço na forma de C_{tot}, D_{tot}, C_{sum} e D_{sum}. C_{tot} e D_{tot}, representam o somatório dos erros ao longo dos caminhos enquanto C_{sum} e D_{sum} representam o somatório dos erros desde o último ponto em que a reserva foi modificada.

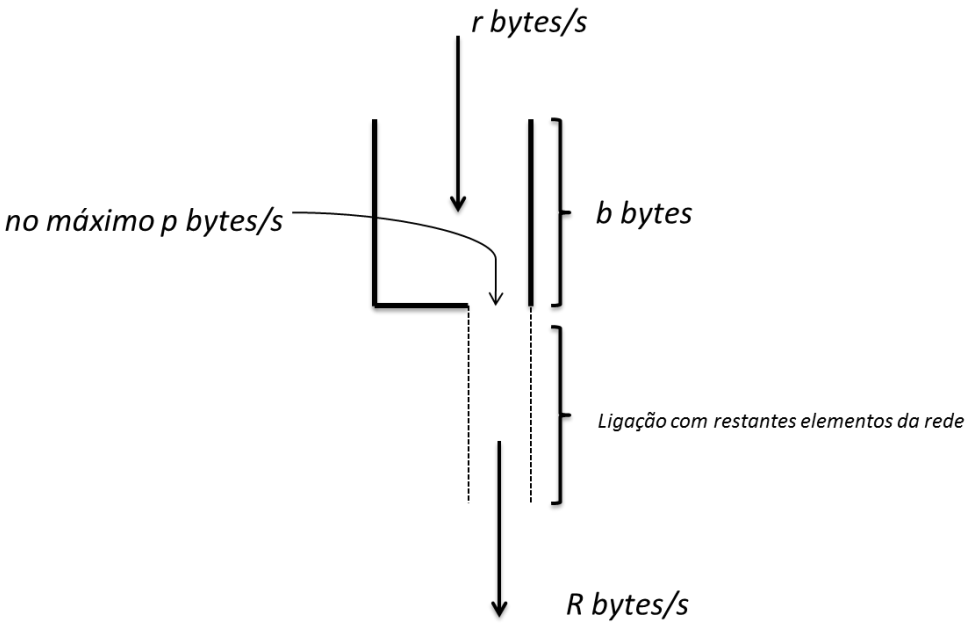

Figura 3.3: Descrição do fluxo de dados em *Guaranteed Service*

Estes valores permitem, em conjunto com o *Tspec,* estimar o valor do atraso que um fluxo de dados pode sofrer devido a filas. De acordo com *White* [36], esse valor é dado por:

$$Q_{delayend2end} = \frac{(b-M)(p-R)}{R(p-r)} + \frac{(M+\ C_{tot})}{R} +\ D_{tot}, \qquad se\ p > R \geq r \qquad \textbf{(1)}$$

$$Q_{delayend2end} = \frac{(M+\ C_{tot})}{R} +\ D_{tot}, \qquad se\ R \geq p \geq r \qquad \textbf{(2)}$$

Assim que um recetor recebe uma mensagem *Path* são extraídos os parâmetros referentes ao *Sender Tspec* e *Adspec* e é calculado o atraso máximo que um pacote pode sofrer devido a filas:

$$Q_{delreq} = latência\ mínima\ do\ caminho - atraso\ requirido\ pela\ aplicação$$

Posteriormente uma primeira análise é feita considerando R = p usando a equação **(2)**. Se o valor obtido for maior ou igual ao de Q_{delreq}, a equação **(2)** é usada para calcular o valor de R assumindo $Q_{delayend2end} = Q_{delreq}$. Caso contrário usa-se a equação (1). Se o valor de R for maior que a largura de banda do caminho, este tem de ser reduzido adequadamente.

A reserva pode então ser efetivada definindo o valor de R calculado e inicializando o valor de S (normalmente a 0) na mensagem *Resv.*

3.1.4 Mapeamento de requisitos de sinalização do *HaRTES* em *RSVP*

O *switch HaRTES* foi desenvolvido de forma a possibilitar o fornecimento de garantias temporais e o uso eficiente da largura de banda em sistemas dinâmicos e distribuídos de Tempo-Real. Contudo, o funcionamento deste baseia-se em protocolos desenvolvidos pelos criadores e não suporta protocolos normalizados, o que dificulta a utilização em cenários reais. Portanto, é necessário procurar formas de conseguir a integração de protocolos normalizados no *switch*.

No seguimento dessa intenção, tornou-se necessário estudar uma possível integração e implementação de um protocolo de reserva de recursos no *switch* capaz de manter o correto funcionamento do mesmo. Para tal, é imprescindível averiguar se é possível fazer um mapeamento entre os parâmetros do *switch* e os protocolos de sinalização de reserva de recursos.

3.1.4.1 Requisitos do switch HaRTES

O *switch HaRTES* define, para a sua correta operação, um conjunto de variáveis. Estas são apresentas na Tabela 3.1.

Tabela 3.1: Variáveis definidas pelo *switch HaRTES*

Variável	**Descrição**
fttElementaryCycleDuration	Duração de um ciclo elementar definido pelo paradigma FTT
fttDurationOfSynchro-nousAndAsynchronousDuration	Duração das janelas destinadas ao transporte de tráfego síncrono e assíncrono
fttGuardingWindow	Janela que acomoda a eventualidade de existência de diferentes tempos de propagação
fttTurnArroundTime	Tempo necessário para descodificação das mensagens TM
fttSwitchingDelay	Atraso introduzido pelo *switch*
fttTransmissionTime	Tempo de colocação dos dados no meio
fttIdleTime	Tempo de *idle* inserido no fim de um ciclo elementar quando já não existe espaço suficiente para a transmissão de determinado pacote
fttSchedulerType	Tipo de escalonador a ser usado pelo *switch*
fttMaximumNumberOfServers	Número máximo de servidores
fttMaximumNumberOfMessag-essAperiodic	Número máximo de mensagens aperiódicas
fttMaximumPacketSize	Tamanho máximo do pacote passível de ser transferido
fttBaudRate	*Baudrate* dos dados

Destas, o *fttMaximumPacketSize* define o tamanho máximo do pacote passível de ser transmitido e o *fttBaudRate* permite definir o número de *bytes* a serem transferidos num segundo, introduzindo uma noção de tempo de acesso. Estas duas variáveis são essenciais para o que *HaRTES* consiga definir e garantir as restrições temporais apresentadas pelas diversas aplicações que o usem. Desta forma, os protocolos de sinalização de recursos devem ser capazes de transportar nas suas mensagens este tipo de informação.

No que diz respeito ao caso do *RSVP*, verifica-se que o *fttMaximumPacketSize* é diretamente mapeado com o valor M transportado no *Tspec* da mensagem *Path.* Contudo, não existe um mapeamento direto do *fttBaudRate* dado que o valor que o modelo fluídico usado pelo *RSVP* para descrever o tráfego, define um valor de pico em *bytes/s* mas não define uma noção de tempo de acesso, isto é, não permite definir um período temporal.

Esta ausência de uma noção de tempo de acesso levanta dificuldades na definição de *deadlines* por parte do *switch* e portanto apresenta-se como uma barreira para a integração do *RSVP* no *HaRTES.*

3.1.4.2 Requisitos do RSVP

No que diz respeito ao *RSVP*, os parâmetros contidos no *Sender Tspec* descrevem o fluxo de dados que o remetente pretende transmitir e portanto são fornecidos pela aplicação.

Relativamente ao *Adspec,* como já foi referido nas secções anteriores, temos de obter informação acerca da largura de banda mínima do caminho, latência mínima do caminho e os erros acumulados ao longo desse caminho (C_{tot} e D_{tot}).

O *switch HArTES* contém no seu ciclo elementar (*EC*) (Figura 3.4), duas zonas distintas para mapeamento de tráfego síncrono e assíncrono, juntamente com uma zona destinada à sincronização do sistema no início de cada *EC*. O tamanho de cada uma das zonas é ajustável de acordo com o tráfego existente, sendo, no entanto, definido um valor mínimo da duração da janela assíncrona (*LAW*). Assim, garante-se um serviço mínimo de atendimento a tráfego assíncrono. A duração de um ciclo elementar (*LEC*) e da zona de sincronização (*LTM + TAT*) são parâmetros do sistema definidos estaticamente antes da inicialização do sistema. No entanto, e como se limita o uso de *RSVP* à janela assíncrona olharemos para o tempo ocupado pelas mensagens de controlo e pela janela síncrona como uma largura de banda reservada.

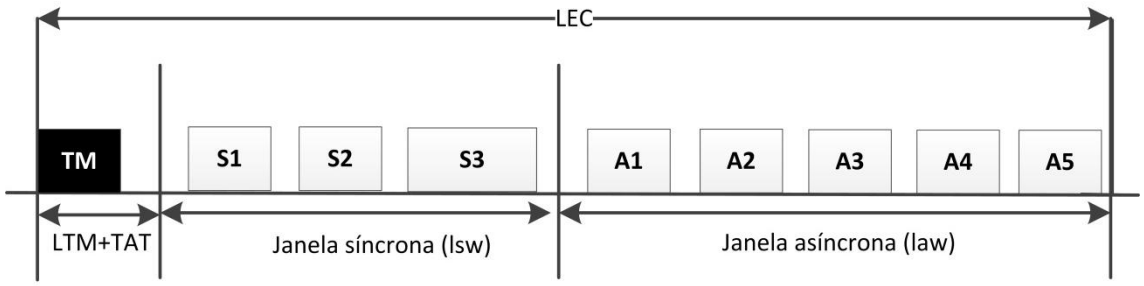

Figura 3.4: Estrutura de um ciclo elementar (*EC*) do switch *HaRTES* (baseado em Santos [35])

O tráfego assíncrono é tratado recorrendo ao uso de servidores, sendo a largura de banda disponível facilmente calculada através da subtração da largura de banda disponível para a janela assíncrona à taxa de utilização dos servidores ativos, dada por $\sum_{i=1}^{n} \frac{C_i}{T_i}$.

A latência mínima de um caminho corresponde ao tempo de transmissão máximo que certos dados podem sofrer na ausência de filas de espera. Assim, considera-se o tempo de resposta de pior caso e que corresponde à ocorrência das seguintes situações:

- Uma tarefa aperiódica chega no final do tempo de execução do servidor, isto é a sua capacidade foi esgotada e nos próximos períodos está bloqueado por outras tarefas de maior prioridade;
- A janela síncrona tem duração 0, isto é, *lsw* = 0;

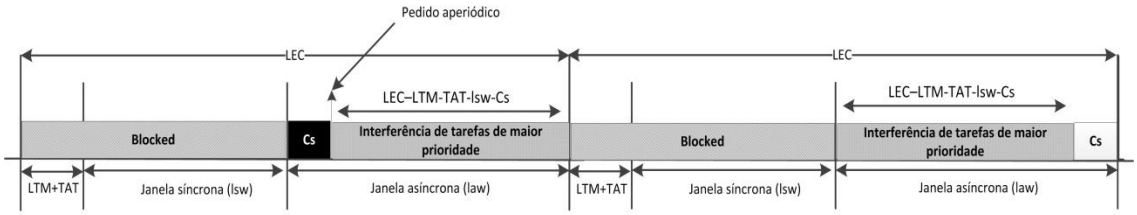

Figura 3.5: Considerações temporais nos ciclos elementares aquando da receção de um pedido aperiódico no fim de execução de um servidor

Como se pode constatar pela observação daFigura 3.5, o tempo de resposta quando existe um pedido de tarefa periódica após o esgotamento da capacidade do servidor e nos próximos períodos do mesmo está bloqueado por tarefas de maior prioridades, é dado por:

$$LTM + TAT + lsw + 2 \times (law - C_s)$$

ou de outra forma,

$$LTM + TAT + lsw + 2 \times (LEC - LTM - TAT - lsw - C_s)$$

Em situações limites a duração da janela síncrona pode ser nula (*lsw* = 0). Isso implica automaticamente um aumento da janela assíncrona e, consequentemente um aumento do tempo de resposta, correspondendo à pior resposta possível.

Em suma, ao contrário da latência mínima de um caminho que pode ser calculada, o protocolo *RSVP* não permite a definição de *deadlines,* elemento essencial aos sistemas de Tempo-Real críticos. Isto fez com que se abandonasse o *RSVP* como hipótese e se procurassem novas soluções, nomeadamente através do *SRP* que será apresentado de seguida.

3.2 *Stream Reservation Protocol (SRP)*

No ponto anterior introduziu-se um protocolo de sinalização que permite a reserva de recursos e, consequentemente, permite fornecer garantias de *QdS* em redes que utilizem *routers.* No entanto, o protocolo apresentado e discutido não é o único no que à reserva de recursos diz respeito. O *Stream Registration Protocol* [8] permite, tal como o *RSVP*, registar, de-registar e manter reservas de recursos ao longo do caminho percorrido pelo fluxo de dados.

Os protocolos, procedimentos e objetos usados pelo *SRP* são especificados no *standard IEEE Std 802.1Qat – 2010* [8]. Este utiliza três protocolos de sinalização diferentes para estabelecer reservas de recursos ao longo de uma *bridged local area network.* Estes são o *Multiple MAC Registration Protocol (MMRP)* [37] que regista endereços *MAC*, o *Multiple VLAN Registration Protocol (MVRP)* [37] que regista e gere *VLANs* e o *Multiple Stream Registration Protocol (MSRP) [8]* que sinaliza a reserva de recursos*.* Todos eles se baseiam e suportam no *Multiple Registration Protocol (MRP) [37],* protocolo este que permite registar ou de-registar atributos ao longo de uma rede.

No *MMRP* os atributos registados são endereços *MAC* e permite controlar a propagação do registo de atributos por parte dos *Talkers* (remetentes de informação) na rede. Já no *MVRP*, os atributos são indentificadores de *VLAN* e permite às estações terminais ou *bridges* registarem-se em *VLAN's.* Por último, os atributos do *MSRP* transportam a informação necessária à reserva de recursos por parte dos elementos da rede.

Dos referidos protocolos de sinalização, focaremos a nossa atenção no protocolo *MSRP* já que é ele que objetivamente permite a reserva de recursos. Assim, estudaremos numa primeira fase o *MSRP*, nomeadamente modelo de operação, formato e o conteúdo das mensagens essenciais ao seu funcionamento e os parâmetros necessários à sua implementação. Numa segunda fase introduziremos o protocolo *MRP*, base do *MSRP.* O capítulo termina com o mapeamento de requisitos entre o *HaRTES* e o *MSRP* e uma comparação do mapeamento para o caso de *RSVP* e do *MSRP*.

3.2.1 *MSRP*

O protocolo *MSRP* consiste num protocolo de sinalização que possibilita a reserva de recursos ao longo de uma rede para que uma *stream*, proveniente de uma fonte e destinada a um ou vários destinatários, possa ser transmitida com as requeridas garantias de *QdS.* O protocolo *MSRP* garante que, caso a reserva seja bem sucedida, as respetivas garantias sejam fornecidas, os recursos associados sejam reservados à transmissão dos

pacotes que compõem a *stream* e que assim as condições de transporte dos pactos respeitem os parâmetros de *QdS* requeridos.

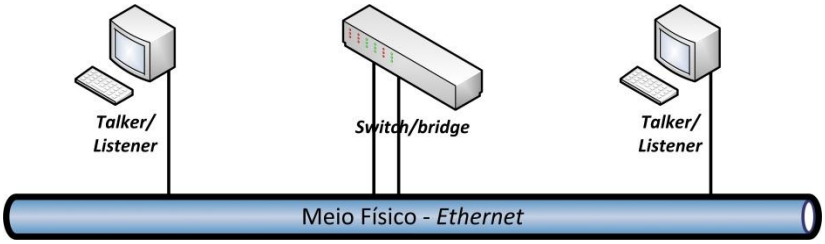

Figura 3.5: Intervenientes numa sessão *MSRP*

Dentro de uma rede em que todos os intervenientes suportem *MSRP,* podem distinguir-se três classes diferentes (Figura 3.5): as estações terminais que pretendem transmitir dados, neste caso dados que requerem *QdS,* denominados *Talkers*; os *Listeners,* também estações terminais mas pretendentes à receção dos dados enviados pelos *Talkers*; e por fim as estações intermédias, *bridges* e/ou *switches*, que são responsáveis por reencaminhar os dados entre os referidos nós terminais, garantido que as garantias dos fluxos de dados são cumpridas. Assim, estes devem intervir diretamente no processo de reserva de recursos, tendo ao seu cargo a tarefa de aceitar, reencaminhar, ajustar e recusar as reservas de recursos.

Sendo o *MSRP* uma aplicação específica do protocolo *MRP*, segue uma arquitetura em tudo semelhante à do protocolo *MRP.* Contudo, cada aplicação *MRP* define os seus próprios atributos e conteúdo das mensagens. A juntar a isso, o *MSRP* define um novo funcionamento para o módulo *MAP*.

Estes e mais detalhes serão apresentados de seguida, começando por se descrever cada módulo individualmente para no fim se clarificar o funcionamento do protocolo como um todo.

3.2.1.1 Declarações de atributos em MSRP

O *MSRP* define distintos atributos de acordo com a estação terminal que os declare e de acordo com as informações de *QdS* recolhidas ao longo da rede. Assim, caso se trate de uma declaração de um atributo por parte de um *Talker,* duas declarações são possíveis: *Talker Advertise* e *Talker Failed.*

Uma declaração do tipo *Talker Advertise* indica que, até esse instante, a *stream* anunciada pelo *Talker* não encontrou qualquer restrição a nível de *QdS.* Caso esta decla-

ração seja transmitida com sucesso até chegar ao *Listener* que a pretenda receber, este poderá efetivar a reserva de recursos através de um *Listener Ready.* A declaração de um *Talker Failed* indica que a *stream* encontrou restrições que não garantem a *QdS* requerida.

Caso esta declaração, *Talker Failed*, chegue ao *Listener* este sabe que existem restrições ao longo do caminho e portanto, a reserva não tem condições para ser concluída. Como consequência, o *Listener* responde ao *Talker Failed* com o envio de um *Listener Asking Failed* que segue em direção ao *Talker.* Na situação anteriormente descrita, assume-se a existência de um único *Listener*. Contudo, é possível a existência de mais do que um *Listener* e, como tal pode ocorrer o caso de haver apenas largura de banda suficiente ao longo da rede para garantir *QdS* a um *Listener.* Nestes casos a declaração do *Listener* é modificada para *Listener Ready Failed*.

3.2.1.2 Controlo das declarações

Uma estação definida como *Talker,* sempre que deseje anunciar as *streams* que pode transferir, comunica-o ao módulo da aplicação *MSRP (*Figura 3.6*)* através da primitiva *REGISTER_STREAM.request.* Esta deve transportar com ela um identificador único da *stream,* o tipo de declaração e os parâmetros descritivos da *stream* que o *Talker* pretende anunciar*.* Caso pretenda terminar o anúncio das mesmas pode fazê-lo através da primitiva *DEREGISTER_STREAM.request* que contém apenas informação acerca do *ID* da mesma. Esta informação é propagada ao longo da rede e quando chega a um *Listener,* o módulo da aplicação informa o *Listener* da existência (ou cancelamento) da *stream* anunciada pelo *Talker* através da primitiva *REGISTER_STREAM.indication (ou DEREGISTER_STREAM.indication).*

Já no que diz respeito à estação *Listener,* quando esta pretende receber determinada *stream,* envia um *REGISTER_ATTACH.request* ao seu módulo de aplicação. Os devidos atributos são propagados em direção ao *Talker* gerador da respetiva *stream* e a primitiva *REGISTER_ATTACH.indication* é usada para informar o mesmo do resultado da reserva de recursos para a *stream* com o identificador transmitido. Isto é possível já que, caso um *Talker* receba um *Listener Ready* associado a uma *stream* e a um T*alker Advertise* para a mesma *stream,* então as devidas alocações de recursos foram feitas e a *stream* terá as garantias de *QdS* requeridas. As primitivas DE*REGISTER_ATTACH.request e DEREGISTER_ATTACH.indication* cancelam o efeito das duas primitivas referidas anteriormente.

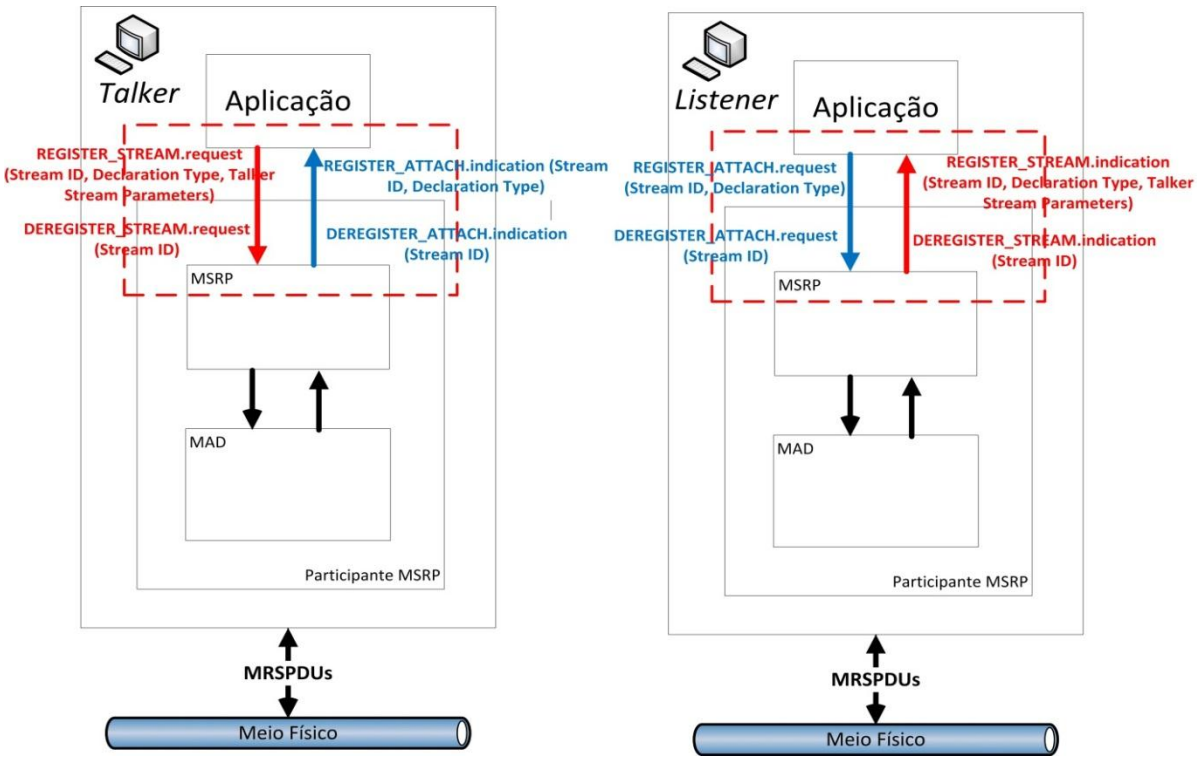

Figura 3.6: Primitivas de controlo das declarações das estações terminais: *Talkers* e *Listeners (*adaptado de *Ghunter* [38])

3.2.1.3 MAD

As primitivas usadas pelo módulo *MAD* são, equivalentes às definidas pelo protocolo *MRP* e identificadas na Figura 3.7. Associados a estas primitivas estão os diferentes tipos de declarações referidos em 3.2.1.1., nomeadamente, *Talker Ready, Talker Failed, Listener Ready, Listener Ready Failed e Listener Asking Failed.* Adicionalmente, além de comunicar com o módulo da aplicação, no sentido de informar a aplicação do estado da reserva, o módulo *MAD* é responsável pela codificação e descodificação dos *PDU's.* Estes representam pacotes de dados que transportam todas as mensagens protocolares através do meio físico entre todos as estações da rede. O formato genérico dos *MRPDU's* será apresentado em 3.2.2.3 enquanto os *MSRPDU's* serão descritos mais detalhadamente no ponto 4.3.

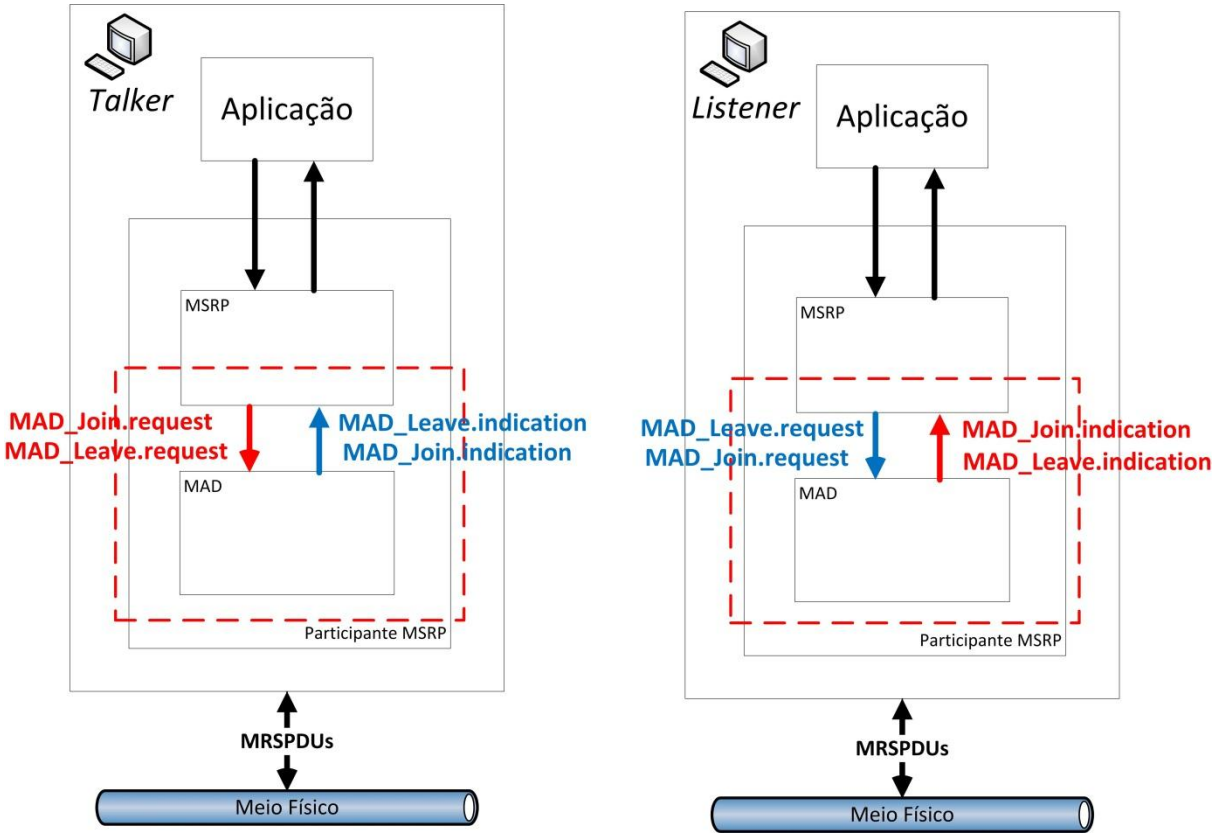

Figura 3.7: Mensagens *MAD (*adaptado de *Ghunter* [38])

3.2.1.4 MAP

A propagação e ajuste das mensagens de sinalização geradas pelas diversas estações ao longo da rede é da responsabilidade do módulo *MAP.* A Figura 3.8 ilustra de que forma uma *bridge,* de duas portas, propaga as mensagens do módulo *MAD*.

Como se pode verificar, a chegada de uma mensagem *MAD_Join.indication* (*MAD_Leave.indication)* ao módulo *MAD* da porta 1, resulta na sua propagação para o módulo *MAD* da porta 2 como *MAD_Join.request (MAD_Leave.request).*

O atributo transportado pelas mensagens referidas depende do tipo de declaração recebido na porta 1, enquanto o atributo propagado para a porta 2 depende, não só do tipo de declaração recebido na porta 1, como também depende da existência de largura de banda e recursos suficientes na *bridge* para dar garantias à *stream* à qual o atributo diz respeito.

Em suma, e considerando-se o caso de uma declaração de um *Talker*, numa rede constituída apenas por um *switch* intermédio de duas portas, caso se receba um *Talker Failed* numa porta, este é propagado para a outra porta também como um *Talker Failed.* No entanto, caso o módulo *MAD* da porta 2 recebe um *Talker Advertise* então o atributo

propagado para a porta 2 depende da capacidade do *switch* em dar garantias de *QdS* à *stream* associada.

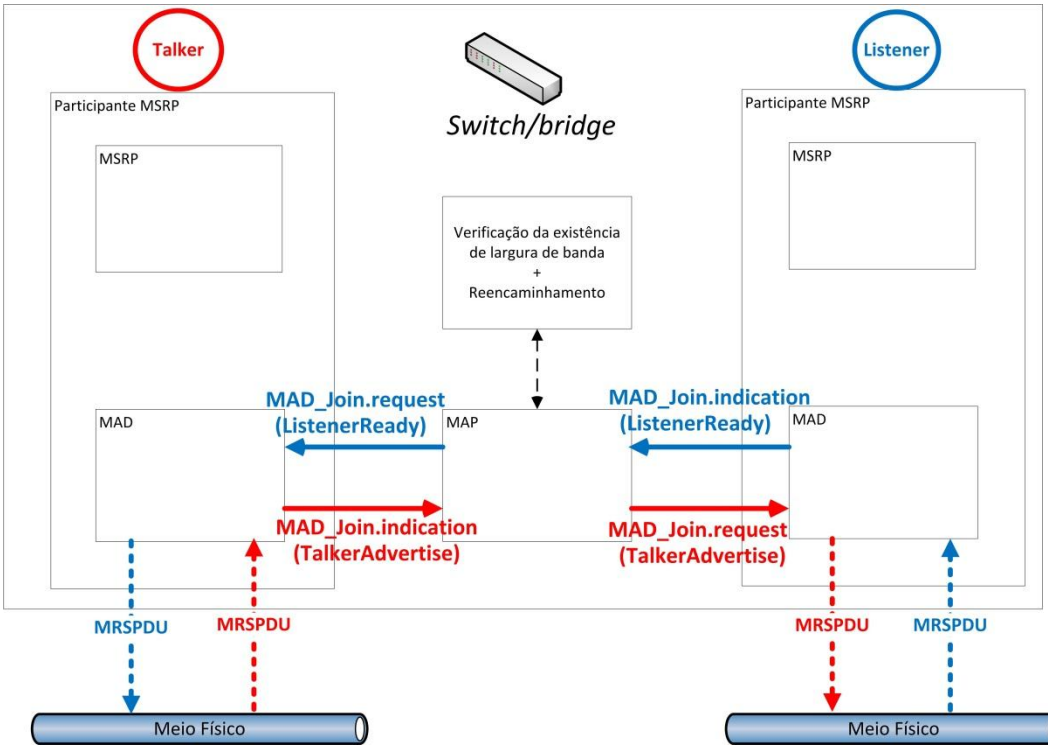

Figura 3.8: Propagação de atributos numa *bridge* - reserva efetuada com sucesso *(*adaptado de *Ghunter* [38])

Se o *switch* oferecer garantias, então um *Talker Advertise* é propagado para a porta 2 em direção ao *Listener (*Figura 3.8*)*, se não um *Talker Failed* é propagado em direção em direção há porta 2 que seguirá em direção aos *Listeners (*Figura 3.9*),* informando-os da inexistência de garantias ao longo da rede para a associada *stream* de dados.

A verificação da existência de largura de banda é efetuada por um módulo específico e o seu funcionamento é especificado no protocolo estandardizado em *IEEE Std.802.1Qav [39].* De salientar que não existe a alocação de recursos aquando da passagem do atributo *Talker Advertise* pelo módulo *MAP*. Isto significa que existe a possibilidade se serem propagados mais *Talker Advertise* do que a largura de banda disponível. Contudo, após a receção do primeiro *Listener Ready* associado a determinado *Talker Advertise,* o módulo responsável pela verificação de existência de largura de banda, faz nova análise aos atributos associados a essa *bridge.* Caso, após a primeira reserva, deixe de ter capacidade de dar garantias às restantes *streams,* os *Talker Advertise* associados a essas *streams* são transformados em *Talker Failed* e o módulo *MAP* encarrega-se de fazer a propagação desta alteração. Em síntese, a reserva é efetuada apenas aquan-

do da receção de um *Listener Ready* associado a um *Talker Advertise*, de uma mesma *stream.*

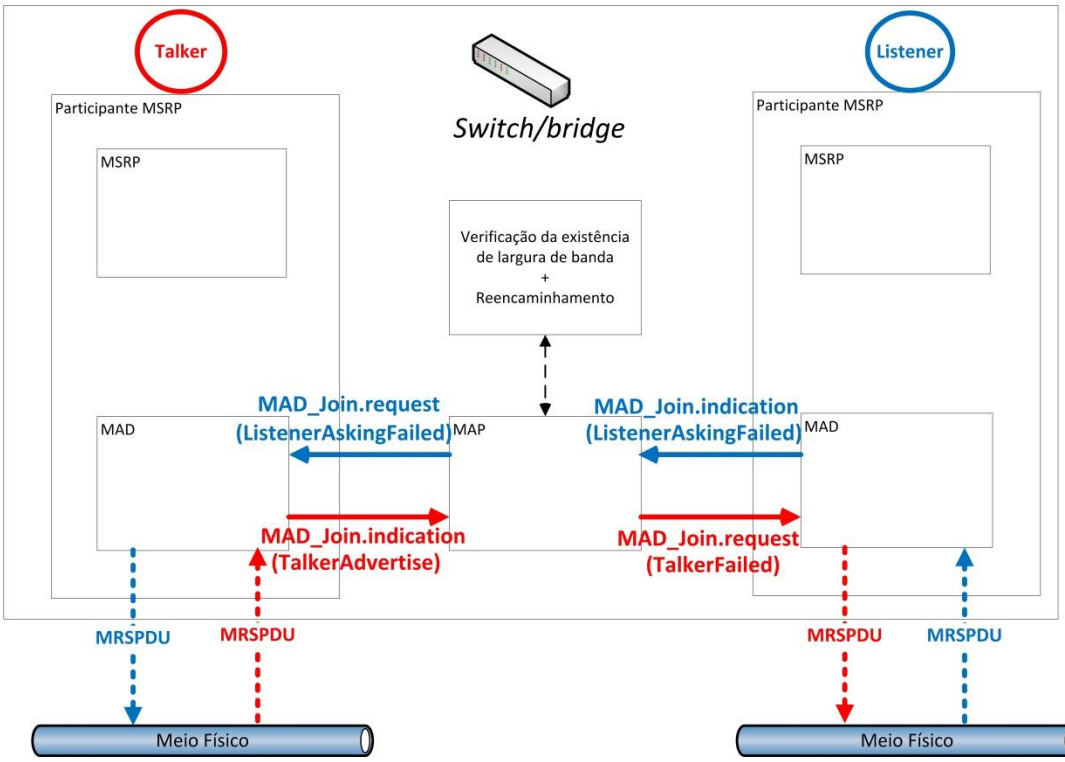

Figura 3.9: Propagação de atributos numa bridge - reserva falha *(*adaptado de *Ghunter* [38])

É também importante referir que a propagação de atributos dentro de uma *bridge* é geralmente efetuado em todas as portas não bloqueadas da *bridge.* Contudo, o *SRP* permite filtrar a propagação destes atributos através da ativação do campo *Talker Pruning.* Esta ativação implica que a propagação apenas se efetuará se o endereço destino da *stream* for encontrando na tabela de endereços do *switch.*

A grande diferença no que diz respeito à propagação dos atributos associados ao *Listener* reside no facto de, no caso dos *Listeners,* poder haver o agrupamento de registos de diferentes *Listeners* associadas à mesma *stream* numa única declaração. Para além disso, a propagação de uma declaração de um atributo *Listener* apenas se processa caso exista um atributo *Talker* associado.

Em suma, registos de atributos de *Listeners* associados à mesma *stream* devem ser agrupados numa única declaração que será propagada em direção à porta que tem o registo do *Talker* associado. Nestes casos, a direção da declaração é estabelecida como

Listener, o tipo de declaração é estabelecido de acordo com o agrupamento dos diferentes registos (Tabela 3.2) e a identificação da *stream* coincide com a que é transportada nos registos.

Tabela 3.2: Agrupamento de declarações de *Listeners*

1ª declaração	2ª declaração	Declaração após agrupamento
Ready	Nenhuma ou *Ready*	*Ready*
	Ready Failed ou Asking Failed	Ready Failed
Ready Failed	-	*Ready Failed*
Asking Failed	*Ready ou Ready Failed*	*Ready Failed*
	Nenhuma ou *Asking Failed*	*Asking Failed*

3.2.1.5 Variáveis MSRP

O protocolo *MSRP* define dois tipos de variáveis que usa para garantir o seu correto funcionamento: um que está associado à categoria de variáveis internas e que são usadas pelas máquinas de estados do *MRP*, e um outro tipo que inclui variáveis protocolares e que são transmitidas através de *MSRPDU's* entre as diferentes estações numa rede.

Na primeira categoria distinguem-se as seguintes variáveis:

- ***Port Media Type:*** esta variável define o modo como é feito o acesso ao meio por parte de uma porta. Pode ser definida como *Acess Control Port,* em que o transmissor controla o acesso ao meio; ou como *Non-DMN shared medium Port,* em que o transmissor acede a um meio partilhado mas não o controla;
- ***Direction*****:** identifica o tipo de nó que originou a declaração do atributo associado à maquina de estados em questão – *Talker ou Listener;*
- ***Declaration Type*****:** tipos de declarações possíveis;
- ***Pârametros SRP:*** descrevem parâmetros importantes para efetivar a reserva de recursos, entre os quais: *portTcmaxLatency, talkerprunning, streamAge, msrpEnabledStatus, msrpPortEnabledStatus, msrpMaxFanInPorts, msrpLatency, MaxFrameSize, SRPdomainBoundary e SR_PVID*. Destes, e para o efeito de cálculo de largura de banda e latência necessários para a alocação de recursos, destacam-se o *msrpLatencyMaxFramSize* que especifica o tamanho máximo dos pacotes de dados que aquela porta permite transmitir e o *portTcmaxLatency* que determina a latência máxima que um pacote pode experimentar por porta e por classe de tráfego;

Na segunda destacam-se as seguintes:

- ***MaxFrameSize:*** representa o tamanho máximo de um pacote que o *Talker* possa gerar. Este tamanho não inclui qualquer *overhead* adicional que deve ser levado em conta no cálculo da largura de banda a reservar para a *stream;*
- ***MaxIntervalFrames:*** indica o número máximo de pacotes que o *Talker* pode enviar em determinada classe de intervalo de tempo;
- ***DataFramePriority:*** define a prioridade da *stream,* valor que é usado para cálculo do *Priority Code Point (PCP);*
- ***Rank:*** marca a *stream* como sendo urgente;
- ***Accumulated Latency:*** este valor representa a latência máxima que uma *stream* pode encontrar no caminho do *Talker* até ao *Listener*.

Das variáveis apresentadas, os valores de *MaxFrameSize* e o *MaxIntervalFrames,* são essenciais, como iremos ver no ponto 3.2.3, para o cálculo da largura de banda a reservar nas *bridges/switches*. Por seu turno, o valor da *Accumulated Latency* é fundamental para que os *Listeners* possam decidir se aceitam esse valor como suficiente para cumprir a *QdS* desejada pela aplicação.

3.2.1.6 Fluxograma descritivo do funcionamento do protocolo

Para finalizar esta secção de introdução ao protocolo *MSRP,* é ilustrado na Figura 3.10, um fluxograma que inclui as principais mensagens e primitivas trocadas entre os diferentes intervenientes numa sessão *MSRP*.

O cenário inclui um *Talker* que pretende transmitir duas *streams*, um *Listener* que pretende receber essas mesmas *streams* e um *switch* com duas portas mas que apenas tem capacidade para dar garantias à transmissão de uma *stream.* Assume-se também que o *Talker* anuncia as duas *streams* consecutivamente antes de receber qualquer informação por parte do *Listener* acerca da reserva de recursos.

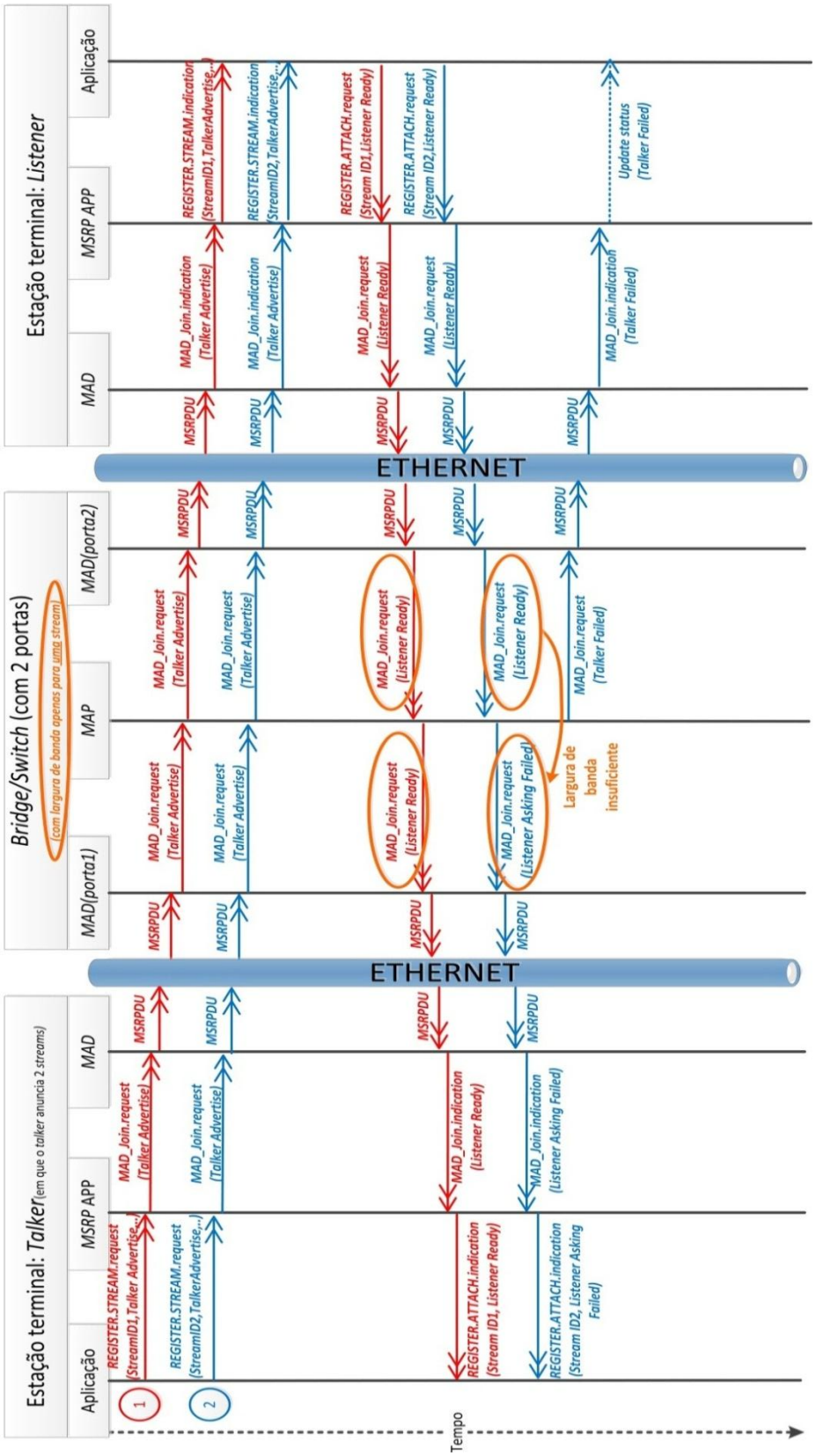

Figura 3.10: Fluxograma descritivo do funcionamento do protocolo *MRSP*: primitivas e mensagens trocadas entre um *Talker* que pretenda transmitir duas *streams*, um *Listener* e um *switch* com duas portas que apenas tem capacidade para dar garantias de transmissão a um *stream*

Descrição do processo

O *Talker* inicia o processo com o anúncio da primeira *stream* (*ID1)* através do envio da primitiva *REGISTER_STREAM.request* para o módulo da aplicação *MSRP.* Com a mesma segue o identificador da *stream,* tipo de declaração (neste caso *Talker Advertise)* e restantes atributos descritivos da *stream.* Consequentemente, este envia um *MAD_Join.request* para o módulo *MAD* contendo toda a informação recebida até esse ponto. O módulo *MAD,* por sua vez codifica toda a informação num pacote protocolar *MSRPDU* e envia-o para a rede.

Este pacote, ao ser recebido numa porta do *switch,* é descodificado pelo módulo *MAD* associado a essa porta e é identificado o tipo de atributo encapsulado. Automaticamente, o tipo de declaração é também descodificado, neste caso *Talker Advertise*. De seguida, esta mensagem *MAD* é recebida pelo módulo *MAP* que, em conjunto com um bloco adicional de verificação de largura de banda, é responsável por verificar a existência de recursos suficientes nesse *switch* para dar as respetivas garantias à *stream.* Neste caso, como o *switch* tem capacidade para dar garantias a uma *stream* e como ainda não foi feita nenhuma reserva de recursos, o atributo *Talker Advertise* é propagado para o módulo *MAD* da segunda porta através de uma mensagem *MAD_Join.request.* Esta, ao ser recebida é codificada de novo num *MSRPDU* com os valores associados a esse *Talker Advertise* que por sua vez é enviado para a rede em direção ao *Listener.* O Listener recebe este *MSRPDU* através do módulo *MAD* e descodifica-o identificando claramente o atributo e valores que transporta. Esta informação é encapsulada numa mensagem *MAD.Join.indication* e enviada para o módulo da aplicação *MSRP.* Esta informa a aplicação da existência de um *Talker* que pretende transmitir a *stream ID1* através da primitiva *REGISTER_STREAM.indication,* que inclui o identificador da *stream,* as características da mesma e o resultado da pré-análise feita ao *QdS* disponível ao longo da rede. Nesta situação, o *Listener* recebe um *Talker Advertise* que lhe indica que não foram encontradas quaisquer restrições ao longo da rede. Portanto, caso a *stream* seja destinada a esse *Listene*r (que neste caso assume-se que é), poderá enviar um pedido de reserva em direção ao *Talker.*

Entretanto, o *Talker* inicia novo processo de anúncio de nova *stream* com *ID2.* O processo desenrola-se de igual forma ao da *stream ID1.* Repare-se que o tipo de declaração que é recebida pelo *Listener* referente à stream *ID2* é o mesmo ao da *stream ID1 (Talker Advertise)*, isto apesar de a rede apenas ter recursos para suportar a transmissão de uma *stream*. Tal acontece, dado que nenhuma reserva foi efetuada aquando do anún-

cio da primeira *stream* por parte do *Talker*. Foi feita apenas uma pré-análise tendo em conta as reservas já efetuadas pelos *Listeners*. Assim sendo, nesse instante, o *Listener* tem conhecimento que existe um *Talker* que pretende transmitir duas *streams* e que, até esse instante, nenhuma restrição foi encontrada. Em conformidade, o *Listener* inicia o processo de reserva de recursos associado à *stream ID1.* Fá-lo através do envio da primitiva *REGISTER_ATTACH.request* da aplicação para o módulo da aplicação *MSRP* com identificador da *stream ID1* e tipo de declaração, neste caso *Listener Ready*.

A propagação deste atributo é feita de forma semelhante ao anteriormente descrito para o caso do *Talker Advertise*. De notar, contudo, algumas diferenças na operação do módulo *MAP* para o caso de um atributo *Listener*.

Ao receber um *Listener Ready* é verificada a existência de largura de banda suficiente para dar garantias à *stream* que o *Talker* pretende transmitir. Adicionalmente, é verificado se existe algum *Talker Advertise* associado ao *Listener Ready* recebido. Caso ambas as condições se verifiquem é requisitado pelo módulo *MAP* que se efetue a reserva de recursos nesse *switch;* o *Listener Ready* é propagado para a porta restante e, consequentemente, propagado até atingir o *Talker.* Ao atingir o módulo de aplicação *MSRP* do *Talker,* este usa a primitiva *REGISTER_ATTACH.indication* para informar o *Talker* do resultado da reserva de recursos (a primitiva transporta o identificador da *stream 1* e um tipo de declaração *Listener Ready).* Com isto, o processo de reserva de recursos associado à *stream ID1* fica concluído.

O processo de reserva de recursos associado à *stream ID2* é iniciado de forma equivalente ao da *stream ID1.* Porém, quando a declaração do atributo *Listener Ready* chega ao módulo *MAP* do *switch* e é verificada a inexistência de largura de banda suficiente para dar garantias a essa *stream* (relembra-se que assumiu-se que o *switch* tem apenas capacidade para dar garantias a uma *stream* e que essa capacidade foi já atribuída à *stream ID1),* este modifica a declaração do atributo *Listener Ready* para *Listener Asking Failed* e modifica a declaração do atributo do *Talker* associado, de *Talker Advertise* para *Talker Failed*. Estas modificações são propagadas até atingirem as respetivas estações terminais, informando, tanto o *Talker* como o *Listener,* que a reserva de recursos associada à *stream ID2* não foi bem sucedida.

A partir deste momento, e até que que o *Talker* cancele o anúncio da *stream ID1,* apenas esta pode ser transmitida com as requeridas garantias de *QdS*. Por outro lado, qualquer alteração na declaração de algum dos atributos expostos ou qualquer alteração no funcionamento do *switch* (como a adição de mais portas) faz com que o módulo *MAP* faça propagar de novo os atributos de acordo com as alterações feitas.

3.2.2 *Multiple Registration Protocol (MRP)*

O *Multiple Registration Protocol (MRP)* é um protocolo que concede a possibilidade de *bridges*, registarem ou libertarem atributos em conjunto com outros elementos da rede. As especificações do protocolo vêm individualizadas na norma I*EEE 802.1ak-2007* [37] e veio substituir o *Generic Attribute Registration Protocol (GARP).*

3.2.2.1 Visão global do protocolo

O protocolo *MRP* permite, a qualquer participante na sessão, declarar (ou libertar) atributos que podem resultar no registo (ou libertação) dos mesmos atributos em outros participantes da rede da qual os primeiros fazem parte. Uma declaração ocorre quando determinado nó pretende iniciar a propagação do atributo e/ou existe a propagação num *switch* de um registo efetuado em determinada porta. Já um registo é uma consequência de uma declaração originada num nó e propagada ao longo da rede, isto é, um nó que não o gerador da declaração ao receber o atributo propagado toma conhecimento dele através de um registo.

Assim, e tomando como exemplo o caso em que a declaração é feita apenas por uma estação terminal, pode observar-se na Figura 3.11 o resultado de uma declaração de um atributo.

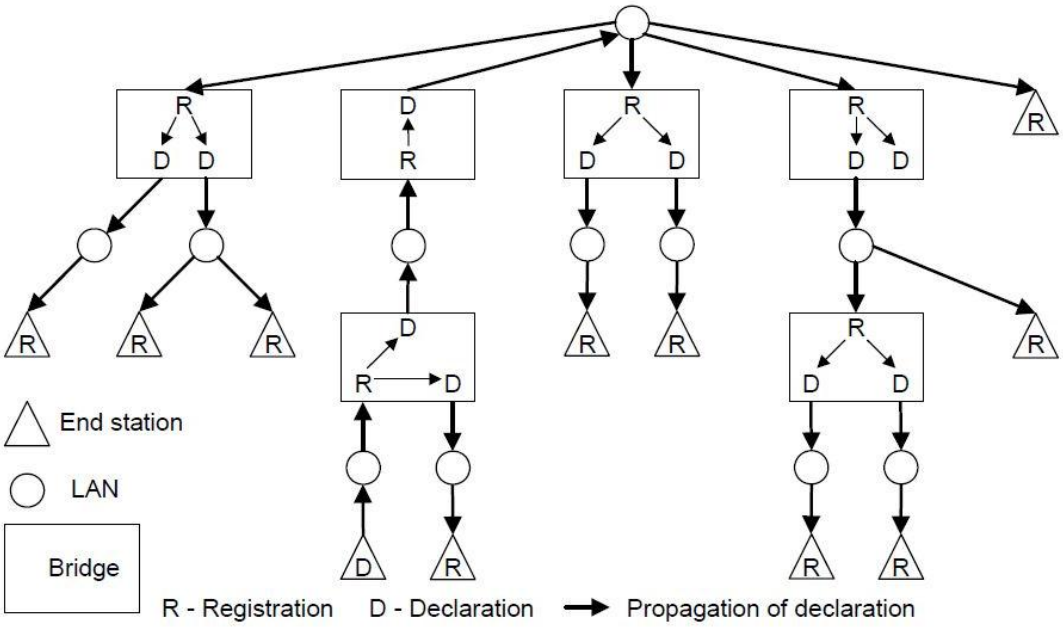

Figura 3.11: Exemplo de propagação de uma declaração de um atributo no caso em que é feito apenas por uma estação terminal [37]

A declaração de um atributo por parte de uma estação terminal que esteja ligada a uma porta de uma *bridge* resulta no registo do mesmo atributo na referida porta. Este registo origina a declaração do mencionado atributo nas restantes portas da *bridge*. A propagação da declaração do registo acontece em todas as portas desde que estas façam parte da topologia ativa, isto é, desde que estas façam parte do conjunto de portas que permanecem no estado de reencaminhamento (*forwarding).* O processo repete-se até que o atributo seja registado em todas as estações finais.

A existência de declarações e registos de atributos sugere a existência de uma forma de gerir e processar esses dados. Assim, as declarações e registos efetuados por um interveniente que suporte *MRP* são preservados em máquinas de estados. Caso sejam declarações, as máquinas de estado denominam-se *Applicant state machine,* se forem registos denominam-se *Registrar state machine*. Cada nó *MRP* deverá manter ambas as máquinas de estado, por porta e por atributo.

Na arquitetura do MRP (Figura 3.12), podem distinguir-se três módulos essenciais: um módulo específico da aplicação (*MRP application*), um módulo *MRP Attribute Declaration (MAD)* e um módulo *MRP Atribute Propagation (MAP)* que introduziremos de seguida.

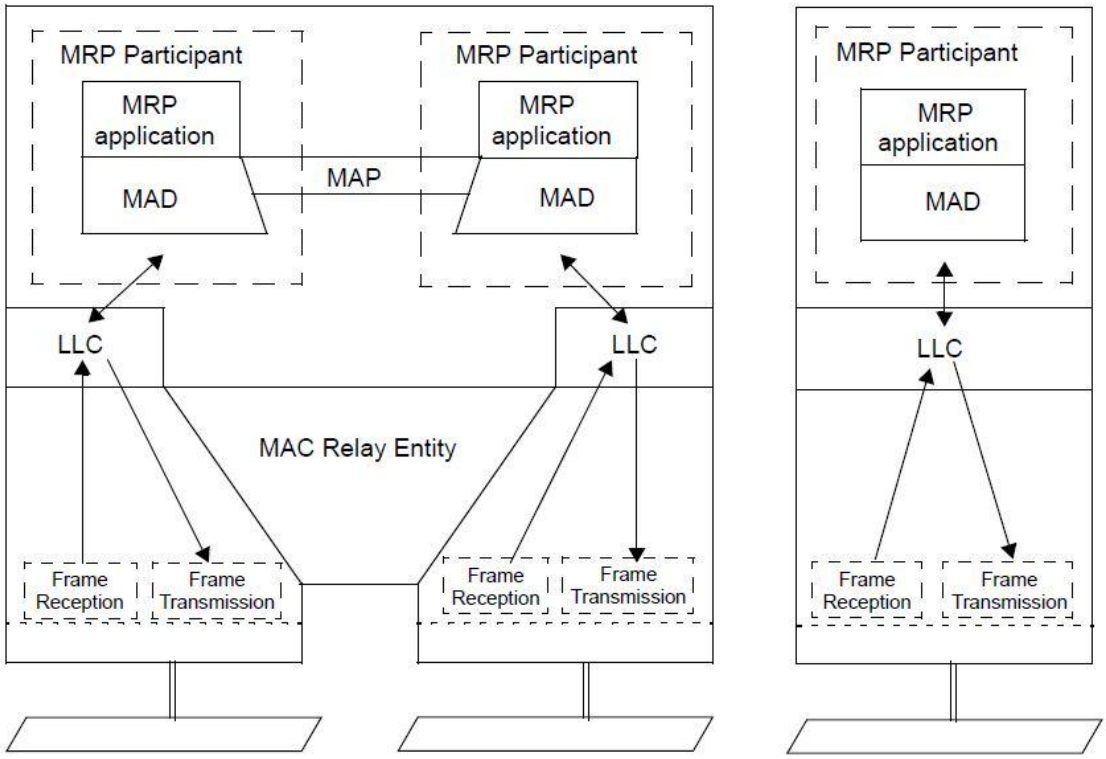

Figura 3.12: Arquitetura do *MRP* numa bridge com duas portas e uma estação final [37]

O módulo específico da aplicação é responsável pela semântica e registo dos valores dos atributos e comunica com o *MAD* com o objetivo de concretizar (ou eliminar) declarações através das seguintes diretivas:

- *MAD_Join.request (attribute_type, attribute_value, new);*
- *MAD_Leave.request (attribute_type, attribute_value);*

Nas referidas mensagens, o *attribute_type*, descreve o tipo de atributo ao qual a mensagem se aplica, e a sua gama de valores depende da aplicação *MRP*. Já o segundo parâmetro, *attribute_value,* transporta os dados necessários ao registo do atributo. Por fim, o parâmetro *new* indica a presença da declaração de um atributo novo.

O módulo *MAD* tem a responsabilidade de implementar/concretizar o protocolo MRP. Isto inclui a geração das mensagens a transmitir, bem como o processamento das mensagens recebidas dos outros participantes. Utiliza as seguintes mensagens para notificar o módulo específico da aplicação de possíveis alterações de declarações:

- *MAD_Join.indication (attribute_type, attribute_value, new);*
- *MAD_Leave.indication (attribute_type, attribute_value);*

A codificação das mensagens protocolares feita pelo *MAD* são dependentes da aplicação *MRP* e denominam-se *Multiple Registration Protocol Data Units* (*MRPDUs*), que são transferidas do *MAD* para o meio e posteriormente recebidas e descodificadas pelo módulo *MAD* de outro participante. O formato básico dos pacotes é definido pelo protocolo *MRP* e será apresentado na secção 3.2.2.3. O conteúdo dos pacotes depende de aplicação para aplicação e portanto é especificado por elas.

Já o módulo *MAP* é responsável pela propagação dos atributos registados numa porta de uma bridge pelas restantes. Contudo, ao contrário do funcionamento do *MAD* que é idêntico qualquer que seja a aplicação *MRP*, o mesmo não acontece com o módulo *MAP*. O seu funcionamento é distinto caso se trate de uma aplicação *MMRP*/*MVRP* ou *MSRP*.

3.2.2.2 MRP Attribute Declaration (MAD)

O módulo *MAD* é, tal como já foi referido, responsável pela execução do protocolo *MRP* segundo as especificações do mesmo. Sendo este módulo idêntico para qualquer que seja a aplicação *MRP* passaremos de seguida a uma descrição mais concreta da sua operação e especificação.

Cada estação de uma *LAN*, seja estação final ou porta de uma *bridge* que suporte *MRP*, contém um módulo *MAD* que é constituído por quatro diferentes tipos de máquinas de estados:

- ***Applicant state machine**:* tem uma função de controlo e existe uma por atributo declarado;
- ***Registrar state machine**:* é responsável por preservar as declarações dos atributos existindo uma por atributo;
- ***LeaveAll state machine**:* que assegura que as declarações são repetidas periodicamente existindo uma por participante;
- ***PeriodicTransmission state machine**:* usada em ambientes suscetíveis à perda de pacotes, garante que o registo dos atributos é feito com sucesso. Existe uma por participante.

Contudo, o protocolo *MRP* possibilita quatro diferentes implementações consoante as máquinas de estados que a aplicação *MRP* necessitará para o seu correto funcionamento. Assim, define-se:

- ***Full Participant**,* quando todas as máquinas de estado são implementadas no seu todo;
- ***Full Participant, point-to-point subset*** quando implementa as mesmas máquinas de estados do *Full Participant* mas ignora certos estados e ações da *Applicant state machine;*
- ***Applicant-Only Participant**,* que implementa apenas a *Applicant state machine* (ainda que, com a omissão de alguns estados e ações) e a *PeriodicTransmission state machine*;
- ***Point-to-point subset of the Applicant-Only Participant**,* quando implementa as mesmas máquinas de estado da *Applicant-Only Participant* mas com a omissão de alguns estados e ações.

As referidas máquinas de estado fazem uso de diferentes tipos de definições e convenções, nomeadamente, convenções de notação e abreviaturas, definições de controlo, *timers,* eventos e ações. A sua introdução será feita gradualmente à medida que se apresentem as diferentes máquinas de estado já explicitadas.

Applicant state machine

A *Applicant state machine* tem a responsabilidade de garantir que uma declaração feita pela respetiva estação é corretamente registada pela *Registrar* dos restantes participantes da *LAN*. Adicionalmente, tem a tarefa de induzir os restantes participantes a repetir o registo dos atributos caso um deles remova uma declaração.

Associado a qualquer *Applicant state machine* existe um parâmetro de controlo global (*Applicant Administrative Control)* que permite perceber se a máquina de estados em questão participa ou não na troca de mensagens *MRP*. Caso participe, esse parâmetro é colocado a *Normal Participant,* caso contrário é colocado a *Non-Participant*.

A referida máquina de estados reconhece três diferentes tipos de participantes. Os participantes ativos, que efetuaram uma declaração através do envio de uma ou mais mensagens; os passivos que não requerem declaração mas requerem registo; e os observadores, que apesar de não requererem registo, seguem o estado do atributo em questão e podem passar a participantes ativos caso o atributo seja registado.

Com o intuito de dar a conhecer o estado de transmissão do atributo de dado participante, quatro tipos de mensagens foram definidas: *Empty,* que traduz a inexistência de algum atributo declarado ou registado; *In,* que significa a existência de registo mas não de declaração; *JoinEmpty*, que implica declaração mas não registo e por fim *JoinIn* que expressa a existência de declaração e registo.

Além das referidas mensagens outras duas complementares são especificadas: *Leave* e *New.* Quando a *Applicant state machine* recebe a indicação de cancelamento de uma declaração, este poderá comunicar esse mesmo estado através do envio de uma mensagem *Leave.* A mensagem *New* transporta a informação de que houve uma declaração por parte de um novo participante.

A máquina de estados tem a si associado um conjunto de estados (que se encontram listados no Anexo A, Figura 1) cujas transições estão dependentes da ocorrência de eventos protocolares (descritos no Anexo A, Figura 2) que ocorram. Como resultado dessas transições certas ações protocolares (registados no Anexo A, Figura 3) deverão ser executadas. A relação entre estados, eventos e ações referente à *Applicant State machine* está resumida na Figura 4 do Anexo A que pode ser interpretada da seguinte forma:

- Cada coluna representa o estado da máquina de estados antes da ocorrência de determinado evento;
- Cada linha corresponde a um determinado evento;
- A interceção de uma linha com uma coluna representa o estado para que transitará a máquina de estados na ocorrência do evento associado a essa linha. Inclui também, caso exista, a ação protocolar que resulta dessa transição. Assim, assumindo que determinada *Applicant state machine* se encontra no estado *VP* e ocorre um evento *tx!*, a máquina de estados mencionada transitará para o estado *AA* e como fruto dessa transição a ação *sJ é* executada.

Registrar state machine

A *Registrar state machine* tem o papel de guardar e preservar as declarações de atributos feitas por outros participantes na *LAN*, não enviando contudo qualquer tipo de mensagens protocolares.

Tal como a *Applicant state machine,* a *Registrar state machine* tem associado um parâmetro de controlo (*Registar Administrative Controlo)* que, neste caso, permite perceber que tipo de registo é admitido para determinado atributo em determinada máquina de estados. Assim, caso o valor desse parâmetro seja *Normal Registration,* a máquina de estados opera de forma regular sem apresentar qualquer tipo de restrições, isto é, seguindo o que é especificado na Figura 5 do Anexo A. No entanto, se tiver *Registration Fixed* a máquina de estados mantém o seu estado em *IN*, isto é o atributo está registado, ignorando assim qualquer mensagem *MRP* que possa ser recebida. Por último, e caso o valor do parâmetro de controlo seja *Registration Forbidden,* o estado da máquina de estados é mantido a *MT*, isto é não está registado nenhum atributo ignorando desta forma qualquer tipo de mensagem *MRP* recebida.

A *Registrar state machine* implementa três distintos estados. O estado *In* que indica que o atributo está registado na máquina de estados, o estado *Lv* que indica que o atributo se encontrava registado mas está em processo de remoção de registo e o estado *MT* que significa que o atributo não está registado.

A relação entre estados, eventos e ações para a *Registrar state machine* é ilustrada na Figura 5 do Anexo A e a sua leitura é idêntica à da *Applicant state machine.*

LeaveAll state machine

A *LeaveAll state machine* tem como missão assegurar que os diferentes participantes de uma sessão MRP repetem os seus registos periodicamente. Assim, evita-se que possíveis falhas no registo ou remoção do registo sejam continuamente propagadas. Esta máquina de estados tem associado um *timer* que após expirar, faz com que seja transmitida uma oportunidade de transmissão e, um *LeaveAll* consequente. De forma a evitar que múltiplas mensagens de *LeaveAll* sejam enviadas por diferentes participantes que registam o mesmo atributo, o timer é reinicializado sempre que uma mensagem de *LeaveAll* seja recebida.

O protocolo *MRP* implementa quatro tipos de timers: *jointimer*, *leavetimer*, *leavealltimer* e *periodictimer*, estando cada um deles articulado com cada uma das máquinas de

estado apresentadas no início da secção 3.2.2.2. O *jointimer* controla o tempo da janela de tempo entre oportunidades de transmissão. O *leavetimer* garante que se guarda um intervalo de tempo na transição do estado *Lv* para o estado *Mt* numa *Registrar state machine*. Quanto ao *leavealltimer,* este controla o tempo decorrido entre sucessivos envios de mensagens *LeaveAll* a todos os participantes que declarem o mesmo atributo. Por fim, o *periodictimer* é responsável por controlar a janela de tempo entre a ocorrência de eventos periódicos por parte da *PeriodicTransmission state machine*.

Os valores que definem cada timer não são fixos, tendo no entanto valores padrão que, em condições normais de funcionamento, otimizam o funcionamento do protocolo e que podem ser consultados na Figura 6 do Anexo A.

A Figura 7 do Anexo A resume o funcionamento da *LeaveAll machine*. Esta apresenta dois diferentes estados, *passive* e *active*. O estado *active* é atingindo quando o *timer* associado a esta máquina de estados (*leavealltimer)* expira. Já o estado *passive* está relacionado com mensagens de *LeaveAll* enviadas por outros participantes que estejam a declarar o mesmo atributo. De notar que o *timer* associado é reiniciado através da ação *start leavealltimer* sempre que uma mensagem de *LeaveAll (rLA!)* é recebida, ou o *timer* associado expire *(leavealltimer!)*.

PeriodicTransmission state machine

Caso o ambiente de operação do protocolo esteja sujeito a perdas de pacotes, a *PeriodicTransmission state machine* assegura que o registo dos atributos é efetuado com sucesso através da geração de eventos periódicos. O funcionamento da máquina de estado é descrito pela Figura 8 do Anexo A.

3.2.2.3 *MRP Protocol Data Units (MRPDUs):* estrutura genérica

As mensagens trocadas entre os diferentes participantes numa sessão *MRP* são codificadas e englobadas em *MRP Protocol Data Units (MRPDUs)*. Adicionalmente, cada *MRPDU* identifica a aplicação que lhe deu origem bem como a aplicação a que se destina. O conteúdo dos diferentes campos que constituem um *MRPDU* é dependente da aplicação *MRP* a que diga respeito. Não obstante, existe uma estrutura genérica que deverá ser seguida qualquer que seja a aplicação *MRP*.

Um *MRPDU* é definido como sendo um conjunto inteiro de bytes em que se diferenciam diferentes campos, ilustrados na Figura 3.13. O primeiro *byte* é dedicado à versão do protocolo à qual o *PDU* diz respeito. Os *bytes* que se seguem destinam-se às mensa-

gens *MRP* a ser transmitidas que, como já foi referido, podem corresponder à comunicação de diferentes eventos. O último *byte* corresponde a um caracter de terminação, denominado *end mark*.

Cada mensagem é por sua vez constituída por quatro diferentes zonas: *AttributeType, AttributeLenght* e *AttributeList.* O conjunto de valores que o campo *AttributeType* pode tomar é específico da aplicação e representam o tipo de atributo à qual a mensagem diz respeito. O *AttributeList* transporta o conteúdo da mensagem e o seu tamanho é codificado no *AttributeLenght.* O campo referente ao *AttributeList* é constituído por diferentes *VectorAttributes* e termina com novo terminador *End Mark*.

Todo o *VectorAttribute* é constituído pelos campos *VectorHeader, FirstValue e Vector.* O *VectorHeader* codifica a existência ou não de um *LeaveAllEvent* bem como um outro campo denominado *NumberOfValues,* que identifica o número de eventos codificados no campo *Vector*. O valor do *FirstValue* é dependente da aplicação *MRP* enquanto o campo *Vector* transporta os eventos aos quais a máquina de estados deve ser sujeita para o atributo definido pelo *AttributeType.*

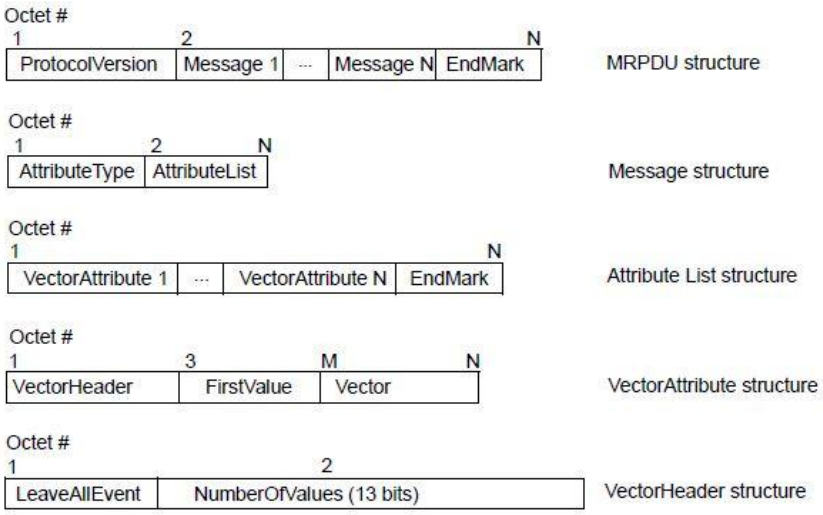

Figura 3.13: Estrutura genérica de um *MRPDU* [37]

3.2.2.4 Aplicações *MRP*

O protocolo *MRP* define um conjunto de requisitos estruturais e funcionais genéricos para que um nó de uma rede possa declarar e registar determinados atributos numa rede *LAN*. De acordo com os atributos que se pretendam declarar/registar podem definir-se diferentes aplicações MRP.

Até ao momento foram definidas e já identificas neste documento três diferentes aplicações *MRP*: *Multiple MAC Registrarion Protocol (MMRP), Multiple VLAN Registration Protocol (MVRP) e Multiple Stream Registration Protocol (MSRP)*.

O *MMRP* permite o registo, quer de endereços, quer de grupos de endereços *MAC,* possibilitando restringir o tráfego apenas aos segmentos da rede onde é requisitado. O *MVRP*, por seu lado, permite o registo dinâmico em *VLANs* enquanto o *MSRP* possibilita a reserva recursos de rede de forma ter garantias de Qualidade de Serviço (*QdS*), aplicação esta já descrita em 3.2.1.

3.2.3 Mapeamento de requisitos de sinalização do *HaRTES* em *MSRP*

3.2.3.1 Requisitos do *HaRTES*

O protocolo *MSRP* descreve as características do tráfego para o qual pretende sinalizar a reserva de recursos através de duas variáveis: *MaxFrameSize* que define o tamanho máximo do pacote de dados que um *Talker* pode transmitir e o *MaxIntervalTime* que limita o número máximo de *frames* por unidade de tempo. Estas duas variáveis podem ser praticamente mapeadas diretamente do *HaRTES* para o *MSRP.* O *MaxFrameSize* correspondendo ao *fttMaxPackedSize* e o *MaxIntervalFrames* através do *fttBaudrate.* Assim, o *MSRP* permite a definição de períodos e portanto permite a existência de uma referência temporal, essencial para a definição de *deadlines.*

3.2.3.2 Requisitos do *MSRP*

Tal como o protocolo *RSVP,* também o protocolo *MSRP* exige às estações intermédias o cálculo da latência máxima que estes podem provocar no tráfego que transportam. O tipo de análise que deve ser feito é idêntico ao do *RSVP* e portanto não será descrito novamente.

Por outro lado, é exigida uma forma de se calcular a largura de banda a reservar em determinado *switch* de modo a garantir a requerida *QdS.* Esta pode ser calculada através de algumas variáveis transportadas nos atributos do protocolo de sinalização *MSRP.* Assim, e segundo Lim et al. [40], a largura de banda pode ser calculada através da seguinte expressão:

$$StreamBW = \frac{(MaxFrameSize + OH) \cdot MaxIntervalFrames}{IntervalTime}$$

Onde *OH* representa o *overhead* associado a uma *stream Ethernet.* Este cálculo é efetuado assumindo situações de pior caso na transmissão de um *stream* e portanto assegura a requerida Qualidade de Serviço.

3.3 Comparação do mapeamento RSVP / MSRP

Depois de se terem apresentados dois protocolos de sinalização de reserva de recursos, nomeadamente *RSVP* e *SRP*, e após discussão do possível mapeamento de parâmetros destes protocolos e o *switch*, torna-se importante fazer um resumo desse mapeamento. Para tal, apresenta-se na Tabela 3.3 um resumo dessa comparação.

Tabela 3.3: Tabela comparativa do mapeamento *RSVP/MSRP*

Característica/ Protocolo	*RSVP*	*MSRP*
Fonte do protocolo	*IETF*	*IEEE*
Serviços	*Controlled-Load e Guaranteed service*	8 diferentes prioridades ([4])
Estações intervenientes	Terminais e *routers*	*Bridges* , *switches* e terminais
Especificação de tempo de acesso	Não	Permite
Definição de *Deadlines*	Não	Permite
Latência acumulada	Exige	Exige
Origem das reservas	*Listeners*	*Listeners*

A grande diferença entre os protocolos de sinalização de reserva de recursos estudados diz respeito aos serviços disponibilizados por ambos, nas estações intervenientes e na existência de uma escala temporal e consequente possibilidade de definição de *deadlines.*

Por um lado, o *RSVP* pode ser usado em dois tipos de serviços propostos pelo *IETF, Controlled-Load e Guaranteed service.* O segundo permite às *streams* obter o QdS desejado por elas, enquanto o primeiro apenas permite um controlo da degradação da QdS. Por outro lado, o *RSVP* foi inicialmente desenvolvido para *routers* enquanto o *MSRP* foi pensado para redes com *bridges*/*switches.*

[4] Prioridade que é usada para gerar o *Priority Code Point (PCP)* tal como descrito no *Std 802.1Qat-2010.* Também segundo este documento, este valor é associado a classes de tráfego descritas na secção 34.5

A não definição de um período temporal, impossibilita ao *RSVP* a definição de *dead-lines,* aspeto essencial para o *HaRTES.* Pelo contrário, o *MSRP* permite-o, garantindo-lhe características favoráveis ao seu uso no *switch HaRTES.*

4 Implementação

A necessidade de se dotar o *switch HaRTES* de mecanismos de reserva de recursos normalizados, levou a que se partisse para a implementação do protocolo *MSRP* num *switch* deste tipo. As próximas secções são centradas nessa implementação, onde se identificam os blocos estruturais essenciais para a implementação do mesmo e se descrevem os blocos desenvolvidos no âmbito desta dissertação.

4.1 Levantamento de requisitos

Qualquer estação que faça uso do protocolo *MSRP* deve conter alguns blocos estruturais essenciais para o correto funcionamento do mesmo. Estes são:

- **Um bloco específico da aplicação**, responsável pela semântica associada aos atributos definidos pelo *MSRP*;
- **Um bloco *MAD*** responsável pela execução do protocolo, que inclui a implementação das máquinas de estado apresentadas em 3.2.2.2 e a codificação e descodificação dos MSR*PDU's;*
- **Um bloco *MAP*,** caso se trate de uma *bridge/switch,* responsável pela propagação dos atributos registados numa porta para as restantes portas.
- Um bloco responsável pela verificação da existência de largura de banda, pela consumação da reserva de recursos e que está associado ao bloco *MAP.*

Dos blocos apresentados, alguns são comuns a todas as aplicações *MRP* enquanto outros são específicos de cada aplicação. Como tal, e como o tema central desta dissertação se centraliza no protocolo de sinalização *MSRP,* procurou-se, numa fase inicial, encontrar algum trabalho já desenvolvido nesta área, nomeadamente, relativo à implementação do protocolo genérico de registo e propagação de registos de atributos (*MRP*). Contudo, esta procura não foi bem sucedida, o que exigiu a adoção de uma nova abordagem.

Devido às restrições temporais e de recursos encontradas, dois cenários foram ponderados. Um passava pela implementação de raiz do protocolo *MRP* deixando o protocolo *MSRP* para trabalho futuro. O outro pela possibilidade de iniciar-se a integração de um suporte *MSRP* através da implementação das mensagens específicas do protocolo *MSRP* deixado o protocolo *MRP* para trabalho futuro.

Dado que o tema central desta dissertação foi, desde início, o protocolo de reserva de recursos *MSRP*, a abordagem adotada para os trabalhos desta dissertação foi a implementação das mensagens protocolares do *MSRP* bem como a construção de uma

máquina de estados simples que permitisse testar o comportamento do protocolo de *MSRP* em determinadas situações.

Assim, o trabalho desenvolvido no âmbito desta dissertação baseou-se na existência de dois nós terminais, que podem funcionar como *Talker* ou *Listener* e que contêm dois blocos essenciais (Figura 4.1):

- **Rotinas de codificação e descodificação dos *MSRPDUs*;**
- **Implementação de um módulo *MAD* de teste**, bastante básico, mas que permitisse testar as funcionalidades dos *MSRPDUs* desenvolvidos bem como testar algumas situações relativas ao comportamento do protocolo *MSRP* no que diz respeito à troca de mensagens protocolares.

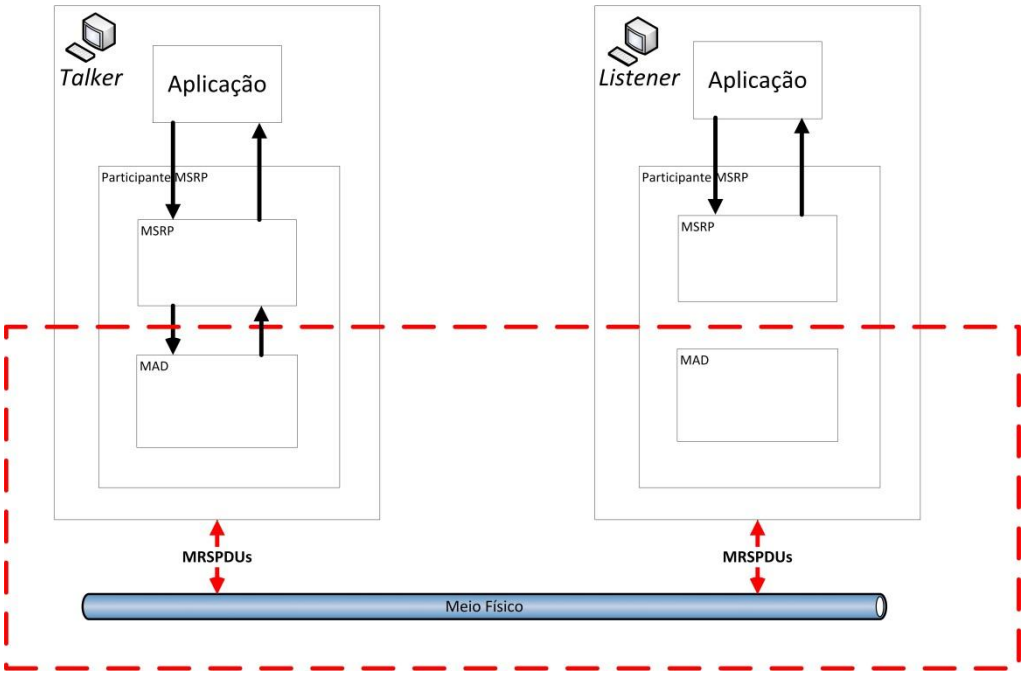

Figura 4.1: Blocos alvos de implementação

A secção seguinte descreve a plataforma de desenvolvimento utilizada para implementação dos blocos referidos, enquanto o restante capítulo é dedicado à descrição mais detalhada dos referidos blocos.

4.2 Plataforma de desenvolvimento

O desenvolvimento dos blocos referidos no ponto anterior foi feito em cima de *Linux* e a linguagem de programação escolhida foi C. Esta escolha baseou-se no facto de ser vantajoso para este tipo de sistemas o uso de uma lingua gem de baixo nível, e na impra-

ticabilidade de ser usar *assembly,* a escolha natural recaiu em C que adicionalmente permite que o código possa ser portado para outras plataformas.

A troca de mensagens *Ethernet* exigiu o uso de *sockets* de internet para possibilitar a troca de mensagens entre os terminais, *Talker* e *Listener*. Adicionalmente, uma sessão *MSRP* pressupõe a existência de pelo menos dois nós terminais, nós estes que podem funcionar como *Talkers* e *Listeners* numa mesma sessão. Assim, cada nó deve ser capaz de enviar e receber mensagens em qualquer instante. Para se alcançar tal especificação foi necessário a implementação de duas *threads,* uma que permitisse o início de um pedido de reserva de recursos através da interação com o utilizador, neste caso, pressionando uma tecla, e uma segunda que permitisse a receção de mensagens de outros nós.

4.3 Mensagens

Esta secção descreverá de uma forma mais detalhada as mensagens protocolares e seus parâmetros possíveis de serem trocados pelo protocolo *MSRP.* Este define cinco tipos de declarações possíveis, sendo estas associadas a três diferentes tipos de atributos transportados por três diferentes tipos de *MSRPDUs*. Estes atributos são *Talker Advertise, Talker Failed* e *Listener*[5]*.*

Estes três atributos são codificados em *PDUs* que seguem uma estrutura em todo semelhante à apresentada na Figura 3.13. Associado a estes existe um conjunto de variáveis indispensáveis à codificação dos mesmos e que passaremos de seguida a descrever.

- ***MSRP application address* (6 *Bytes*)**: todos os *MSRPDUs* devem ter como endereço *MAC* destino um valor específico, *(Tabela 9, Anexo A*) e que pertence ao grupo de endereços *MAC "Individual LAN Scope group address, Nearest Bridge group address";*
- ***EtherType* (1 *Byte*)*:*** identifica o tipo de protocolo que é encapsulado na carga útil de um trama *Ethernet* (*Tabela 10, Anexo A*);
- ***Protocol Version* (1 *Byte*)*:*** especifica a versão do protocolo a que a mensagem diz respeito;
- ***Attribute Type* (1 *Byte*)*:*** específica o tipo de atributo codificado em determinada mensagem, já explicitados no início desta secção *(Tabela 11, Anexo A*);
- ***AttributeLenght* (1 *Byte*)*:*** identifica o número de *bytes* que cada instância do atributo contido numa mensagem ocupa *(Tabela 12, Anexo A*);

[5] Existe um quarto atributo, denominado *Domain Vector* mas que não se enquadra no âmbito desta dissertação de mestrado.

- ***AttributeListLength (2 Bytes):*** presente apenas em *MSRUs* (não é usado em *MMRPDUs* ou *MVRPDUs*), expressa o número de bytes codificados no campo *AttributeList.* Este valor permite saber quantos bytes contem cada mensagem codificada num *MSRPDU*.

As variáveis até agora descritas são únicas no *MSRPDU* e codificadas no início de qualquer *MSRPDU.* Estes campos são essenciais para a identificação do protocolo e do atributo codificado no pacote. Por outro lado, cada *MSRPDU* pode transportar diversas mensagens e cada mensagem, associada a um único tipo de atributo, pode codificar diversas instâncias do mesmo atributo. Estas diferentes instâncias são codificadas em diferentes *VectorAttributes*, cuja composição é descrita de seguida.

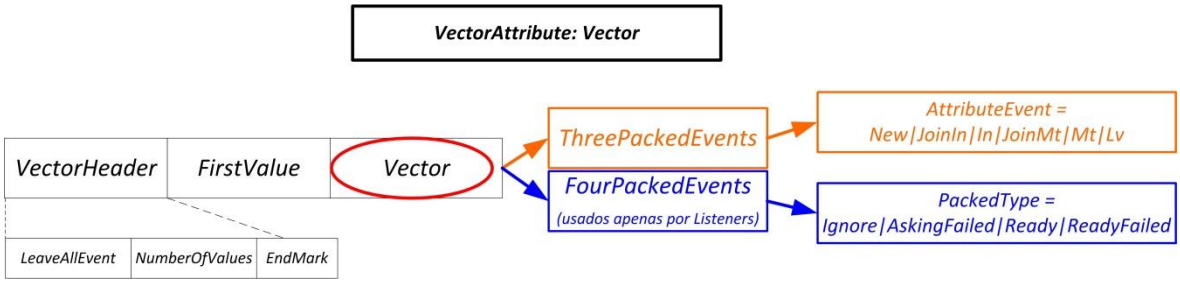

Figura 4.2: Constituição de um *Vector Attribute* e constituição do campo *Vector*

A Figura 4.2 identifica os campos de cada *VectorAttribute* e explicita as variáveis codificadas num campo *Vector.* Cada um contém um cabeçalho (*VectorHeader – 2 Bytes)* constituído por um *bit LeaveAllEvent* e uma variável, *NumberOfValues*, que descreve o número de eventos codificados no campo *Vector.* Este campo, *Vector,* é constituído por um ou mais *bytes* que constituem o *ThreePackedEvents.* Cada *byte* é denominado de *ThreePackedEventByte* e pode encapsular até três eventos associados a um atributo. Estes eventos aplicam-se às máquinas de estado associadas ao protocolo *MRP* e são respetivamente *New, JoinIn, In, JoinMT, Mt, Lv (Tabela 13, Anexo A*). O número de *ThreePackedEventBytes* é definido através da variável *NumberOfValues*, nomeadamente *NumberOfValues/3+1.*

A variável *ThreePackedEvent* é comum aos três atributos *MSRP* apresentados, *Listener, Talker Advertise* e *Talker Failed.* Contudo, para o caso do atributo *Listener*, é também encapsulado em cada *Vector* uma outra variável, o *FourPackedEvent.* Este permite distinguir diferentes tipos de declarações do tipo *Listener: Listener Ready, Listener Asking Ready e Listener Asking Failed (Tabela 14, Anexo A*). Em cada *FourPackedEventByte* podem ser codificados até quatro declarações. O número de *FourPackedEventBytes* codificadas é, tal como no caso do *ThreePackedEvent*Byte, calculado através da variável

NumberOfValues, sendo que neste caso o total de *bytes* é dado por *NumberOfValues/4 + 1.*

No campo *FirstValue* são codificadas as restantes variáveis associadas aos atributos *MSRP*. A Figura 4.3 sumaria as diferentes variáveis e a que atributos dizem respeito.

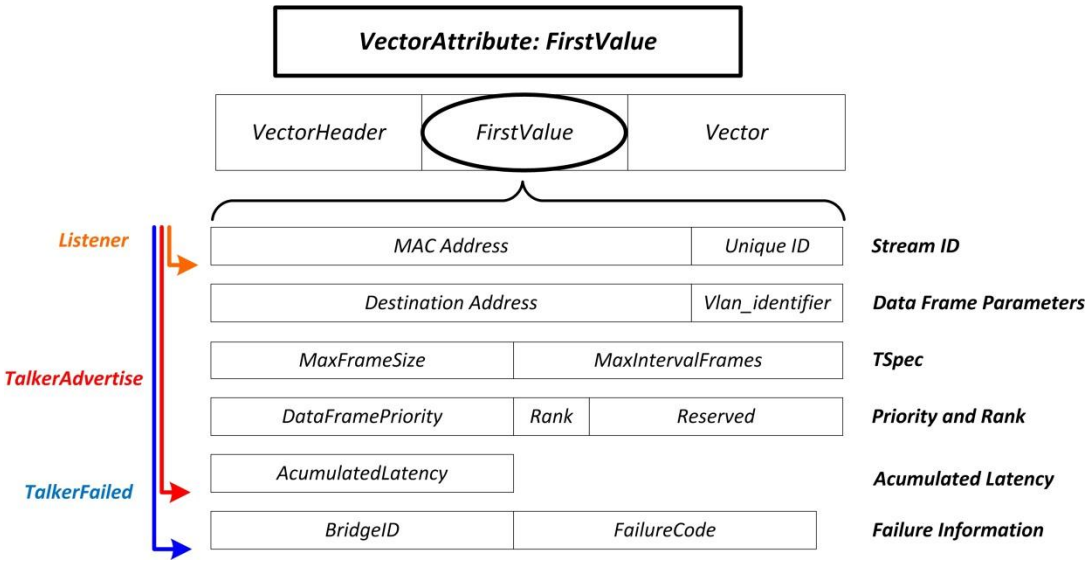

Figura 4.3: Composição de uma estrutura *FirstValue*

Como se pode observar, o identificador da *stream* é comum aos três tipos de atributos e é composto pelo endereço *MAC* do gerador da *stream (6 Bytes)* e um identificador único (2 Bytes).

Caso se trate de um atributo *Talker Advertise,* além do identificador da *stream,* o campo *FirstValue* contém as seguintes estruturas:

- ***Data Frame Parameters:*** informação acerca do destino a que se destina o *PDU;*
- ***Tspec:*** informação acerca das características da *stream* que o *Talker* pretende transmitir;
- ***Priority and Rank:*** informação acerca da prioridade da *stream* associada;
- ***Accumulated Latency:*** informação acerca da latência acumulada pela *stream*

As variáveis codificadas no *Data Frame Parameters* incluem o endereço *MAC* da estação a que se destina a *stream* (6 Bytes) e um identificador da *VLAN* (2 Bytes) caso a estação o permita.

Já no *TSpec* são codificados o *MaxFrameSize* (2 Bytes) e o *MaxIntervalFrames* (2 Bytes). O primeiro representa o tamanho máximo de uma *frame* que o *Talker* pode gerar, excluindo-se qualquer *overhead.* Este valor é usado para alocação dos recursos neces-

sários de forma a garantir a *QdS* requerido pelo gerador da *stream.* O *MaxIntervalFrames* é usado para os mesmos fins do anterior, representado, no entanto, o número máximo de *frames* que o *Talker* é capaz de transmitir num determinado período de tempo, denominado no protocolo por *class measurement interval.*

O valor de *Data Frame Priority* (1 Bytes) é usado para gerar o *Priority Code Point (PCP)* e permite especificar a prioridade da *frame* a ser transmitido pelo *Talker.* Já o valor de *rank* permite distinguir *streams* com estatuto especial de urgência.

A latência acumulada especificada no campo *Accumulated Latency* (4 Bytes) é usada para determinar a latência máxima que uma *stream* pode sofrer ao longo do caminho do *Talker* até ao *Listener.* Este valor é inicializado como sendo igual ao definido na variável *portTcmaxLatency* do *Talker* e é incrementado em cada *bridge/switch* intermédio. Este valor é usado pelo *Listener* no momento de decisão do pedido de reserva de recursos.

A definição do campo *First Value* para um atributo *Talker Advertise* acaba com a variável *Accumulated Latency*. Caso se trate de um *Talker Failed*, uma estrutura extra é adicionada *aos* descritos até aqui, nomeadamente a estrutura *Failure Information* que identifica a causa que levou a que um *Talker Advertise* fosse modificado para um *Talker Failed.* O código de falha (*Tabela 15, Anexo A*) associado é codificado no campo *FailureCode* (1 Bytes) enquanto a identificação da *bridge/switch* onde a falha ocorreu é codificada no campo *bridge ID* (8 Bytes).

Como referido anteriormente, cada *MSRPDU* pode transportar diversas mensagens e estas podem transportar diversas declarações de um mesmo atributo. Contudo, no âmbito deste trabalho considerou-se apenas a codificação de uma mensagem e uma declaração em cada *MSRPDU*, ou seja, fixou-se a variável NumberOfValues a 1.

4.4 Módulo *MAD* de teste

De forma a simular o comportamento do protocolo MSRP no processo de sinalização de reserva de recursos, nomeadamente no que diz respeito ao fluxo de mensagens protocolares, foi desenvolvido um módulo *MAD* de teste para o efeito.

O módulo foi desenvolvido de forma a permitir a um *Talker* iniciar um pedido de reserva e obter resposta do *Listener* ao qual a *stream* se destina. Como pode ser observado através da Figura 4.4, um *Talker* inicia um pedido de reserva através do envio de *MSRPDU* com um atributo *Talker Advertise.* Ao ser recebido pelo *Listener*, este verifica se a *stream* associada lhe é destinada. Sendo, a mensagem recebida é ignorada e descartada. Não sendo, verifica se tem recursos disponíveis para satisfazer o pedido de re-

serva encapsulado no *Talker Advertise* recebido. Caso tenha, é codificado um MSRPDU com um *Listener Ready* associado ao ID da *stream* recebida e enviado para o originador da reserva. Pelo contrário, caso o *Listener* não tenha capacidade de garantir a reserva requerida, este envia um *Listener Asking Failed*.

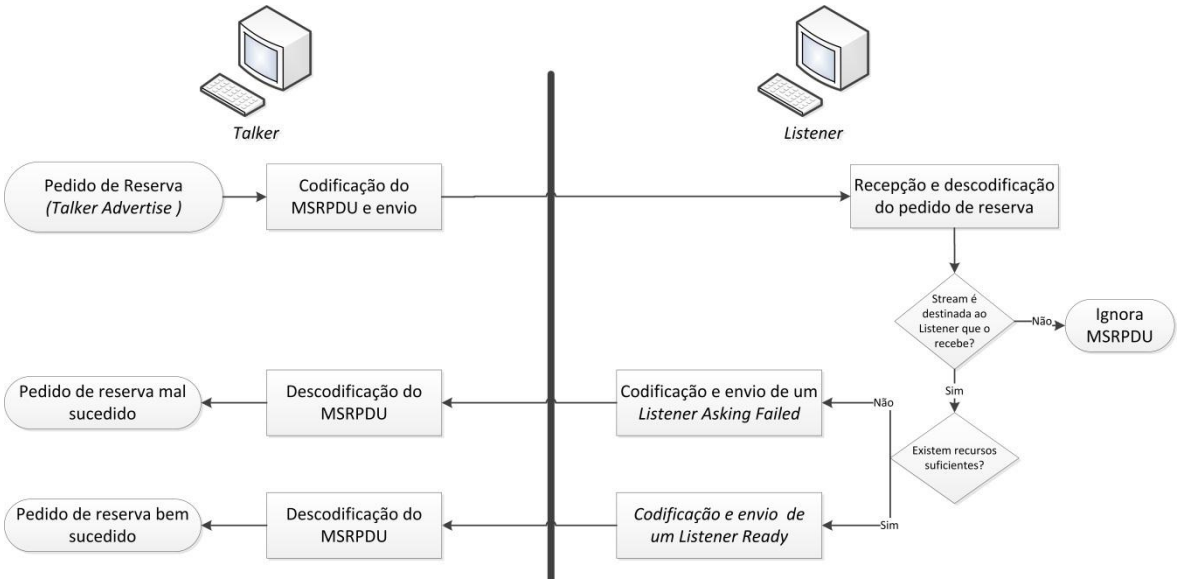

Figura 4.4: Descrição do modo de funcionamento do módulo MAD desenvolvido

A verificação de existência de recursos por parte do *Listener* é um assunto que não é desenvolvida no âmbito desta implementação. Contudo, para teste do módulo desenvolvido desenvolveu-se uma pequena função que retorna “sim” ou “não” consoante os valores recebidos. Fixou-se uma variável representativa da largura de banda disponível no *Listener* (valor virtual) e este, ao receber um *Talker Advertise*, verifica se o valor é suficiente para dar resposta às caraterísticas do tráfego, descritas através dos parâmetros *MaxFrameSize* e *MaxIntervalFrames.*

5 Resultados

A implementação descrita no capítulo anterior requer uma validação experimental. O capítulo que agora se inicia apresentará a metodologia adotada para realização dos testes e respetiva obtenção de resultados, também apresentados neste capítulo.

5.1 Metodologia

No capítulo anterior, em que se descreveu o trabalho desenvolvido no âmbito desta dissertação, distinguiram-se dois blocos essenciais. Um primeiro bloco respeitante à codificação e descodificação de *MSRPDU's* e um segundo bloco simulador do comportamento do protocolo *MSRP*. Assim, os testes e resultados obtidos seguiram também este tipo de abordagem, isto é, foram desenvolvidos um conjunto de testes que validassem a codificação e descodificação dos diferentes *MSRPDU's* e um conjunto de testes que validassem o módulo *MAD* de teste desenvolvido.

O primeiro conjunto de testes consistiu na codificação de vários valores nos diversos campos presentes nos *MSRPDU's*. Dada a impraticabilidade de testar todos os valores possíveis, foram testados os casos limites e alguns casos aleatórios. O segundo conjunto de testes consistiu na verificação da correta resposta de um *Listener* aquando de um pedido de reserva.

De forma a facilitar e agilizar o conjunto de testes realizados, foi também desenvolvido um interface com o utilizador utilizando a biblioteca *ncurses* de modo a separar na interface a informação obtida da *thread* cliente, da informação da *thread* servidora.

Adicionalmente, para que o conteúdo das mensagens enviadas para a rede pudesse ser confirmado de uma forma independente, um analisador de tráfego de rede foi usado, nomeadamente o *wireshark.* As duas ferramentas anteriormente apresentadas permitem verificar a correta codificação e descodificação das mensagens.

As secções que a seguir se apresentam descrevem os resultados obtidos, começando por se apresentar o tipo de interface disponibilizado para depois se apresentarem os resultados obtidos aquando da realização dos testes.

5.2 Interface

A interface desenvolvida utilizando a biblioteca *ncurse* permite ao utilizador iniciar uma simulação de pedido de reserva bem como testar a codificação dos diferentes MSRPDU's (Figura 5.1). Para além disso, facilmente se conseguem identificar na Figura 5.1 as duas *threads* implementadas, sendo que a *thread* responsável pela interação com o utilizador aparece na parte superior da consola enquanto a *thread* responsável pela receção de mensagens aparece na parte inferior da consola.

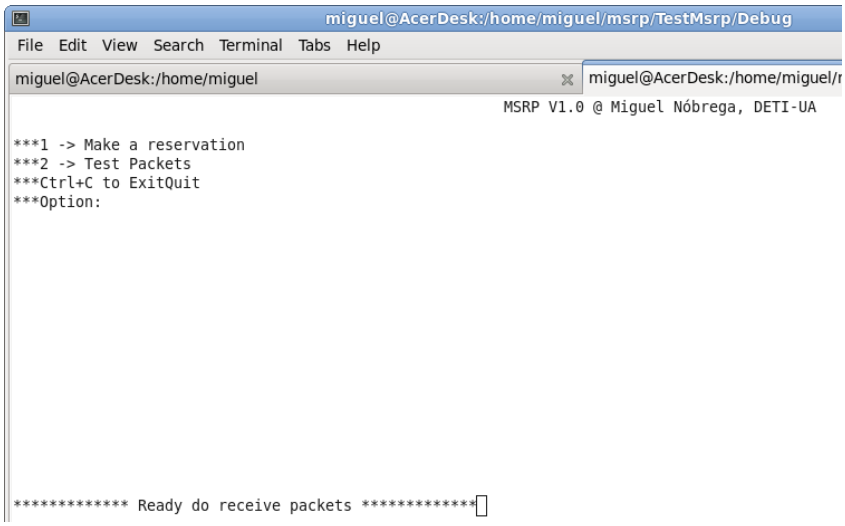

Figura 5.1: Menu principal do interface desenvolvido para teste do sistema

A seleção de qualquer uma das opções do menu principal permite ao utilizador escolher os parâmetros que devem ser codificados no *MSRPDU* a ser enviado para a rede (um exemplo para o caso de um *Talker Advertise* pode ser observado na Figura 5.2).

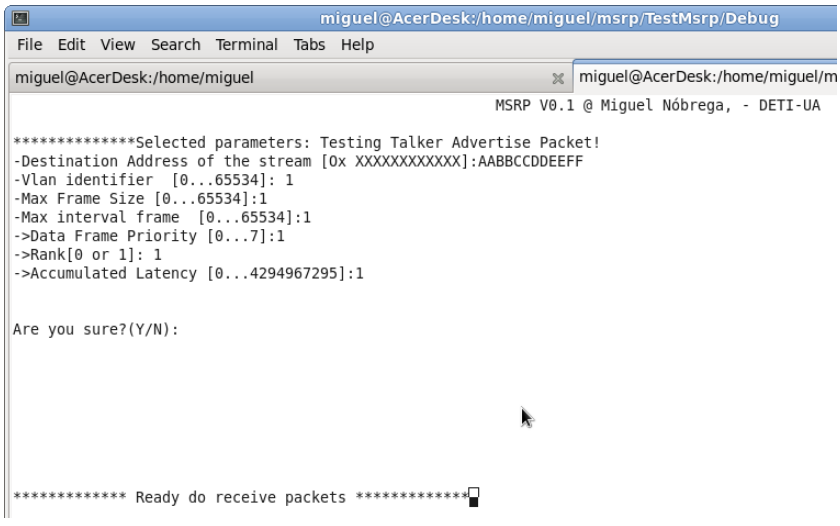

Figura 5.2: Parâmetros passíveis de serem introduzidos para teste no caso do início de um pedido de reserva (*Talker Advertise)*

5.3 Mensagens

Para testar os diferentes tipos de mensagens e parâmetros, diversos testes foram realizados. Como seria impraticável apresentar aqui todos os testes efetuados, escolheram-se quatro diferentes casos achados essenciais para testar o sistema desenvolvido. Dois casos em que se testaram valores limites dos parâmetros e dois casos com valores totalmente aleatórios.

Para confirmação da correta codificação e descodificação das mensagens é usado o interface desenvolvido em conjunto com o *wireshark,* tanto no nó *Talker* como no nó *Listener.* As próximas subsecções apresentam a captura das mensagens trocadas referentes aos casos anteriormente referidos.

5.3.1 Mensagens *Talker Advertise*

Esta secção apresenta as capturas efetuadas que comprovam a correta codificação e descodificação de uma mensagem *Talker Advertise* através da apresentação dos resultados dos quatro casos de teste (um caso com valores mínimos, 2 casos com valores aleatórios e um caso com valores máximos). Para cada um desses casos são apresentados um grupo de duas capturas feitas no *Talker* e no *Listener*, uma usando o interface desenvolvido e outra usando o *wireshark.*

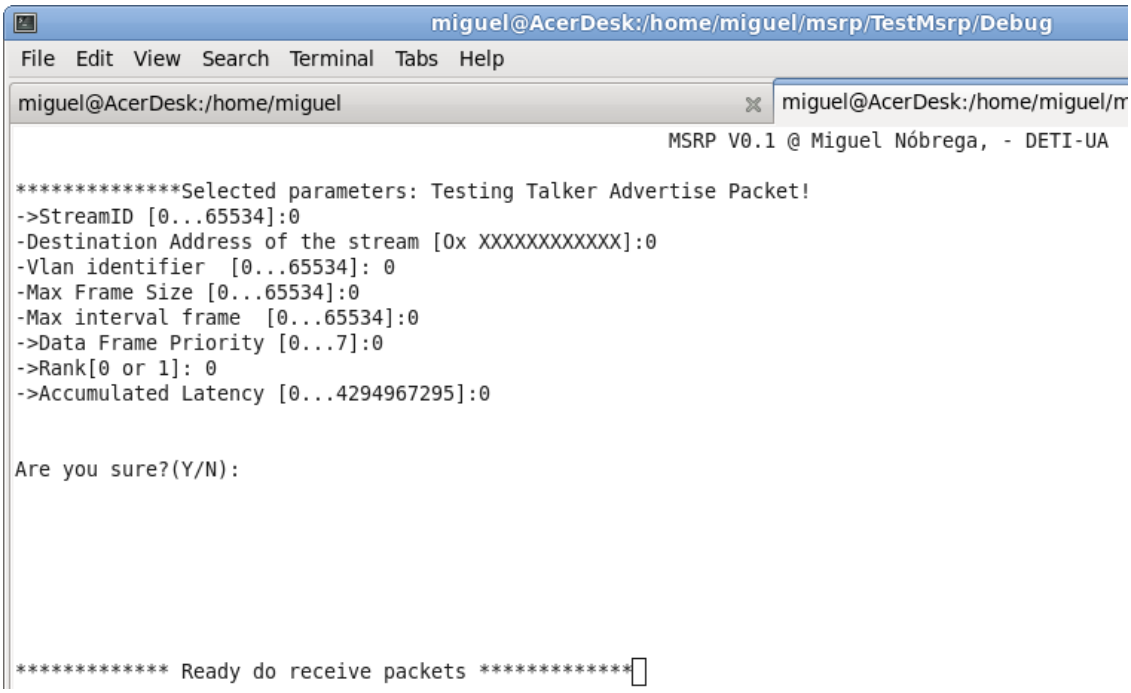

Figura 5.3: Introdução na consola dos valores a codificar num *Talker Advertise* : valores mínimos

O primeiro caso testado correspondeu à codificação dos diferentes campos da mensagem *Talker Advertise* com o valor mínimo possível. Esses valores foram introduzidos manualmente pelo utilizador usando o interface desenvolvido sendo que este não permitiu, como desejado, a introdução de valores inferiores ao valor mínimo que a variável correspondente permite. Os parâmetros introduzidos usando a interface referida podem ser visualizados na Figura 5.3 enquanto a captura da mensagem enviada usando *wireshark* é apresentada na Figura 5.4.

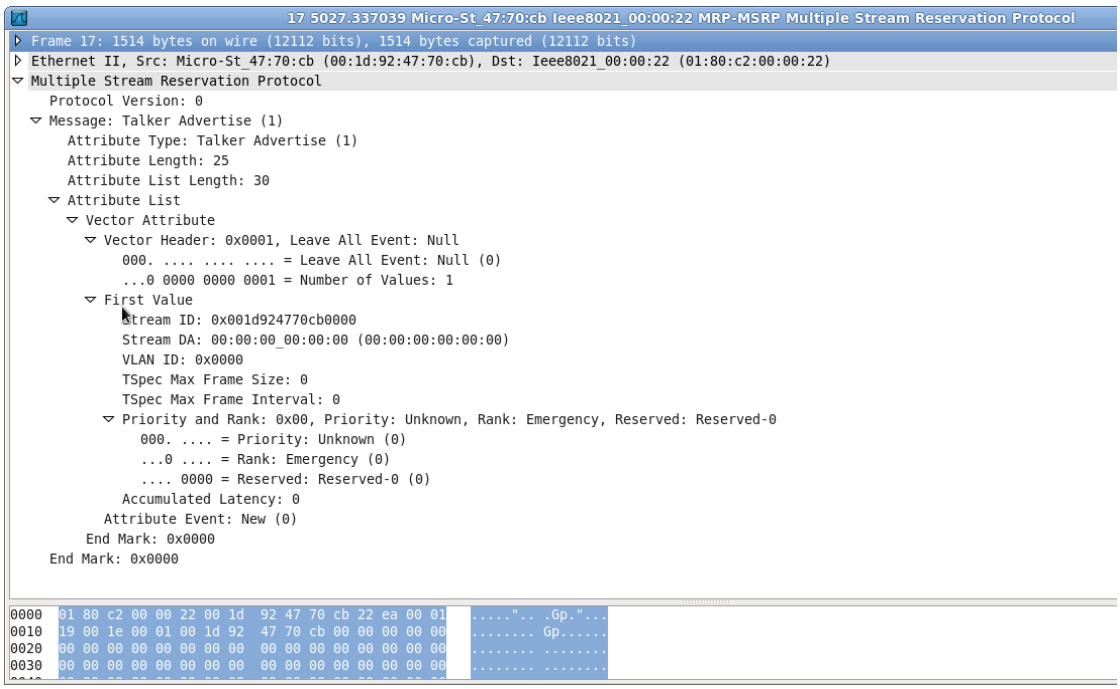

Figura 5.4: Confirmação do MSRPDU codificado usando *wireshark:* valores mínimos

A análise das figuras anteriores permite concluir a correta codificação de uma mensagem *Talker Advertise* considerando-se o caso do uso dos valores mínimos. De notar que o campo *stream ID* vem corretamente codificado com o valor do endereço MAC do dispositivo e que a variável *NumberOfValues* apresenta o valor 1 como pré-definido. Por fim, notar que o valor codificado no *Attribute Event* não tem qualquer relevância para efeito desta dissertação e como tal também foi fixado como *New* em todos os casos.

Para finalizar este caso de teste, a Figura 5.5 ilustra o resultado da descodificação da mensagem *Talker Advertise* por parte do *Listener* e cujos valores descodificados são apresentados usando o interface desenvolvido. De notar que apesar de a nível lógico não fazer sentido a codificação do valor 0 em alguns parâmetros nada é especificado nas normas do protocolo que impossibilite a codificação desse valor dentro da gama de valores disponíveis.

```
   ************* Ready do receive packets **********
Received packet from: 1D924770CB,
AttributeType encoded: 0X1 ->   Talker Advertise
Decoding Talker Advertise Packet.........

Protocol Version: 0
Attribute Type: 1
Attribute Lenght 19
Attribute List Lenght: 1E
Leave All Event: 0
Number of values/events:  1
Talker MAC Address: 1D924770CB
Stream Unique ID: 0
Stream Destination Address: 0
Vlan ID: 0
TSpec Bandwith: 0
TSpec Frame Rate: 0
Traffic class: 0
Rank: 0
Reserved: 0
Acummulated Latency: 0
Three Packed Event Byte: 0
EndMark1: 0
EndMark2: 0
```

Figura 5.5: Recepção, descodificação e impressão na consola do *MSRPDU* recebido pelo *Listener* referente ao *Talker Advertise* ilustrado na Figura 5.3

Depois de apresentado o primeiro caso teste referente à codificação de uma mensagem *Talker Advertise* com valores mínimos, apresentar-se-á de seguida um outro caso teste, considerando-se desta vez a codificação de uma mensagem *Talker Advertise* com valores aleatórios (dentro da gama possível). Os valores utilizados são representados na Figura 5.6.

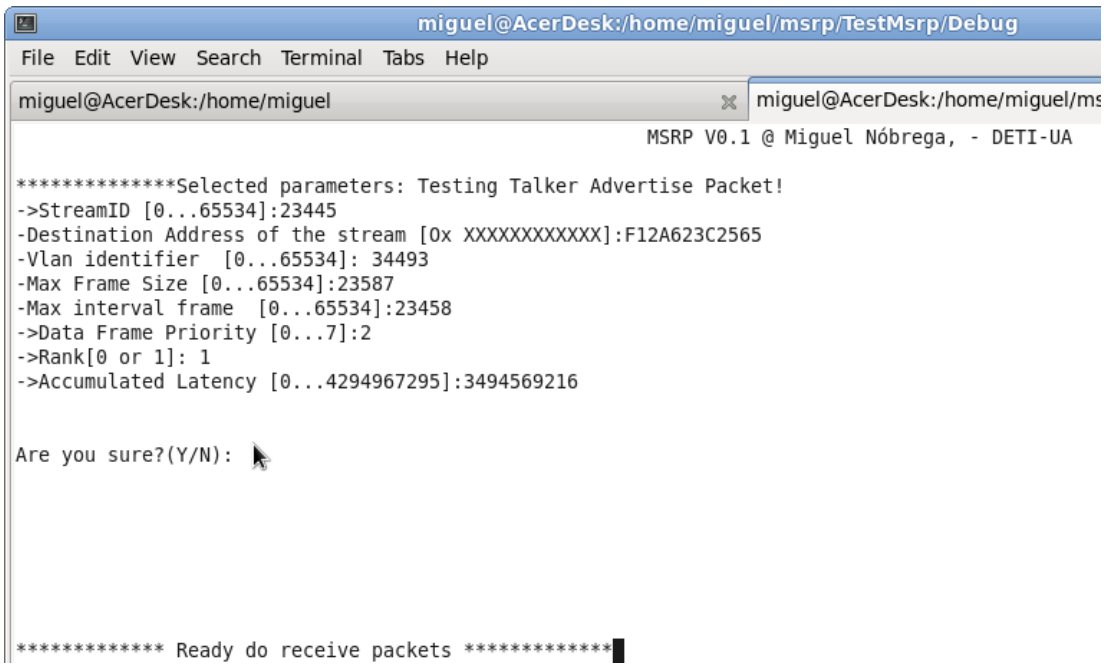

Figura 5.6: Introdução na consola dos valores a codificar num *Talker Advertise* : valores aleatórios (1)

A Figura 5.7 permite ilustra a captura efetuada pelo *wireshark* da mensagem *Talker Advertise* codificada com os valores indificados na Figura 5.6 e recebida por um *Listener.*

Como se pode observar os valores apresentados pelo *wireshark* permitem concluir que a codificação foi efetuada com sucesso e que a mensagem foi enviada pela rede corretamente preenchida. De notar que o campo *Stream ID* se encontra corretamente preenchido com o valor do endereço MAC do dispositivo físico que o originou mais o endereço único introduzido pelo utilizador (23445 equivale a 0x5b95).

Outra nota de destaque prende-se com o facto de o *wireshark* associar o valor 2 do campo *DataFramePriority* a uma classe de tráfego B. De facto até agora apenas se definiram duas classes de tráfego, A e B [41], associadas a uma prioridade de valor 3 e 2 respetivamente. Consequentemente, o *wireshark* apenas é capaz de identificar esse dois tipos de classes de tráfego.

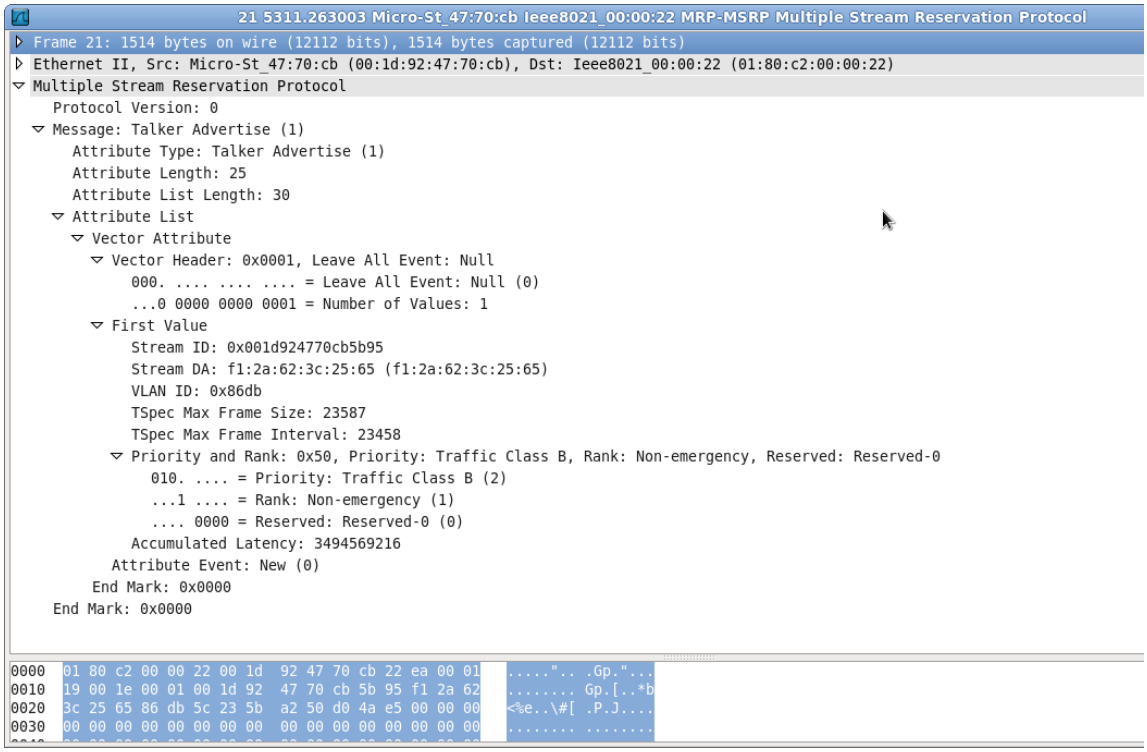

Figura 5.7: Confirmação do MSRPDU codificado usando *wireshark:* valores aleatórios (1)

A mensagem recebida e descodificada no *Listener* é detalhada na Figura 5.8. Comparando os valores apresentados após a descodificação e aqueles apresentados através do uso do analisador de redes *wireshark* verifica-se existir uma concordância absoluta.

```
************* Ready do receive packets *************
Received packet from: 1D924770CB,
AttributeType encoded: 0X1 ->   Talker Advertise
Decoding Talker Advertise Packet..........

Protocol Version: 0
Attribute Type: 1
Attribute Lenght 19
Attribute List Lenght: 1E
Leave All Event: 0
Number of values/events:  1
Talker MAC Address: 1D924770CB
Stream Unique ID: 5B95
Stream Destination Address: F12A623C2565
Vlan ID: 86BD
TSpec Bandwith: 23587
TSpec Frame Rate: 23458
Traffic class: 2
Rank: 1
Reserved: 0
Acummulated Latency: 3494569216
Three Packed Event Byte: 0
EndMark1: 0
EndMark2: 0
```

Figura 5.8: Receção, descodificação e impressão na consola do MSRPDU recebido pelo Listener referente ao *Talker Advertise* ilustrado na Figura 5.6

Por forma a verificar que a correta codificação e descodificação da mensagem apresentada no caso teste anterior não foi um mero acaso, apresentam-se de seguida os resultados obtidos para um novo caso de teste usando novos valores aleatórios. Os valores utilizados são apresentados na Figura 5.9.

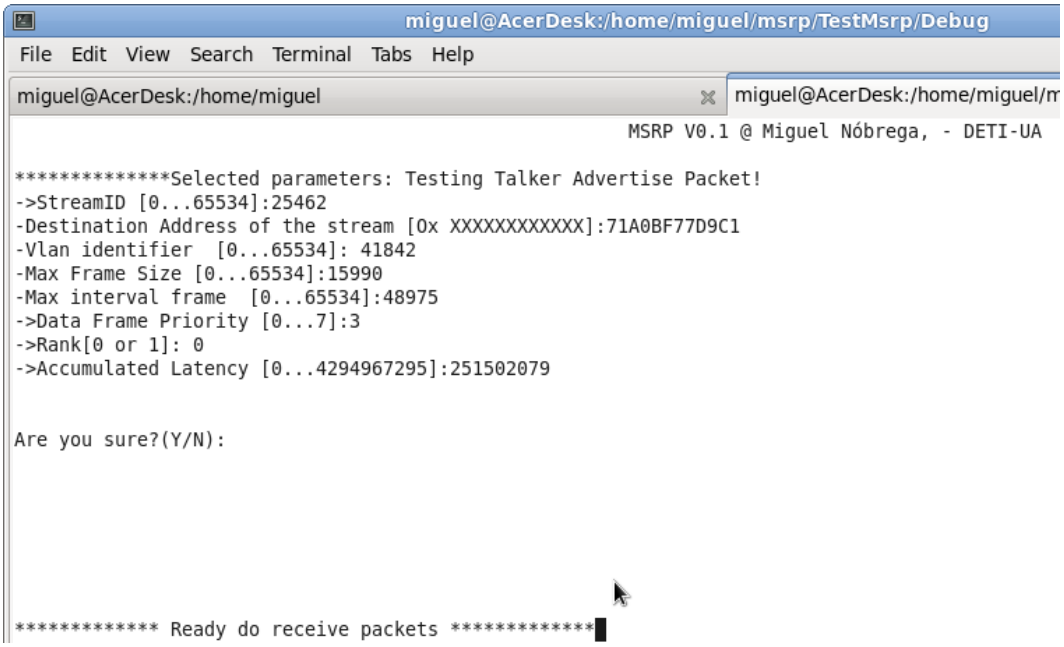

Figura 5.9: Introdução na consola dos valores a codificar num *Talker Advertise*: valores aleatórios (2)

A Figura 5.10 ilustra a captura efetuada à mensagem codificada no *Talker,* enviada para a rede e recebida pelo *Listener* usando o *wireshark*. Como se pode observar os parâmetros descodificados coincidem com os introduzidos pelo utilizador e identificados na Figura 5.9. Mais uma vez, notar que o *wireshark* associa o valor de prioridade 3 à classe de tráfego A, como esperado e especificado.

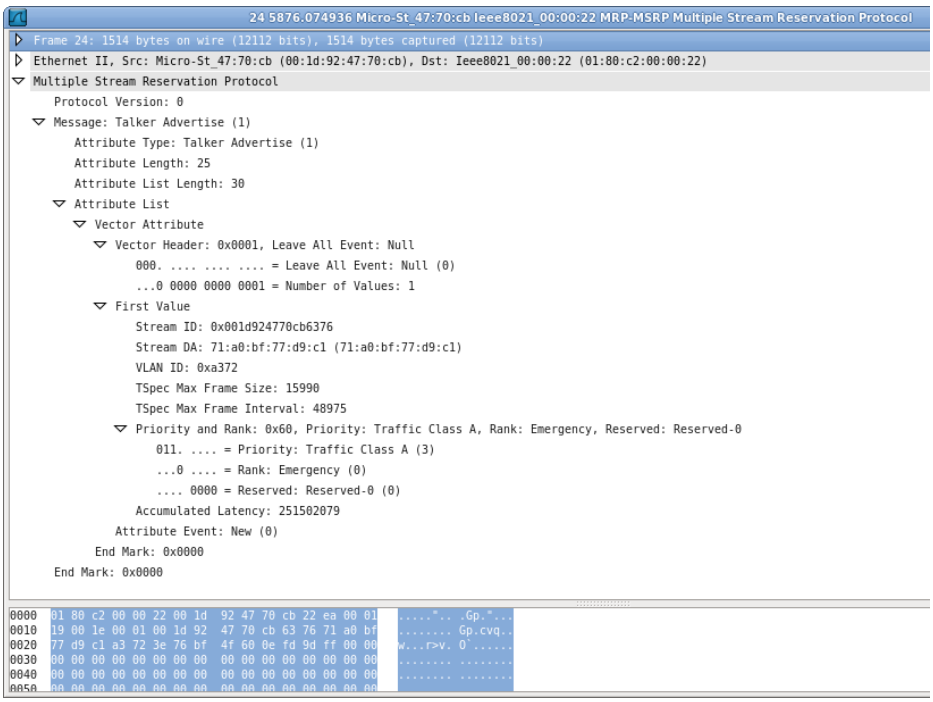

Figura 5.10: Confirmação da correta descodificação do MSRPDU ilustrado na Figura 5.9 usando *wireshark*

A Figura 5.11, e para finalizar este caso de teste, corresponde à impressão dos valores descodificados usando o interface desenvolvido. Estes correspondem ao esperado e portanto validam mais um caso de teste.

```
************* Ready do receive packets *************
Received packet from: 1D924770CB,
AttributeType encoded: 0X1 ->   Talker Advertise
Decoding Talker Advertise Packet..........

Protocol Version: 0
Attribute Type: 1
Attribute Lenght 19
Attribute List Lenght: 1E
Leave All Event: 0
Number of values/events:  1
Talker MAC Address: 1D924770CB
Stream Unique ID: 6376
Stream Destination Address: 71A0BF77D9C1
Vlan ID: A372
TSpec Bandwith: 15990
TSpec Frame Rate: 48975
Traffic class: 3
Rank: 0
Reserved: 0
Acummulated Latency: 251502079
Three Packed Event Byte: 0
EndMark1: 0
EndMark2: 0
```

Figura 5.11: Receção, descodificação e impressão na consola do *MSRPDU* recebido pelo *Listener* referente ao *Talker Advertise* ilustrado na Figura 5.9

Para terminar esta secção de testes apresentam-se as capturas feitas para o caso em que se codificam os diversos parâmetros de uma mensagem *Talker Failed* utilizando os seus valores máximos (ver Figura 5.12).

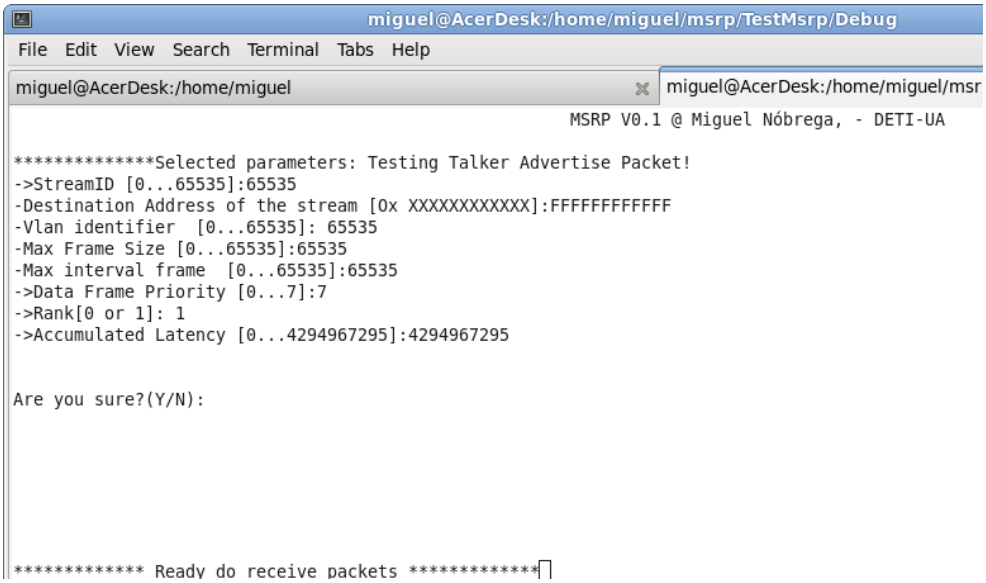

Figura 5.12: Introdução na consola dos valores a codificar num *Talker Advertise* : valores máximos

A Figura 5.13 ilustra a captura feita à mensagem codificada pelo *Talker*, enviada para a rede e recebida pelo *Listener*. Como se pode verificar todos os campos apresentam o seu valor máximo com exceção do campo destinado à identificação da fonte da *stream* que apresentada o endereço MAC do dispositivo que a gerou.

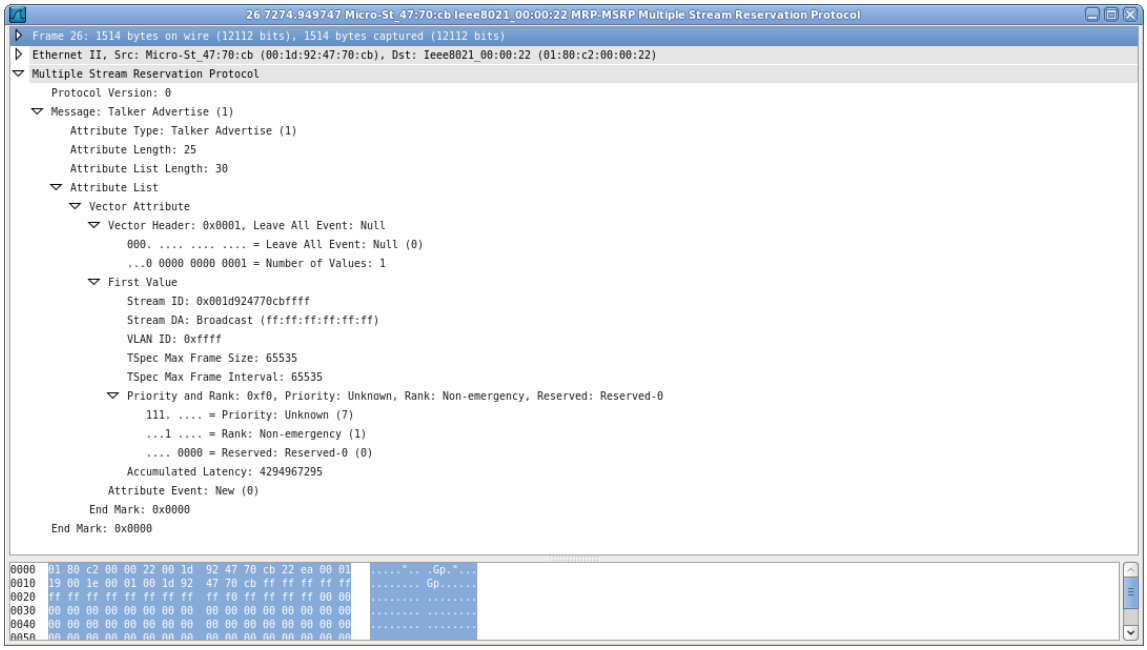

Figura 5.13: Confirmação da correta descodificação do MSRPDU ilustrado na Figura 5.12 usando *wireshark*

Por fim, comprova-se através da observação da Figura 5.14 a correta descodificação da mensagem recebida no *Listener.* De facto, todos os valores apresentam o seu valor máximo tal como codificado pelo *Talker.*

Este caso de teste encerra esta seção de testes, concluindo-se que as rotinas desenvolvidas que permitem a codificação, envio e descodificação de mensagens do tipo *Talker Advertise* funcionam corretamente dentro das condições assumidas.

```
************* Ready do receive packets *************
Received packet from: 1D924770CB,
AttributeType encoded: 0X1 ->   Talker Advertise
Decoding Talker Advertise Packet..........

Protocol Version: 0
Attribute Type: 1
Attribute Lenght 19
Attribute List Lenght: 1E
Leave All Event: 0
Number of values/events:  1
Talker MAC Address: 1D924770CB
Stream Unique ID: FFFF
Stream Destination Address: FFFFFFFFFFFF
Vlan ID: FFFF
TSpec Bandwith: 65535
TSpec Frame Rate: 65535
Traffic class: 7
Rank: 1
Reserved: 0
Acummulated Latency: 4294967295
Three Packed Event Byte: 0
EndMark1: 0
EndMark2: 0
```

Figura 5.14: Receção, descodificação e impressão na consola do MSRPDU recebido pelo *Listener* referente ao T*alker Advertise* ilustrado na Figura 5.12

5.3.2 Mensagens *Talker Failed*

As semelhantes implementações das mensagens *Talker Advertise* e *Talker Failed* fez com que os procedimentos de teste para estes dois tipos de mensagens fossem muito idênticos. A mensagem *Talker Failed* contém, apenas dois campos adicionais referentes à identificação do tipo de falha e identificação do nó que originou a falha.

Assim, e dado que os resultados obtidos se assemelham grandemente aos já apresentados para o caso de uma mensagem *Talker Advertise*, não serão apresentadas novamente neste capítulo as capturas efetuadas. Contudo, as mesmas podem ser consultadas no Anexo B.

5.3.3 Mensagens *Listener*

As mensagens *Listener* apresentam um número bastante menor de parâmetros a testar, sendo o mais importante o *fourPackedEvents* que identifica o tipo de mensagem *Listener* a que a mensagem diz respeito. Como verificado, cada *byte* do referido

fourPackedEvents pode codificar até quatro diferentes tipos de instâncias *Listener.* Com isto, o protocolo *MSRP* permite o agrupamento de diversas mensagens enviadas por diferentes *Listeners,* otimizando assim o funcionamento do mesmo. Contudo, e como a implementação apresentada pressupôs a existência de apenas dois nós terminais, ignora-se a possibilidade de existência de agrupamento de mensagens e portanto assume-se, para efeitos de teste, a existência de apenas uma declaração do tipo *Listener* dentro do *fourPackedEvent.*

Porém, é necessário testar se o sistema desenvolvido permite a codificação dos três[6] tipos de declarações associadas ao atributo *Listener: Listener Ready, Listener Ready Failed, Listener Asking Failed.* Para tal, o interface desenvolvido para efeitos de teste permite a codificação de *MSRPDU's* com estes diferentes tipos de declarações.

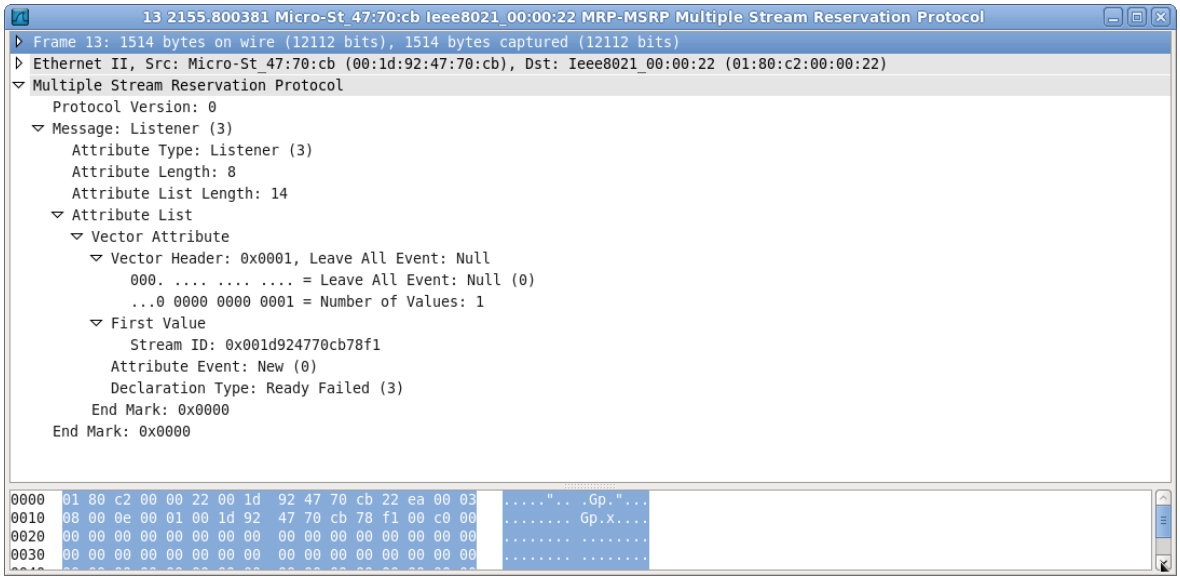

Figura 5.15: Mensagem enviada contendo um *Listener Ready Failed*

As figuras Figura 5.15 e Figura 5.16 ilustram os testes efetuados para o caso de uma declaração do tipo *Listener Ready Failed*. De notar que no âmbito destes testes usou-se um número arbitrário para o valor do identificador único da *stream,* assim como se considerou o *MAC* originador da *stream*.

[6] Na verdade existem 4 tipos de declarações, contudo dado que o agrupamento de mensagens (*merging)* foi ignorado para efeitos deste trabalho, o tipo de declaração *ignore* foi igualmente ignorado.

```
************* Ready do receive packets *************
Received packet from: 1D924770CB,
AttributeType encoded: 0X3 -> Listener;
Decoding Listener Packet..........
Received Listener Ready Failed.

Protocol Version: 0
Attribute Type: 3
Attribute Lenght 8
Attribute List Lenght: E
Leave All Event: 0
Number of values/events:  1
Stream ID TALKER MAC: 1D924770CB
Stream ID Unique ID: 78F1
Three Packed Event Byte: 0
Four packed Event Byte: C0
EndMark1: 0
EndMark2: 0
```

Figura 5.16: Mensagem recebida referente à ilustrada na Figura 5.15

5.4 Módulo *MAD* de teste

A verificação do correto funcionamento deste módulo foi efetuada usando a mesma metodologia usada anteriormente para testes dos MSRPDU's. Neste ponto a chave está na confirmação do correto funcionamento do módulo MAD de teste desenvolvido e ilustrado na Figura 4.4. Para tal é necessário provar que o sistema funciona corretamente considerando as seguintes situações:

1. A sinalização de reserva de recursos é efetuada com sucesso, terminando com a receção por parte do *Talker* de um *Listener Ready*. Este *Listener Ready* deverá ser enviado pelo *Listener* a que se destina a *stream* associada*,* após este receber um *Talker Advertise;*
2. A sinalização de reserva de recursos não é efetuada com sucesso, terminado com a receção por parte do *Talker* de um *Listener Asking Failed.* Este deverá ser enviado por um *Listener* a que se destina a *stream* associada, após este receber um *Talker Advertise*.

A primeira situação descrita será detalhadamente comprovada de seguida recorrendo às ferramentas já apresentadas. Já a segunda, devido à sua semelhança com a primeira é deixada para anexo (Anexo C).

5.4.1 Sinalização de reserva bem sucedida

Os passos efetuados para realização dos testes para o caso de uma reserva bem sucedida bem como os dados recolhidos são descritos e apresentadas de seguida.

O pedido de sinalização de reserva de recursos e introdução dos parâmetros referentes ao pedido de reserva de recursos é feito através do interface desenvolvido. A Figura 5.17 representa os parâmetros a serem codificados numa mensagem *Talker Advertise* que iniciará o processo de sinalização. De notar que os parâmetros referentes ao *Tspec* deverão resultar num pedido bem sucedido dado existirem recursos suficientes para que sejam transferidos os pactos pertencentes à *stream.*

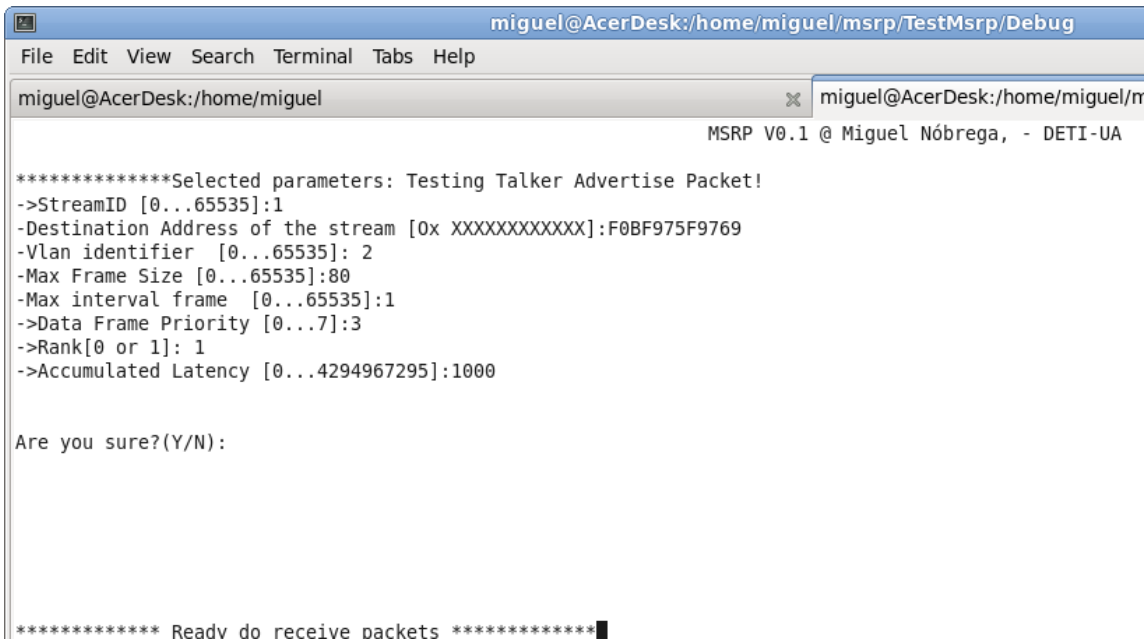

Figura 5.17: Parâmetros descritivos de um pedido de reserva de recursos. Este pedido é iniciado com o envio de um *Talker Advertise*

A codificação dos parâmetros é confirmada através da Figura 5.18 através do recurso do *wireshark.* A mensagem *Talker Advertise* é então enviada pelo *Talker* em direção ao *Listener* que a descodifica e analisa. Dado que o endereço destino da *stream* coincide com o endereço *MAC* do *Listener* e este tem recursos suficientes para suportar o pedido de reserva, este responde com o envio de um *Listener Ready* em direção ao remetente do pedido tal como se irá comprovar de seguida.

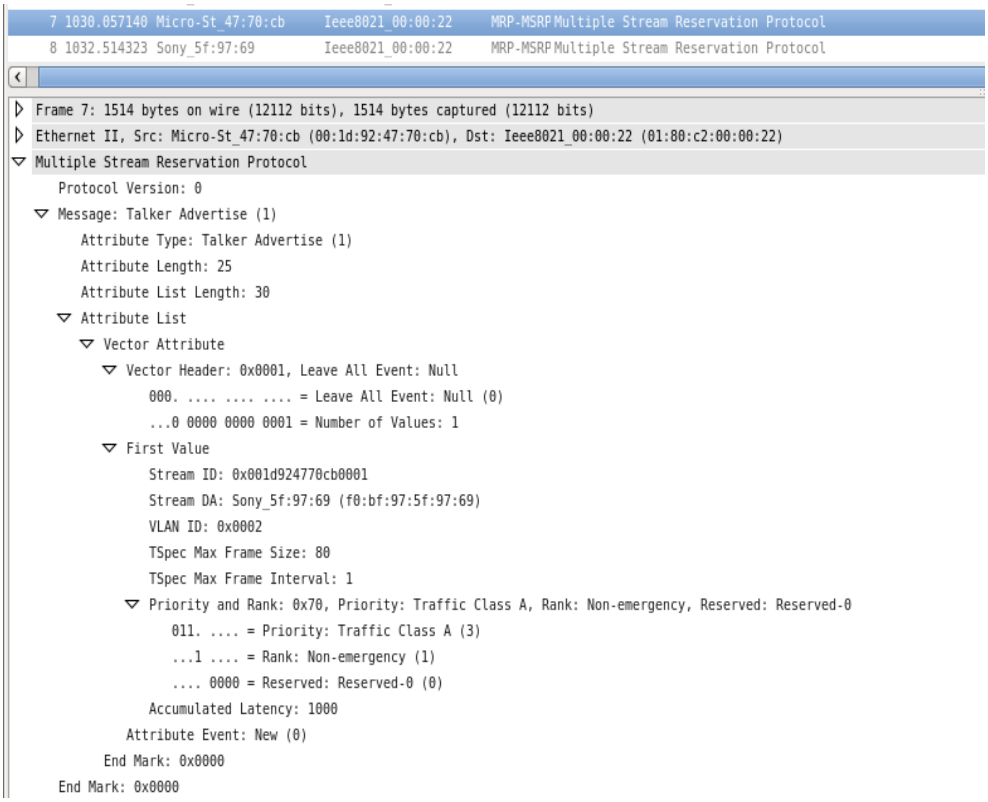

Figura 5.18: Talker Advertise recebido pelo *Listener* (*wireshark*)

A Figura 5.19 confirma então a esperada reação do *Listener* após recebimento de um pedido de reserva de recursos. Como se pode observar o *Listener* recebe uma mensagem *Talker Advertise,* descodifica-a corretamente, verifica o destinatário da *stream* referenciada, verfica a existência de largura de banda e responde com uma mensagem *Listener Ready.*

```
************ Ready do receive packets ************
Received packet from: 1D924770CB,
AttributeType encoded: 0X1 ->   Talker Advertise
Decoding Talker Advertise Packet..........

Protocol Version: 0
Attribute Type: 1
Attribute Lenght 19
Attribute List Lenght: 1E
Leave All Event: 0
Number of values/events:  1
Talker MAC Address: 1D924770CB
Stream Unique ID: 1
Stream Destination Address: F0BF975F9769
Vlan ID: 2
TSpec Bandwith: 80
TSpec Frame Rate: 1
Traffic class: 3
Rank: 1
Reserved: 0
Acummulated Latency: 1000
Three Packed Event Byte: 0
EndMark1: 0
EndMark2: 0
The stream is destined to this Listener!
Bandwidth available
Answering with a Listener Ready MSRPDU!
Releasing memory.....
```

Figura 5.19: Reação do *Listener* ao *Talker Advertise* recebido

```
************* Ready do receive packets *************
Received packet from: F0BF975F9769,
AttributeType encoded: 0X3 -> Listener;
Decoding Listener Packet..........
Received Listener Ready.

Protocol Version: 0
Attribute Type: 3
Attribute Lenght 8
Attribute List Lenght: E
Leave All Event: 0
Number of values/events:  1
Stream ID TALKER MAC: 1D924770CB
Stream ID Unique ID: 1
Three Packed Event Byte: 0
Four packed Event Byte: 80
EndMark1: 0
EndMark2: 0
Signalization completed!
Releasing memory.....
```

Figura 5.20: *Listener Ready* recebido pelo *Talker,* confirmando o pedido de reserva bem sucedida

Por fim, as Figura 5.20 e Figura 5.21 confirmam a receção do *Listener Ready* enviado pelo *Listener* e correta descodificação da mensagem. Este passo conclui o processo de sinalização de reserva de recursos com sucesso, sendo que a partir deste momento o *Talker* poderá começar a transmitir os dados correspondentes ao pedido de reserva.

```
    7 1029.823929   Micro-St_47:70:cb    Ieee8021_00:00:22    MRP-MSRP    Multiple Stream Reservation Protocol
    8 1032.281325   Sony_5f:97:69        Ieee8021_00:00:22    MRP-MSRP    Multiple Stream Reservation Protocol

▷ Frame 8: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits)
▷ Ethernet II, Src: Sony_5f:97:69 (f0:bf:97:5f:97:69), Dst: Ieee8021_00:00:22 (01:80:c2:00:00:22)
▽ Multiple Stream Reservation Protocol
    Protocol Version: 0
  ▽ Message: Listener (3)
      Attribute Type: Listener (3)
      Attribute Length: 8
      Attribute List Length: 14
    ▽ Attribute List
      ▽ Vector Attribute
        ▽ Vector Header: 0x0001, Leave All Event: Null
            000. .... .... .... = Leave All Event: Null (0)
            ...0 0000 0000 0001 = Number of Values: 1
        ▽ First Value
            Stream ID: 0x001d924770cb0001
          Attribute Event: New (0)
          Declaration Type: Ready (2)
        End Mark: 0x0000
  End Mark: 0x0000
```

Figura 5.21: Confirmação usando *wireshark* da receção por parte do *Talker* de um *Listener Ready,* finalizando desta forma a sinalização de um pedido de reserva de recurso

6 Conclusões

As comunicações de Tempo-Real, até agora constantemente associadas aos *fieldbuses*, passaram a ser ligadas à tecnologia de comunicação mais comummente usada em redes de área local, a *Ethernet.* A sua massificação, aliada aos baixos preços e velocidades elevadas levou ao desenvolvimento de soluções que permitissem o seu uso em sistemas distribuídos de Tempo-Real.

Contudo, os protocolos Tempo-Real que foram sendo desenvolvidos apresentam algumas limitações. Em particular, a obtenção de garantias temporais é normalmente obtida à custa de uma abordagem estática. O switch HaRTES foi desenvolvido por forma a conciliar garantias temporais com flexibilidade operacional, suportando nomeadamente QdS variável e suporte a diferentes tipos de tráfego.

Porém, o *switch* foi desenvolvido com base em protocolos próprios, não normalizados internacionalmente, sendo esta uma limitação do mesmo. Por conseguinte, tornou-se de todo o interesse a integração de um protocolo normalizado internacionalmente capaz de gerir as funcionalidades do *switch* de uma forma integrada.

Foram apresentados e discutidos dois protocolos de sinalização de reserva de recursos, *RSVP* e *SRP* e avaliada a sua possível integração no *HaRTES.* O facto de o *SRP* ter sido desenvolvido para redes com *bridges/switches* e o facto de permitir ao *switch HaRTES* definir *deadlines,* essenciais para garantir um escalonamento que forneça garantias às aplicações, fez com que fosse escolhido em detrimento do *RSVP*.

O protocolo *SRP* assenta no protocolo *MRP* que permite o registo e propagação de registos de atributos ao longo de uma rede, tendo este uma arquitetura bastante complexa. A aplicação *SRP* depende do *MRP*, pelo que exige uma implementação do mesmo. Tanto quanto pudemos determinar, não existem na atualidade implementações *MRP* disponíveis, pelo que o trabalho apresentado começou pela implementação das mensagens protocolares exigidas pelo *SRP* bem como pela implementação de uma versão teste do módulo *MAD*, para efeitos de teste

Foram conduzidos testes que mostraram o correto funcionamento do módulo *MAD* desenvolvido no que respeita à correta codificação e interpretação das mensagens trocadas, correção que pode ser verificada através da utilização de um interpretador de pacotes. Os testes permitiram ainda comprovar o correto funcionamento do módulo *SRP* no que respeita às máquinas de estados, que entretanto não puderam ser concluídas.

6.1 Trabalho futuro

O presente trabalho incluiu o desenvolvimento de um protocolo de reserva de recursos que pertence a um conjunto de normas definidas pelo *IEEE* que não foram implementadas. A sua implementação dotaria o *switch HaRTES* de suporte de *VLAN's,* suporte de redundância em *bridging* e suporte de um sistema de sinalização de reserva de recursos normalizado. Para o seu suporte, deveria ser desenvolvido:

- Módulo *MVRP*
- Módulo *MMRP*
- Standard *STP* ou módulo *spanning tree*
- Módulo *MSRP*

Adicionalmente, e dada a falta de tempo e quantidade de trabalho necessário à completa implementação da norma *SRP,* alguns detalhes de implementação ficaram por ser concluídos:

- Conclusão da implementação das máquinas de estado do *MRP:* como foi referido, o protocolo *SRP* assenta no funcionamento do protocolo *MRP* que, para o seu funcionamento, define um conjunto de máquinas de estados não concluídas no âmbito desta dissertação;
- O trabalho desenvolvido não incluiu o desenvolvimento do módulo que implementa o controlo de admissão (*MAP)* essencial à gestão de recursos. Assim, a componente da norma implementada só pode ser utilizada em estações terminais, sendo necessário implementar o módulo *MAP* para que o *software* possa ser aplicado num *switch;*
- Módulo da aplicação *MSRP* e módulo *MAD:* estes dois módulos deverão ser completados e integrados de forma a seguir as especificações presentes na norma que o definem;
- Integração do *MAP* no *switch HaRTES.*

7 Referências

[1] R. Marau, L. Almeida, and P. Pedreiras, "Enhancing real-time communication over COTS Ethernet switches," 2006, pp. 295-302.

[2] *ETHERNET Powerlink Protocol* Available: http://www.ethernet-powerlink.org/

[3] S. Varadarajan and T. Chiueh, "EtheReal: A host-transparent real-time Fast Ethernet switch," 1998, pp. 12-21.

[4] H. Hoang, M. Jonsson, A. Kallerdahl, and U. Hagström, "Switched real-time Ethernet with earliest deadline first scheduling-protocols and traffic handling," *Parallel and Distributed Computing Practices,* vol. 5, pp. 105-115, 2002.

[5] (2008). *Profinet and IT, A PI White Paper*. Available: http://us.profibus.com/docs/pi_white_paper_profinet_it_en_v1_0.pdf

[6] H. Charara and C. Fraboul, "Modelling and simulation of an avionics full duplex switched ethernet," in *Telecommunications, 2005. advanced industrial conference on telecommunications/service assurance with partial and intermittent resources conference/e-learning on telecommunications workshop. aict/sapir/elete 2005. proceedings*, 2005, pp. 207-212.

[7] I. A. T. Group. *IEEE 802.1 Audio/Video Bridging (AVB)*. Available: http://www.ieee802.org/1/pages/avbridges.html

[8] "IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks. Amendment 14: Stream Reservation Protocol (SRP)," *IEEE Std 802.1Qat-2010 (Revision of IEEE Std 802.1Q-2005),* 2010.

[9] G. C. Buttazzo. (2005, Hard Real-Time Computing Systems : Predictable Scheduling Algorithms and Applications. Available: http://148.201.94.3:8991/F?func=direct&current_base=ITE01&doc_number=000177144

[10] R. G. V. Santos, "Enhanced ethernet switching technology for adaptive hard real-time applications: Tecnologia de comutação ethernet melhorada para aplicações adaptativas e críticas de tempo-real," Departamento de Eletrónica, Telecomunicações e Informática, Universidade de Aveiro, 2011.

[11] P. Pedreiras, "Modelos Computacionais, Apontamentos da Disciplina de Sistemas de Tempo-Real," ed: Universidade de Aveiro, 2011.

[12] J. W. S. W. Liu, *Real-time systems*: Prentice Hall PTR, 2000.

[13] J. D. Decotignie, "Ethernet-based real-time and industrial communications," *Proceedings of the IEEE,* vol. 93, pp. 1102-1117, 2005.

[14] R. Lagmann. Interbus Basics.

[15] E. Tovar and F. Vasques, "Real-time fieldbus communications using Profibus networks," *Industrial Electronics, IEEE Transactions on,* vol. 46, pp. 1241-1251, 1999.

[16] K. Tindell, H. Hansson, and A. J. Wellings, "Analysing real-time communications: controller area network (CAN)," 1994, pp. 259-263.

[17] J. R. Aschenbrenner, "Open systems interconnection," *IBM systems journal,* vol. 25, pp. 369-379, 1986.

[18] IEEE. *IEEE 802.3 Ethernet Working Group*. Available: http://www.ieee802.org/3/

[19] *Carrier Sense Multiple Access Collision Detect (CSMA/CD) Explained*. Available: http://learn-networking.com/network-design/carrier-sense-multiple-access-collision-detect-csmacd-explained

[20] P. Pedreiras, R. Leite, and L. Almeida, "Characterizing the real-time behavior of prioritized switched-ethernet," *2nd RTLIA,* 2003.

[21] P. Pedreiras and A. Luis, "The flexible time-triggered (ftt) paradigm: An approach to qos management in distributed real-time systems," 2003, p. 9 pp.

[22] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: a new resource reservation protocol," *IEEE Communications Magazine,* vol. 40, pp. 116-127, 2002.

[23] P. Ji, Z. Ge, J. Kurose, and D. Towsley, "A comparison of hard-state and soft-state signaling protocols," 2003, pp. 251-262.

[24] C. Topolcic, "Experimental internet stream protocol: Version 2 (ST-II)," 1990.

[25] L. Zhang, D. Estrin, S. Berson, S. Herzog, and S. Jamin, "Resource reservation protocol (RSVP)--Version 1 functional specification," *Resource,* 1997.

[26] P. White and J. Crowcroft, "A dynamic sender-initiated reservation protocol for the Internet," 1998, pp. 327-350.

[27] A. Eriksson and C. Gehrmann, "Robust and secure light-weight resource reservation for unicast IP traffic," 1998, pp. 168-170.

[28] P. Pan and H. Schulzrinne, "YESSIR: A simple reservation mechanism for the Internet," *ACM SIGCOMM Computer Communication Review,* vol. 29, pp. 89-101, 1999.

[29] G. Feher, K. Nemeth, M. Maliosz, I. Cselenyi, J. Bergkvist, D. Ahlard, and T. Engborg, "Boomerang-a simple protocol for resource reservation in ip networks," 1999.

[30] V. Firoiu and D. Towsley, "Call admission and resource reservation for multicast sessions," 1996, pp. 94-101.

[31] W. Almesberger, T. Ferrari, and J. Y. Le Boudec, "SRP: a scalable resource reservation protocol for the Internet," *Computer Communications,* vol. 21, pp. 1200-1211, 1998.

[32] *Internet Engineering Task Force (IETF)*. Available: www.ietf.org/

[33] C. Metz, "RSVP: general-purpose signaling for IP," *Internet Computing, IEEE,* vol. 3, pp. 95-99, 1999.

[34] D. Clark, "The design philosophy of the DARPA Internet protocols," *ACM SIGCOMM Computer Communication Review,* vol. 18, pp. 106-114, 1988.

[35] S. Shenker and L. Breslau, "Two issues in reservation establishment," 1995, pp. 14-26.

[36] P. P. White, "RSVP and integrated services in the Internet: A tutorial," *Communications Magazine, IEEE,* vol. 35, pp. 100-106, 1997.

[37] "IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 7:Multiple Registration Protocol," *IEEE Std 802.1ak - 2007 (Revision of IEEE Std 802.1Q-2005),* 2007.

[38] G. Gunther, "MSRP Attribute Declaration, Propagation and Bandwidth Allocation," in *IEEE 802.1 Interim - Seoul*, 2008.

[39] "IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams," *IEEE Std 802.1Qav - 2009 (Revision of IEEE Std 802.1Q-2005),* 2009.

[40] H. T. Lim, D. Herrscher, M. J. Waltl, and F. Chaari, "Performance analysis of the IEEE 802.1 ethernet audio/video bridging standard," in *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, 2012, pp. 27-36.

[41] "Media Acess Control (MAC) Bridges and Virtual Bridged Local Area Networks," *IEEE Std 802.1Qtm - 2011 (Revision of IEEE Std 802.1Q-2005),* 2011.

Anexo A: Valores de alguns parâmetros especificados no protocolo *MRP* e *MSRP*

Tabela 1: Estados referentes à *Applicant state machine [37]*

Estados da *Applicant state machine*	
VO - Very anxious Observer	O *Applicant* não está a declarar o atributo e não recebeu nenhuma mensagem *Join*In desde que a máquina de estados foi inicializada ou desde que recebeu uma mensagem *Leave* ou *LeaveAll*
VP – Very anxious Passive	O *Applicant* está a declarar o atributo mas não enviou nenhuma mensagem *Join* ou *JoinIn* desde que a máquina de estados foi inicializada ou desde que recebeu uma mensagem *Leave* ou *LeaveAll*
VN - Very anxious New	O *Applicant* está a declarar o atributo mas não enviou nenhuma mensagem desde que recebeu um *MAD.join_request* referente a uma nova declaração
AN - Anxious New	O *Applicant* está a declarar o atributo e enviou uma única mensagem *New* desde que recebeu um *MAD.join_request* referente a uma nova declaração
AA - Anxious Active	O *Applicant* está a declarar o atributo e enviou uma mensagem *Join* desde a última mensagem *Leave* ou *LeaveAll.* Não recebeu, contudo, outra *JoinIn* ou *In,* ou uma mensagem a especificar o estado *Empty* do *Registrar*
QA – Quiet Active	O *Applicant* está a declarar o atributo e enviou pelo menos uma das mensagens *Join* ou *New* requeridas desde a última mensagem *Leave* ou *LeaveAll,* e viu ou enviou a outra e não recebeu nenhuma mensagem a especificar e estado *Empty* do *Registrar.*
LA – Leaving Active	O *Applicant* enviou uma mensagem de *Join* ou *New* desde a última *Leave* ou *LeaveAll,* mas recebeu entretanto um *MAD.leave_request* e não enviou ainda uma mensagem *Leave*
AO – Anxious Observer	O *Applicant* não está a declarar o atributo mas recebeu um *JoinIn* desde o último *Leave* ou *LeaveAll*
QO – Quiet Observer	O *Applicant* não está a declarar o atributo mas recebeu dois *JoinIns* desde o último *Leave* ou *LeaveAll* e pelo menos um desde que recebeu uma mensagem a especificar e estado *Empty* do *Registrar.*
AP – Anxious Passive	O *Applicant* está a declarar o atributo mas não enviou nenhuma mensagem *Join* ou *New* desde o último *Leave* ou *LeaveAll* mas recebeu mensagens tal como o AO
QP – Quiet Passive	O *Applicant* está a declarar o atributo mas não enviou nenhuma mensagem *Join* ou *New* desde o último *Leave* ou *LeaveAll* mas recebeu mensagens tal como o QO
LO – Leaving Observer	O *Applicant* não está a declarar o atributo e recebeu uma mensagem de *Leave* ou *LeaveAll*

Tabela 2: Eventos protocolares passíveis de ocorrer no protocolo *MRP [37]*

Eventos protocolares	
Begin!	Inicializa a máquina de estados
New!	Nova declaração
Join!	Declaração sem sinalizar novo registo
Lv!	Remover a declaração
tx!	Oportunidade de transmissão sem a existência de um *LeaveAll*
txLA!	Oportunidade de transmissão com a existência de um *LeaveAll*
txLAF!	Oportunidade de transmissão com a existência de um *LeaveAll* mas sem espaço
rNew!	Recebida mensagem *New*
rJoinIn!	Recebida mensagem *JoinIn*
rIn!	Recebida mensagem *In*
rJoinMt!	Recebida mensagem *JoinEmpty*
rMt!	Recebida mensagem *Empty*
rLv!	Recebida mensagem *Leave*
rLA!	Recebida mensagem *LeaveAll*
Flush!	Alteração na topologia da rede
Re-Declare!	Alteração na topologia da rede
periodic!	Evento periódico o ocorre
leavetimer!	*Leavetimer* expirou
leavealltimer!	*Leavealltimer* expirou
periodictimer!	*Periodictimer* expirou

Tabela 3: Identificação das ações protocolares especificados pelo MRP [36]

Ações protocolares	
New	Envia indicação de *New* ao módulo MAP e aplicação MRP
Join	Envia indicação de *Join* ao módulo MAP e aplicação MRP
Lv	Envia indicação de *Lv* ao módulo MAP e aplicação MRP
sN	Envio de mensagem *New*
sJ	Envio de mensagem *JoinIn* e *JoinMT*
sL	Envio de mensagem Lv
s	Envio de mensagem *Empty*
[s]	Envio de mensagem *In* ou *Empty,* se requerido otimização da codificação do PDU(*Protocol Data Unit)*
[sL]	Envio de mensagem *Lv,* se requerido otimização da codificação do PDU
[sJ]	Envio de mensagem *Join,* se requerido otimização da codificação do PDU
sLA	Envio de mensagem *Leave All*
periodic	Transmissão de evento periódico
leavetimer	*Leave period timer*
leavealltimer	*Leave All period timer*
periodictimer	Periodic transmission timer
-x-	Nenhum ação ou transição de estado ocorre

Tabela 4: *Applicant State machine*: estados, eventos e ações [37]

EVENT	STATE											
	VO[11]	**VP[6]**	**VN[6]**	**AN[6]**	**AA[6]**	**QA**	**LA[6]**	**AO[3,11]**	**QO[3,11]**	**AP[3,6]**	**QP[3]**	**LO[6]**
Begin!	—	VO	VO	VO	VO	VO	VO	VO	VO	VO	VO	VO
New!	VN	VN	—	—	VN	VN	VN	VN	VN	VN	VN	VN
Join!	VP	—	—	—	—	—	AA	AP	QP	—	—	VP
Lv!	—	VO	LA	LA	LA	LA	—	—	—	AO	QO	—
rNew!	—	—	—	—	—	—	—	—	—	—	—	—
rJoinIn!	AO[4]	AP[4]	—	—	QA	—	—	QO	—	QP	—	—
rIn!	—	—	—	—	QA[5]	—	—	—	—	—	—	—
rJoinMt! || rMt!	—	—	—	—	—	AA	—	—	AO	—	AP	VO
rLv! || rLA! || Re-declare!	LO[1]	—	—	VN	VP[9]	VP[9]	—[10]	LO[1]	LO[1]	VP	VP	—
periodic!	—	—	—	—	—	AA	—	—	—	—	AP	—
tx![7]	[s] —	sJ AA	sN AN	sN QA[8]	sJ QA	[sJ] —	sL VO	[s] —	[s] —	sJ QA	[s] —	s VO
txLA![2]	[s] LO	s AA	sN AN	sN QA	sJ QA	sJ —	[s] LO	[s] LO	[s] LO	sJ QA	sJ QA	[s] —
txLAF![2]	LO	VP	VN	VN	VP	VP	LO	LO	LO	VP	VP	—

Notes to the table:
[1]Applicant-Only participants exclude the LO state, and transition to VO.
[2]These events do not occur for Applicant-Only participants.
[3]Point-to-point subset participants exclude the AO, QO, AP, and QP states.
[4]Ignored (no transition) if point-to-point subset or if operPointToPointMAC is TRUE.
[5]Ignored (no transition) if operPointToPointMAC is FALSE.
[6]Request opportunity to transmit on entry to VN, AN, AA, LA, VP, AP, and LO states.
[7]If the MRPDU is full and cannot convey a required message there is no change of state and an additional transmit opportunity is requested if that has not been done already.
[8]QA if the Registrar is IN, and AA otherwise.
MRP design notes:
[5]On shared media the receipt of In does not confirm registration by all Participants, and the In could have been sent by an Applicant-Only participant.
[8]Since New messages do not convey registrar state, a Leave could have been received without an Empty or JoinEmpty prompt being sent, the transition to AA guards against loss of that Leave by another Applicant.
[9]The design accepts a small possibility of a continued registration (after rLv! if a Lv! occurs before a further Join is sent) in return for not accumulating many Active participants when Join!s and Lv!s are frequent. rLv! processing is deliberately not optimized for point-to-point.
[10]If a Leave has been received, the Registrar for the transmitting participant is very probably IN, as this Applicant has not yet sent a Leave, so the pending Leave is required. The small savings from avoiding transmission of Leaves pending on receipt of LeaveAlls does not merit distinguishing the rLv! and rLA! cases.
[11]The VO, AO, and QO states represent states where the attribute is neither being declared by the Participant nor being registered by any other station on the LAN. In implementations where dynamic creation and discarding of state machines is desirable, the state machine can be discarded when in any of these states, pending a future requirement to declare or register that attribute value.

Tabela 5: *Registrar State machine*: estados, eventos e ações [37]

		STATE		
		IN	**LV**	**MT**
EVENT	**Begin!**	MT	MT	MT
	rNew!	New IN	New Stop leavetimer IN	New IN
	rJoinIn! || rJoinMt!	IN	Stop leavetimer IN	Join IN
	rLv! || rLA! || txLA! || Re-declare!	Start leavetimer LV	-x-	-x-
	Flush!	MT	Lv MT	MT
	leavetimer!	-x-	Lv MT	MT

Tabela 6: Valores padrão dos timers protocolares MRP [37]

Parameter	Value (centiseconds)
JoinTime	20
LeaveTime	60–100
LeaveAllTime	1000

Tabela 7: *LeaveAll* state *machine*: estados, eventos e ações [37]

		STATE	
		Active	**Passive**
EVENT	**Begin!**	Start leavealltimer Passive	Start leavealltimer Passive
	tx!	sLA Passive	-x-
	rLA!	Start leavealltimer Passive	Start leavealltimer Passive
	leavealltimer!	Start leavealltimer Active	Start leavealltimer Active

Tabela 8: *Period Transmission state machine*: estados, eventos e ações [37]

		STATE	
		Active	**Passive**
EVENT	**Begin!**	Start periodictimer Active	Start periodictimer Active
	periodicEnabled!	-x-	Start periodictimer Active
	periodicDisabled!	Passive	-x-
	periodictimer!	Start periodictimer periodic Active	-x-

Tabela 9: grupo MAC associado a cada aplicação MRP [37]

Aplicação	Valor
Endereço MMRP	01-80-C2-00-00-20
Endereço MVRP	01-80-C2-00-00-21
Endereço MSRP	01-80-C2-00-00-22
Reservado	01-80-C2-00-00-22….2F

Tabela 10: *Ethertype* associado a cada aplicação MRP [37]

Aplicação	Valor
EtherType MMRP	88-F6
EtherType MVRP	88-F5
EtherType MSRP	22-EA

Tabela 11: Valores dos diferentes *Attribute Type* [37]

Attribute Type	Valor
Talker Advertise	1
Talker Failed	2
Listener	3
Domain	4

Tabela 12: Valores dos diferentes *Attribute Lenght* [37]

Attribute Lenght	Valor
Talker Advertise	25 (0x19)
Talker Failed	34 (0x22)
Listener	8
Domain	4

Tabela 13: Valores associados ao diferentes *Attribut Event* codificados nos *ThreePackedEvents* [37]

AttributeEvent	Valor
New	0
JoinIn	1
In	2
JoinMt	3
Mt	4
Lv	5

Tabela 14: Valores associados ao diferentes *FourPackedType* codificados nos *FourPackedEvents* [37]

FourPackedType	Valor
Ignore	0
Asking Failed	1
Ready	2
Ready Failed	3

Tabela 15: Descrição dos códigos de falha associados ao atributo *Talker Failed* [37]

Failure code	Descrição
1	Largura de banda insuficiente
2	Recursos insuficientes no *switch /bridge*
3	Largura de banda insuficiente para a classe de tráfego
4	*StreamID* em uso por outro *Talker*
5	Endereço destino da *stream* em uso
6	*Stream* interrompida por outra de maior *rank*
7	Alteração no valor da latência anteriormente reportado
8	Porta não suporta transmissão
9	Endereço destino inválido
10	Recursos MSRP esgotados
11	Recursos MMRP esgotados
12	Impossível guardar endereço destino
13	Prioridade requisitada não faz parte de uma classe de tráfego SR
14	*MaxFrameSize* demasiado grande
15	*Valor msrpMaxFanInPorts* atingido
16	Alterações no valor do *FirstValue* para uma *stream* registada
17	*VLAN* bloqueada
18	*VLAN* tagging desativado
19	Classe de prioridade SR incompatível

Anexos B: Resultados obtidos para validação de uma mensagem *Talker Failed*

Valores mínimos

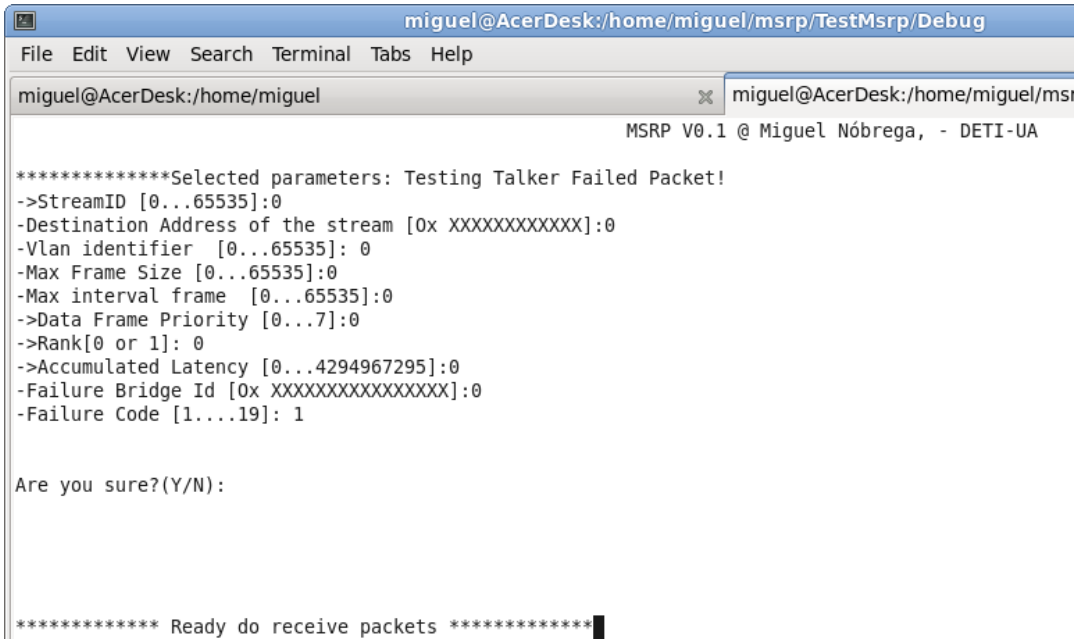

Figura 1: Introdução na consola dos valores a codificar num *Talker Failed*: valores mínimos

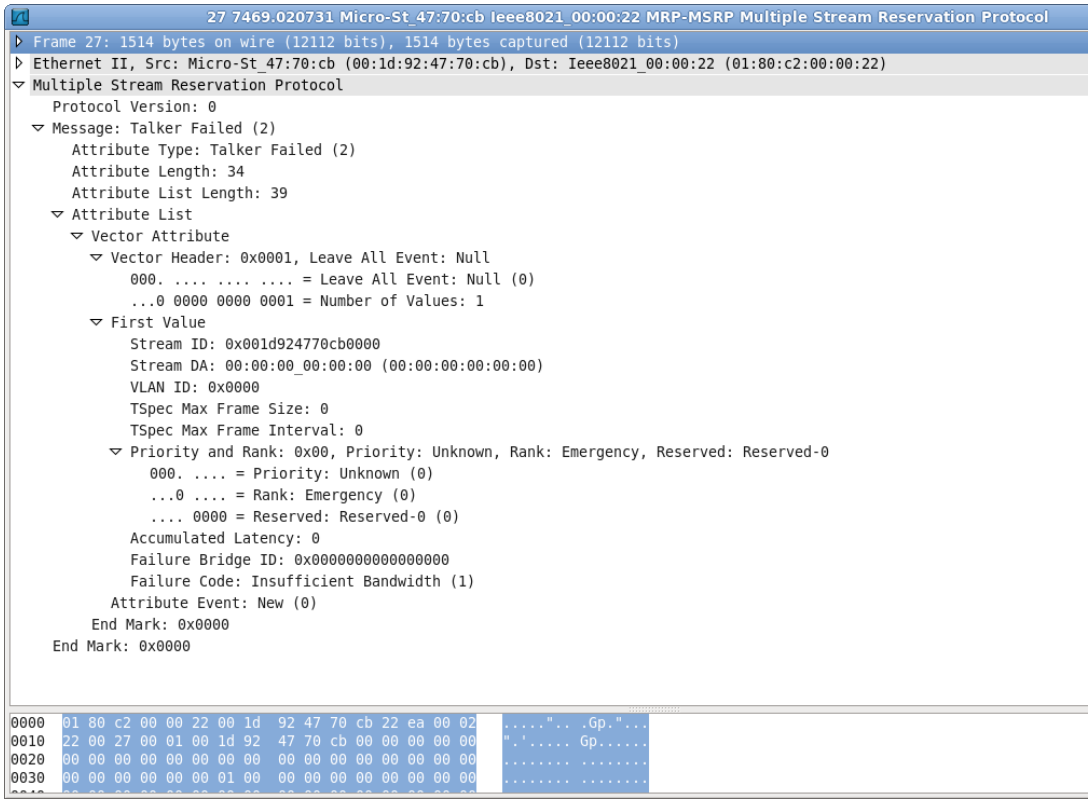

Figura 2: Confirmação do *MSRPDU* codificado usando *wireshark:* valores mínimos

```
************* Ready do receive packets *************
Received packet from: 1D924770CB,
AttributeType encoded: 0X2 -> Talker Failed;
Decoding Talker Failed Packet.........

Protocol Version: 0
Attribute Type: 2
Attribute Lenght 22
Attribute List Lenght: 27
Leave All Event: 0
Number of values/events:  1
Talker MAC Address: 1D924770CB
Stream Unique ID: 0
Stream Destination Address: 0
Vlan ID: 0
TSpec Bandwith: 0
TSpec Frame Rate: 0
Traffic class: 0
Rank: 0
Reserved: 0
Acummulated Latency: 0
Failure Information Bridge ID: 0
Failure Information Failure Code: 1
Three Packed Event Byte: 0
EndMark1: 0
EndMark2: 0
```

Figura 3: Receção, descodificação e impressão na consola do *MSRPDU* recebido pelo *Listener* referente ao T*alker Advertise* ilustrado na Figura 1

Valores aleatórios (1)

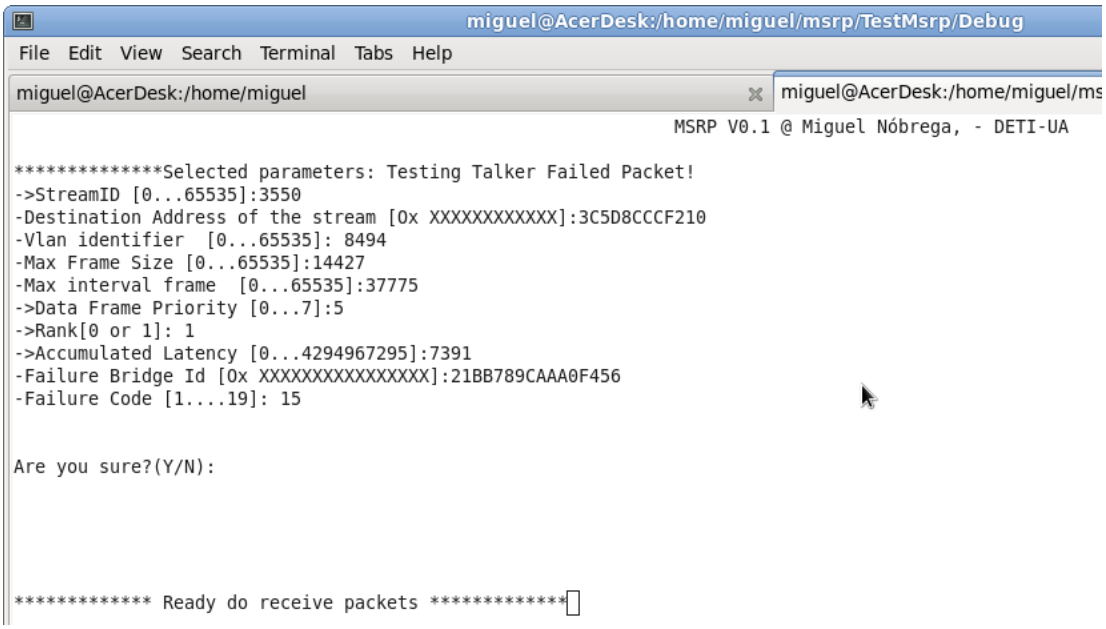

Figura 4: Introdução na consola dos valores a codificar num *Talker Failed* : valores aleatórios (1)

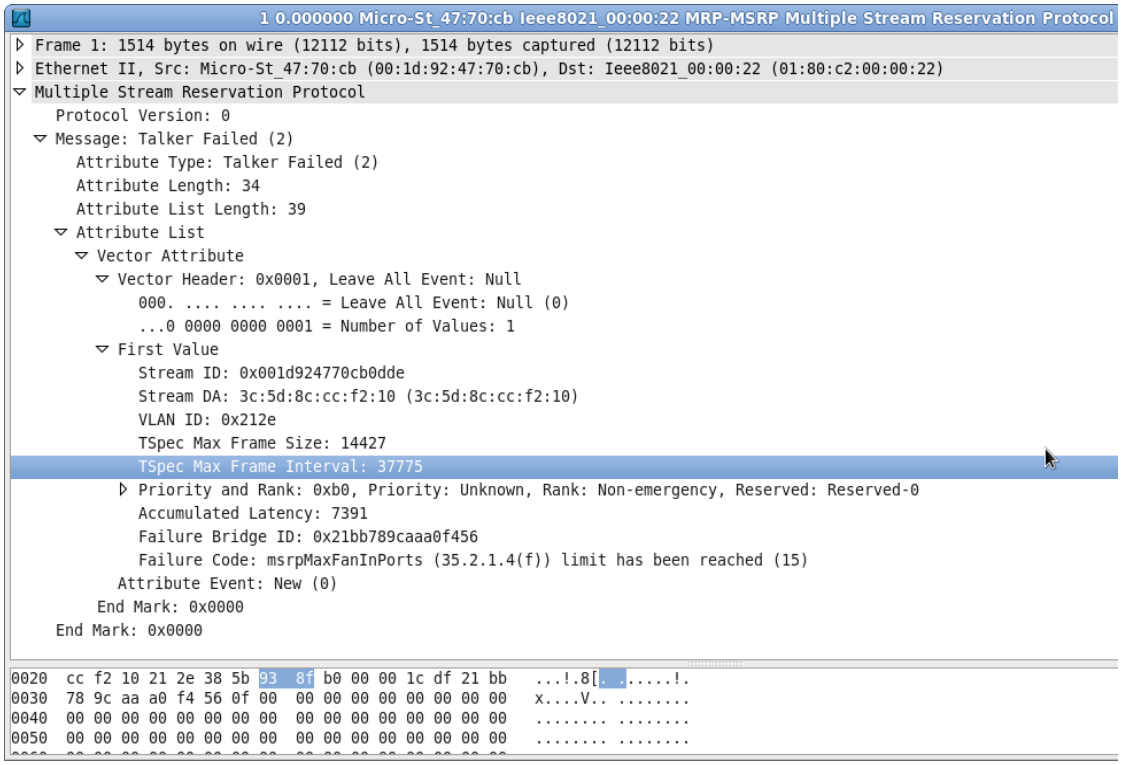

Figura 5: Confirmação do *MSRPDU* codificado usando *wireshark:* valores aleatórios (1)

```
************* Ready do receive packets *************
Received packet from: 1D924770CB,
AttributeType encoded: 0X2 -> Talker Failed;
Decoding Talker Failed Packet..........

Protocol Version: 0
Attribute Type: 2
Attribute Lenght 22
Attribute List Lenght: 27
Leave All Event: 0
Number of values/events:  1
Talker MAC Address: 1D924770CB
Stream Unique ID: DDE
Stream Destination Address: 3C5D8CCCF210
Vlan ID: 212E
TSpec Bandwith: 14427
TSpec Frame Rate: 37775
Traffic class: 5
Rank: 1
Reserved: 0
Acummulated Latency: 7391
Failure Information Bridge ID: 21BB789CAAA0F456
Failure Information Failure Code: 15
Three Packed Event Byte: 0
EndMark1: 0
EndMark2: 0
```

Figura 6: Receção, descodificação e impressão na consola do *MSRPDU* recebido pelo *Listener* referente ao *Talker Failed* ilustrado na Figura 4

Valores aleatórios (2)

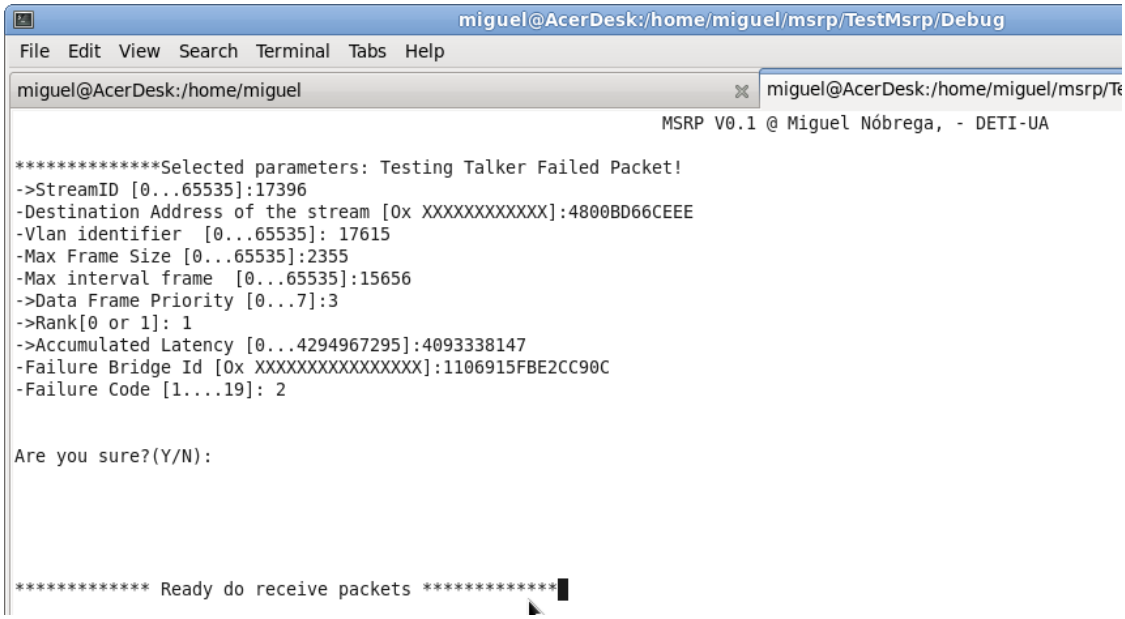

Figura 7: Introdução na consola dos valores a codificar num *Talker Failed*: valores aleatórios(2)

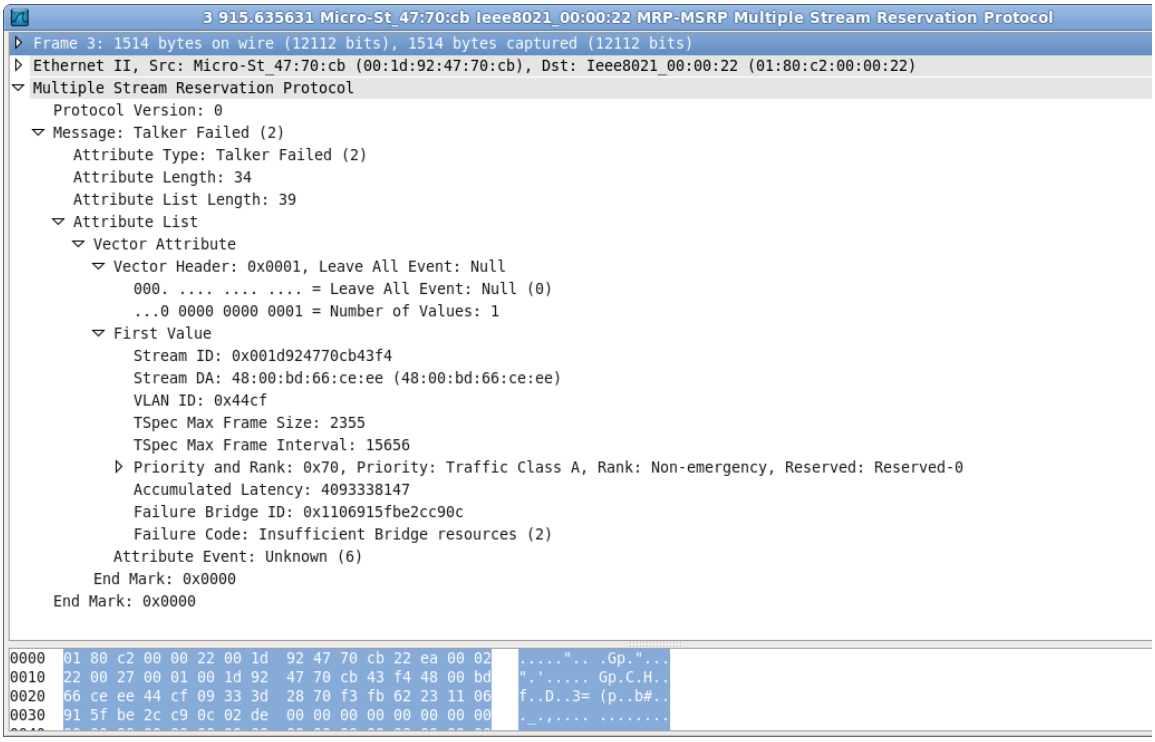

Figura 8: Confirmação do MSRPDU codificado usando *wireshark:* valores aleatórios (2)

```
************* Ready do receive packets *************
Received packet from: 1D924770CB,
AttributeType encoded: 0X2 -> Talker Failed;
Decoding Talker Failed Packet..........

Protocol Version: 0
Attribute Type: 2
Attribute Lenght 22
Attribute List Lenght: 27
Leave All Event: 0
Number of values/events:  1
Talker MAC Address: 1D924770CB
Stream Unique ID: 43F4
Stream Destination Address: 4800BD66CEEE
Vlan ID: 44CF
TSpec Bandwith: 2355
TSpec Frame Rate: 15656
Traffic class: 3
Rank: 1
Reserved: 0
Acummulated Latency: 4093338147
Failure Information Bridge ID: 1106915FBE2CC90C
Failure Information Failure Code: 2
Three Packed Event Byte: 0
EndMark1: 0
EndMark2: 0
```

Figura 9: Receção, descodificação e impressão na consola do *MSRPDU* recebido pelo *Listener* referente ao *Talker Failed* ilustrado na Figura 7

Valores máximos (2)

```
miguel@AcerDesk:/home/miguel/msrp/TestMsrp/Debug
File  Edit  View  Search  Terminal  Tabs  Help
miguel@AcerDesk:/home/miguel                    miguel@AcerDesk:/home/miguel/m
                                                MSRP V0.1 @ Miguel Nóbrega, - DETI-UA

**************Selected parameters: Testing Talker Failed Packet!
->StreamID [0...65535]:65535
-Destination Address of the stream [Ox XXXXXXXXXXXX]:FFFFFFFFFFFF
-Vlan identifier  [0...65535]: 65535
-Max Frame Size [0...65535]:65535
-Max interval frame  [0...65535]:65535
->Data Frame Priority [0...7]:7
->Rank[0 or 1]: 1
->Accumulated Latency [0...4294967295]:4294967295
-Failure Bridge Id [Ox XXXXXXXXXXXXXXXX]:FFFFFFFFFFFFFFFF
-Failure Code [1....19]: 19


Are you sure?(Y/N):




************* Ready do receive packets *************
```

Figura 10: Introdução na consola dos valores a codificar num *Talker Failed*: valores máximos

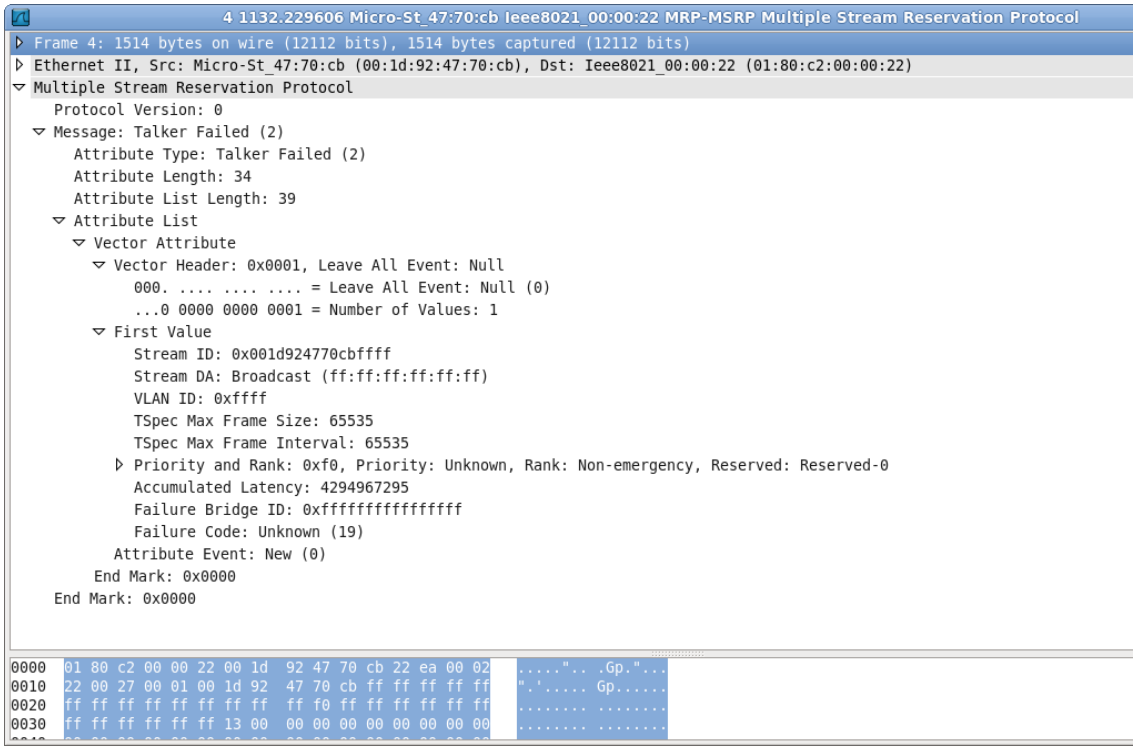

Figura 11: Confirmação do *MSRPDU* codificado usando *wireshark:* valores máximos

```
************* Ready do receive packets *************
Received packet from: 1D924770CB,
AttributeType encoded: 0X2 -> Talker Failed;
Decoding Talker Failed Packet..........

Protocol Version: 0
Attribute Type: 2
Attribute Lenght 22
Attribute List Lenght: 27
Leave All Event: 0
Number of values/events:  1
Talker MAC Address: 1D924770CB
Stream Unique ID: FFFF
Stream Destination Address: FFFFFFFFFFFF
Vlan ID: FFFF
TSpec Bandwith: 65535
TSpec Frame Rate: 65535
Traffic class: 7
Rank: 1
Reserved: 0
Acummulated Latency: 4294967295
Failure Information Bridge ID: FFFFFFFFFFFFFFFF
Failure Information Failure Code: 19
Three Packed Event Byte: 0
EndMark1: 0
EndMark2: 0
```

Figura 12: Receção, descodificação e impressão na consola do *MSRPDU* recebido pelo *Listener* referente ao *Talker Advertise* ilustrado na Figura 10

Anexo C: Resultados obtidos para o caso de uma sinalização de reserva mal sucedida

Os passos efetuados para realização dos testes para o caso de uma reserva mal sucedida bem como os dados recolhidos são descritos de seguida:

1. Pedido de início de sinalização de reserva de recursos e introdução dos parâmetros referentes ao pedido de reserva de recursos (Figura 13). De notar que os parâmetros referentes ao *Tspec,* de acordo com a implementação feita, deverão resultar num pedido mal sucedido.

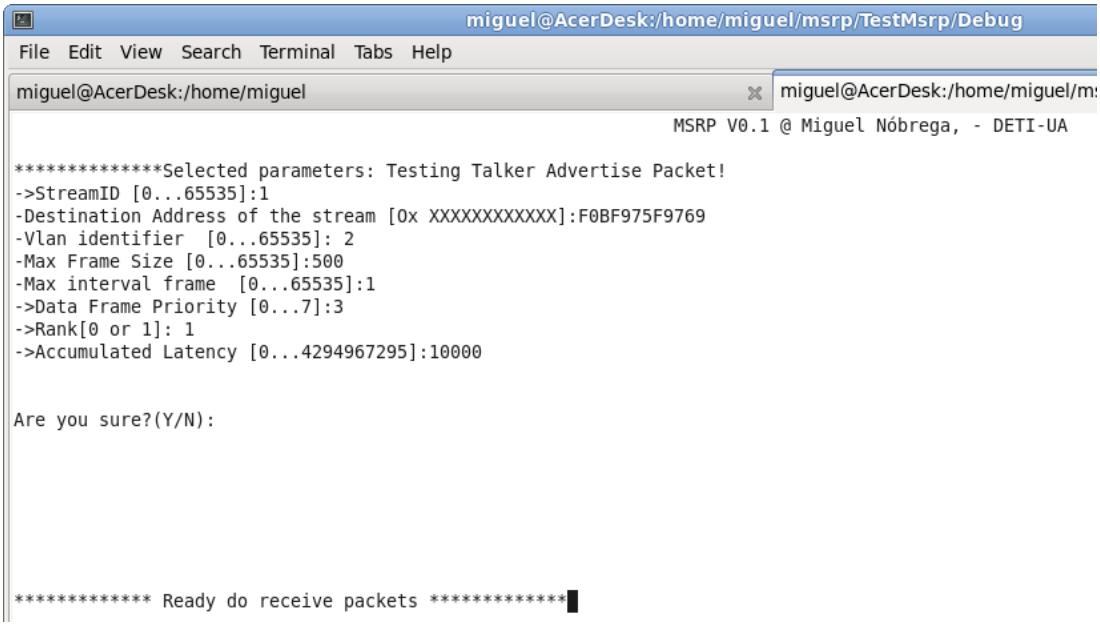

Figura 13: Parâmetros descritivos de um pedido de reserva de recursos. Este pedido é iniciado com o envio de um *Talker Advertise* (caso de um pedido mal sucedido)

2. A mensagem *Talker Advertise* ao ser recebida pelo *Listener* (Figura 14) é corretamente descodificada e o seu conteúdo é analisado. Dado que o endereço destino da *stream* coincide com o endereço *MAC* do *Listener* e este **não** tem recursos suficientes para suportar o pedido de reserva, este responde com o envio de um *Listener Asking Failed* em direção ao remetente do pedido. Tal pode ser comprovado nas Figuras 15, 16 e 17.

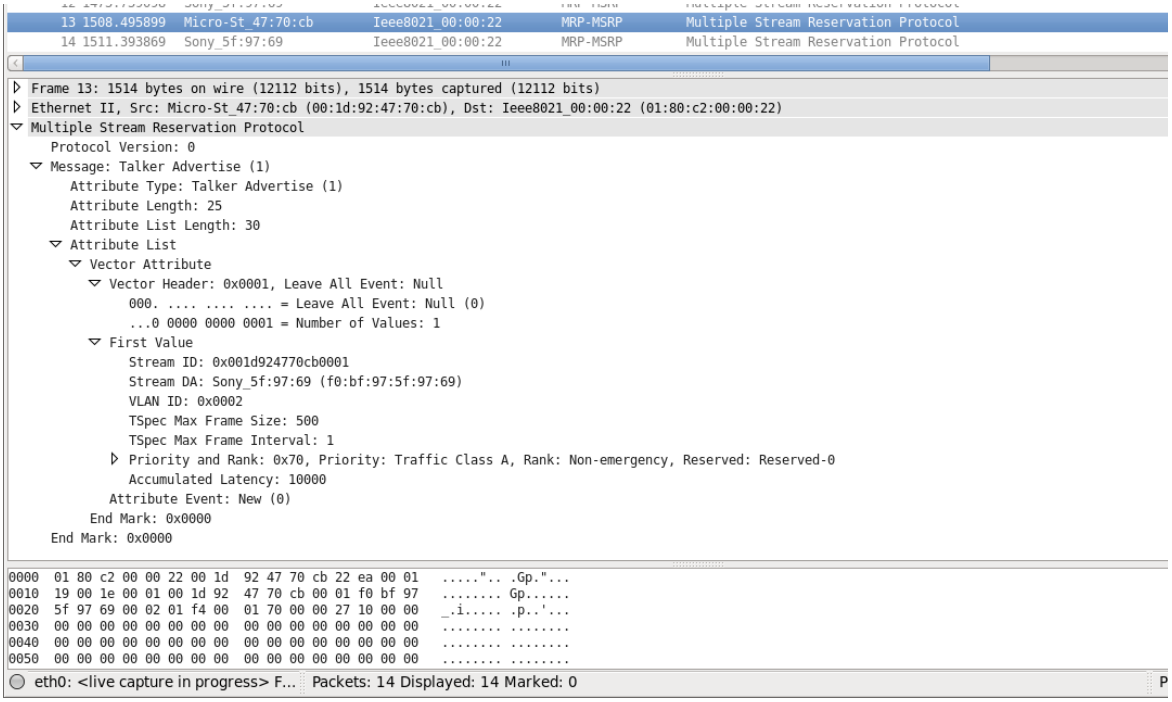

Figura 14: Confirmação do *MSRPDU* enviado pelo *Talker* com respetivos parâmetros descritivos do pedido de reservas (caso de um pedido mal sucedido)

```
************ Ready do receive packets ************
Received packet from: 1D924770CB,
AttributeType encoded: 0X1 ->   Talker Advertise
Decoding Talker Advertise Packet..........

Protocol Version: 0
Attribute Type: 1
Attribute Lenght 19
Attribute List Lenght: 1E
Leave All Event: 0
Number of values/events:  1
Talker MAC Address: 1D924770CB
Stream Unique ID: 1
Stream Destination Address: F0BF975F9769
Vlan ID: 2
TSpec Bandwith: 500
TSpec Frame Rate: 1
Traffic class: 3
Rank: 1
Reserved: 0
Acummulated Latency: 10000
Three Packed Event Byte: 0
EndMark1: 0
EndMark2: 0
The stream is destined to this Listener!
Bandwidth NOT available
Answering with a Listener Asking Failed MSRPDU!
Releasing memory.....
```

Figura 15: Reação do *Listener* ao *Talker Advertise* recebido (caso de pedido mal sucedido)

```
************* Ready do receive packets *************
Received packet from: F0BF975F9769,
AttributeType encoded: 0X3 -> Listener;
Decoding Listener Packet..........
Received Listener Asking Failed.

Protocol Version: 0
Attribute Type: 3
Attribute Lenght 8
Attribute List Lenght: E
Leave All Event: 0
Number of values/events:  1
Stream ID TALKER MAC: 1D924770CB
Stream ID Unique ID: 1
Three Packed Event Byte: 0
Four packed Event Byte: 40
EndMark1: 0
EndMark2: 0
Signalization completed!
Releasing memory.....
```

Figura 16: *Listener Asking Failed* recebido pelo *Talker,* sinalizando um pedido de reserva mal sucedido

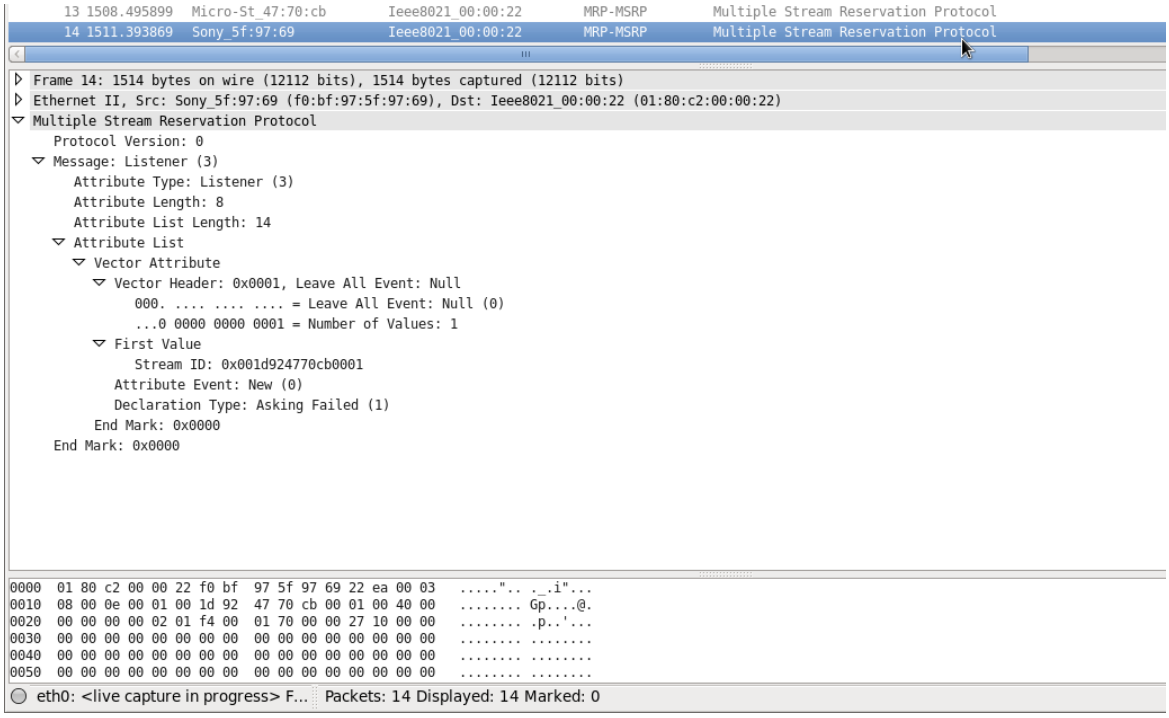

Figura 17: Confirmação usando *wireshark* da receção por parte do *Talker* de um *Listener Asking Failed,* sinalizando assim um pedido de reserva mal sucedido