Controle de Admissão para QoS em Sistemas Distribuídos Híbridos, Tolerantes a Falhas

Sérgio Gorender, Raimundo José de Araújo Macêdo, Waltemir Lemos Pacheco Júnior

[1]Laboratório de Sistemas Distribuídos (LaSiD)
Departamento de Ciência da Computação
Universidade Federal da Bahia
Campus de Ondina - Salvador - BA - Brasil

{gorender,macedo}@ufba.br, brwaltemir@gmail.com

Abstract. *Hybrid distributed systems have synchronous and asynchronous processes and communication channels. Depending on the amount of synchronous components in this system, it is possible to solve classical problems of distributed systems, such as consensus, with a higher level of fault tolerance. Hybrid models for fault tolerant distributed systems have been presented with these features. Synchronous communication channels can be obtaines through the use of QoS Architectures. These architectures, while based on different mechanisms, usually show some kind of service that provides a communication isochronous service (synchronous channel). Admission control mechanisms are fundamental for providing isochronous services for new communication channels. Using this mechanism it is possible to ensure that there is no overloading of the network resources reserved for the isochronous classe of service. In this paper we present an admission control module for the QoS Provider, which is a mechanism for managing QoS and is used by models for hybrid distributed systems such as HA and Spa.*

Resumo. *Sistemas distribuídos híbridos são compostos por processos e canais de comunicação que podem ser síncronos ou assíncronos. Dependendo da quantidade de componentes síncronos presente no sistema, é possível resolver problemas clássicos dos sistemas distribuídos, como o consenso, com um maior nível de tolerância a falhas. Modelos para sistemas distribuídos híbridos, tolerantes a falhas têm sido apresentados com estas características. Uma das formas de se obter canais de comunicação síncronos é através do uso de arquiteturas para prover QoS. Estas arquiteturas, embora baseadas em mecanismos diferentes, em geral apresentam alguma classe de serviço que fornece um serviço de comunicação isócrono (síncrono). Para que estes serviços isócronos sejam possíveis, é fundamental o uso de um mecanismo de controle de admissão para novos canais de comunicação, para garantir não haver sobrecarga dos recursos de rede utilizados para prover o serviço. Apresentamos neste artigo um módulo de controle de admissão para o QoS Provider, o qual é um mecanismo para gerenciamento de QoS sendo utilizado por modelos para sistemas distribuídos híbridos como o HA e o Spa.*

Palavras-chave: QoS, Controle de Admissão, modelos para sistemas distribuídos híbridos, tolerância a falhas, detecção de defeitos.

1. Introdução

Sistemas distribuídos híbridos são compostos por componentes (processos e canais de comunicação) que podem apresentar um comportamento síncrono ou assíncrono. Não são portanto sistemas síncronos, mas possuem componentes síncronos em execução. Nestes sistemas é possível executar protocolos distribuídos, com o consenso, tolerando falhas, proporcionalmente ao número de componentes síncronos existentes no sistema, sendo o próprio consenso um importante bloco de construção para a criação de sistemas distribuídos tolerantes a falhas.

Nos sistemas híbridos é possível tirar proveito do nível de sincronismo existente para resolver problemas não solucionados nos sistemas assíncronos, tolerando falhas. Os modelos HA [Gorender et al. 2007] e Spa [Macêdo and Gorender 2009] para sistemas distribuídos híbridos apresentam propriedades e características utilizadas na solução destes problemas.

Ambientes de execução híbridos se tornam comuns nos dias atuais, e existem diversas formas de se implementar processos e canais de comunicação síncronos e assíncronos. É possível executar ações dos processos com limites de tempo garantidos através do uso de sistemas operacionais de tempo real, como por exemplo, o Xenomai ou o RTLinux. Também podemos obter a execução síncrona de processos utilizando computadores dedicados, dimensionados para executar os processos com limites de tempo garantidos. Também é possível obter canais de comunicação síncronos com o uso de redes de controle dedicadas, dimensionadas para a comunicação a ser efetuada. Assim como podemos obter um serviço de comunicação síncrono com o uso de Qualidade de Serviço (QoS).

Qualidade de Serviço diz respeito à possibilidade de se reservar recursos de rede (largura de banda e memória nos roteadores) para alguns fluxos de comunicação (canais de comunicação) e de se priorizar estes fluxos, no encaminhamento das mensagens nos roteadores. A reserva e priorização podem ser efetuadas por fluxos de comunicação ou por agrupamentos de fluxos de comunicação (classes). A arquitetura DiffServ, desenvolvida pelo IETF, provê reserva de recursos e prioridade para classes de encaminhamento de pacotes, sendo que os fluxos de comunicação são atribuídos a estas diferentes classes. Estas classes são configuradas nos roteadores, os quais passam a prover o serviço especificado.

Para garantir o fornecimento de um serviço síncrono, é necessário que a quantidade de fluxos de comunicação alocados à classe de serviço não gere uma sobrecarga nos recursos reservados para esta classe, garantindo que, no pior caso, todos os pacotes recebidos pelos roteadores e atribuídos a esta classe serão encaminhados, não havendo perdas de pacotes. Para tal, torna-se necessário a utilização de um mecanismo de Controle de Admissão, o qual irá verificar a disponibilidade de recursos nos roteadores, e só admitirá um novo fluxo de comunicação para uma classe de serviço, se esta classe ainda tiver recursos disponíveis em quantidade suficiente para tal.

Neste artigo apresentamos a implementação de um mecanismo de Controle de Admissão para o QoS Provider [Gorender et al. 2004], o qual é um mecanismo para a criação de canais de comunicação e gerenciamento de informações sobre os serviços de comunicação fornecidos aos canais criados. O QoS Provider foi desenvolvido para prover

informações a sistemas distribuídos híbridos, sobre o estado dos componentes do sistema, entre síncronos e assíncronos (timely e untimely), assim como, permitir a comunicação destes sistemas com mecanismo do ambiente de execução, como arquiteturas para prover QoS e sistemas operacionais de tempo real. Um protótipo simplificado deste mecanismo foi apresentado em [Gorender et al. 2004], porém sem um módulo de controle de admissão.

Estes canais de comunicação síncronos, fornecidos a partir de um controle de admissão, são utilizados por detectores de defeitos para a execução de detecções perfeitas. Desta forma é possivel implementar, nestes ambientes híbridos, detectores de defeitos híbridos, que realizam detecções não confiáveis (suspeitas), e detecções confiáveis, assim como um detector de defeitos perfeito, dependendo da quantidade e organização dos componentes híbridos do sistema [Macêdo and Gorender 2009].

Este artigo está organizado da seguinte forma: a seção a seguir apresenta algumas características dos modelos para sistemas distribuídos híbridos, a seção 3 apresenta os conceitos básicos sobre Qualidade de Serviço, e a relevância e estratégias adotadas para módulos de Controle de Admissão, a seção seguinte, 4, apresenta a estrutura geral do mecanismo de controle de admissão desenvolvido para o QoS Provider, a seção 5 apresenta diversos aspectos da implementação do mecanismos de controle de admissão, a seção 6 mostra os resultados de testes realizados com canais de comunicação com e sem QoS, admitidos com o uso do controle de admissão, e a seção 7 apresenta conclusões a este trabalho.

2. Sistemas Distribuídos Híbridos

Modelos para sistemas distribuídos são definidos a partir de suas propriedades e características. O modelo síncrono apresenta restrições temporais para a execução de ações dos processos e para a transferência de mensagens entre estes processos, além de relógios com desvios limitados. O modelo assíncrono não apresenta restrições temporais. Diversos modelos ditos parcialmente síncronos têm sido propostos, caracterizados por inserir algum nível de sincronismo ao sistema assíncrono. Os sistemas híbridos possuem componentes (processos e canais de comunicação) síncronos e assíncronos.

Um modelo para sistemas distribuídos híbrido é composto por processos e canais de comunicação que podem ser síncronos ou assíncronos. O modelo HA considera que todos os processos são síncronos, mas os canais de comunicação podem ser síncronos ou assíncronos [Gorender et al. 2007]. Também assume que os canais de comunicação podem alterar seu estado entre síncrono e assíncrono. Para este modelo foi desenvolvido um detector de defeitos também híbrido, que realiza suspeitas e notificações de processos, e que se adapta a alterações no estado dos canais de comunicação, entre síncrono e assíncrono. Já o modelo Spa assume que tanto processos quanto canais de comunicação podem ser síncronos e assíncronos, mas que este estado não se modifica [Macêdo and Gorender 2009]. Neste modelo, os processos síncronos interligados por canais de comunicação síncronos são agrupados em partições síncronas. Nestas partições podemos realizar detecção de defeitos perfeita. Se todos os processos são síncronos e pertencem a alguma partição síncrona, implementamos no sistema um detetor de defeitos perfeito (P).

Para obter estes modelos, necessitamos de mecanismos que permitam obter

canais de comunicação com características isócronas (síncrono), assim como processos que executem tarefas síncronas. Para obter os canais de comunicação utilizamos uma arquitetura para prover Qualidade de Serviços em redes de computadores. Para obter processos síncronos utilizamos um sistema operacional de tempo real.

3. QoS e Controle de Admissão

Qualidade de Serviço (QoS), de uma forma geral, diz respeito ao modo como um serviço (execução de tarefas ou comunicação, por exemplo) pode ser oferecido a uma aplicação com alta eficiência ou, simplesmente, sem que haja perdas em seu desempenho. O serviço é especificado por um conjunto de requisitos. Estes requisitos podem ser qualificados, quantificados e utilizados como parâmetros para caracterizar o tipo de qualidade de serviço oferecido. Atualmente, a qualidade de serviço pode ser aplicada a vários níveis arquiteturais, tais como a nível de sistemas operacionais, subsistemas de comunicação e na comunicação de dados [Aurrecoechea et al. 1998]. O conceito de QoS é amplamente utilizado na área de redes para referenciar a qualidade de serviço aplicada a um determinado serviço ou fluxo de dados (WANG, 2001). Em uma rede com QoS, alguns fluxos de comunicação podem ser privilegiados, ou seja, parte dos recursos da rede podem ser reservados para atender às necessidades especiais destes em detrimento de outros. Estes fluxos de comunicação poderão obter reserva de largura de banda e *buffer* nos roteadores, assim como maior prioridade para o encaminhamento de seus pacotes de dados. Existem diversas arquiteturas desenvolvidas para prover QoS a redes de comunicação, tais como: Quartz [Siqueira and Cahill 2000], Omega [Nahrstedt and Smith 1995], QoS-A [Campbell et al. 1994], IntServ [Braden et al. 1994] e DiffServ [Blake et al. 1998]. As arquiteturas IntServ e DiffServ foram padronizadas pelo IETF (*Internet Engeneering Task Force*)

Estas arquiteturas gerenciam os recursos das redes, e fornecem serviços com qualidade para fluxos de comunicação. Para que um melhor serviço de comunicação seja provido a um novo fluxo de comunicação é necessário que a arquitetura verifique a disponibilidade de recursos na rede, e realize uma reserva de recursos, em quantidade suficiente, para prover o serviço requisitado. A verificação da disponibilidade dos recursos é feita por um mecanismo de controle de admissão. Este mecanismo verifica a disponibilidade de recursos na rede, em toda a rota a ser percorrida pelo fluxo de comunicação, e só admite o novo fluxo, com o compromisso de prover a Qualidade de Serviço solicitada, se existirem recursos suficientes na rede. No caso da admissão do fluxo, os recursos são reservados, não estando mais disponíveis para a admissão de um outro fluxo de comunicação.

3.1. Controle de Admissão em um Domínio DIFFSERV

Um domínio de rede que provê QoS utilizando a arquitetura DiffServ é chamado de Domínio DiffServ. Esta arquitetura provê QoS classificando os diversos fluxos de comunicação em diferentes níveis de serviço. O IETF padronizou três classes de serviço para o DiffServ: Serviço Melhor Esforço (padrão da Internet), Serviço Assegurado (provê níveis diferentes de prioridade, e de probabilidade em não haver perdas de pacotes) e o Serviço Expresso (garante não haver perdas de pacotes, e um atraso limitado na transferência destes pacotes). Enquanto o Serviço Melhor Esforço se caracteriza por uma comunicação não isócrona (assíncrona), o Serviço Expresso provê uma comunicação

isócrona (síncrona). Uma vez que um novo fluxo de comunicação é admitido para uma classe, os pacotes gerados para este fluxo são marcados, na origem (ou pelo primeiro roteador do domínio DiffServ, chamado roteador de borda), com um código (*DSCP - DiffServ Codepoint*) que identifica para os roteadores a classe de serviço à qual o fluxo de pacotes foi admitido.

Para garantir que os fluxos atribuídos ao Serviço Expresso recebam este serviço é fundamental a utilização de um serviço de controle de admissão. Nesta arquitetura, o controle de admissão irá garantir que não haverá sobrecarga sobre os recursos alocados à classe Serviço Expresso. É garantido que todo pacote recebido é encaminhado.

Para implementar o Controle de Admissão, existem basicamente duas abordagens: a distribuída e a centralizada. No Controle de Admissão distribuído, cada roteador pertencente a um domínio deve ter a capacidade de negociar e administrar os seus recursos, ou seja, eles são responsáveis tanto pela admissão dos fluxos quanto pela alocação dos recursos requisitados. Para executar este mecanismo, a aplicação solicita a QoS desejada diretamente aos roteadores, os quais trocam mensagens entre sí. Para isto, utilizam um protocolo de sinalização como, por exemplo, o protocolo RSVP (Resource Reservation Protocol)[Braden et al. 1997]. Os roteadores irão receber a solicitação de requisitos de QoS necessários para a aplicação e, com base em políticas de admissão, irão aceitar ou não a admissão do novo fluxo, determinando assim a melhor forma de alocação dos recursos solicitados. No Controle de Admissão centralizado, um agente central é responsável por administrar os recursos de um domínio e admitir novos fluxos com base em políticas de aceitação. Este agente recebe os pedidos de QoS das aplicações e, com base em informações obtidas da rede, pode decidir se é possível admitir um novo fluxo ou não. Para tal, ele deve ter a capacidade de se comunicar com os roteadores da rede, colher informações e armazená-las. Em algumas arquiteturas é proposta a existência de um agente chamado Bandwidth Broker [Nahrstedt and Smith 1995], com esta responsabilidade.

A estratégia adotada para verificar a possibilidade de se admitir um novo fluxo na rede depende dos requisitos da aplicação, e do nível de serviço solicitado. Se existem requisitos, por parte da aplicação, para um alto nível de QoS, a rede precisará fornecer garantias rígidas em relação ao serviço oferecido ao novo fluxo de comunicação admitido, e neste contexto, é mais adequado o uso de controle de admissão baseado na disponibilidade de recursos. Se a aplicação não requerer um alto nível de serviço, não sendo portanto necessário que a rede forneça garantias rígidas com relação ao serviço fornecido, o controle de admissão pode ser probabilístico, sendo Baseado em Medidas.

4. Controle de Admissão no QoS Provider

O QoS Provider (QoSP) é um mecanismo desenvolvido para gerenciar as informações mantidas por arquiteturas para prover QoS, provendo canais de comunicação com serviços síncrono e assíncrono, e fornecendo aos sistemas distribuídos informação a cerca do serviço provido a cada canal. Além disto, o QoSP fornece controle de admissão a um domínio de rede executando a arquitetura DiffServ, focando nos serviços Expresso e Melhor Esforço (serviço síncrono e assíncrono). O objetivo imediato do desenvolvimento deste mecanismo é a construção de ambientes de execução híbridos, capazes de fornecer componentes (processos e canais de comunicação) síncronos e

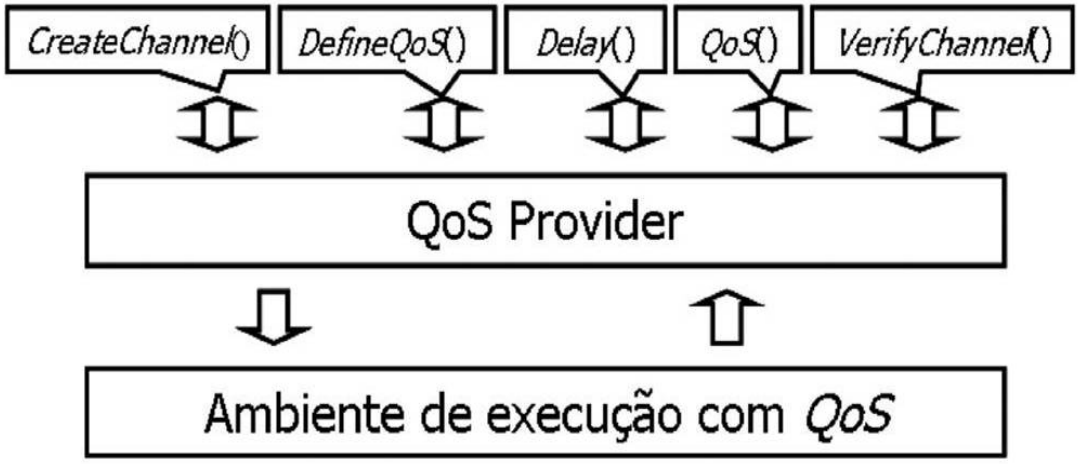

Figura 1. Interface do QoS Provider.

assíncronos aos sistemas distribuídos e o fornecimento de informações sobre a QoS provida a cada canal, informação utilizada por detectores de defeitos híbridos ao realizar detecções, as quais podem ser confiáveis se for estiver sendo utilizado um canal síncrono. Modelos para sistemas distribuídos como o HA [Gorender et al. 2007] e o Spa [Macêdo and Gorender 2009] são modelos híbridos, baseados na existência de canais de comunicação síncronos e assíncronos (no caso do modelo HA), e canais e processos síncronos e assíncronos (no caso do modelo Spa). Para ambos os modelos, uma das formas de se fornecer canais síncronos é através do uso de arquiteturas para prover QoS.

O módulo de controle de admissão desenvolvido para o QoS Provider executa interagindo com uma implementação da arquitetura DiffServ disponível em roteadores Cisco. Este módulo foi baseado na abordagem centralizada, e na estratégia de controle de admissão baseada em disponibilidade de recursos. A função do QoS Provider que representa o controle de admissão é *defineQoS()*, como apresentada na Figura 1. As demais funções do QoSP são responsáveis por criar canais de comunicação (CreateChannel, verificar o atraso máximo para a transferência de mensagens (Delay), verificar se um canal de comunicação é *timely* ou *untimely* (QoS) e verificar se um canal de comunicação remoto apresenta tráfego (VerifyChannel). As justificativas e implementação destas funções está fora do escopo deste trabalho.

O QoSP gerencia e provê dois níveis de serviço, fornecendo canais de comunicação chamados *timely* e *untimely*. Canais *timely* (síncronos) são atribuídos ao Serviço Expresso, definido pela arquitetura DiffServ através do *PHB Expedited Forwarding* [Davie et al. 1999] enquanto o canais *untimely* são providos com o Serviço Melhor Esforço, definido pela arquitetura DiffServ como *PHB Default* [Blake et al. 1998].

Cada *host* do sistema contém um módulo ativo do QoSP, o qual funciona como um servidor local para as aplicações que estão rodando no mesmo *host*. Os serviços são requisitados através do envio de mensagens de solicitações, utilizando para isto as funções fornecidas por uma API cliente, a qual tem uma interface de comunicação para a troca de mensagens com o QoSP. O QoSP, localizado no host dos processos p_x e p_y , é definido, respectivamente, como QoSP$_x$ e QoSP$_y$.

O QoSP Bandwidth Broker (QoSPBB) foi desenvolvido baseado no Bandwidth Broker [Nahrstedt and Smith 1995], para dar suporte ao Controle de Admissão do QoSP.

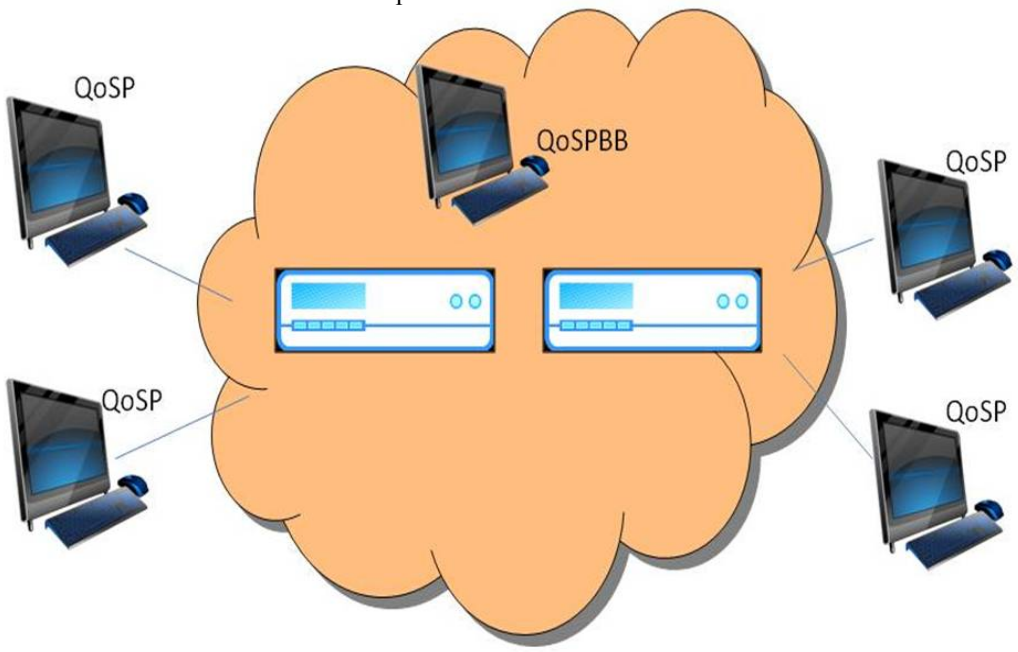

Figura 2. Domínio QoS Provider

O QoSPBB executa em um servidor ligado a algum roteador do domínio de rede, estabelecendo comunicação com todos os roteadores do domínio, e com os módulos QoSP nos *hosts*. A comunicação entre os módulos QoSP cliente e o QoSPBB se dá através de um protocolo de comunicação desenvolvido para tal. O QoSPBB foi projetado para aceitar apenas as requisições dos QoSPs distribuídos pelo domínio de rede, utilizando para isto um mecanismo de autenticação.

Na inicialização de um módulo QoSP, a mensagem $QOSP\_REGISTER$ é enviada ao QoSPBB, com o intuito de registrar o módulo QoSP como cliente do QoSPBB. O controle de admissão é iniciado por um dos processos da aplicação, através de uma solicitação ao módulo QoSP existente em seu *host*. O módulo QoSP estabelece uma comunicação com o QoSPBB e com o módulo QoSP no *host* destino. O resultado da verificação de disponibilidade de recursos para a classe Serviço Expresso nos roteadores do domínio, caso positiva, é armazenada pelo QoSPBB como um Acordo de Nível de Serviço (*Service Level Agreement - SLA*).

5. Implementação do Controle de Admissão no QoSP

O QoSP foi implementado em C++, sobre uma rede de computadores composta por desktops com o sistema operacional linux, e roteadores CISCO 871, com o sistema *IOS() Advanced IP Service*, o qual dá suporte à arquitetura DiffServ. Descrevemos a seguir a API do QoSP, o protocolo de comunicação entre as aplicações e os módulos QoSP, o protocolo de comunicação entre os módulos QoSP eo QoSPBB e o mecanismo de comunicação entre o QoSPBB e os roteadores. A API e as mensagens descritas consideram também funções para criar (*CreateChannel()*) e excluir canais de comunicação. As funções *Delay()*, *QoS()* e *VerifyChannel()*, apresentadas na figura 1 não são descritas neste trabalho.

5.1. A API do QOSP

Os serviços do QoSP são solicitados pelas aplicações, utilizando para isso as funções de uma API. As solicitações são feitas para o módulo do QoSP, o qual está localizado no host

onde as aplicações solicitantes estão rodando. A API funciona como um programa cliente, o qual deve ser anexado às aplicações através do arquivo cabeçalho $api\_qosprovider.h$. Quando uma função da API é utilizada para requisitar um serviço, isto implicará no envio de uma mensagem ao QoSP, conforme o serviço solicitado. A API contém uma função para cada serviço do QoSP. As funções relacionadas com o controle de admissão são:

- $qosp\_init\_api()$: Esta função é utilizada para inicializar as estruturas de comunicação com o módulo do QoSP. Inicializa também a estrutura *tableSockChannel*. Esta função não tem parâmetros de entrada e não solicita serviços ao QoSP.
- $qosp\_createChannel()$: Esta função é utilizada quando a aplicação solicita o registro da criação de um canal de comunicação. Ela recebe os endereços IP e as portas que serão utilizadas pelos processos para a comunicação. Esta função envia a mensagem $CHANNEL\_REQUEST$ para o módulo do QoSP e recebe a resposta através da mensagem $CHANNEL\_REPLY$.
- $qosp\_deletChannel()$: Esta função é solicitada quando uma aplicação deseja finalizar um canal de comunicação. Ela recebe como entrada um valor que identifica o canal a ser finalizado, e envia a mensagem $CHANNEL\_DELET$ ao módulo do QoSP.
- $qosp\_defineQos()$: A aplicação utiliza esta função para solicitar a alteração da QoS provida ao seu canal de comunicação. Isto implica na mudança do tipo de canal utilizado, ou seja, de *timely* para *untimely* ou vice-versa. Os parâmetros de entrada são o identificador do canal de comunicação e uma estrutura que foi definida para conter os parâmetros de QoS que a aplicação necessita. Além dos parâmetros de QoS, esta estrutura contém um valor inteiro, que representa o tipo de canal solicitado (*timely* ou *untimely*). Os parâmetros de QoS só serão enviados para o módulo QoSP através da mensagem $DEFINEQOS\_REQUEST$, caso a solicitação seja de *untimely* para *timely*.

A estrutura *tableSockChannel* citada na função $qosp\_init\_api()$ é um vetor onde são armazenadas as identificações dos canais, ou seja, para cada valor de *socket* criado existirá um *idChannel* equivalente. O *socket* é um valor inteiro vinculado a uma porta de comunicação, o qual identifica a porta a ser utilizada no momento do envio da mensagem. Sendo assim, este identifica o canal para a aplicação. Já o *idChannel* é um valor utilizado pelo módulo QoSP como identificador de um canal. Logo, a estrutura *tableSockChannel* foi criada para relacionar ambos os identificadores.

5.2. Protocolo de comunicação de aplicações com o módulo QoSP

As mensagens que a aplicação envia ao QoSP através da API são constituídas de dois campos padronizados. O primeiro campo da mensagem é o identificador (*Id*) do tipo de mensagem enviada, ou seja, o módulo QoSP vai utilizar este *Id* para identificar o tipo de solicitação, como por exemplo, a criação de um canal, a negociação de QoS, etc. O segundo campo da mensagem (*Information*) contém informações que serão relevantes para o serviço solicitado. Segue abaixo uma descrição das mensagens de solicitação de serviços enviadas a um módulo QoSP:

- $CHANNEL\_REQUEST$: Mensagem enviada pela aplicação ao módulo do QoSP para requisitar o registro de um canal de comunicação. Para enviar esta

mensagem, a aplicação utiliza a função $qosp\_CreateChannel()$ da API. Esta mensagem é formada pelas seguintes informações: *Id* da mensagem, endereço IP do processo p_x, porta do processo p_x, endereço *IP* do processo p_y e porta do processo p_y.

- $CHANNEL\_REPLY$: Esta mensagem é enviada em resposta à solicitação do registro de criação de canal, ou seja, em resposta à mensagem $CHANNEL\_REQUEST$. Esta mensagem contém o identificador do canal criado, o qual será armazenado na estrutura *tableSockChannel* junto com o valor do *socket* equivalente para o canal.
- $CHANNEL\_DELET$: Mensagem enviada ao módulo QoSP para solicitar a exclusão de um canal. Além do *Id* equivalente à solicitação, esta mensagem leva como informação o *idChannel* do canal a ser excluído. Ao receber esta solicitação, o módulo QoSP excluirá as informações do canal armazenado e, caso o canal seja *timely*, os recursos alocados para este serão liberados.
- $DEFINEQOS\_REQUEST$: Esta mensagem é enviada ao QoSP para solicitar a negociação de QoS para um determinado canal de comunicação. Esta mensagem contém, além do *Id* equivalente à solicitação, o *idChannel* do canal e os parâmetros de QoS a serem negociados. Entretanto, caso a negociação seja de *timely* para *untimely*, os parâmetros de QoS não irão compor a mensagem.
- $DEFINEQOS\_REPLY$: Esta mensagem é enviada em resposta à solicitação de negociação de QoS, ou seja, em resposta à mensagem $DEFINEQOS\_REQUEST$. O conteúdo da mensagem identificará se a solicitação foi atendida ou não.

5.3. Protocolo de comunicação entre os módulos QoSP e o QoSPBB

Um módulo QoSP comunica-se com outros módulos QoSP distribuídos na rede e com o QoSPBB, com o intuito de executar seus serviços. Foi criado um protocolo de comunicação para a execução das diversas funcionalidades implementadas, para a realização do controle de admissão.

- $CHANNEL\_REGISTER$: Esta mensagem é enviada de um módulo QoSP para outro, quando um canal é criado entre dois processos. Por exemplo, quando o processo p_x solicita a criação de um canal ao $QoSP_x$, uma mensagem $CHANNEL\_REGISTER$ é enviada ao $QoSP_y$ para que o mesmo registre o canal. Além do *Id*, que identifica o tipo de mensagem, a mensagem também contém as informações referentes ao canal.
- $CHANGE\_QOS$: Esta mensagem é enviada de um módulo QoSP para outro, quando a QoS fornecida a um canal é alterada, o que significa alterar entre os dois tipos de canais possíveis (*timely* e *untimely*). Esta mensagem contém o identificador da mensagem, o identificador do canal, o identificador do acordo SLA registrado no QoSPBB e a nova QoS do canal.
- $CLOSE\_CHANNEL$: Esta mensagem é enviada de um módulo QoSP para outro, quando um canal é encerrado. Quando um processo p_x solicitar o encerramento de um canal ligando os processo p_x e p_y, a mensagem $CLOSE\_CHANNEL$ é enviada ao $QoSP_y$ para que o mesmo também exclua o canal do seu banco de dados. Esta mensagem contém como informação, além do *Id* da mensagem, o identificador do canal a ser excluído.

- $QOSP\_REGISTER$: Esta mensagem é enviada de um módulo QoSP para o QoSPBB. Esta mensagem tem a finalidade de registrar o módulo QoSP no QoSPBB, sendo que este registro é utilizado para autenticar o cliente no momento em que o mesmo solicitar algum serviço ao QoSPBB. Esta mensagem contém o identificador da mensagem e uma senha (*password*) gerada pelo cliente (o módulo QoSP).
- $REGISTER\_REPLY$: Esta mensagem é enviada do QoSPBB para um módulo QoSP, em resposta à mensagem $QOSP\_REGISTER$. A finalidade desta mensagem, além de confirmar o registro do cliente, é de verificar se o QoSPBB está ativo.
- $QOS\_REQUEST$: Esta mensagem é enviada de um módulo QoSP para o QoSPBB quando é feita uma solicitação de reserva de recursos para um canal. Esta mensagem contém como informação, além de seu *Id*, os parâmetros de QoS solicitados pela aplicação, a senha do módulo QoSP e os roteadores que constituem o canal.
- QOS_REPLY: Esta mensagem é enviada do QoSPBB para um módulo QoSP, em resposta à mensagem $QOS\_REQUEST$. Esta contém como informação o resultado da negociação e o identificador do acordo SLA armazenado no QoSPBB.
- QOS_LET: Esta mensagem é enviada de um módulo QoSP para o QoSPBB quando um canal é alterado de *timely* para *untimely*, e quando um canal é encerrado. Esta mensagem tem a finalidade de autorizar a liberação dos recursos que estavam reservados para o canal. Além do *Id* que identifica a mensagem, ela contém o identificador do acordo SLA do canal e a senha do QoSP.

5.4. Comunicação entre o QoSPBB e os roteadores

O QoSPBB precisa comunicar-se com os roteadores do domínio, com o intuito de colher as informações de que necessita. Para esta finalidade, foi utilizado o protocolo SNMP (Simple Network Management Protocol) [Case et al. 1990], o qual é muito utilizado no contexto de gerenciamento dos dispositivos de rede (roteadores, switches, etc). O funcionamento do SNMP é baseado em uma comunicação entre o agente e o gerente. Os agentes são elementos de software instalados nos equipamentos da rede, com a finalidade de colher informações sobre os dispositivos e enviá-las ao gerente, o qual utilizará estas informações para gerenciar os dispositivos.

As informações dos dispositivos, colhidas pelo agente, estão disponíveis em variáveis. Estas variáveis estão estruturadas hierarquicamente em um formato de árvore, sendo esta estrutura conhecida como MIB (Management Information Base). Para obter informações sobre a QoS fornecida pelo roteador foi utilizada a MIB CISCO-CLASS-BASED-QOS-MIB, fornecida pela Cisco para a comunicação com seus roteadores. Foi utilizado um roteador Cisco 871, com o sistema Cisco IOS.

6. Resultados

O ambiente utilizado para a realização de testes foi composto por um roteador CISCO 871, provido com o sistema IOS Security Bundle with Advanced IP Services, o qual dá suporte à arquitetura DiffServ, e três computadores (hosts) configurados com o sistema operacional Debian Gnu/Linux Lenny. Optamos por não utilizar um SO de tempo real, uma vez que, para testar o mecanismo de admissão, não utilizariamos tarefas

de tempo real. Neste ambiente, foram criadas três redes locais, Rede1(192.168.1.0), Rede2(192.168.2.0) e Rede3(192.168.3.0), onde cada host foi configurado em uma destas redes. O *host* C (Rede 3) foi utilizado para rodar o QOSPBB, e cada um dos outros dois hosts executam um módulo do QoSP e uma ou mais aplicações de teste. O ambiente é formado por um domínio DiffServ, o qual interliga as três redes locais. Este domínio é representado apenas por um roteador de núcleo, que tem o papel de encaminhar os pacotes conforme a classe de serviço à qual eles pertencem. O papel de marcar os pacotes fica como atribuição do QoSP ativo no host de origem do fluxo.

O roteador foi configurado com duas classes de serviços: o Serviço Expresso para os canais *timely*, e o de Melhor Esforço(*best-effort*) para os canais *untimely*. É importante salientar que as mensagens trocadas, entre os módulos QoSP e entre estes módulo e o QoSPBB, utilizam canais *timely*.

Uma aplicação simples, do tipo cliente/servidor, foi implementada para testar tanto os módulos desenvolvidos, o QoSP e o QoSPBB, como os protocolos de comunicação criados. A aplicação de teste utiliza as funções da API do QoSP descritas, para solicitar serviços ao QoSP. O objetivo principal desta aplicação é de testar a integração e comunicação entre os componentes, ou seja, entre a aplicação com o módulo QoSP, entre módulos QoSP e de módulos QoSP com o QoSPBB. A aplicação de teste apresenta um menu de opções, onde o usuário pode optar por criar um ou mais canais de comunicação, utilizando para isto a função $qosp\_createChannel()$. Os canais criados são inicialmente *untimely*. Após a criação, o usuário pode solicitar uma QoS para um determinado canal, através da função $qosp\_defineQos()$, sendo que, os requisitos de QoS desejados são passados como parâmetro, para que sejam negociados. Antes de utilizar as funções $qosp\_createChannel()$ e $qosp\_defineQos()$, responsáveis por criar um canal e negociar uma QoS, respectivamente, as aplicações devem utilizar a função $qosp\_init_api()$. Esta função tem o objetivo de inicializar a comunicação com o módulo do QoSP ativo no *Host*.

Foi utilizado um programa chamado Wireshark para analisar o conteúdo das mensagens trocadas entre os módulos desenvolvidos, validando assim os tipos de mensagens trocadas.O Wireshark é um programa que possibilita capturar os pacotes que chegam ou saem de um *Host*, sendo possível ver o cabeçalho e o conteúdo dos pacotes.

Muitos testes foram realizados utilizando a aplicação de teste. Vários canais *untimely* foram criados e admitidos como canais *timely* em um domínio DiffServ, enquanto existiam recursos para admiti-los. Realizamos diversos testes utilizando os canais criados como *timely* e *untimely*, para verificar e validar o controle de admissão implementado. Os resultados que serão mostrados a seguir, referem-se ao RTT (*Round Trip Time*) das mensagens trocadas, ou seja, o tempo que um pacote demora para ir de uma origem até um destino e retornar, em milissegundos, e às perdas de pacotes. Pacotes perdidos não são retransmitidos. Estes valores foram obtidos a partir de quatro fluxos de dados (F1, F2, F3 e F4), sendo que, para cada fluxo de dados, existe uma aplicação de teste rodando, localizada no *Host A*. Os fluxos gerados pela aplicação de teste correspondem ao envio de 10.000 pacotes, com uma taxa de 3 Kbits/s. O canal utilizado para enviar os pacotes pode ser *timely* ou *untimely*. Em cada experimento, foi calculada a média aritmética do RTT dos pacotes transmitidos, em cada canal, assim como o seu desvio padrão.

O DSCP utilizado para o Serviço Expresso foi o recomendado pela IETF, ou seja, o valor 46. Qualquer outro valor de DSCP é considerado pertencente à classe Serviço Melhor Esforço, o qual representa o serviço oferecido pelos canais *untimely*. As larguras de banda configuradas no roteador para as classes de serviço foram: 10 Kbits para os canais *timely* e 10 Kbits para os canais *untimely*. A associação da largura de banda configurada para as classes com a taxa de transferência dos canais permitiu exaurir os recursos reservados, testando assim o mecanismo de admissão.

A tabela 1 mostra quatro fluxos gerados. Os três primeiros foram admitidos e estão utilizando canais *timely*. O fluxo F4 está utilizando um canal *untimely*, pois não pôde ser admitido em decorrência da falta de recursos. Estes fluxos foram gerados do *Host A* para o *Host B*. Neste teste, não foi gerada sobrecarga dos canais. Pela tabela, pode-se perceber que os tempos médios de RTT dos pacotes, que estão utilizando tanto os canais *timely* quanto o canal *untimely*, tiveram valores muito próximos, e não houve perdas de pacotes para nenhum dos fluxos. Já a tabela 2, mostra os mesmos fluxos da tabela anterior, porém com a utilização do programa *ping* do Linux para gerar carga no sistema. Sendo assim, a tabela 2 mostra que o fluxo F4, que utilizou um canal *untimely*, apresentou $4,45\%$ de perdas de pacotes. Isto se deve à sobrecarga no sistema. Os tempos de RTT continuaram próximos para os dois tipos de canais.

Fluxo	Média	Desvio Padrão	Perdas($\%$)
F1	1,651	1,99	0,000
F2	1,209	1,75	0,000
F3	1,144	0,87	0,000
F4	1,343	1,04	0,000

Tabela 1. Canais sem carga

Fluxo	Média	Desvio Padrão	Perdas($\%$)
F1	1,356	1,27	0,00
F2	1,069	1,00	0,00
F3	1,309	1,20	0,00
F4	1,155	0,99	4,45

Tabela 2. Canais com carga

A tabela 3 apresenta um ambiente sem Controle de Admissão, onde os quatro fluxos gerados estão utilizando os recursos existentes da rede. Para este experimento, também foi utilizado o ping para gerar carga no sistema. Podem-se observar perdas de pacotes para os quatro fluxos de dados. Isso ocorreu devido à falta de recursos para atender completamente a todos os fluxos. Já a tabela 4 mostra um ambiente com Controle de Admissão, onde os fluxos F3 e F4 foram admitidos e passaram a utilizar canais *timely*. Pode-se perceber que não houve perdas de pacotes para os fluxos F3 e F4.

Os testes mostram que com o controle de admissão, os fluxos admitidos como *timely*, que são alocados à classe Serviço Expresso, recebem um serviço de fato diferenciado pelo roteador, não apresentando perda de pacotes em suas mensagens. Os fluxos são admitidos para esta classe apenas quando existem recursos suficientes

Fluxo	Média	Desvio Padrão	Perdas(%)
F1	1,486	1,7	11,20
F2	1,228	1,21	22,90
F3	1,470	1,36	15,85
F4	2,906	2,9	18,54

Tabela 3. Canais sem controle de admissão

Fluxo	Média	Desvio Padrão	Perdas(%)
F1	1,144	0,89	3,32
F2	1,083	0,89	2,80
F3	1,323	1,21	0,00
F4	1,220	1,13	0,00

Tabela 4. Fluxos F3 e F4 timely.

para tal. Não havendo perda de pacotes, não haverá a necessidade de retransmissão, gerando atraso na comunicação. Além disto, como a classe Serviço Expresso tem prioridade no encaminhamento de pacotes, o tempo gasto para o encaminhamento destes pacotes será limitado, relativo ao tamanho do *buffer* reservado para a classe. Entretanto, como este *buffer* é grande, o tempo dos pacotes esperando na fila antes de seu encaminhamento variou bastante, proporcional à quantidade de pacotes recebidos e ainda não encaminhados pelo roteador, o que caracteriza o desvio padrão elevado para a média aritmética obtida. No entanto, como o *buffer* tem um tamanho limitado e não há perda de pacotes nesta classe, existe um limite máximo para o atraso no encaminhamento destes pacotes, o qual é razoavelmente superior ao limite mínimo. No caso de canais de comunicação *untimely*, a perda de pacotes implica na necessidade de retransmissão para que as mensagens sejam entregues a seu destino, implicando em um tempo de comunicação não limitado. Os testes mostraram a importância de um Controle de Admissão para criar canais de comunicação com QoS, evidenciando que, em um ambiente sem Controle de Admissão, os recursos podem não ser suficientes para garantir a comunicação através dos canais estabelecidos.

7. Conclusões

Apresentamos neste artigo o módulo de controle de admissão do QoS Provider. Este módulo executa a função *defineQoS()* do QoSP, sendo responsável por verificar a disponibilidade de recursos nos roteadores da rede (largura de banda e memória) para a admissão de novos canais de comunicação a serem providos com Serviço Expresso pela arquitetura DiffServ em execução nestes roteadores. Estes canais apresentarão um comportamento síncrono, com garantias na entrega das mensagens, enquanto não ocorrerem falhas.

O mecanismo de controle de admissão é composto por módulos que são componentes dos módulos QoSP, em execução nos *hosts* dos clientes, e por módulos QoSPBB, que gerenciam a disponibilidade de recursos nos roteadores de um domínio.

Com os testes realizados, verificamos que os canais admitidos para o Serviço Expresso não apresentam perdas de pacotes, tendo todas as suas mensagens entregues ao

seu destino, enquanto que os canais *untimely* (qualquer outro serviço, em geral Serviço Melhor Esforço), apresentam perdas de pacotes, dependendo da sobrecarga gerada na rede.

Este mecanismo é um bloco de construção fundamental para a obtenção de canais de comunicação síncronos fim-a-fim, atuando em conjunto, no QoS Provider, com um mecanismo de admissão de tarefas de tempo real (a partir da utilização de um sistema operacional de tempo real, no caso o Xenomai), e também com a utilização de QoS no acesso do *host* à rede local, com a utilização de placas de rede RtLink com a reserva de tempo de acesso à rede. Utilizando estes canais síncronos é possível executar protocolos para detecção de defeitos, os quais monitoram processos de forma confiável, e no caso de falhas, detectam os processo faltosos de forma confiável.

Referências

Aurrecoechea, C., Campbell, A. T., and Hauw, L. (1998). A survey of qos architectures. *ACM Multimedia Systems Journal, Special Issue on QoS Architecture*, 6(3):138–151.

Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and Weiss, W. (1998). An architecture for differentiated services. *RFC 2475*.

Braden, B., Clark, D., and Shenker, S. (1994). Integrated services in the internet architecture: an overview. *RFC 1633*.

Braden, B., Zhang, L., Berson, S., Herzog, S., and Jamin, S. (1997). Resource reservation protocol (rsvp) - version 1 functional specification. *RFC 2205*.

Campbell, A., Coulson, G., and Hutchison, D. (1994). A quality of service architecture. *ACM Computer Communications Review*, 24(2):6–27.

Case, J., Fedor, M., Schoffstall, M., and Davin, J. (1990). A simple network management protocol (snmp). *RFC 1157*.

Davie, B., Charny, A., Bennet, J. C. R., Benson, K., Boudec, J. Y. L., Courtney, W., Davari, S., Firoiu, V., and Stiliadis, D. (1999). An expedited forwarding phb (per hop behavior). *RFC 3246*.

Gorender, S., Macêdo, R. J. A., and Cunha, M. (2004). Implementação e análise de desempenho de um mecanismo adaptativo para tolerância a falhas em sistemas distribuídos com qos. In *Anais do Workshop de Testes e Tolerância a Falhas, V WTF - SBRC2004*, pages 3–14.

Gorender, S., Macêdo, R. J. A., and Raynal, M. (2007). An adaptive programming model for fault-tolerant distributed computing. *IEEE Transactions on Dependable and Secure Computing*, 4(1):18–31.

Macêdo, R. J. A. and Gorender, S. (2009). Perfect failure detection in the partitioned synchronous distributed system model. In *Proceedings of the The Fourth International Conference on Availability, Reliability and Security (ARES 2009), IEEE CS Press*.

Nahrstedt, K. and Smith, J. M. (1995). The qos broker. *IEEE Multimedia*, 2(1):53–67.

Siqueira, F. and Cahill, V. (2000). Quartz: A qos architecture for open systems. In *International Conference on Distributed Computing Systems*, pages 197–204.