UNIVERSIDADE DE ÉVORA

ESCOLA DE CIÊNCIAS E TECNOLOGIA

DEPARTAMENTO DE INFORMÁTICA

QoS em servidores HTTP Apache

José Carlos Ferreira da Silva Martins

Orientação: Prof. Doutor Luís Miguel Mendonça Rato

Mestrado em Engenharia Informática

Dissertação

Évora, 2015

Resumo

Os serviços baseados na Internet têm registado um crescimento contínuo e acelerado nos últimos anos, dependendo o seu sucesso, em larga medida, da qualidade de serviço prestada. A sociedade moderna tornou-se fortemente dependente da Internet e dos vários serviços nela disponibilizados.

Nesta dissertação é abordado o problema da qualidade de serviço em servidores HTTP, com particular ênfase no servidor HTTP Apache. Neste trabalho, é definido e implementado um sistema original de controlo em cadeia fechada de QoS, baseado nas metodologias da teoria de controlo, demonstrando-se a sua robustez, estabilidade e capacidade para gerir os recursos de forma dinâmica. Por fim, é comprovada a capacidade do sistema realizar a diferenciação de serviço (*DiffServ*) entre duas classes de sítios: *Premium* e *Outros*. Esta qualidade é comprovada para os objetivos *erro nulo*, *rejeição de perturbações* e *seguimento de referência*.

Palavras-chave: Servidor HTTP, Apache, Teoria de controlo, Qualidade de Serviço, QoS, Erro nulo, Rejeição de perturbações, Seguimento de referência.

Abstract

QoS for Apache HTTP server

Internet-based services have registered a continuous and rapid growth in recent years, depending their success to a large extent on the provided quality of service. Modern society has become heavily dependent on the Internet and the various services it provides.

This thesis addresses the issue of quality of service in HTTP servers, with particular emphasis on the Apache HTTP server. In this work, a closed-loop system with QoS, based on control theory methodologies, is defined and implemented. The proposed system robustness, stability and ability to manage resources dynamically is shown. Finally, it is confirmed the system's ability to provide differentiated services (*DiffServ*) between two classes of sites: *Premium* and *Others*. This quality is proven to the objectives *regulatory control*, *disturbance rejection* and *time-varying reference*.

Keywords: HTTP server, Apache, Control theory, Quality of Service, QoS, Regulatory control, Disturbance rejection, Time-varying reference.

"Sem ambição, nada se começa. Sem esforço, nada se completa."
Ralph Waldo Emerson

Agradecimentos

Quero agradecer profunda e sinceramente ao meu orientador, Prof. Doutor Luís Rato, pela sua preciosa ajuda e apoio que sempre manifestou. Agradeço a sua constante disponibilidade e motivação nos momentos mais difíceis.

Quero também agradecer à minha família pelo seu amor, compreensão e paciência ao longo destes anos.

Agradeço, ainda, a todos aqueles que, de forma direta ou indireta, contribuíram para que este trabalho chegasse ao seu término.

A todos vós,

Muito Obrigado.

Índice Geral

Índice de Quadros

Índice de Figuras e Gráficos

Índice de Abreviaturas Utilizadas

ACK	Acknowledgement
ANS	Acordo de Nível de Serviço
APR	Apache Portable Runtime
CBQ	Class Based Queueing
CGI	Common Gateway Interface
CPU	Central Processing Unit
CRLF	Carriage Return, Line Feed
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DoS	Denial of Service
FIFO	First in, First out
HTB	Hierarchical Token Bucket
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPD	Apache HyperText Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
IaaS	Infrastructure-as-a-Service
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IIS	Internet Information Services
I/O	Input/Output
IP	Internet Protocol
ISC	Internet Systems Consortium
MIMO	Multiple Input, Multiple Output
MPM	Multi-Processing Modules
MTU	Maximum Transmission Unit
NCSA	National Center for Supercomputing Applications
NetEm	Network Emulation
NTP	Network Time Protocol
PaaS	Platform-as-a-Service
PHP	Hypertext Preprocessor

PID	Proporcional-Integral-Derivativo
POSIX	Portable Operating System Interface
PR	Percentagens Relativas
QoS	Quality of Service
RAM	Random Access Memory
RED	Random Early Detection
RSVP	Resource reSerVation Protocol
RTO	Retransmittion Timeout
SaaS	Software-as-a-Service
SFQ	Stochastic Fairness Queueing
SGBD	Sistemas de Gestão de Bases de Dados
SISO	Single Input, Single Output
SL	Service Level
SLA	Service Level Agreements
SSI	Server Side Includes
SSL	Secure Socket Layer
tc	Traffic Control
TBF	Token Bucket Filter
TCP	Transmission Control Protocol
ToS	Type of Service
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WAN	Wide Area Network

1. Introdução

Os serviços baseados na Internet têm registado um crescimento contínuo e acelerado nos últimos anos, dependendo o seu sucesso, em larga medida, da qualidade de serviço (QoS) prestada. A sociedade moderna tornou-se fortemente dependente da Internet e dos vários serviços nela disponibilizados.

Com o advento da "computação em núvem"[1], os serviços baseados na Internet desempenham um papel cada vez mais importante, sendo o *software-as-a-service* (SaaS), *platform-as-a-service* (PaaS), *data-storage-as-a-service* (dSaaS) e *infrastructure-as-a-service* (IaaS), alguns desses exemplos. O sucesso da "computação em núvem" depende, em grande medida, da qualidade de serviço (QoS) prestada [31], o que eleva a necessidade da monitorização da qualidade de serviço de modo a impor um "Acordo de Nível de Serviço"[2] (ANS) [47]. É, igualmente, importante não esquecer os "serviços tradicionais/clássicos": notícias, correio electrónico, *homebanking*, *etc.*

O aumento do uso de servidores HTTP representa um constante desafio ao seu desenho e desenvolvimento no sentido de lhes permitir melhorar o desempenho, a disponibilidade e a escalabilidade. Embora na maior parte dos casos não seja técnica e economicamente possível alocar recursos dimensionados para os picos de utilização é, ainda assim, possível fornecer uma melhor QoS a sítios privilegiados (*Premium*), mesmo quando os servidores estão sobrecarregados com pedidos [22]. Deste modo, o uso de sistemas de QoS é, cada vez mais, uma exigência/tendência no desenho e desenvolvimento de servidores HTTP.

Por outro lado, a teoria de controlo é uma ferramenta poderosa no desenho de sistemas de QoS, permitindo alocar de forma dinâmica diferentes recursos a diferentes classes de pedidos baseados em amostras de *feedback*. Pese embora a teoria de controlo seja utilizada em várias áreas, no caso concreto de sistemas computacionais, o objetivo principal é torná-los mais robustos e estáveis. As áreas mais populares de pesquisa têm sido as redes informáticas, sistemas operativos, servidores HTTP, sistemas de gestão de bases de dados (SGBD), multimédia e gestão de energia [28].

Analistas e designers estão extremamente interessados nas características de desempenho de

[1] Do inglês, *Cloud computing*
[2] Do inglês, *Service Level Agreement (SLA)*

sistemas computacionais, em especial em tempos de resposta, débito[1] e gestão de filas de espera. Embora as características estáticas possam ser compreendidas usando a "teoria de filas de espera"[2], na prática faltam ferramentas conceptuais para tratar a gestão de recursos dinâmicos, em especial alterações de carga e configurações [28].

As dinâmicas de sistemas computacionais são questões importantes para a rendibilidade e disponibilidade de vários negócios. Os sítios de *e-commerce*, por exemplo, têm frequentemente cargas que variam tão rapidamente que podem ocorrer situações de degradação, ou até quebra de serviço. Administradores de sistemas experientes sabem que deixar a gestão de aspetos dinâmicos a cargo de operadores não é aceitável, já que as alterações ocorrem tão rapidamente que não é humanamente possível responder em tempo útil [28].

Nos últimos anos podemos apontar várias ocasiões em que os recursos disponíveis em determinados servidores HTTP não foram suficientes para satisfazer a quantidade de pedidos: *Big Brother* em Portugal (2000), ataques de 11 de setembro nas torres gémeas de Nova Iorque (2001), *tsunami* no Oceano Índico (2004), *Wiki Leaks* (novembro 2010), *etc.* Em todos os casos mencionados, o acesso aos sítios de conteúdos informativos foi tão elevado que os servidores HTTP não conseguiram satisfazer os pedidos. No caso concreto do sítio WikiLeaks (`http://wikileaks.org`), no dia 28 de novembro de 2010, o mesmo começou a divulgar documentos confidenciais de embaixadas americanas. O acesso ao sítio foi tão elevado que durante a maior parte do dia e mesmo no dia seguinte quem tentasse aceder obtinha, quase sempre, a seguinte página[3]:

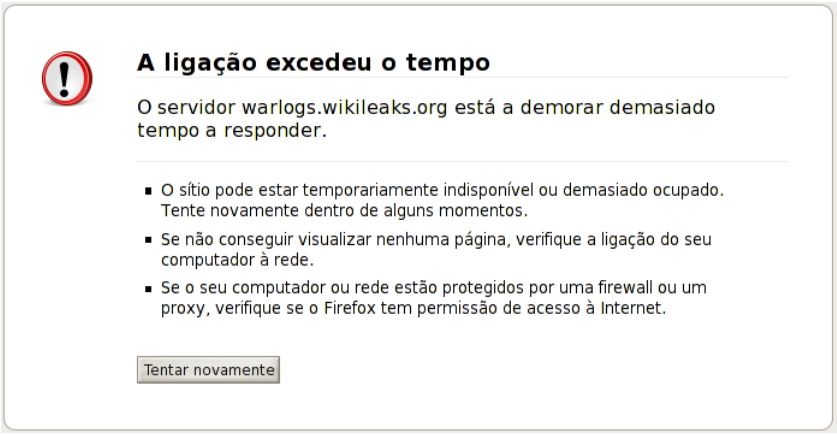


Figura 1.1: Acesso ao sítio "WikiLeaks" durante o período de sobrecarga do servidor HTTP.

[1] Do inglês, *Throughput*
[2] Do inglês, *Queueing theory*
[3] Acedido 29 novembro 2010 às 12h50m em Portugal

1.1 Objetivos

Não é possível ter uma rede informática com bom desempenho se não existir um controlo de tráfego implementado. Aumentar os recursos disponíveis nem sempre é a solução, já que geralmente o sistema só demora mais tempo até saturar assumindo, a partir desse momento, o mesmo comportamento deficiente. Além disso, aumentar os recursos, por si só, não permite fazer a diferenciação de serviço, muitas vezes necessária e desejada.

O algoritmo de escalonamento no servidor HTTP Apache é `FIFO` (*First In First Out*), ou seja, os pedidos são satisfeitos por ordem de chegada, sem qualquer prioridade.

Este trabalho procura atingir os seguintes objetivos:

- Analisar e alterar o código fonte do servidor HTTP Apache, de modo a permitir recolher a informação necessária à implementação de um sistema original de controlo em cadeia fechada de QoS, baseado em teoria de controlo;
- Demonstrar um sistema de controlo em cadeia fechada, baseado na informação obtida dos processos do servidor HTTP Apache, que permite fazer a diferenciação de QoS entre duas classes de sítios: sítio *Premium* ("sítio privilegiado", ao qual eram alocados mais recursos) e sítio *Outros* ("sítio não privilegiado", ao qual eram alocados os recursos remanescentes);
- Atestar a robustez e estabilidade do sistema implementado, através da simulação com várias cargas de pedidos e com várias metodologias, designadamente: *erro nulo*, *seguimento de referência* e *rejeição de perturbações*;
- Comparar o comportamento do servidor HTTP Apache com o sistema de QoS e sem o sistema de QoS.

1.2 Principais contribuições desta dissertação

As principais contribuições desta dissertação são: em primeiro lugar, a definição e implementação de um sistema original de controlo em cadeia fechada de QoS, aplicado ao servidor HTTP Apache; em segundo lugar, demonstrar que, com este sistema de QoS, é possível implementar uma ferramenta robusta e estável de recolha, análise e gestão de recursos dinâmicos em servidores HTTP Apache.

1.3 Estrutura desta dissertação

Esta dissertação é composta por cinco (5) capítulos e dois (2) anexos.

No presente capítulo, 1. Introdução, é feita uma apresentação do trabalho, dos objetivos a atingir e das principais contribuições desta dissertação.

No capítulo 2. Revisão Bibliográfica, é apresentada uma pesquisa bibliográfica sobre o tema da dissertação, a qual tem como objetivo conhecer os aspetos mais relevantes dos servidores HTTP, em particular o servidor HTTP Apache, bem como conhecer o "estado da arte" da teoria de controlo e de sistemas de QoS aplicados a servidores HTTP.

No capítulo 3. Equipamentos e Metodologia é explicada, em detalhe, a infraestrutura informática implementada, bem como a metodologia seguida designadamente: quais as otimizações realizadas, as ferramentas de teste, como foi recolhida a informação, bem como outros aspetos fundamentais para a compreensão do sistema de QoS implementado.

No capítulo 4. Apresentação e Discussão de Resultados, são apresentados os resultados obtidos pela aplicação da metodologia descrita no capítulo anterior e a sua respetiva análise.

No capítulo 5. Conclusões e Trabalho Futuro, são apresentadas as conclusões baseadas nos resultados obtidos neste trabalho. São, ainda, apontadas algumas linhas de investigação em aberto e possíveis complementos a este trabalho.

No final da dissertação encontram-se os anexos, onde se apresenta o código fonte desenvolvido, bem como os quadros resumo dos principais resultados dos clientes HTTP.

2. Revisão Bibliográfica

O grande interesse por problemas e dados cada vez mais complexos, bem como as exigências colocadas pelos utilizadores e instituições tem conduzido à necessidade de desenvolver sistemas computacionais cada vez mais potentes e complexos. No entanto, limitações económicas e tecnológicas impedem que os sistemas tenham os recursos necessários em todas as situações. Mesmo nos casos em que os sistemas computacionais têm elevados recursos, eles são sempre finitos, o que faz com que possam ocorrer situações em que os mesmos não sejam suficientes. Como é natural, estas limitações aplicam-se também aos servidores HTTP, o que exige que os mesmos sejam otimizados, sempre que possível. Importa, assim, conhecer as características mais importantes destes sistemas e quais os aspetos mais relevantes na sua estrutura, configuração e otimização.

2.1 Estado da Arte

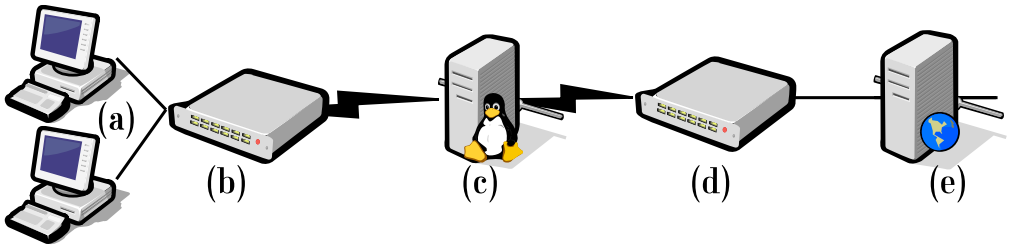
Nas últimas duas décadas foram realizados vários estudos sobre serviços baseados na Internet. Alguns desses trabalhos propõem sistemas SISO e MIMO de controlo em cadeia fechada com enfoque no CPU e na memória de referência e, na sua relação com os parâmetros de configuração `MaxClients` e `KeepAlive` do servidor HTTP Apache [28]. São igualmente propostos sistemas baseados na diferenciação de serviço (*DiffServ*), os quais comportam duas classes de clientes: `Premium` e `Básico`. Nesta área, diferentes autores sugerem e avaliam uma arquitetura na qual a restrição é imposta nos recursos disponíveis, *threads* ou processos, para os clientes do tipo `Básico` [27]. Outros autores utilizam mecanismos de controlo de admissão e algoritmos de escalonamento para fornecer um melhor serviço aos clientes `Premium` [2, 16, 34]. Alguns desses autores argumentam que a melhor abordagem para fornecer QoS aos serviços baseados na Internet passa por realizar o controlo de admissão e técnicas de modelação nos pontos de entrada dos sítios da Internet [16]. Existem, igualmente, estudos onde é proposta uma arquitetura na qual são mantidas filas de serviço separadas para clientes `Premium` e `Básicos`, facilitando assim, um tratamento diferenciado [15].

Foi, igualmente, desenvolvido um módulo de QoS para o servidor HTTP Apache: `mod_qos` [18]. Este módulo tem capacidade para, por exemplo, limitar o número de pedidos para

um determinado URI ou *Virtual host*, limitar o número de pedidos de um determinado IP, número de pedidos por segundo, *etc.*

Linhas mais recentes de investigação abordam os controladores *fuzzy*, os quais são controladores dinâmicos, não-lineares. Os controladores *fuzzy* são bem conhecidos pela sua capacidade em se adaptar a ambientes dinâmicos, imprecisos e com picos, tal como sucede com o tráfego Internet [40]. É sabido que o tráfego e as cargas na Internet tem características estocásticas e apresentam variações significativas ao longo do tempo. Assim sendo, o principal desafio passa por saber como implementar um controlo eficiente e com bom desempenho nas várias condições de carga, sabendo do comportamento não linear de um servidor Internet em resposta aos recursos alocados. A maior desvantagem dos controladores *fuzzy* diz respeito ao elevado número de parâmetros a ajustar [21]. É, em especial, difícil definir os ajustes iniciais adequados [45]. Mais, os ajustes dos parâmetros assentam nos conhecimentos de quem implementa este tipo de controlador.

Nesta dissertação a abordagem ao problema será realizada através da implementação de uma infraestrutura informática constituída por vários componentes, designadamente: um servidor HTTP Apache modificado, dois clientes HTTP, uma *Gateway* (onde reside todo o mecanismo de QoS: controlador PID/transdutor, *etc.*), dois *switch* e respetiva cablagem (figura 2.1).



(a) Clientes HTTP, (b) *Switch* A, (c) *Gateway*, (d) *Switch* B, (e) Servidor HTTP Apache.

Figura 2.1: Esquema simplificado da infraestrutura informática.

Na figura 2.2 apresenta-se um esquema simplificado da "ligação física ao sistema de controlo em cadeia fechada". São representados, entre outros aspetos, o fluxo de pacotes, a sua classificação, as filas de espera e o controlador PID.

Os clientes HTTP efetuam os pedidos ao sistema, sendo os mesmos enviados para a *Gateway*. Os pacotes com os pedidos são colocados na fila de espera única da interface `eth1`, sendo posteriormente encaminhados (`ip_forwarding`) para a interface `eth0`. Quando os pacotes chegam à interface `eth0`, são classificados de acordo com o sítio a que se destinam, isto é: `Premium` ou `Outros`, existindo uma fila de espera associada a cada tipo de sítio. Caso os pacotes não sejam destinados a nenhum destes sítios, são colocados na fila de espera `Default`. Aos pacotes colocados nas filas `Premium` e `Outros` é eventualmente aplicado um atraso no seu envio (τ). O valor deste atraso é determinado pelo controlador PID e transdutor do sistema.

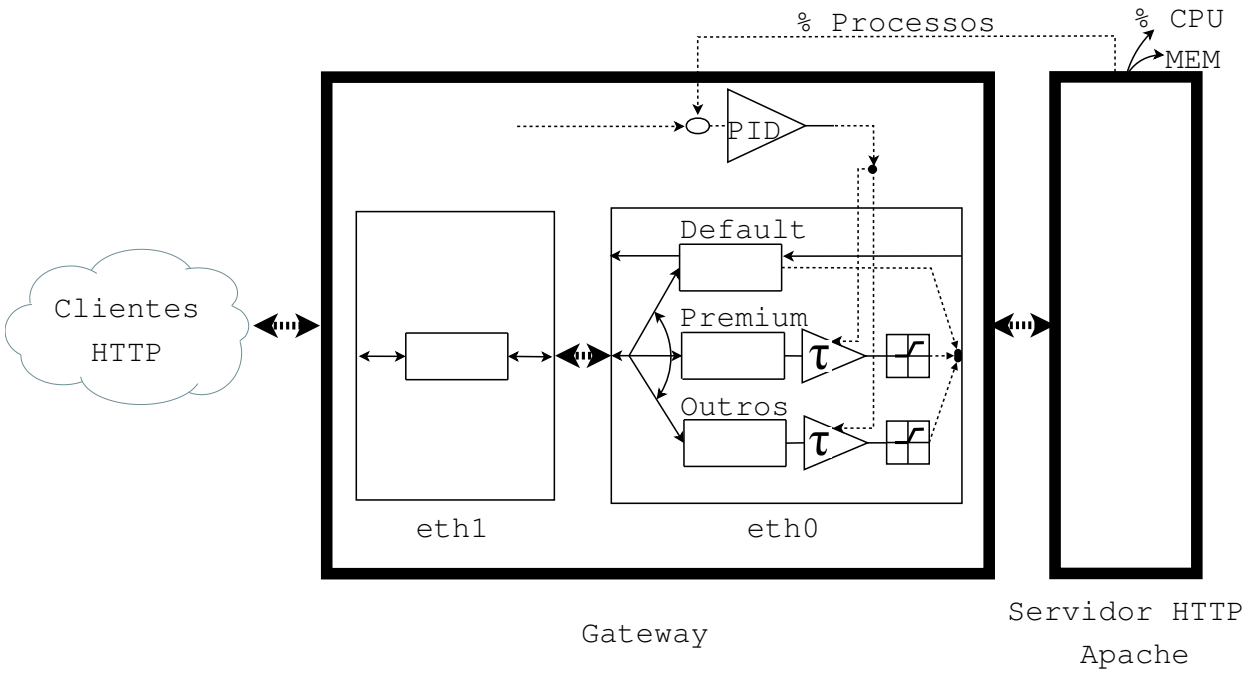


Figura 2.2: Esquema simplificado da "ligação física ao sistema de controlo em cadeia fechada, aplicado ao servidor HTTP Apache".

O controlador recebe a informação enviada pelo servidor HTTP Apache, nomeadamente, a percentagem de processos `Ativos` e `Premium`, CPU e memória, em intervalos de cerca de 5 segundos. Os pacotes com os pedidos são finalmente enviados pela *Gateway* para o servidor HTTP Apache, para processamento e resposta.

2.2 Servidores HTTP

2.2.1 Introdução

Um servidor HTTP resulta da combinação de equipamento informático[1] e do programa informático[2] nele instalado. O servidor HTTP interage com o cliente através de um navegador de Internet[3]. Nesta comunicação é normalmente utilizado o protocolo HTTP ou HTTPS.

De uma forma genérica o fluxo de informação entre o servidor HTTP e o navegador do cliente é o seguinte:

Passo 1: Cliente - Análise[4] do URL

O primeiro passo que um navegador tem de fazer é verificar de que forma vai solicitar ao

[1]Do inglês, *Hardware*
[2]Do inglês, *Software*
[3]Do inglês, *Browser, ou Web Browser*
[4]Do inglês, *Parsing*

servidor HTTP uma determinada página, ou documento, a partir de um URL. O URL tem a seguinte forma básica: "protocolo://servidor/URIdoPedido". O protocolo é quase auto-explicativo e identifica qual o protocolo que deve ser utilizado para a comunicação entre o cliente e servidor, por exemplo: http, https. Em seguida, o navegador é informado qual o servidor que deve contatar. Por fim, o `URIdoPedido` (URI do Pedido) é o "nome" utilizado pelo servidor HTTP para identificar o objeto (página ou documento).

Passo 2: Cliente - Pedido HTTP[1]

Habitualmente, o protocolo utilizado é HTTP. Neste passo, o navegador envia o pedido ao servidor sob a forma `GET /URIdoPedido HTTP/versão`, em que a versão informa o servidor qual a versão de HTTP a usar (HTTP 1.0 ou HTTP 1.1 - ligações persistentes).

Convém salientar que o pedido deve conter toda a informação necessária para o servidor poder satisfazer o pedido, já que o servidor não sabe qual a origem do pedido. O servidor apenas processa o pedido e devolve o resultado. Saliente-se que processar o pedido pode implicar várias etapas como, por exemplo, o acesso a Sistemas de Gestão de Bases de Dados (SGBD), processamento de CGI, *etc.*

Passo 3: Servidor - Resposta HTTP[2] [53]

Após receber e interpretar o pedido, o servidor responde com uma resposta HTTP.

A primeira linha de uma resposta HTTP é a "linha de estado"[3], a qual é composta pela versão do protocolo, seguido de um código numérico de estado e da sua frase textual associada (Descrição). Cada um destes elementos está separado por um espaço e a linha termina como a sequência CRLF[4]. Deste modo, a primeira linha da resposta (linha de estado, LE) tem a seguinte forma:

LE = versão HTTP ␣Código de estado ␣Descrição ↵

O código de estado/resposta é um número inteiro de 3 dígitos, o qual indica o resultado da tentativa do servidor compreender e satisfazer o pedido do cliente. A frase textual associada (Descrição) pretende fornecer uma pequena descrição do significado do código de estado. Assim sendo, pode-se afirmar que o código de estado é para ser interpretado pelo navegador, enquanto a descrição é para ser entendida pelo utilizador. O navegador não tem necessariamente de analisar, ou até de apresentar a descrição. Estes códigos padrão de resposta HTTP foram definidos pela IETF[5] e são atualmente geridos pela IANA[6].

[1]Do inglês, *HTTP Request*
[2]Do inglês, *HTTP Response*
[3]Do inglês, *Status-Line*
[4]Do inglês, *Carriage return line feed*
[5]Internet Engineering Task Force, `http://www.ietf.org`
[6]Internet Assigned Numbers Authority, `http://www.iana.org`

O primeiro dígito do código de estado define a classe da resposta. Os dois últimos dígitos não têm qualquer papel na classificação da resposta. Tem por missão fornecer informações mais específicas. Os cinco valores possíveis para o primeiro dígito são os seguintes [33]:

1xx: Informação - Pedido recebido, em processamento;

2xx: Sucesso - A ação foi recebida com sucesso, compreendida e aceite;

3xx: Redirecionamento - Outras ações devem ser tomadas a fim de completar o pedido;

4xx: Erro de cliente - O pedido contém sintaxe inválida, ou não pode ser cumprido;

5xx: Erro de servidor - O servidor não conseguiu executar um pedido aparentemente válido.

Resumidamente, na sua forma mais simples e, tal como é apresentado na figura 2.3, a comunicação HTTP consiste numa mensagem do tipo *HTTP Request* do cliente para o servidor e numa resposta *HTTP Response* por parte do servidor.

É possível simular o pedido do navegador ao servidor HTTP com o IP 192.168.100.100, recorrendo à ferramenta `telnet`, da seguinte forma:

```
telnet 192.168.100.100 80

Trying 192.168.100.100...
Connected to 192.168.100.100.
Escape character is '^]'.

GET /exemplo.html HTTP/1.1
Host: 192.168.100.100
```

Resposta do servidor HTTP Apache instalado em GNU/Linux:

```
HTTP/1.1 200 OK
Date: Wed, 14 Sep 2014 16:00:00 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Wed, 14 Sep 2014 15:50:21 GMT
ETag: "1800f-35-4acdb87ae3940"
```

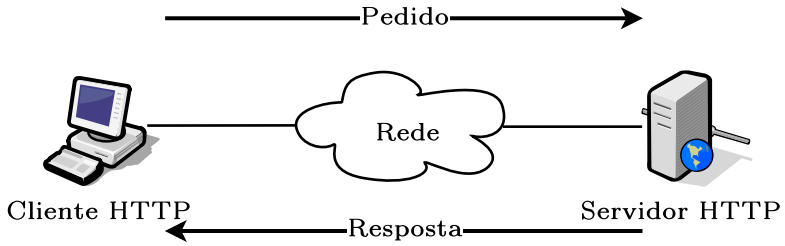


Figura 2.3: Esquema simplificado da comunicação entre um cliente e o servidor HTTP Apache.

```
Accept-Ranges: bytes
Content-Length: 53
Content-Type: text/html; charset=UTF-8

<html>
 <body>
  <p>Olá mundo!</p>
 </body>
</html>
Connection closed by foreign host.
```

2.2.2 Implantação

A implantação de servidores HTTP é muito grande e encontra-se dispersa em todo o mundo. Em Fevereiro de 2015, o sítio Netcraft [42] obteve resposta de 883.419.935 nomes diferentes de anfitrões[1] e 177.296.984 sítios ativos. Neste sítio é igualmente possível obter várias estatísticas, entre elas, o número total de sítios (figura 2.4) e a quota de mercado dos principais servidores HTTP (figura 2.5, figura 2.6 e Quadro 2.1).

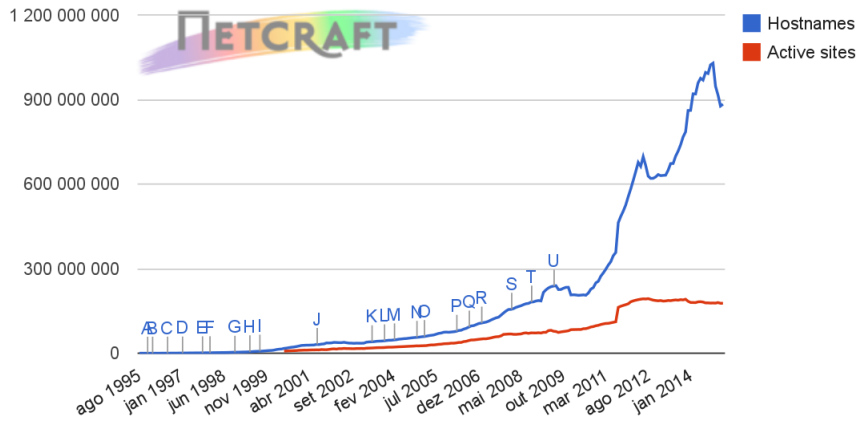


Figura 2.4: Estatísticas do sítio Netcraft. Número total de sítios. Agosto 1995 a Fevereiro 2015.

[1] Do inglês, *Hosts*

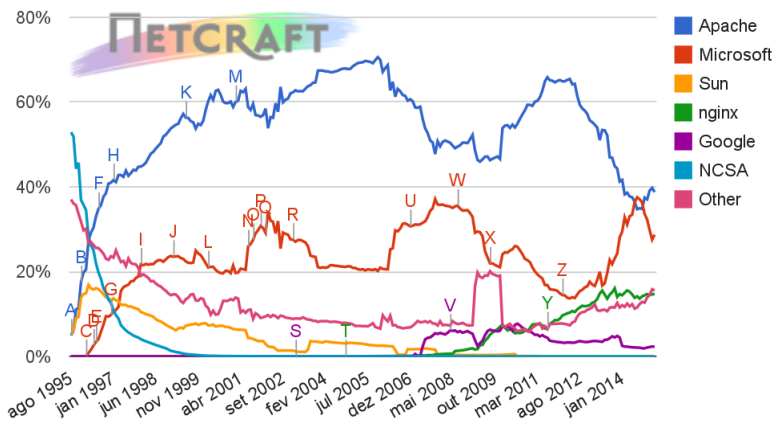


Figura 2.5: Estatísticas do sítio Netcraft. Quota de mercado dos principais servidores HTTP (Todos). Agosto 1995 a Fevereiro 2015.

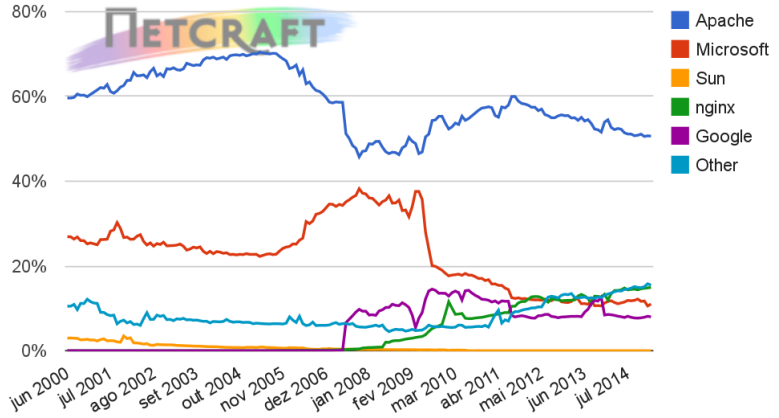


Figura 2.6: Estatísticas do sítio Netcraft. Quota de Mercado dos principais servidores HTTP (Ativos). Junho 2000 a Fevereiro 2015.

A metodologia pormenorizada seguida pela Netcraft para a obtenção destes dados está descrita em `http://www.netcraft.com/active-sites/`.

É possível constatar que o servidor HTTP Apache é o servidor HTTP mais utilizado, contabilizado cerca de 39% da totalidade dos sítios e mais de 50% dos sítios ativos. Esta realidade, enquadrada com o exposto anteriormente justifica a importância de conhecer, em pormenor,

	Todos os sítios		Sítios ativos	
	Número	%	Número	%
Apache	342.480.920	38,77	89.780.592	50,64
nginx	253.484.221	28,69	26.481.450	14,94
Microsoft	130.093.899	14,73	19.504.039	11,00
Google	20.238.057	2,29	14.137.227	7,97

Quadro 2.1: Estatísticas do sítio Netcraft. Quota de mercado dos principais servidores HTTP. Fevereiro 2015.

o funcionamento deste servidor, bem como os seus mecanismos de ajustamento de modo a ser possível desenvolver um sistema de QoS baseado em teoria de controlo.

2.3 Servidor HTTP Apache

2.3.1 Introdução

Em Fevereiro de 1995 o servidor HTTP [12] desenvolvido por Rob McCool no *National Center for Supercomputing Application's* (NCSA) da Universidade de Illinois era o mais popular. Contudo, o desenvolvimento desse servidor parou depois de Rob McCool ter deixado o NCSA, em meados de 1994. Entretanto, alguns programadores continuaram a desenvolver as suas próprias extensões e a resolver erros de programação[1].

Um pequeno grupo de *webmasters*, contactados via email, juntaram-se com o propósito de coordenarem as suas alterações, sob a forma de *patches*. Brian Behlendorf e Cliff Skolnick disponibilizaram uma *mailing list*, um espaço de partilha de informação e acessos para os programadores principais num servidor. No final de Fevereiro oito programadores principais formaram a fundação original do *Apache Group*; eles eram: Brian Behlendorf, Roy T. Fielding, Rob Hartill, David Robinson, Cliff Skolnick, Randy Terbush, Robert S. Thau e Andrew Wilson. Também contribuiram Eric Hagberg, Frank Peters e Nicolas Pioch.

Usando o servidor HTTPD 1.3 da NCSA como base, foram corrigidos todos os erros conhecidos e adicionadas todas as funcionalidades disponíveis consideradas relevantes. Foi testado o resultado e lançada a primeira versão pública do servidor HTTP Apache (0.6.2) em Abril de 1995. Este lançamento teve um grande sucesso na altura. A partir deste momento existiu um esforço concertado na correção dos erros e na disponibilização de novas funcionalidades.

Menos de um ano após a constituição deste grupo, o servidor HTTP Apache ultrapassou o servidor HTTP da NCSA como o servidor n.º 1, lugar que de acordo com o inquérito da Netcraft (apresentado anteriormente) ainda mantém.

Em 1999, os membros do *Apache Group* formaram a *Apache Software Foundation*, para fornecer suporte organizacional, legal e financeiro ao servidor HTTP Apache.

2.3.2 Arquitetura

O servidor HTTP Apache é constituído por um pequeno conjunto de funcionalidades base e um conjunto de módulos [35], estando representado na figura 2.7. Os módulos podem

[1] Do inglês, *Bugs*

ser compilados estaticamente no servidor ou, mais frequentemente, existirem no diretório `/modules` ou `/libexec` sendo carregados dinamicamente em tempo de execução. De igual forma, o servidor conta com as bibliotecas do *Apache Portable Runtime* (APR), as quais fornecem uma camada[1] multiplataforma e utilitários de modo a que os módulos não tenham de contar com chamadas ao sistema operativo não portáveis. Um módulo especial denominado *Multi-Processing Module* (MPM) permite otimizar o Apache para o sistema operativo subjacente. O modulo `MPM` conjuntamente com as bibliotecas APR são os que normalmente devem aceder diretamente ao sistema operativo.

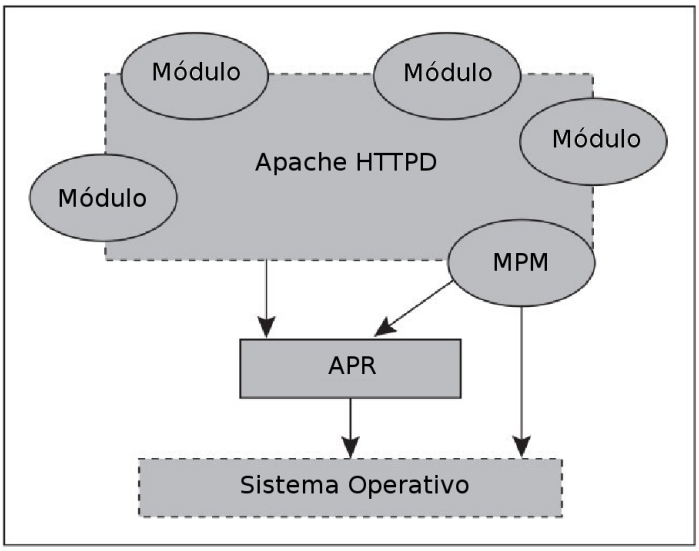


Figura 2.7: Arquitetura do servidor HTTP Apache.

O Apache funciona em duas fases distintas: inicial e operacional. O arranque do sistema tem lugar com o utilizador "root" e inclui a leitura dos ficheiros de configuração, o carregamento dos módulos e a inicialização dos recursos do sistema, tais como ficheiros de *log*, segmentos de memória partilhada e ligações a SGBD. Para o funcionamento normal o Apache "abandona" os seus privilégios de sistema e é executado como um utilizador não-privilegiado, antes de aceitar e processar as ligações dos clientes. Esta medida básica de segurança ajuda a impedir que um simples erro no Apache, num módulo, ou *script*, possa se tornar numa vulnerabilidade de todo o sistema, tal como sucedeu com os casos do "Code Red" [19] e "Nimda" [20] no *Internet Information Services* (IIS) da Microsoft.

O servidor HTTP Apache tem uma fila[2] para os pedidos dos clientes (figura 2.8) e um número de processos/`workers` prontos a servir esses pedidos [28] (figura 2.9).

[1] Do inglês, *Layer*
[2] Do inglês, *Queue*

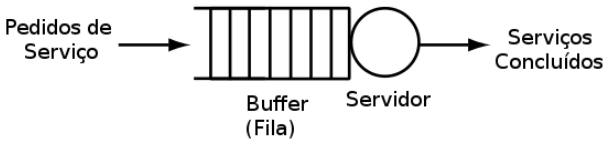


Figura 2.8: Esquema simplificado do sistema de fila única num servidor HTTP.

Na figura 2.9 apresenta-se, de forma simplificada, o funcionamento do servidor HTTP Apache [28].

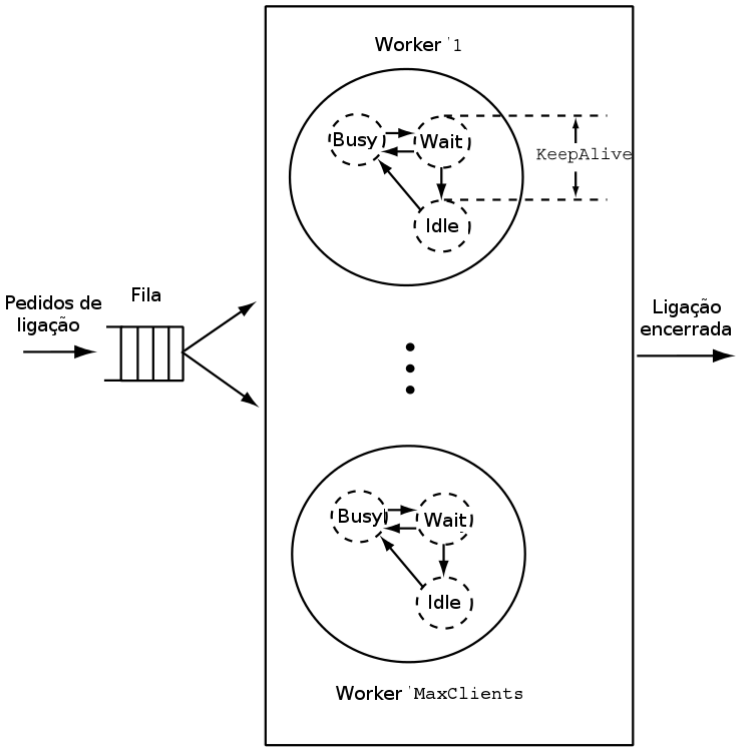


Figura 2.9: Esquema simplificado do funcionamento do servidor HTTP Apache (Detalhe `worker`).

Resumidamente, os pedidos dos clientes são colocados numa fila de espera onde aguardam pelo serviço, o qual basicamente se traduz na resposta aos mesmos. Os pedidos concluídos abandonam o sistema.

O comprimento da fila de espera é limitado existindo, por isso, um tempo limite[1] para um pedido ficar na fila. Caso este tempo seja ultrapassado, o pedido é removido sem ser processado.

[1]Do inglês, *Timeout*

2.3.3 Sobrecarga

Tendo em consideração que existe uma fila com um comprimento fixo no servidor HTTP Apache, existe sempre a possibilidade da mesma ficar cheia. Quando a fila fica cheia, o sistema não tem capacidade para aceitar mais pedidos dos clientes, os quais ficarão assim sem receber a resposta aos seus pedidos. Uma das razões para existir uma fila limitada é evitar a sobrecarga[1] do servidor, já que pretender que todos os pedidos dos clientes, durante uma fase de grande carga, fossem satisfeitos, conduziria a uma fila muito extensa de pedidos, que a serem executados sequencialmente demoraria muito tempo a serem satisfeitos, em particular os mais recentes/últimos.

Outro aspeto a ter em consideração é que os pedidos requerem processos/`workers` para os processar e cada processo/`worker` requer recursos do sistema (e.g. processador e memória). Deste modo, se não fosse imposto um limite à fila de pedidos, os processos/`worker` podiam eventualmente conduzir a um bloqueio do sistema. Convém recordar que o servidor HTTP está a ser executado num equipamento que disponibiliza simultaneamente os seus recursos (CPU, memória RAM, processos, ligações TCP, buffer TCP, buffers de sistema, *etc.*) a vários serviços. A sobrecarga de um sistema pode causar problemas graves, sendo algo que deve ser evitado, sempre que possível.

Existem algumas soluções de modo a minimizar este problema, designadamente:

Fila de espera de tamanho fixo: Embora seja uma das formas de atenuar este problema, não é totalmente satisfatória, pois pode impedir que alguns pedidos importantes sejam atendidos, já que não entram na fila de espera. O peso que cada pedido dos clientes exerce sobre os servidores pode variar consideravelmente. Um pedido pode ser bastante simples e de resposta rápida, enquanto outros podem ser bastante complexos e necessitarem de muitos recursos de sistema. Este facto impede que se possa definir *a piori* uma dimensão fixa ótima para a fila de espera, de modo a que o sistema nunca esteja sobrecarregado.

Fila de espera de tamanho variável: Esta solução resolve algumas das limitações referidas no ponto anterior, mas exige que seja definido um controlo dinâmico eficiente do comprimento da fila de espera. Convém, contudo, salientar que ainda assim, será sempre uma fila de tamanho finito.

***Timeout*:** É igualmente possível definir um tempo limite (*timeout*) para os pedidos que estão na fila, de modo a definir quanto tempo um pedido pode ficar na fila antes de ser removido.

De qualquer foma, controlar apenas o número de pedidos na fila não será uma garantia para evitar uma sobrecarga do servidor. Haverá ainda um limite para os recursos do sistema que precisam ser divididos entre todos os serviços do servidor. Conforme referido anteriormente, podem existir grandes diferenças no uso dos recursos do sistema de acordo com o pedido do

[1] Do inglês, *Overload*

cliente. Por isso, provavelmente também seria interessante ser capaz de controlar o número de processos/`worker` que executam, dependendo da carga.

Outra abordagem no controlo de sobrecarga do servidor HTTP pode passar por adoptar determinadas técnicas de controlo de admissão e de adaptação de conteúdo [37].

2.3.4 `MPM`

O servidor HTTP Apache foi desenhado para ser um servidor potente e flexível, capaz de operar num vasto número de ambientes e plataformas. Plataformas e ambientes diferentes frequentemente necessitam de funcionalidades diferentes, ou podem ter diferentes formas de implementar eficientemente a mesma funcionalidade. O Apache sempre acomodou um vasta variedade de ambientes no seu desenho modular. Este desenho permite ao administrador de sistemas escolher quais as funcionalidades a incluir no servidor e selecionar quais os módulos a carregar em tempo de compilação e em tempo de execução.

O Apache 2.0 alarga este desenho modular a algumas das funcionalidades mais básicas de um servidor HTTP. O servidor já tem à partida uma pré-seleção de módulos `MPM` (*Multi-Processing Modules*) [4], os quais são responsáveis pela ligação às portas de rede de uma máquina, aceitar pedidos e criar processos-filho para tratar de pedidos.

Alargar este desenho modular a este nível do servidor permite dois benefícios importantes:

- O Apache pode suportar de uma forma mais limpa e eficiente uma grande variedade de sistemas operativos, em particular, a versão Windows do Apache é agora muito mais eficiente, já que o módulo `mpm_winnt` pode utilizar as funcionalidades nativas de rede em vez da camada `POSIX` que era utilizada no Apache 1.3. Este benefício também se estende a outros sistemas operativos que implementam `MPM` especializados.
- O servidor pode ser mais adaptado às necessidades de um sítio particular. Por exemplo, sítios que necessitem de lidar com escalabilidade podem escolher entre usar `MPM` com *threads*, como `worker` ou `event`, enquanto sítios que requeiram estabilidade ou compatibilidade podem utilizar a `MPM prefork`.

Os `MPM` devem ser escolhidos durante a fase de configuração e compilação no servidor. Para escolher os `MPM` pretendidos deve ser utilizado o argumento --with-mpm=NOME no *script* de configuração, em que `NOME` é o nome do `MPM` pretendido.

Após o servidor ter sido compilado é possível determinar qual o `MPM` escolhido executando o comando `./httpd -l`. Este comando lista todos os módulos que foram compilados no servidor, incluindo o `MPM`.

Nos sistemas operativos `UNIX`, o `MPM` pré-definido é o `prefork`.

MPM Worker

Este MPM implementa um servidor híbrido "multi-processo"/"multi-*thread*" [7]. Ao utilizar *threads* para servir pedidos, ele é capaz de servir um número elevado de pedidos com menos recursos do que um servidor baseado em processos. No entanto, o facto de ter múltiplos processos disponíveis, cada um com várias *threads* pode, em certos casos, evidenciar uma menor estabilidade do que um servidor baseado em processos.

O parâmetro mais importante usado para controlar este MPM é ThreadsPerChild, o qual controla o número de *threads* disponibilizadas por cada processo-filho e MaxClients, o qual controla o número total máximo de *threads* que pode ser lançado.

Neste MPM, um único processo de controlo, processo-pai, é responsável por lançar processos-filhos. Cada processo-filho cria um número fixo de *threads* de servidor de acordo com o parâmetro ThreadsPerChild e cria uma estrutura designada listener thread, a qual fica à espera de ligações passando-as para as *threads* de servidor para processamento assim que o pedidos chegam.

O servidor HTTP Apache tenta sempre manter uma *pool* de *threads* de servidores de reserva/livres para servir pedidos que cheguem. Deste modo, os clientes não necessitam de esperar que sejam criadas *threads*, ou processos, antes de os seus pedidos serem servidos. O número de processos que são lançados no início são definidos pelo parâmetro StartServers. O Apache durante o seu funcionamento avalia o número total de *threads* inativas em todos os processos e faz *fork* ou encerra processos para manter este número entre os limites especificados por MinSpareThreads e MaxSpareThreads. Tendo em consideração que este processo é auto-regulado raramente é necessário modificar estes parâmetros dos seus valores padrão.

O parâmetro ServerLimit é o limite do número de processos-filho ativos e deve ser maior, ou igual, ao rácio MaxClients/ThreadsPerChild. O parâmetro ThreadLimit é o limite do número de *threads* do servidor e deve ser maior, ou igual, ao parâmetro ThreadsPerChild.

Além do conjunto de processos-filho ativos, podem existir outros processos-filho que estejam a terminar. Estes processos têm ainda, pelo menos, uma *thread* de servidor a tratar uma ligação/pedido de um cliente. É possível existir até o valor configurado em MaxClients de processos a terminar, apesar de ser expectável que o número seja inferior. Este comportamento pode ser evitado desabilitando o encerramento de processos-filhos individuais, o que é conseguido definindo o valor de MaxRequestsPerChild para zero, ou definindo o valor de MaxSpareThreads para o mesmo valor de MaxClients.

Uma configuração típica de `worker MPM` é a seguinte:

```
ServerLimit 16
StartServers 2
MaxClients 150
MinSpareThreads 25
MaxSpareThreads 75
ThreadsPerChild 25
```

`MPM Prefork`

O `MPM prefork` utiliza vários processos-filho com uma *thread* cada [6].

Este `MPM` implementa um servidor HTTP sem *threads* e baseado em *pré-fork*, o qual trata os pedidos de uma forma semelhante ao Apache 1.3, sendo o `MPM` adequado para sítios que necessitem evitar as *threads* por questões de compatibilidade com bibliotecas *non-thread-safe*. É, igualmente, o melhor `MPM` para isolar cada pedido, de modo a que um pedido não afete outro.

Este `MPM` é muito auto-regulador pelo que, geralmente, não é necessário ajustar os seus parâmetros de configuração. A condição mais importante é que o valor do parâmetro `MaxClients` seja elevado para o Apache poder processar o número de pedidos simultâneos expectáveis e razoável de modo a assegurar que exista memória RAM suficiente para todos os processos.

Neste `MPM` um único processo é responsável por lançar os processos-filho, os quais ficam à espera de ligações servindo-as quando chegam. O Apache tenta sempre manter vários processos de reserva/livres, os quais estão disponíveis para servir pedidos. Deste modo, os clientes não necessitam de esperar que seja feito o *fork* de novos processos-filho antes do seus pedidos serem satisfeitos.

Os parâmetros `StartServers`, `MinSpareServers`, `MaxSpareServers` e `MaxClients` regulam a forma como o processo-pai cria os processos-filho. Tal como em outras situações, o servidor HTTP Apache é muito auto-regulador e muitos sites não necessitam de ajustar os parâmetros mencionados anteriormente. Sítios que necessitem de servir simultaneamente mais de 256 pedidos podem necessitar aumentar o valor de `MaxClients`, enquanto sítios com memória limitada podem necessitar baixar o valor de `MaxClients` de modo a manter o servidor sem *thrashing*[1].

Enquanto o processo-pai é habitualmente iniciado pelo utilizador `root`, de modo a estabelecer ligação ao porto 80 (porto privilegado), os processos-filho são lançados pelo utilizador `apache` (ou `http`), o qual tem menos privilégios do que o utilizador `root`. Os parâmetros `User` e `Group` são utilizados para definir o utilizador e grupo dos processos-filho do servidor HTTP

[1] *Swap* de memória para disco rígido e o processo inverso.

Apache. Um aspeto bastante importante é que os processos-filho devem ter permissão para ler todo o conteúdo a disponilizar nos pedidos mas, devem ter o menor número de privilégios possível.

O parâmetro `MaxRequestsPerChild` controla a frequência com que o servidor recicla os processos, eliminado os processos antigos e criando novos processos.

`MPM worker ou prefork?`

Tendo em consideração os aspetos e diferenças referidas anteriormente para os `MPM worker` e `MPM prefork`, qual escolher?

Os processos, ao contrário das *threads*, não partilham memória ou recursos diretamente. Deste modo, os processos consomem mais recursos do sistema do que as *threads* e sobrecarregam mais o servidor. Alternar entre processos, também designado como *context switch*, é habitualmente uma tarefa mais pesada que alternar entre *threads*.

No caso de `MPM` "multi-processo"/"multi-*thread*"(ex.: `worker`) o maior problema é que alguns módulos externos[1] (ex. PHP), que muitos sítios necessitam incluir, não conseguem executar de uma forma fiável (*thread-safe*) no Linux/Unix.

A estabilidade e desempenho são dois dos aspetos mais importantes no que concerne a um sítio Internet e, neste momento nenhum dos outros `MPM` pode competir com o `prefork` nestes aspetos. No caso do `MPM prefork`, quando ocorre um erro com um pedido, o problema é encapsulado num processo e não afeta os outros pedidos, o que pode não ser o caso quando se utiliza um servidor "multi-processo"/"multi-*thread*".

Um dos motivos da escolha do `MPM` *prefork* neste trabalho está relacionado com o objetivo de se pretender utilizar o PHP. Para melhor perceber a dispersão do PHP apresenta-se na figura 2.10 as linguagens de programação, *server-side*, mais utilizadas nos sítios Internet [49].

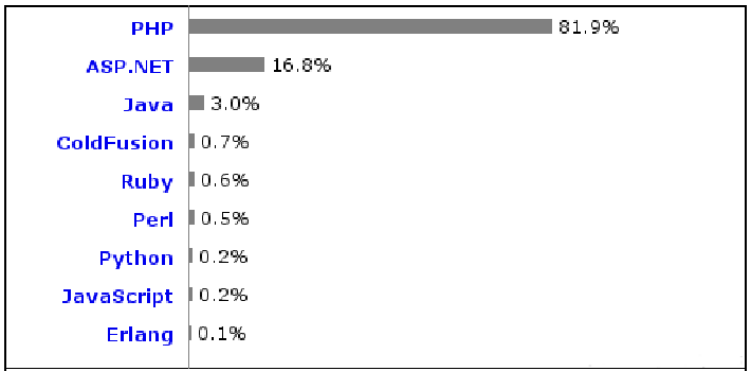


Figura 2.10: Estatísticas do sítio W3Techs. Quotas das linguagens de programação *server-side* mais utilizadas nos sítios Internet.

[1] Do inglês, *Third-party*

Conforme se pode constatar, o PHP é utilizado em cerca de 81% dos sítios Internet, o que justifica a escolha do `MPM prefork` neste trabalho bem como, de um modo geral, em ambientes de produção dos vários sítios Internet.

2.3.5 `Scoreboard`

O processo-pai necessita de um mecanismo de comunicar e saber o estado de todos os processos-filho. Este mecanismo é levado a cabo através de um estrutura denominada `Scoreboard`. O `Scoreboard` é uma estrutura de dados simples (*struct*, como é denominado em C), armazenando numa área partilhada, habitualmente memória partilhada, mas que pode ser um ficheiro. Esta estrutura pode ser acedida tanto pelo processo pai, como pelo processo filho.

O `Scoreboard` consiste nas seguintes três áreas:

global : Uma estrutura que contém os parâmetros globais que o processo-pai e os processos-filho podem aceder, exemplo: `generation id` (usado no *graceful restart*) e o parâmetro `ServerLimit`;

parent : Uma estrutura que contém vários parâmetros, usados principalmente pelo processo-pai, exemplo `process ID (PID)` de cada processo-filho;

servers : Uma tabela que contém várias informações sobre os processos-filho, tais como o estado do processo - `status`, (ex.: `dead`, `ready`, `starting`), o tempo de início (`start_time`) e o tempo de finalização (`stop_time`) de um pedido, quantos bytes foram servidos num pedido *etc.* O `Scoreboard` tem funções para atualizar estes parâmetros. Quando um processo-filho é criado, ele obtém um `id` único que é utilizado como índice nesta tabela. Ao utilizar este `id` o processo-pai consegue obter informação sobre um processo-filho específico. No código-fonte, a instância do `Scoreboard` é denominado `ap_scoreboard_image`. Este componente foi utilizado nesta tese para obter a informação necessária para o sistema de QoS, nomeadamente o número de processos-filho a processar pedidos do sítio `Premium` e do sítio `Outros`.

No quadro 2.2 apresenta-se a lista de estados em que os "servidores" podem estar. Estes estados estão definidos no ficheiro `include/scoreboard.h`:

Quando o processo-pai entra no seu ciclo[1] principal fica exclusivamente a aguardar que um processo-filho esteja no estado `dead`) ou, que o tempo de espera seja superior a `timeout` + 1 *segundo*. É através da função `perform_idle_server_maintenance` que o processo-pai controla a sua *pool* de processos-filho e determina quantos filhos deve iniciar, ou encerrar ("matar"). Esta função ao arrancar inicia um ciclo que percorre todos os processos-filho já presentes no `Scoreboard` e examina o seu estado. Cada processo com o estado `dead`

[1] Do inglês, *Loop*

Código - Estado	Descrição do estado
0 - `SERVER_DEAD`	"Morto"
1 - `SERVER_STARTING`	A arrancar
2 - `SERVER_READY`	A aguardar por um ligação (*accept() lock*)
3 - `SERVER_BUSY_READ`	A ler o pedido de um cliente
4 - `SERVER_BUSY_WRITE`	A processar o pedido de um cliente
5 - `SERVER_BUSY_KEEPALIVE`	A aguardar mais pedidos do mesmo cliente via `keepalive`
6 - `SERVER_BUSY_LOG`	A registar o pedido de um cliente
7 - `SERVER_BUSY_DNS`	A procurar o *hostname*
8 - `SERVER_CLOSING`	A fechar uma ligação
9 - `SERVER_GRACEFUL`	A encerrar graciosamente um pedido
10 - `SERVER_IDLE_KILL`	A encerrar clientes inativos (estado `idle`)
11 - `SERVER_NUM_STATUS`	Número de estados

Quadro 2.2: Servidor HTTP Apache - Lista de estados.

terá o seu *slot* marcado como "livre" no `Scoreboard`. Assim que o número de *slots* livres é igual ao número de filhos que pretende criar, o ciclo é interrompido. Se não existirem *slots* antigos suficientes com o estado `dead`, a função incrementará o número de *slots* em uso até serem suficientes, ou até que o número de *slots* em uso ser inferior ao valor de `MaxClients`. Quando a função faz o ciclo através dos processos-filho, ela também regista quantos servidores estão no estado `idle`. Se o número de servidores no estado `idle` excede o parâmetro `MaxSpareServers`, um desse processos-filho será terminado. Se o número de servidores no estado `idle` é inferior ao parâmetro `MinSpareServers` a criação de processos é ativada e da próxima vez que a função for executada esses processos extra serão criados.

2.3.6 *Pool* de memória

Para evitar problemas de fugas de memória[1] o Apache disponibiliza *pools* de recursos. Cada recurso como, memória, ficheiros abertos, *etc.* será ligado a uma *pool*. Os recursos são automaticamente libertados quando o servidor termina a *pool*. Por exemplo, cada pedido tem a sua própria *pool* para conter os recursos. Logo que o servidor termine de responder aos pedidos, todos os recursos ligados à sua *pool* são libertados.

2.3.7 Parâmetros de configuração

Em seguida explicam-se alguns dos conceitos/parâmetros mais importantes do servidor HTTP Apache e que mereceram especial atenção nesta dissertação.

[1] Do inglês, *Memory leak*

`Timeout`

O parâmetro `Timeout` define o tempo que o Apache deve esperar por operações de Entrada/Saída (E/S)[1] em determinadas circunstâncias [3]:

- Quando o servidor recebe dados do cliente: é o tempo de espera pela chegada de um pacote TCP, se o *buffer* de receção está vazio;
- Quando envia dados para o cliente: é o tempo de espera pela confirmação[2] de um pacote, se o *buffer* de envio está cheio;
- No `mod_cgi`: é o tempo de espera pelo *output* do *script* de CGI (*Computer Graphics Interface*);
- No `mod_ext_filter`: é o tempo de espera pelo *output* do processo de filtragem;
- No `mod_proxy`: é valor pré-definido de `timeout` se o parâmetro `ProxyTimeout` não está configurado.

`KeepAlive`

A extensão `Keep-Alive` ao HTTP/1.0 adiciona a funcionalidade de ligações persistentes. As sessões HTTP/1.1 têm maior tempo de vida, o que permite que vários pedidos sejam enviados na mesma ligação TCP [3]. Em alguns casos, foi demonstrado que pode resultar numa redução de quase 50% de latência de documentos HTML com muitas imagens.

Para clientes HTTP/1.0, as ligações *Keep-Alive* só serão usadas se forem especificamente solicitadas pelo cliente. Adicionalmente, uma ligação *Keep-Alive* com cliente HTTP/1.0 só pode ser utilizada quando o tamanho do conteúdo é conhecido previamente. Este facto implica que o conteúdo dinâmico tal como, CGI, páginas SSI (*Server Side Includes*), e listagens geradas pelo servidor geralmente não utilizam ligações *Keep-Alive* com clientes HTTP/1.0. Para clientes HTTP/1.1, as ligações persistentes são utilizadas por padrão, a não ser que seja dada indicação em contrário.

Quando um cliente utiliza uma ligação *Keep-Alive*, a mesma será contabilizada como um único pedido para o parâmetro `MaxRequestsPerChild`, independentemente do número de pedidos que são enviados na ligação.

`MaxRequestsPerChild`

O parâmetro `MaxRequestsPerChild` define o número limite de pedidos que um processo-filho irá processar [5]. Se o número de pedidos exceder o definido, o processo é encerrado. Se este parâmetro estiver definido como zero, então o processo nunca expirará. Definir o parâmetro para um valor diferente de zero limita a memória que um processo pode consumir por uma eventual fuga de memória[3].

[1] Do inglês, *Input/Ouput* (I/O)
[2] Do inglês, *Acknowledgement*
[3] Do inglês, *Memory leak*

`MaxKeepAliveRequests`

O parâmetro `MaxKeepAliveRequests` limita o número de pedidos permitidos, por ligação, quando a opção *KeepAlive* está ativa [3]. Se este parâmetro estiver definido como zero, o número de pedidos permitido será ilimitado. Na documentação Apache é recomendado definir um valor elevado para um melhor desempenho do servidor, exemplo: "`MaxKeepAliveRequests 500`".

`KeepAliveTimeout`

Este parâmetro define o número de segundos que o Apache aguardará por pedidos subsequentes, antes de encerrar a ligação [3]. Após ter chegado um pedido, é tido em conta o valor definido no parâmetro `Timeout`. Definir o parâmetro `KeepAliveTimeout` para um valor alto pode causar problemas de desempenho em servidores sobrecarregados. Quanto maior for o valor, mais processos do servidor estarão ocupados aguardando ligações de clientes no estado `idle`.

`StartServers`

Define o número de processos-filho a criar no arranque do Apache [5]. Como o número de processos é controlado dinamicamente, dependendo da carga, não existem geralmente muitas situações em seja necessário ajustar este parâmetro. Deve ser salientado que, neste trabalho, o valor definido para o parâmetro `StartServers` é igual aos parâmetros `MaxClients` e `ServerLimit`. Tal opção tem três motivos:

- O servidor instalado é utilizado exclusivamente para o servidor Apache;
- O servidor tem capacidade para alocar, logo no início, os recursos necessários;
- O servidor deve estar a funcionar à sua máxima capacidade desde o início.

`ServerLimit`

No caso do `MPM prefork`, este parâmetro define o valor máximo configurado para `MaxClients` durante toda a vida do processo Apache [5]. Para o `MPM worker`, este parâmetro combinado com o parâmetro `ThreadLimit` define o valor máximo configurado para o `MaxClients` durante toda a vida do Apache. O parâmetro `MaxClients` pode ser alterado, mas só produzirá efeito após reiniciar o servidor. Com o `MPM prefork`, este parâmetro só deve ser utilizada se for necessário definir o parâmetro `MaxClients` para um valor superior a 256 (valor prédefinido).

Deve ser atribuída especial atenção a este parâmetro. Se estiver definido com um valor muito superior ao necessário, será alocada muita memória partilhada. Se os parâmetros `ServerLimit` e `MaxClients` forem definidos para valores muito elevados, o sistema pode não ter recursos para os suportar. Nestas situações, o Apache pode não conseguir iniciar ou, mesmo que inicie, todo o sistema pode ser muito instável.

`MaxClients`

O parâmetro `MaxClients` define o limite do número de pedidos simultâneos que são satisfeitos [5]. Qualquer pedido/ligação após este limite entrará para uma fila de espera, até ao limite definido pelo parâmetro `ListenBacklog`.

Para sistemas sem *threads*, ex.: `MPM prefork`, o parâmetro `MaxClients` define o número máximo de processos-filho que serão lançados para servir os pedidos. Tal como referido anteriormente, o valor pré-definido é 256. Se for necessário aumentar este valor, também deve ser aumentado o valor do parâmetro `ServerLimit`.

Para sistemas híbridos, ex.: `MPM worker`, o parâmetro `MaxClients` restringe o número total de *threads* que estarão disponíveis para servir os clientes. Para `MPM` híbridos, o valor pré-definido é 400, o qual resulta da multiplicação dos valores dos parâmetro `ServerLimit` (16) e `ThreadsPerChild` (25). Deste modo, para aumentar o valor de `MaxClients` para um valor que exija mais do que 16 processos, também é necessário aumentar o valor de `ServerLimit`.

`MinSpareServers`

O parâmetro `MinSpareServers` define o número mínimo desejável de processos-filho que estão no estado `idle` [6]. Um processo no estado `idle` é aquele que não está a tratar de nenhum pedido. Se existir um número inferior de `MinSpareServers` no estado `idle`, o processo-pai cria um novo processo-filho à taxa de um por segundo. Otimizar este parâmetro só deve ser necessário em sites com muita carga.

`MaxSpareServers`

O parâmetro `MaxSpareServers` define o número máximo desejável de processos-filho que estão no estado `idle` [6]. Se existir um número superior a este valor, então o processo-pai terminará o número de processos em excesso.

Tal como no parâmetro `MinSpareServers`, alterar este valor só deve ser necessário em sítios com carga elevada. Se este parâmetro for definido para um valor igual, ou inferior, a `MinSpareServers`, então o processo-pai definirá automaticamente este valor para `MinSpareServers` + 1.

2.3.8 *Virtual host*

O termo *Virtual Host* designa a prática de disponibilizar mais do que um sítio Internet numa única máquina, por exemplo: www.sitio1.com e www.sitio2.com. Os *Virtual Host* podem

ser baseados em IP[1], o que significa que cada sítio terá um endereço IP diferente ou, baseado no nome[2], o que significa que vários nomes podem estar associados ao mesmo endereço IP. O facto de vários sítios estarem a ser executados na mesma máquina física não é percetível para o utilizador final [11].

Virtual Host baseados em IP

Como o próprio termo indica, o servidor deve possuir vários IP diferentes, um por cada *virtual host*, já que deve existir uma combinação IP/porta única diferente para cada *virtual host*. Este objetivo pode ser conseguido tendo a máquina várias interfaces de rede ou, utilizando o que é designado de "*IP aliases*", sendo esta última forma suportada pela maioria dos sistemas operativos recentes.

Existem duas formas de configurar o Apache para suportar vários *virtual host*. Uma consiste em executar um demónio[3] HTTPD para cada *virtual host*, outra consiste em executar um único demónio, o qual suporta todos os *virtual host*.

Devem/podem ser utilizados vários demónios, nas seguintes situações:

- Por questões de segurança e privacidade. Neste caso, pretende-se isolar o tráfego entre os sítios, de modo a que não seja possível alguém da "empresa 1" ler os dados da "empresa 2", a não ser via sítio Internet. Neste caso, são necessários dois demónios, sendo cada um executado com diferentes definições nos parâmetros `User`, `Group`, `Listen` e `ServerRoot`;
- Existem recursos suficientes para disponibilizar memória e descritores de ficheiros distintos a cada *alias* existente na máquina física.

Devem/podem ser utilizados demónios únicos, nas seguintes situações:

- É aceitável partilhar a configuração HTTP entre vários *virtual host*;
- A máquina física serve um número elevado de pedidos e a execução em demónios separados pode representar uma quebra significativa de desempenho.

Virtual Host baseados em nome

Os *virtual host* baseados no nome são geralmente mais simples já que só é necessário configurar o servidor de DNS (*Domain Name System*) para resolver cada nome para o respetivo endereço IP e configurar o servidor HTTP Apache para reconhecer nomes diferentes. Utilizar *virtual host* baseados no nome também facilita as exigências do número de IP públicos, os quais são um recurso escasso. Neste último aspeto, convém referir que é possível utilizar um *reverse proxy* (ex.: nginx[4]) existindo, contudo, vários fatores importantes a ter em conta, tais como, por exemplo, o correto funcionamento dos sítios e sua compatibilidade.

[1]Do inglês, *Apache IP-based Virtual Host Support*
[2]Do inglês, *Apache Name-based Virtual Host Support*
[3]Do inglês, *Daemon*
[4]`http://nginx.org/`

2.3.9 `Apache Portable Runtime`

A `Apache Portable Runtime` (APR) é uma biblioteca de suporte disponível para o servidor HTTP Apache, sendo responsável por "mapear funções" do sistema operativo. Originalmente fazia parte do servidor HTTP Apache, mas foi promovida a projeto independente pela *Apache Software Foundation* [9].

A missão da APR é criar e manter uma interface previsível e consistente com as várias plataformas, designadamente sistemas operativos. Com esta biblioteca os programadores podem desenvolver código e estarem relativamente seguros que o mesmo terá um comportamento previsível e, eventualmente até igual, independentemente da plataforma onde o software seja desenvolvido/executado. Não é necessário codificar para condições específicas de modo a que o código funcione ou, seja possível tirar partido de determinadas funcionalidades.

A APR é atualmente utilizada por outras aplicações que necessitam de independência de plataforma [10]. Exemplo de projetos *Open Source* que utilizam a APR:

- Servidor HTTP Apache (`http://httpd.apache.org/`)
- Flood(`http://httpd.apache.org/test/flood/`)
- FreeSwitch (`http://www.freeswitch.org/`)
- mod_jk v2 e mod_webapp (parte do Tomcat - `http://jakarta.apache.org/tomcat/`)
- Subversion (`http://subversion.tigris.org/`)
- serf (`http://code.google.com/p/serf/`)
- ActiveMQ CPP (`http://activemq.apache.org/cms/`)
- OpenAMQ (`http://www.openamq.org/`)
- managelogs (`http://managelogs.tekwire.net/`)

Como exemplo de "projeto comercial", que utiliza a APR, podemos mencionar o W2ML (The Web 2 Markup Language - `http://w2ml.com/`).

2.4 Teoria de controlo

Nas redes informáticas o controlo de tráfego é um assunto de gestão essencial e um desafio permanente para os engenheiros de redes de comunicação e de informação [13].

Existem basicamente dois tipos de sistemas de controlo: Sistemas de controlo em cadeia fechada[1] e Sistemas de controlo em cadeia aberta[2].

[1] Do inglês, *Closed loop ou feedback*
[2] Do inglês, *Open Loop ou feedforward*

2.4.1 Sistemas de controlo em cadeia fechada

A ideia principal dos sistemas de controlo em cadeia fechada é utilizar as medições dos dados de saída[1] tais como, por exemplo, o tempo de resposta e o débito[2], para determinar quais as entradas[3] a fornecer ao sistema, de modo a ir ao encontro dos objetivos previamente definidos. Isto é conseguido ajustando as entradas do sistema, tais como os parâmetros que afetam o tamanho da fila, as políticas de agendamento e níveis de concorrência.

Tendo em consideração que os dados de saída medidos são utilizados para determinar as entradas e, as entradas afetam depois as saídas, a arquitetura é designada por controlo em cadeia fechada ou *feedback*. É o fluxo circular de informação que motiva esta designação. Praticamente qualquer sistema que seja considerado automático tem algum elemento de controlo em cadeia fechada [28].

O objetivo do controlo em cadeia fechada é regular as características de um sistema [28]. Hoje em dia já existem, no uso comum, vários exemplos de sistemas de controlo em cadeia fechada tais como, por exemplo, o *cruise control* de um automóvel e o termóstato numa casa. Os conceitos de teoria de controlo em sistemas em cadeia fechada também se aplicam a sistemas computacionais. Uma das áreas estudadas é a dos servidores HTTP. Nestes sistemas, a saída medida depende tipicamente na natureza dos pedidos servidos, ou da carga [28]. A carga é frequentemente caracterizada em termos da chegada de pedidos e da distribuição dos tempos de serviço dos recursos usados, exemplo: CPU, memória, *locks* de bases de dados. Nos estudos realizados por [28] com o servidor HTTP Apache, a utilização do CPU depende da carga e do controlo de entrada.

A figura 2.11 apresenta um exemplo de um sistema de controlo em cadeia fechada SISO (Single Input, Single Output) [28]. Em sistemas computacionais é mais frequente trabalhar-se com sistemas de controlo em cadeia fechada MIMO (Multiple Input, Multiple Output), os quais têm várias entradas (parâmetros de configuração) e várias saídas (tempos de resposta, débito por classe de serviço, etc).

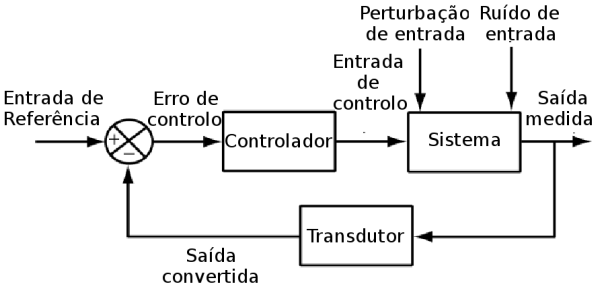


Figura 2.11: Sistema de controlo em cadeia fechada. Diagrama em bloco.

[1]Do inglês, *Output*
[2]Do inglês, *Throughput*
[3]Do inglês, *Input*

Tal como representado na figura 2.11, os elementos essenciais de um sistema de controlo fechado são os seguintes [28]:

Erro de controlo [1]: É a diferença entre o valor de referência de entrada e o valor de saída medido;

Entrada de controlo [2]: Parâmetro que afeta o comportamento do sistema alvo e pode ser ajustado dinamicamente (exemplo: O parâmetro `MaxClients` no servidor HTTP Apache);

Controlador [3]: Determina a definição da entrada e o controlo necessário para obter a entrada de referência. O controlador obtém valores de controlo de entrada baseados nos valores actuais e anteriores do erro;

Perturbação de entrada [4]: É qualquer alteração que afete a entrada de controlo e influencie os valores de saída medidos. Um exemplo de perturbação/interferência de entrada no servidor HTTP Apache é a execução de outras tarefas, designadas "tarefas administrativas", tais como cópias de segurança[5] e verificações de vírus, as quais afetam a relação entre a entrada de controlo (ex.: `MaxClients`) e o saídas medidas (ex.: Utilização de CPU e tempos de resposta);

Saída medida [6]: Característica mensurável do sistema, tal como a utilização de CPU e/ou tempo de resposta;

Ruído de entrada [7] Qualquer efeito que altere os valores de saída medidos produzidos pelo sistema. Também é designado como "ruído do sensor" ou "ruído de medição";

Entrada de referência [8]: É o valor desejado do output medido (ou transformações do mesmo), tal como CPU utilizado (exemplo 66%). Algumas vezes, a entrada de referência é designado como "saída desejada" ou *setpoint*;

Sistema [9]: Sistema computacional que se pretende controlar;

Transdutor [10]: Converte os valores de saída medidos de modo a ser comparado com a entrada de referência. Neste trabalho, o transdutor converte a percentagem medida de processos do sítio *Premium* e do sítio *Outros* em atraso a aplicar aos pacotes destinados ao sítio *Premium* e ao sítio *Outros*, respetivamente.

O servidor HTTP é um sistema MIMO [28]. Vários estudos [1, 25, 41, 50] fornecem exemplos da aplicação de sistemas em cadeia fechada lineares ao servidor HTTP Apache (exemplo figura 2.12). Esquemas mais elaborados incluem sistemas de controlo não-linear [46].

[1] Do inglês, *Control Error*
[2] Do inglês, *Control input*
[3] Do inglês, *Controller*
[4] Do inglês, *Disturbance input*
[5] Do inglês, *Backup*
[6] Do inglês, *Measured input*
[7] Do inglês, *Noise input*
[8] Do inglês, *Reference input*
[9] Do inglês, *Target sistem*
[10] Do inglês, *Transducer*

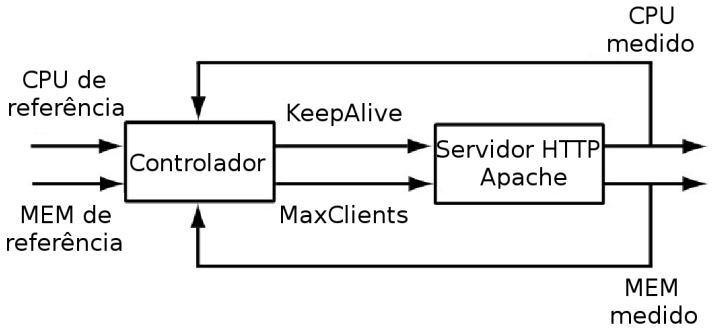


Figura 2.12: Sistema de controlo em cadeia fechada aplicado ao servidor HTTP Apache. Diagrama em bloco.

O processo de desenho de um sistema de controlo geralmente envolve vários passos. Um cenário típico é o seguinte: [26]

1. Estudar o sistema a ser controlado;
2. Decidir quais os tipos de sensores e atuadores a serem usados e respetiva localização;
3. Modelar o sistema a ser controlado;
4. Simplificar o modelo, se necessário, de modo a ser exequível;
5. Analisar o modelo resultante e determinar as suas propriedades;
6. Decidir as especificações de desempenho;
7. Decidir o tipo de controlador a utilizar;
8. Desenhar o controlador de modo a ir ao encontro das especificações, se possível. Caso contrário, modificar as especificações ou, generalizar o tipo de controlador pretendido;
9. Simular o sistema controlado resultante, seja num computador, seja num protótipo;
10. Repetir o passo 1 e 2, se necessário;
11. Escolher o *hardware* e o *software* e implementar o controlador;
12. Otimizar o controlador, se necessário.

2.4.2 Sistemas de controlo em cadeia aberta

Apesar dos sistemas de controlo em cadeia aberta serem apelativos, o seu uso exige, entre outros aspetos, medições em tempo real e um desenho cuidadoso para assegurar que o sistema tem as propriedades desejáveis, especialmente: estabilidade, precisão, curto tempo de estabilização e um pequeno deslize relativamente ao objetivo. Uma alternativa ao sistemas de controlo em cadeia fechada são os sistemas de controlo em cadeia aberta, os quais não utilizam as saídas medidas para ajustar o controlo de entrada. O controlo em cadeia aberta é por vezes designado como *feedforward control*. Na figura 2.13 apresenta-se um diagrama em bloco de um sistema de controlo em cadeia aberta [28]. Este sistema utiliza o controlo de

referência e algumas vezes a perturbação de entrada, para determinar o controlo de entrada necessário para atingir as saídas pretendidas. As saídas medidas não são utilizadas. Um esquema de controlo em cadeia aberta no qual a entrada de controlo é uma função determinística da referência de entrada (e/ou da perturbação de entrada) é estável se o sistema alvo é estável [28]. No entanto, construir um sistema deste tipo implica ter um modelo preciso do sistema alvo/destino, a partir do qual a definição do controlo de entrada é determinado. Este modelo pode ser elaborado a partir da análise dos resultados de várias experiências. O modelo deve ser robusto a alterações do sistema e seu ambiente de funcionamento, incluindo perturbações tais como a execução de tarefas administrativas.

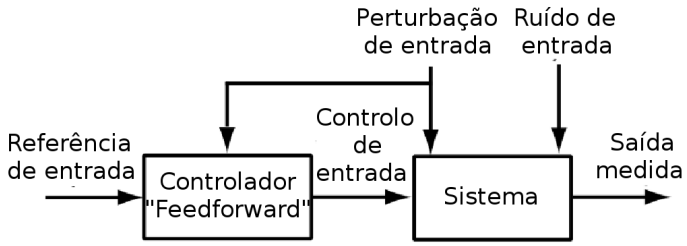


Figura 2.13: Sistema de controlo em cadeia aberta. Diagrama em bloco.

Na figura 2.13 é possível constatar que contrariamente ao sistema de controlo em cadeia fechada, as saídas medidas não são utilizadas pelo controlador para determinar as entradas de controlo necessárias para atingir as entradas de referência.

2.4.3 Sistemas de controlo em cadeia fechada *vs.* sistemas de controlo em cadeia aberta

Para ilustrar este aspeto, vamos considerar o servidor HTTP Apache.

O administrador do sistema pode experimentar vários valores do parâmetro `MaxClients` e encontrar um que permite atingir o objetivo pretendido (ex.: utilização do CPU a 66%). No entanto, se as cargas variarem ao longo do dia, ou semana, o valor do parâmetro `MaxClients` necessita de ser ajustado para manter a desejada utilização de CPU. Contudo e ao contrário dos sistemas de controlo em cadeia fechada, os sistemas de controlo em cadeia aberta não conseguem compensar perturbações ou ruído.

De igual modo, os sistemas de controlo em cadeia fechada não necessitam de modelos muito precisos, algo que é muito difícil obter na prática.

O quadro 2.3 apresenta, de forma resumida, a comparação entre sistemas de controlo em cadeia aberta e sistemas de controlo em cadeia fechada [28].

	Sistema de controlo em cadeia aberta	Sistema de controlo em cadeia fechada
É necessário usar os valores de saída	Não	Sim
Pode tornar um sistema estável num sistema instável	Não	Sim
Modelo de sistema simples	Não	Sim
Adapta-se às perturbações	Não	Sim

Quadro 2.3: Comparação entre sistemas de controlo em cadeia aberta e sistemas de controlo em cadeia fechada.

Apesar dos modelos de sistemas de controlo em cadeia aberta serem atrativos pelo facto da complexidade do seu desenho ser inferior e assegurar a estabilidade do sistema, raramente são utilizados na prática. Isto porque não conseguem adaptar-se às mudanças e é quase impossível obter modelos muito precisos dos sistemas.

Tendo em consideração a aplicabilidade dos sistemas de controlo em cadeia fechada convém referir que é extremamente importante implementar controladores que sejam estáveis, minimizem os erros, tenham um baixo tempo de estabilização e não ultrapassem largamente o objetivo[1].

2.4.4 Controladores

Um dos pontos nevrálgicos de um sistema de controlo em cadeia fechada é o controlador [44]. Nesta matéria, os controladores **PID** (***P**roporcional*, ***I**ntegral*, ***D**iferencial*) assumem papel de destaque, sendo um dos algoritmos de controlo mais comuns. Estes controladores são desenhados com determinados propósitos/objetivos de controlo. Os objetivos mais comuns são [28]:

Controlo regulador ou "Erro nulo(0)" [2]: Assegura que a saída medida é igual, ou próxima, da entrada de referência. Por exemplo, a utilização do servidor HTTP deve ser mantido a 66%. Neste caso, a atenção centra-se nas alterações da entrada de referência, tal como alterar o objetivo de 66% para 75%, caso um quarto servidor esteja disponível;

Rejeição de perturbações [3]: Assegura que as perturbações que atuam sobre o sistema não afetam significativamente as saídas medidas. Por exemplo, quando ocorre uma cópia de segurança, ou uma verificação de antivírus num servidor HTTP, a utilização do sistema é mantido a 66%. Este critério difere do controlo regulador no sentido em que o foco está nas alterações das perturbações na entrada e não na referência de entrada [28].

[1] Do inglês, *Overshoot*
[2] Do inglês, *Regulatory control*
[3] Do inglês, *Disturbance rejection*

Otimização do sistema : Consiste em obter o "melhor valor" de saída medida. Por exemplo, ajustar o parâmetro `MaxClients` no servidor HTTP Apache de modo a minimizar os tempos de resposta.

Outro objetivo importante a salientar é o denominado "**Seguimento de referência**"[1]. Nesta situação, o objetivo varia ao longo do tempo e o sistema de controlo deve ser capaz de se adaptar rapidamente e com precisão às diferentes entrada de referência.

A designação de controlador PID advém dos seus três componentes/termos, a saber: **P**: Termo Proporcional, **I**: Termo Integral e **D**: Termo Derivativo.

Controlador Proporcional (P)

O controlador proporcional é o controlo em cadeia fechada mais simples de implementar e é, talvez, o controlador mais comum dentro dos sistemas de controlo em cadeia fechada. Um controlador proporcional consiste simplesmente no cálculo do produto entre o erro e uma constante (contante proporcional, K_p). Este valor é depois injetado no atuador. O termo proporcional (P) é calculado da seguinte forma:

$P_{out} = k_p e(t)$, em que:

P_{out}: Termo proporcional de saída;
k_p: Ganho/Constante proporcional; parâmetro de otimização;
e: Erro = objetivo - valor atual medido;
t: Tempo ou tempo instantâneo (presente).

A figura 2.14 mostra o que sucede quando se adiciona um controlador proporcional a um sistema semelhante ao *cruise-control* de um carro [51].

Para ganhos pequenos (K_p=1) o sistema atinge o objetivo pretendido, mas muito lentamente. Aumentar o ganho (K_p=2) aumenta a velocidade com que se atinge o objetivo. Com valores superiores (K_p=5, K_p=10) o sistema ultrapassa o valor pretendido, oscila e leva praticamente o mesmo tempo a atingir o objetivo que com ganhos inferiores (K_p=2). Apresenta, no entanto, mais situações em que o objetivo é ultrapassado. Caso se continuasse a aumentar o ganho, provavelmente o sistema oscilaria ainda mais em torno do objetivo e nunca estabilizaria.

[1] Do inglês, *Time-varying reference*

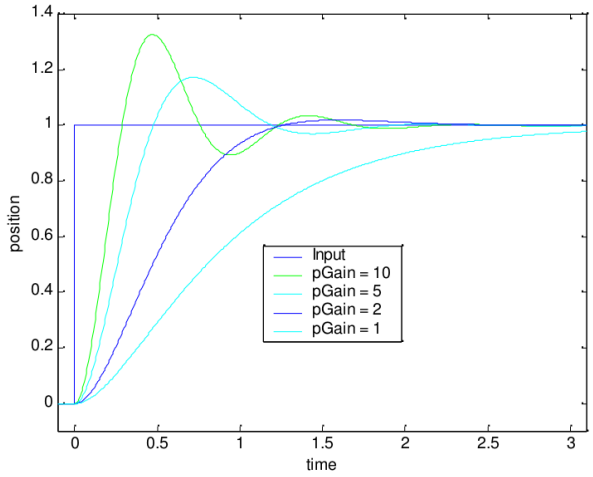


Figura 2.14: Controlador Proporcional com vários valores da constante proporcional (k_p).

Controlador Integral (I)

O controlo integral é utilizado para adicionar precisão a "longo-prazo" a um controlo em cadeia fechada. É quase sempre utilizado em conjunto com um controlador proporcional.

O termo integral (I) é calculado da seguinte forma:

$I_{out} = k_i \int_0^t e(\tau)d\tau$, em que:

I_{out}: Termo integral de saída;
k_i: Ganho/Constante integral; parâmetro de otimização;
t: Tempo ou tempo instantâneo (presente)
e: Erro = objetivo - valor atual medido;
τ: Variável *dummy* de integração.

Um controlador integral, por si só, reduz a estabilidade de um sistema, ou até a destrói [51]. A figura 2.15 mostra um controlador integral puro ($P_{out} = 0$) num sistema semelhante ao mencionado no controlador proporcional. É possível constatar que o sistema não estabiliza. Tal como sucedeu no controlador proporcional, para ganhos elevados, o sistema com o controlador integral isolado oscila cada vez com maior amplitude, até que eventualmente atinja um determinado limite.

A figura 2.16 mostra o mesmo sistema com um controlador proporcional-integral [51].

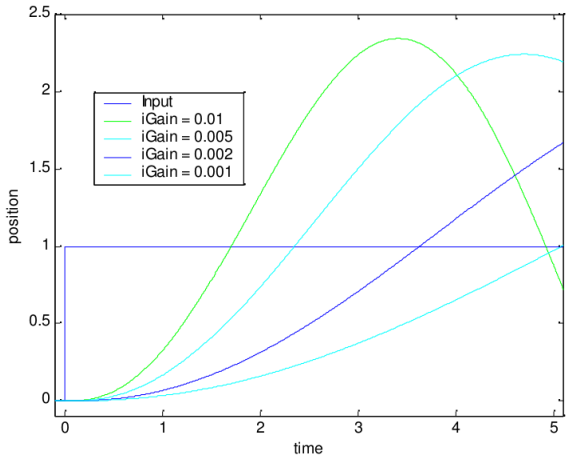


Figura 2.15: Controlador Integral com vários valores da constante integral (k_i).

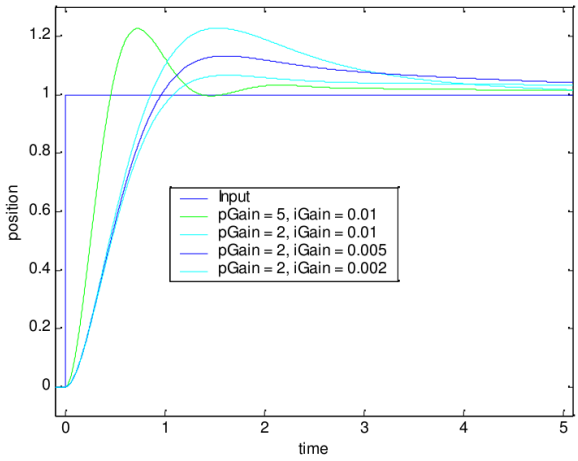


Figura 2.16: Controlador Proporcional-Integral com vários valores da constante proporcional (k_p) e da constante integral (k_i)

Controlador Derivativo (D)

De um modo geral, se um sistema não consegue estabilizar com um controlador proporcional, também não estabiliza com um controlador proporcional-integral. O controlador proporcional está relacionado com o comportamento atual do sistema e o controlador integral com o comportamento passado (histórico) do sistema. Se existisse algum elemento que estimasse o comportamento futuro, então ele poderia ser utilizado para estabilizar o sistema. É esse o objetivo do termo derivativo.

O termo derivativo (D) é calculado da seguinte forma:

$D_{out} = k_d \frac{d}{dt} e(t)$

D_{out}: Termo diferencial de saída;
k_d: Ganho/Constante diferencial; parâmetro de otimização;
e: Erro = objetivo - valor atual medido;
t: Tempo ou tempo instantâneo (presente).

O controlador derivativo é muito poderoso, mas é igualmente o mais problemático [51]. Os três problemas que podem existir são: irregularidade de amostragem, ruído e grandes oscilações de frequência.

Controlador Proporcional-Integral-Derivativo (PID)

Neste controlador, os termos proporcional, integral e diferencial são somados para obter a saída do controlador PID. De uma forma simples, podemos apresentar o controlador PID como:

$$V(t) = P_{out} + I_{out} + D_{out}$$

Definindo $u(t)$ como a saída do controlador, a forma final do controlo PID é:

$$u(t) = V(t) = k_p e(t) + k_i \int_0^t e(\tau)d\tau + k_d \frac{d}{dt} e(t)$$

A figura 2.17 mostra um sistema de aquecimento com um controlador PID [51]. Neste caso é possível verificar que o sistema tem as características desejadas, isto é: não oscila, é rápido a estabilizar e mantém-se estável.

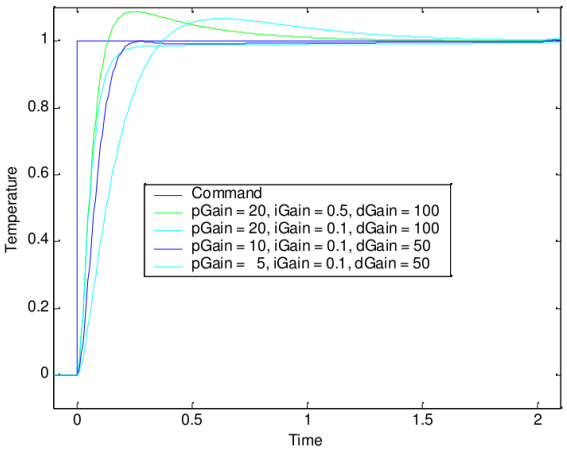


Figura 2.17: Controlador Proporcional-Integral-Derivativo (PID) com vários valores da constante proporcional (k_p), constante integral (k_i) e da constante derivativa (k_d).

Em seguida apresentamos o algoritmo genérico/"pseudo-código" de um controlador PID:

```
kp = <constanteProporcional>
ki = <constanteIntegral>
kd = <constanteDiferencial>
valorReferencia = <objetivo>
erro = integral = diferencial = 0
Inicio:
    erro = valorReferencia − valorAtual
    integral = integral + erro
    diferencial = valorUltimo − valorAtual
```

```
    def = erro * K_p + integral * K_i + diferencial * K_d
    valorUltimo = valorAtual
    esperar(tempo)
GOTO Inicio
```

Uma das principais dificuldades na definição dos controladores PID reside na definição das várias constantes (K_p, K_i e K_d). Trata-se de um processo que exige experimentação, caracterizando-se frequentemente por "tentativa/erro". Existem, no entanto, diversos métodos de afinação de controladores PID, tais como os métodos *Ziegler-Nichols* e o método *Cohen-Coon*. Neste trabalho também houve a necessidade de recorrer ao processo de "tentativa/erro" até se conseguir obter estas constantes.

Uma explicação teórica para estimar de que modo a quantificação afeta um sistema em cadeia fechada é extremamente difícil. A solução para esse problema passa por tratar os efeitos da quantificação como ruído [52].

2.4.5 Amostragem

A amostragem também tem um papel relevante nos sistemas de controlo. Se a amostragem é muito baixa, pode não ser possível atingir o desempenho pretendido, porque se adicionou atraso à amostragem; se a amostragem é muito elevada, podemos ter problemas com o ruído na componente diferencial e um excesso de valores no componente integral. Como ponto de partida em controlo de sistemas digitais, a amostragem deve estar compreendida entre 1/10 e 1/100 do tempo de estabilização do sistema desejado [51].

2.5 Qualidade de Serviço

O termo "Qualidade de Serviço"[1] (QoS) designa a capacidade de fornecer um serviço de acordo com determinadas exigências/objetivos, como por exemplo: tempo de resposta, débito[2], disponibilidade, *etc.*

Uma solução fácil para fornecer uma boa qualidade de serviço passa por implementar uma rede informática com capacidade suficiente para satisfazer qualquer tráfego que passe por ela. Esta solução é designada por *overprovisioning*. A rede informática resultante será capaz de transportar o tráfego aplicacional sem perdas significativas e, assumindo que existe um esquema decente de encaminhamento, a mesma entregará os pacotes com pouca latência. Existe simplesmente tanta capacidade disponível que a procura pode, quase sempre,

[1] Do inglês, *Quality of Service*
[2] Do inglês, *Throughput*

ser satisfeita. Basicamente, consiste em resolver o problema com recursos financeiros. Os mecanismos de QoS permitem que redes informáticas com menos capacidades satisfaçam as necessidades aplicacionais a custos mais baixos. Mais, o *overprovisioning* é baseado em tráfego estimado, o que faz com que perante alterações de padrões, o mesmo possa estar incorrecto. Com mecanismos de QoS, as redes informáticas podem satisfazer o desempenho, mesmo perante picos, embora à custa de alguns pedidos [48].

Quatro aspetos devem ser avaliados para assegurar o QoS:

· Quais as aplicações que necessitam de rede;

· Como regular o tráfego que entra na rede;

· Como reservar recursos nos encaminhadores para garantir o desempenho;

· Se a rede pode aceitar mais tráfego com segurança.

Nenhuma técnica, por si só, resolve de forma eficiente todas estas questões [48]. As soluções práticas de QoS passam pela combinação de várias técnicas. Foram desenvolvidas várias técnicas ao nível das camadas de rede e transporte[1].

2.5.1 Níveis de serviço

O termo "nível de serviço"[2] define o nível de exigência para a capacidade de fornecer um determinado serviço. Definem-se, geralmente, três níveis de QoS:

Melhor esforço [3]: Não fornece nenhuma diferenciação entre os vários fluxos de rede e não permite nenhuma garantia. Este nível de serviço é também designado como *lack of QoS*.

Serviço diferenciado [4]: Permite definir níveis de prioridade para os diferentes fluxos rede, sem contudo fornecer uma garantia estrita. Este nível de serviço é também designado como *soft QoS*.

Serviço garantido [5]: Consiste em reservar recursos de rede para certos tipos de fluxos. O principal mecanismo utilizado para obter tal nível de serviço é o "Protocolo de reserva de recursos[6]" (RSVP). Este nível de serviço é também designado como *hard QoS*.

[1] Do inglês, *Network and transport layer*
[2] Do inglês, *Service Level (SL)*
[3] Do inglês, *Best effort*
[4] Do inglês, *Differenciated service*
[5] Do inglês, *Guaranteed service*
[6] Do inglês, *Resource reSerVation Protocol*

2.5.2 Critérios de qualidade de serviço

Os principais critérios que permitem apreciar a qualidade de serviço são os seguintes:

Débito : É definido como o volume máximo de informação (bits) por unidade de tempo;

Flutuação [1]: É a variação estatística do atraso na entrega de pacotes. Pode ser definida como a medida de variação do atraso entre sucessivos pacotes de dados;

Latência [2]: Caracteriza o atraso entre a emissão e a receção de um pacote;

Perda de pacotes [3]: Corresponde à não entrega de um pacote de dados;

Desordenação de pacotes [4]: Diz respeito à modificação na ordem de chegada dos pacotes.

Muitas vezes, quando se fala num sistema servidor é importante pensar sobre a QoS que um sistema oferece aos utilizadores. Existem várias propriedades que podem ser observadas nos servidores HTTP, mas os mais importantes são os seguintes:

Disponibilidade : Percentagem de tempo que o sistema está disponível;

Tempo de resposta : Tempo total que o sistema leva a responder ao pedido de um cliente;

Débito : Taxa na qual o sistema pode manipular as solicitações.

Dificilmente um sistema tem um desempenho ótimo nestes três parâmetros, até porque muitas vezes há objetivos contraditórios entre eles. Uma melhoria no tempo de resposta pode levar a uma diminuição no débito e na disponibilidade de serviço. O débito e a disponibilidade tem sinergias; um melhor débito, provavelmente, também será bom para a disponibilidade, mas pode aumentar o tempo de resposta.

Importa, assim, definir quais as propriedades críticas para o sistema em análise. Em seguida, devem ser obtidas observações dessas propriedades para avaliar o desempenho do sistema.

Disponibilidade

A disponibilidade está intimamente ligada à fila de espera. Perante uma situação de sobrecarga, se a fila de espera estiver cheia, o sistema não aceitará mais pedidos do cliente e, portanto, estará indisponível. De igual forma, se o servidor estiver muito ocupado, um pedido pode estar muito tempo na fila de espera e pode ser removido antes de ter sido cumprido.

Para controlar a disponibilidade do sistema é necessário ter em atenção o comprimento da fila de espera, o tempo de espera da fila e evitar, sempre que possível, a sobrecarga.

[1] Do inglês, *Jitter*
[2] Do inglês, *Delay*
[3] Do inglês, *Packet loss*
[4] Do inglês, *Desequencing*

Débito

No sentido de melhorar o rendimento do sistema, o número de servidores pode ser aumentado. Essa solução permite ao sistema lidar com mais solicitações em simultâneo. Também é possível aumentar o comprimento da fila e o tempo de espera da fila (*timeout*). Esta alteração admite que existam mais pedidos no sistema e, consequentemente, reduzir o tempo de inatividade de cada servidor, o que permite melhorar o seu rendimento. É importante ter consideração que, nesta segunda hipótese, a alteração desses parâmetros, vai consumir mais recursos do sistema. Se o consumo de recursos do sistema se tornar muito elevado, existirá uma sobrecarga do servidor e uma consequente redução na taxa de transferência.

Tempo de resposta

É difícil dizer o que permite melhorar o tempo de resposta de um sistema, pois este é altamente dependente da carga do servidor. Um ajuste que permite um bom tempo de resposta durante a carga leve pode não funcionar adequadamente sob carga mais pesada. É desejável manter o tempo que cada pedido passar na fila o mais baixo possível. A fila de espera é uma boa funcionalidade sob a perspectiva da disponibilidade e da taxa de transferência de um servidor, pois torna possível ao sistema aceitar mais pedidos. No entanto, a fila de espera não é algo que melhore o tempo resposta. Deste modo, se o tempo de resposta é o parâmetro mais crítico de um determinado sistema é preferível manter o comprimento da fila e o tempo limite da fila o mais baixo possível, mesmo sob pena do pedido poder ser descartado.

2.5.3 Controlo de tráfego

O mecanismo de controlo de tráfego entra em ação quando um pacote entra na fila para transmissão numa interface, mas antes do mesmo ser efetivamente enviado pelo *driver*. O kernel toma a decisão do controlo de tráfego depois do pacote estar na fila de espera para ser transmitido. A figura 2.18 identifica o local onde as decisões são tomadas no que diz respeito à transmissão do pacote pelas camada física de rede e transporte [14].

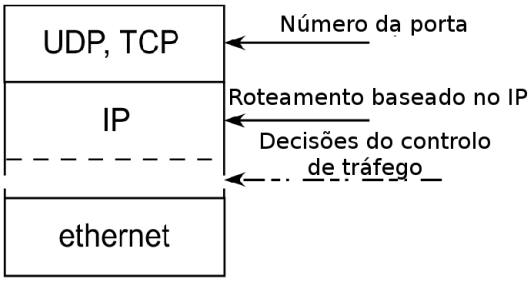


Figura 2.18: Identificação do local onde ocorrem as decisões de controlo de tráfego.

O controlo de tráfego consiste nas seguintes acções:

Modelar [1]: Quando o tráfego é modelado, a sua taxa de transmissão está sob controlo. A modelagem pode consistir em mais do que reduzir a largura de banda disponível; é igualmente utilizada para suavizar picos de tráfego de modo a se registar um melhor desempenho da rede. A modelagem atua no controlo de saída (*egress shapping*);

Escalonar [2]: Ao se escalonar a transmissão de pacotes é possível melhorar a interactividade do tráfego que dela necessita, ao mesmo tempo que se garante a largura de banda para o restante tráfego. O processo de reordenar é igualmente designado como priorização e, só atua no controlo de saída;

Fiscalizar [3]: Se a modelação diz respeito à transmissão de pacotes, a fiscalização trata o tráfego que é recebido, pelo que ocorre na entrada (*ingress*);

Descartar [4]: O tráfego que excede a largura de banda pode ser descartado tanto na entrada, como na saída.

O processamento do tráfego é controlado por três tipos de objetos, designadamente: disciplinas de serviço (`qdiscs`), classes(`classes`) e filtros (`filters`).

Disciplinas de serviço [5]: São o elemento básico para compreender o controlo de tráfego. Sempre que o kernel necessita de enviar um pacote para uma interface, ele é colocado na fila da disciplina de serviço (`qdisc`) configurada para essa interface. Em seguida, o kernel tenta obter tantos pacotes quanto possível da `qdisc` para os entregar ao *driver* da placa de rede. A `qdisc` pré-definida do kernel do Linux é do tipo *pfifo_fast*, a qual não efetua qualquer processamento sendo uma fila puramente "*First in, First out*" (FIFO). Consegue, no entanto, armazenar tráfego quando uma interface de rede não o consegue gerir temporariamente.

Classes : Algumas `qdiscs` conseguem conter classes, as quais contêm mais `qdiscs`. O tráfego pode ser inserido em qualquer uma das `qdiscs` presentes no interior das classes. Quando o kernel tenta retirar um pacote dessa `qdisc` *classful*, ele pode provir de qualquer uma das classes. Uma `qdisc` pode dar prioridade a certo tipo de tráfego tirando os pacotes de algumas classes antes de outras.

Fitros : Os filtros residem dentro das `qdiscs` e são utilizados pelas `qdisc` *classful* para determinar em que classe um pacote deve ser colocado. Sempre que o tráfego chega a uma classe com sub-classes, ele deve ser classificado. Todos os filtros pertencentes a uma classe são invocados, até que um deles é aplicado. Se nenhum filtro é aplicado, podem ser avaliados outros critérios.

Na figura 2.19 apresenta-se um esquema simplificado destes objetos [14].

[1] Do inglês, *Shapping*
[2] Do inglês, *Scheduling*
[3] Do inglês, *Policing*
[4] Do inglês, *Dropping*
[5] Por vezes designadas como "Disciplinas de filas de espera"

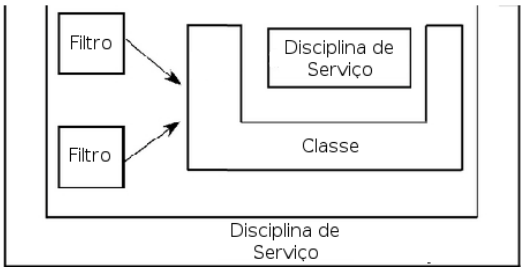


Figura 2.19: Esquema simplificado de disciplinas de serviço, classes e filtros.

2.5.3.1 Disciplina de serviço

O kernel do Linux trabalha com dois tipos de disciplina de serviço: *classful* e *classless*.

2.5.3.2 Disciplinas de serviço *classless*

As disciplinas de serviço *classless* são aquelas que só podem aceitar dados e só conseguem re-agendar, atrasar ou descartar. Estas classes só podem ser utilizadas para modelar o tráfego de uma interface no seu todo, sem qualquer subdivisão [32].

Cada uma destas disciplinas de serviço pode ser utilizada como `qdisc` principal de uma interface ou, pode ser utilizada dentro de uma classe filha de uma `qdisc` *classful*. Estas são as unidades de escalonamento fundamentais usadas no Linux.

Algumas das disciplinas de serviço *classless* usadas no Linux são:

[p|b]fifo : Esta é a `qdisc` mais simples e tem um comportamento FIFO puro;

pfifo_fast : É a `qdisc` pré-definida no Linux. Esta disciplina está dividida em três bandas diferentes e só dentro de cada banda é que se aplicam as regras de FIFO. No entanto, desde que existam pacotes à espera na banda 0, os pacotes da banda 1 não são processados. O mesmo sucede para a banda 1 e banda 2. O kernel utiliza o campo *Type of Service* (ToS) presente no cabeçalho dos pacotes e insere na primeira fila de espera (banda 0) os pacotes cujo serviço associado seja sensível a atrasos;

***Random Early Detection* (RED)** : Simula um congestionamento físico e descarta aleatoriamente pacotes quando a largura de banda usada se aproxima do valor configurado;

***Stochastic Fairness Queueing* (SFQ)** : Re-ordena o tráfego nas filas de espera de modo a que alternadamente cada "sessão" consiga enviar um pacote. Isto leva a um comportamento justo de envio de dados. O SFQ tem na sua designação a referência estocástico porque não armazena propriamente uma fila de espera para cada sessão, tem um algoritmo que divide o tráfego por um número limite de filas usando um algoritmo de *hashing*. Devido ao *hashing*, várias sessões podem acabar no mesmo *bucket*. Para prevenir esta situação de ser percetível, o SFQ muda o seu algoritmo (*hashing*) algumas vezes de modo a que duas

sessões em colisão apenas o façam por segundos. Normalmente, não se coloca em prática o SFQ na interface de rede;

***Token Bucket Filter* (TBF)** : Disciplina de serviço simples que apenas vai deixando passar pacotes até que o seu número não exceda valores pré-definidos. Existe, contudo, a possibilidade de permitir que temporariamente esses valores sejam excedidos. A implementação do TBF consiste num *buffer*/*bucket*, que está constantemente a ser preenchido por *tokens* a uma taxa específica (*token rate*). Cada *token* representa um pacote. O parâmetro mais importante do *bucket* é o seu tamanho, isto é: o número de *tokens* que consegue armazenar. Esta disciplina de serviço escala bem em larguras de banda elevada.

2.5.3.3 Disciplinas de serviço *classful*

As disciplinas de serviço *classful* podem ter filtros associadas a elas, permitindo que os pacotes sejam direcionados para determinadas classes e subfilas [17]. As `qdisc` *classful* são bastante úteis quando existem diferentes tipos de tráfego e é necessário fazer um tratamento diferenciado.

Algumas das disciplinas de serviço *classless* usadas no Linux são:

***Class Based Queueing* (CBQ)** : Implementa uma hierarquia de classes rica em funcionalidades. Contém elementos de modelação e priorização. O CBQ funciona certificando-se que a ligação se encontra parada tempo suficiente para reduzir a largura de banda real à taxa configurada. Para tal, é necessário calcular o período de tempo entre pacotes. É de difícil implementação e pode não produzir os resultados desejados;

***Hierarchical Token Bucket* (HTB)** : O HTB pretende ser um substituto mais compreensível, intuitivo e rápido do que a `qdisc` CBQ [24]. Ambos permitem que um *link* físico simule vários *links* mais lentos, permitindo-lhes enviar diferentes tipos de tráfego em diferentes *links* simulados. Em ambos os casos é necessário especificar como o *link* físico é dividido em vários *link* e como selecionar qual o *link* a usar para um determinado pacote. Podemos, no entanto, de uma forma resumida, referir que o HTB assegura que a quantidade de serviço fornecida a cada classe é pelos menos o mínimo entre o que ela solicita e o que lhe está atribuído. Se uma classe solicita menos do que lhe está atribuído, a largura de banda remanescente (excesso) é distribuída por outras classes que solicitem serviço.

PRIO : Esta `qdisc` é um *container* para um número configurável de classes. Esta `qdisc` não realiza modelação e os pacotes são retirados das classes por ordem. Este último aspeto permite uma forma fácil de priorização de tráfego em que, as classes menos importantes só são capazes de enviar se as classes mais importantes não tem pacotes disponíveis para envio. Para facilitar a configuração, os bits relativos ao `ToS` dos cabeçalhos dos pacotes são tidos em consideração.

Tendo em consideração os objetivos principais deste trabalho e as principais funcionalidades das `qdisc` existentes, a disciplina de serviço escolhida é o `HTB`.

2.5.3.4 Qualidade de Serviço em Linux com `tc` e `Netem`

O Linux utiliza a ferramenta/comando `tc`, existente no pacote `iproute` [38], para configurar o controlo de tráfego. Esta ferramenta realiza toda a configuração necessária das estruturas do kernel para suportar o controlo de tráfego [17]. O `tc` utiliza bibliotecas partilhadas e outros ficheiros, geralmente localizados em `/usr/lib/tc` [39].

É possível controlar e visualizar as definições do controlo de tráfego. Em seguida apresenta-se um exemplo de *output* simples:

```
# tc qdisc show

qdisc pfifo_fast 0: dev eth1 root refcnt 2 bands 3 priomap 1 2 2 2 1 2 0 0 1 1 1 1 1 1 1
    ↪1
```

É igualmente possível solicitar informação adicional, exemplo: estatísticas, sendo neste caso o *output* algo do género:

```
# tc -d -s -p qdisc show

qdisc pfifo_fast 0: dev eth0 root refcnt 2 bands 3 priomap 1 2 2 2 1 2 0 0 1 1 1 1 1 1 1
    ↪1
 Sent 5896513294 bytes 5213682 pkt (dropped 0, overlimits 0 requeues 22205)
 rate 0bit 0pps backlog 0b 0p requeues 22205
```

Em tráfego de rede a base é decimal e não binária, pelo que: 1 MB/s = 1.000 KB/s. Tendo em consideração que 1 byte são 8 bits, então 250 Kb/s = 31,25 KB/s[1].

O `tc` usa as seguintes regras para a especificação da largura de banda:

mbps = 1024 kbps = 1024 * 1024 bps => byte/s
mbit = 1024 kbit => kilo bit/s
mb = 1024 kb = 1024 * 1024 b => byte
mbit = 1024 kbit => kilo bit

Os valores são armazenados internamente em bps e b.

Quando `tc` imprime o débito usa a seguinte notação:

1Mbit = 1024 Kbit = 1024 * 1024 bps => bit/s

[1] B = Byte e b = bit

Algumas das opções do comando `tc`, para o HTB, são as seguintes:

parent x:y | root : Este parâmetro determina o posicionamento da instância HTB, podendo estar no topo de uma interface (`root`), ou dentro de uma classe existente;

handle x:y : Identificador da estrutura. De um modo geral (não exclusivamente para o HTB, mas também para todas as `qdisc` e classes no `tc`) os `handle` são representados na forma `x:y`, em que `x` é um número inteiro que identifica a `qdisc` e o `y` é um número inteiro que identifica a classe a que pertence essa `qdisc`. O valor de y deve ser 0 numa `qdisc`, enquanto numa classe este valor deve ser diferente de 0.

default : Identificador da classe para a qual são enviados os pacotes não classificados;

rate : Taxa alocada à classe; a classe pode pedir emprestado a outras classes;

burst : Máximo de bytes que podem ser acumulados num *burst* durante o período ocioso;

ceil : Taxa máxima/limite da classe; a classe não pode pedir emprestado a outras classes;

cburst : Máximo de bytes num *burst* aplicável à variável `ceil`;

mtu : Tamanho máximo do pacote;

prio : Prioridade das classes folha[1]. As classes folha com menor valor tem maior prioridade, isto é: são servidas primeiro;

quantum : Número de bytes que são servidos de cada vez de uma classe folha. O *quantum* é calculado pela fórmula $rate\,(bytes)/r2q$. O valor pré-definido de `r2q` é 10, mas pode ser alterado quando se cria uma `qdisc` HTB. O valor de `quantum` pode ser alterado caso se crie uma classe HTB. O `quantum` deve ser superior ao MTU (1500) para que se possa enviar um pacote de uma só vez e, inferior a 60000 (valor *hard coded* na `qdisc` HTB para prevenir a privação de outras classes e atrasos muito elevados). Erros no `quantum` não afetam a funcionalidade, só a precisão. Estes eventos também são registados pelo que pode ocupar muito espaço em disco. Os pacotes que são enviados quando uma determinada classe pode enviar são, ainda assim, validados, tendo em consideração os parâmetros `rate`/`ceil`/`burst`/`cburst`. Deste modo, um elevado `quantum` não criará *bursts*, se tal não for permitido pelos parâmetros `burst`/`cburst`.

Em seguida apresenta-se alguns exemplos do comando `tc`:

```
tc qdisc add dev eth0 root handle 1:0 htb default 10
tc class add dev eth0 parent 1:0 classid 1:1 htb rate 100kbps ceil 100kbps
tc class add dev eth0 parent 1:1 classid 1:10 htb rate 30kbps ceil 100kbps
tc class add dev eth0 parent 1:1 classid 1:11 htb rate 10kbps ceil 100kbps
tc class add dev eth0 parent 1:1 classid 1:12 htb rate 60kbps ceil 100kbps
```

Na secção 3.2.1.3 *Gateway* são explicadas, em detalhe, as opções usadas neste trabalho.

[1]Do inglês, *Leaf classes*

O `NetEm` (*Network Emulation*) fornece funcionalidades de simulação de redes para testar protocolos ao emular propriedades de redes de longa distância[1]. A versão atual simula atrasos variáveis, perdas, duplicação e re-ordenação de pacotes [39].

Em distribuições de Linux recentes (ex.: Fedora, OpenSuse, Gentoo, Debian, Mandriva, Ubuntu) com o kernel 2.6, o NetEm já está habilitado no kernel, sendo habilitado em:

```
Networking -->
Networking Options -->
QoS and/or fair queuing -->
Network emulator
```

O `NetEm` consiste em dois componentes: um pequeno módulo de `kernel`, integrado no kernel 2.6.8/2.4.28 (ou superior), para a disciplina de serviço e o comando/utilitário `tc` (abordado anteriormente) para o configurar [23, 29].

Em seguida apresenta-se alguns exemplos do comando `tc` com o `NetEm`:

```
tc qdisc add dev eth0 parent 1:11 handle 10: netem delay 10
tc qdisc add dev eth0 parent 1:11 handle 20: netem delay 20
```

Neste trabalho, o `NetEm` é utilizado para introduzir atraso nos pacotes destinados ao sítio `Premium` e ao sítio `Outros`, de acordo com a informação disponibilizada pelo *Scoreboard* do servidor HTTP Apache.

[1]Do inglês, *Wide area network (WAN)*

3. Equipamentos e Metodologia

3.1 Equipamentos

No sentido de implementar um sistema de controlo em cadeia fechada, aplicado a servidores HTTP Apache, baseado em teoria de controlo, é implementada uma infraestrutura informática dedicada, tal como apresentada anteriormente (ver figura 2.1 Esquema simplificado da infraestrutura informática).

Nos quadros 3.1 e 3.2 apresentam-se as principais características dos equipamentos utilizados.

	Clientes HTTP	*Gateway*	Servidor HTTP Apache
Processador	Intel P8700 2.53GHz	Intel Atom330 1.60GHz	Intel T3400 2.16GHz
Memoria (GB)	2	2	2
Disco Rígido (GB)	250	250	250
Interface de Rede	10/100/1000Base-T	2 x 10/100/1000Base-T	10/100/1000Base-T
Sistema Operativo	GNU Linux	GNU Linux	GNU Linux
Distribuição	Fedora 19	Fedora 19	CentOS 5.10
Kernel	3.11.9-200 64bit	3.11.9-200 64bit	2.6.18-371.1.2 64bit

Quadro 3.1: Principais características dos clientes, *Gateway* e do servidor HTTP Apache.

	Switch A	*Switch* B
Número de portas	8	5
Velocidade	10/100/1000Base-T	
Switch Fabric	2Gb/s por porta em *full duplex mode*	

Quadro 3.2: Principais características dos *Switch*.

O funcionamento dos vários componentes é o seguite:

- Os clientes HTTP obtêm o seu endereço IP (rede 192.168.1.0/24) via servidor DHCP[1], o qual está instalado na *Gateway*.
- A *Gateway* tem duas placas rede instaladas e estabelece a ligação entre as duas redes, a saber: rede 192.168.1.0/24 (Clientes HTTP) e 192.168.100.0/24 (servidor HTTP Apache). Os dois IP fixos definidos são: 192.168.100.1 (*interface* eth0) e 192.168.1.1 (*interface* eth1). De modo a ser possível a transferência de pacotes entre as duas interfaces de rede, a *Gateway* tem a opção de kernel `ip_forward` habilitada no ficheiro `/etc/sysctl.conf` (*net.ipv4.ip_forward* = 1).
- O servidor HTTP Apache tem um endereço fixo principal: 192.168.100.100 (*interface* eth0) e dois endereços fixos virtuais/*alias*: 192.168.100.101 (*interface* eth0:0), 192.168.100.102/24 (*interface* eth0:1). Os dois endereços virtuais existem de modo a ser possível criar os *virtual host* no servidor HTTP Apache. O endereço 192.168.100.101 está associado ao sítio `Premium` e o endereço 192.168.100.102 está associado ao sítio `Outros`.
- Todos os cabos de rede são de Categoria 6 "10/100/1000Base-T".

3.2 Metodologia

3.2.1 Otimizações

De um modo geral, os sistemas operativos e os servidores na sua configuração pré-definida, não estão otimizados para fazer pleno uso dos recursos disponíveis. Estão configurados para uma utilização mais genérica, isto é: não específica, podendo falhar sob determinadas ambientes, nomeadamente elevadas cargas de pedidos. No sentido de otimizar os recursos disponíveis dos equipamentos e devido ao facto de se registar um elevado volume de tráfego de rede e de pedidos HTTP, são realizadas algumas otimizações ao nível dos clientes HTTP, da *Gateway* e do servidor HTTP Apache (de agora em diante referido como servidor Apache).

3.2.1.1 Clientes HTTP

Após o arranque dos clientes HTTP é executado um *script* (`tune.sh`) cujo objetivo é otimizar os recursos disponíveis. Em seguida apresenta-se o conteúdo do mesmo.

Clientes - `tune.sh`

```
1 #!/bin/bash
2 ulimit -n 65535
3 ulimit -u 2048
```

[1] *Internet Systems Consortium* (ISC) *DHCP server* - `http://www.isc.org/`

```
4 ulimit -c unlimited
5 modprobe tcp_htcp
6 sysctl -w net.ipv4.tcp_congestion_control=htcp
7 ifconfig eth0 txqueuelen 10000
```

- As linhas 2 a 4 tem como objetivo reduzir a probabilidade do número de descritores de ficheiros[1] em uso (-n), número de processos disponíveis para o utilizador (-u) e o tamanho dos ficheiros com imagens de processos em execução[2] (-c) definidos a nível do sistema operativo coloquem limitações ao sistema e impeçam o seu correto funcionamento, convertendo-se num ponto de estrangulamento/congestionamento.
- As linhas 5 e 6 alteram o algoritmo de gestão de congestão de pacotes TCP para `htcp`[3]. Este protocolo foi especialmente desenvolvido e otimizado para controlo de congestão de pacotes em redes com elevado atraso[4]. O método selecionado neste trabalho para priorização/QoS dos pacotes/pedidos HTTP baseia-se na introdução de atraso na entrega de pacotes considerados menos prioritários (Clientes `Outros`, isto é: "não `Premium`").
- A linha 7 tem como objetivo aumentar o tamanho da fila de transmissão da placa de rede, de modo a ser possível, caso necessário, atingir velocidades de Gigabit.

Deve ser salientado que a partir do kernel 2.4.17/2.6.7 o parâmetro `tcp_moderate_rcvbuf` (`/proc/sys/net/ipv4/tcp_moderate_rcvbuf`) está habilitado, o que permite ao sistema operativo GNU/Linux ajustar dinamicamente o *buffer* de chegada do TCP para otimizar o débito [36]. Assim sendo, o tamanho dos *buffer* é o pré-definido, mas os valores máximos são ajustados dinamicamente de modo a permitir um maior débito. É possível alterar este parâmetro no ficheiro `/etc/sysctl.conf`, mas o valor ajustado dinamicamente será sempre inferior ao do parâmetro `tcp_rmem`, o qual regula o tamanho do *buffer* de recepção.

3.2.1.2 Servidor HTTP Apache

Neste trabalho, o código fonte do servidor Apache é alterado de modo a ser possível recolher informação do seu `Scoreboard`. Essa informação é transmitida para a *Gateway*, estrutura onde é implementado o sistema de controlo em cadeia fechada de QoS, baseado em teoria de controlo.

A versão do Servidor Apache utilizada e modificada neste trabalho é a 2.2.3.

Foi implementado um *script* (`tune.sh`) semelhante ao executado nos clientes HTTP e na *Gateway*. Os objetivos deste *script* são idênticos às outras situações, sendo o mesmo executado após o arranque do servidor físico onde está alojado o servidor Apache.

[1] Do inglês, *File descriptors*

[2] Do inglês, *Maximum size of core files*. Um "ficheiro core" é uma imagem de um processo que é criado pelo sistema operativo quando um processo é encerrado de uma forma inesperada.

[3] `http://www.hamilton.ie/net/htcp.htm`

[4] Do inglês, *Delay*

Servidor Apache - `tune.sh`

```
#!/bin/bash
ulimit -c unlimited
ulimit -n 65535
ulimit -u 2048
modprobe tcp_htcp
sysctl -w net.ipv4.tcp_congestion_control=htcp
sysctl -w net.core.netdev_max_backlog=30000
ifconfig eth0 txqueuelen 10000
```

São, igualmente, alterados alguns dos parâmetros do servidor Apache, sendo os mais importantes os seguintes:

Servidor Apache - `httpd.conf` (excerto)

```
Timeout 5
KeepAlive On
MaxKeepAliveRequests 0
KeepAliveTimeout 3
<IfModule prefork.c>
StartServers 256
MinSpareServers 5
MaxSpareServers 10
ServerLimit 256
MaxClients 256
MaxRequestsPerChild 0
</IfModule>
Listen 80
ExtendedStatus On
HostnameLookups Off
<Location /server-status>
    SetHandler server-status
    Order deny,allow
    Deny from all
    Allow from 127.0.0.1
    Allow from 192.168.1.0/24
    Allow from 192.168.100.0/24
</Location>
ServerName 192.168.100.100
<VirtualHost 192.168.100.101>
    ServerName 192.168.100.101
    DocumentRoot /var/www/html/site1
</VirtualHost>
<VirtualHost 192.168.100.102>
    ServerName 192.168.100.102
    DocumentRoot /var/www/html/site2
</VirtualHost>
```

O significado destes parâmetros foi explicado em detalhe na secção 2.3 Servidor HTTP Apache.

No anexo A apresenta-se, na íntegra, o ficheiro de configuração do servidor Apache.

Nos quadros 3.3 e 3.4 apresentam-se alguns dos valores pré-definidos dos parâmetros do servidor Apache e dos configurados neste trabalho. A função destes parâmetros de configuração foi explicada detalhadamente no ponto 2.3 Servidor HTTP Apache. Algumas das otimizações efetuadas estão descritas na documentação Apache [8], nomeadamente ao ní-

vel dos parâmetros `MaxClients`, `HostnameLookups`, `MinSpareServers`, `MaxSpareServers`, `StartServers`.

Tal como referido anteriormente, o principal motivo da escolha do `MPM prefork` [1] prende-se com o facto deste `MPM` permitir a coexistência do servidor Apache e do PHP nos sítios Internet, onde potencialmente pode ser implementado o mecanismo de QoS proposto. Esta escolha está naturalmente relacionada com a elevada utilização do PHP, tal como evidenciado na revisão bibliográfica.

O servidor HTTP Apache é modificado de modo a obter e disponibilizar à *Gateway* a informação pretendida, a saber:

- Percentagem de processos `Ativos`;
- percentagem de processos `Premium`;
- CPU;
- Memória.

A informação obtida é enviada, em intervalos de cerca de 5 segundos, para a *Gateway*.

As alterações introduzidas no código fonte do servidor HTTP são realizadas nos ficheiros `scoreboard.h` (ver Listagem 3.1), `scoreboard.c` (ver Listagem 3.2), `mst_ue.h` e `mst_ue.c`.

Listagem 3.1: Servidor Apache - `scoreboard.h` (excerto)

```
#ifndef MST_UE
#define MST_UE
#endif
```

Listagem 3.2: Servidor Apache - `scoreboard.c` (excerto)

```
#ifdef MST_UE
#include "mst_ue.c"
#endif
#ifdef MST_UE
        mst_ue(server_limit);
#endif
```

São estas alterações que permitem a inclusão do código desenvolvido especificamente para este trabalho.

No anexo A apresenta-se, na íntegra, o código dos ficheiros `mst_ue.h` e `mst_ue.c`.

3.2.1.3 *Gateway*

Após o arranque da *Gateway* é executado um *script* (`tune.sh`) cujo objetivo é otimizar os recursos disponíveis, à semelhança dos clientes e do servidor HTTP. Em seguida apresenta-se

[1] *Apache Multi-Processing Modules - Prefork*

Parâmetro	Valor pré-definido	Valor configurado
Timeout	60	5
KeepAlive	Off	On
MaxKeepAliveRequests	100	0
KeepAliveTimeout	5	3
<IfModule prefork.c>		
StartServers	8	256
MinSpareServers	5	5
MaxSpareServers	20	10
ServerLimit	256	256
MaxClients	256	256
MaxRequestsPerChild	4000	0
</IfModule>		
ExtendedStatus	Off	On

Quadro 3.3: Valores pré-definidos e configurados de alguns dos parâmetros do servidor HTTP Apache.

Parâmetro	Valor pré-definido	Valor configurado
<Location /server-status> SetHandler server-status Order deny,allow Deny from all Allow from 127.0.0.1 Allow from 192.168.1.0/24 Allow from 192.168.100.0/24 </Location>	"desabilitado"	"habilitado"
ServerName	"comentado"	192.168.100.100
<VirtualHost 192.168.100.101> ServerName 192.168.100.101 DocumentRoot /var/www/html/site1 </VirtualHost> <VirtualHost 192.168.100.102> ServerName 192.168.100.102 DocumentRoot /var/www/html/site2 </VirtualHost>	"inexistente"	"configurado"

Quadro 3.4: Valores pré-definidos e configurados de alguns dos parâmetros do servidor HTTP Apache (cont.).

o conteúdo do mesmo.

Gateway - tune.sh

```
1 #!/bin/bash
2 ulimit -n 65535
3 ulimit -u 2048
4 ulimit -c unlimited
5 modprobe tcp_htcp
6 sysctl -w net.ipv4.tcp_congestion_control=htcp
7 sysctl -w net.core.netdev_max_backlog=30000
8 ifconfig eth0 txqueuelen 10000
9 ifconfig eth1 txqueuelen 10000
```

- As linhas 2 a 6, 8 e 9 tem o mesmo objetivo do *script* dos Clientes HTTP.
- A linha 7 tem como objetivo minimizar as falhas de ligações, resultantes de um elevado número de ligações recebidas[1].

O ficheiro `/etc/sysctl.conf` é igualmente alterado, tendo sido introduzidas as seguintes linhas:

Gateway - `/etc/sysctl.conf` (excerto)

```
net.ipv4.ip_forward = 1
net.ipv4.netfilter.ip_conntrack_max = 32768
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_orphan_retries = 1
net.ipv4.tcp_fin_timeout = 25
net.ipv4.tcp_max_orphans = 8192
net.ipv4.ip_local_port_range = 32768 61000
```

Em seguida explica-se o significado de cada parâmetro[2]:

net.ipv4.ip_forward: Habilita o encaminhamento de pacotes entre as interfaces de rede `eth0` e `eth1`. Tal como referido anteriormente, a *Gateway* faz a interligação entre os clientes HTTP e o servidor Apache e tem duas interfaces de rede;

net.ipv4.netfilter.ip_conntrack_max: Número de ligações que o servidor pode servir, antes de ter que esperar o valor configurado em `tcp_fin_timeout` (expresso em segundos) nas ligações já realizadas para estar novamente disponível. A tabela `conntrack` é a estrutura que o kernel utiliza para controlar o estado das ligações. Assim que esta tabela está cheia, o kernel começa a descartar pacotes e reportar esse facto no seu ficheiro de *log*, exemplo: `Set 10 15:39:14 XXXX-XXX kernel: ip_conntrack: table full, dropping packet`. É de salientar que o valor definido nesta variável (32768) é superior ao número de portas locais definido na variável `ip_local_port_range`: 28232 (61000 - 32768). Na distribuição *GNU/Linux CentOS 5* o valor pré-definido desta variável é 65536;

net.ipv4.tcp_tw_recycle: Permite ao kernel reciclar as ligações que estejam no estado `TIME_WAIT` em vez de descartar os pacotes;

[1] Do inglês, *Incoming connection*

[2] `http://lartc.org/howto/lartc.kernel.obscure.html`

net.ipv4.tcp_tw_reuse: Permite a reutilização de *sockets* que estão no estado `TIME_WAIT` para novas ligações.

Devido ao modo como o TCP/IP funciona, as ligações não podem ser encerradas imediatamente. Os pacotes podem chegar fora de ordem ou serem retransmitidos após a ligação ter sido fechada. O estado `CLOSE_WAIT` indica que o destino (componente remota da ligação) encerrou a ligação. O estado `TIME_WAIT` indica que a origem (componente local da ligação) fechou a ligação. A ligação é mantida de modo a que os pacotes em atraso possam ser recebidos e tratados adequadamente. Podemos dizer que o estado `TIME_WAIT` é um mecanismo de proteção no TCP. O lado que encerra a ligação de um "modo normal", mantém a ligação no estado `TIME_WAIT` de modo a proteger contra a corrupção ou a perda de dados [1].

net.ipv4.tcp_orphan_retries: Controla a número de tentativas antes de um *socket* ser considerado órfão e ser encerrado. O valor pré-definido (7) corresponde a um período de 50 segundos a 16 minutos, dependendo do `RTO` (*Retransmittion Timeout*: Determina quanto tempo o TCP aguarda a confirmação (`ACK`) do pacote transmitido).

net.ipv4.tcp_fin_timeout: Determina o tempo que deve decorrer antes do TCP/IP poder libertar uma ligação fechada e reutilizar seus recursos, ou seja: o tempo que deve manter um *socket* no estado `FIN-WAIT-2`, se foi encerrado localmente. A outra parte (remota) pode nunca encerrar o seu lado, ou até já não existir. O valor pré-definido é 60 segundos, sendo 180 segundos no kernel 2.2. Convém, no entanto, ter em atenção que embora seja possível alterar este valor, corre-se o risco de ter um *overflow* de memória com centenas de *sockets* mortos, mesmo num servidor HTTP com pouca carga. Os *sockets* em estado `FIN-WAIT-2` são menos perigosos do que os em estado `FIN-WAIT-1` porque consomem 1,5Kb de memória, mas tendem a ter uma duração maior (ver `tcp_max_orphans`). Ao reduzir o valor deste parâmetro, o TCP/IP pode libertar ligações fechadas mais rapidamente, disponibilizando mais recursos para novas ligações. Se o valor fosse 60 segundos e mantendo os restantes parâmetros inalterados, o sistema só conseguia ter 470 portas simultaneamente: $(61000 - 32768)/60 \approx 470$ (Intervalo de portas locais / `tcp_fin_timeout`);

net.ipv4.tcp_max_orphans: Número máximo de *sockets* TCP/IP que não estão ligados a nenhum *file handle* de nenhum utilizador e são geridos pelo sistema. Se este número é excedido, é feito um *reset* das ligações orfãs e é exibida uma mensagem de aviso `TCP: too many of orphaned sockets`. Este limite existe para prevenir ataques simples de *DoS*. Esta variável é um número inteiro e o seu valor pré-definido é 8192. O seu valor configurável depende muito da memória disponível, sendo que cada socket órfão necessita de 64Kb de memória (não *swap*). Assim sendo, muita memória pode ser utilizada se ocorrer algum problema. Caso a mensagem mencionada anteriormente ocorra, é necessário/aconselhável aumentar a memória instalada e analisar quais as otimizações a efetuar nas variáveis `tcp_fin_timeout` e `tcp_orphans_retries`, de modo a reduzir a quantidade de *sockets* órfãos;

[1] `https://wiki.apache.org/HttpComponents/FrequentlyAskedConnectionManagementQuestions`

net.ipv4.ip_local_port_range: Define o intervalo das portas locais que o TCP e UDP podem utilizar localmente. Este parâmetro é composto por dois números: O primeiro número é a primeira porta que o TCP e UDP podem utilizar, o segundo valor é a última porta. Para sistemas com carga elevada, os valores pré-definidos (exemplo: *Red Hat Linux*: "1024 4999") devem/podem ser alterados para valores superiores (exemplo: "32768 61000"), o que significa que no máximo podemos ter 28232 ligações abertas em simultâneo: $61000 - 32768$.

A *Gateway* recebe a informação enviada pelo servidor Apache e calcula alguma informação necessária para o controlador PID, designadamente:

- Percentagem de processos `Outros`;
- Erro relativamente ao objetivo definido (`Setpoint`);
- Atraso a aplicar aos pacotes.

3.2.2 Ferramentas de teste

É utilizado o software `httperf` [30] nos clientes HTTP para gerar pedidos HTTP/1.1 ao servidor Apache. São igualmente registadas, para posterior análise, as estatísticas produzidas por esta ferramenta (ver Capítulo 4 Apresentação e Discussão de Resultados).

O `httperf` é uma ferramenta robusta que permite medir o desempenho de um servidor HTTP. Ele fornece flexibilidade para gerar várias cargas HTTP e medir o desempenho do servidor. O objetivo do `httperf` é implementar um bancada de testes específica e fornecer uma ferramenta robusta com elevado desempenho. Tem suporte para os protocolos HTTP/1.1 e SSL. A sua robustez permite-lhe gerar e manter uma carga elevada de pedidos. Durante a sua execução, o `httperf` mantém registo de um conjunto de métricas de desempenho, as quais são resumidas e exibidas sob a forma de estatísticas no final de cada teste.

O `httperf` pode ser configurado através de um conjunto vasto de parâmetros. Os parâmetros utilizados neste trabalho serão explicados na secção "Clientes HTTP".

3.2.3 Execução e recolha de informação

3.2.3.1 Clientes HTTP

Os clientes HTTP solicitam as páginas Internet ao servidor Apache executando um *script* onde é invocado o `httperf`.

Clientes `Premium`

`httperf` - Clientes `Premium`

```
1 #!/bin/bash
2 clear
3 /usr/local/bin/httperf --hog --server 192.168.100.101 --uri /index.html --wsess
    ↪=300000,50,2 --burst-len 5 --rate 50 --timeout 5
```

Clientes `Outros`

`httperf` - Clientes `Outros`

```
1 #!/bin/bash
2 clear
3 /usr/local/bin/httperf --hog --server 192.168.100.102 --uri /index.html --wsess
    ↪=300000,50,2 --burst-len 5 --rate 50 --timeout 5
```

Em seguida apresenta-se, por ordem alfabética, a explicação genérica dos vários parâmetros utilizados:

--burst-length=*N*: Especifica o tamanho dos *bursts*. Cada *burst* consiste em N chamadas ao servidor. O significado exato deste parâmetro depende do gerador de tráfego. Ver a opção **--wsess**.

--hog: Esta opção faz com que o `httperf` tente utilizar tantas portas TCP quanto necessárias. Sem esta opção, o `httperf` está tipicamente limitado ao uso das portas efémeras (no intervado entre 1024 e 5000). Este limite de portas pode rapidamente tornar-se um ponto de estrangulamento[1], sendo por esse motivo aconselhável especificar esta opção para testes mais exaustivos.

--rate=*N*: Especifica a taxa[2] fixa à qual as ligações, ou sessões, são criadas (Por omissão são criadas ligações). Se as opções --wsess ou --wsesslog foram especificadas são criadas sessões. Em ambos os casos, a taxa de 0 (zero) resulta em ligações, ou sessão, sequenciais. Uma nova sessão/ligação é iniciada logo que a anterior terminou. O valor pré-definido para esta opção é 0 (zero).

--server=*S*: Especifica o IP, ou nome, do servidor. Por omissão o nome *localhost* é usado. Esta opção deve ser sempre especificada, já que não é aconselhável executar o cliente e o servidor na mesma máquina.

--timeout=*X*: Especifica a quantidade de tempo X que o `httperf` está disposto a esperar por uma reação do servidor. Este valor é especificado em segundos e pode ser um valor fracionário (exemplo: --timeout 3.5). O valor de *timeout* é usado quando estabelece a ligação TCP, quando envia um pedido, quando espera por uma resposta e quando recebe uma resposta. Se a duração de alguma destas atividades num pedido não consegue progredir, no tempo definido, o `httperf` considera que o pedido morreu, fecha a ligação ou sessão associada e aumenta o contador da estatística `client-timo`. O valor real de *timeout* usado

[1] Do inglês, *Bottleneck*
[2] Do inglês, *Rate*

pelo `httperf` quando espera por uma resposta é a soma deste parâmetro e do parâmetro `think-timeout` (ver opção **--think-timeout**). Por omissão, o valor do *timeout* é infinito.

--uri=*S*: Especifica que o URI *S* deve ser acedido no servidor. Para alguns geradores de tráfego (ex.: **--wset**), esta opção especifica o prefixo dos URI a serem acedidos.

--wsess=*N1,N2,X*: Solicita a criação e medição de sessões em vez de pedidos individuais. Uma sessão consiste numa sequência de *bursts*. Cada *burst* consiste num número fixo *L* de chamadas ao servidor (*L* é especificado pela opção **--burst-lenth**). As chamadas num *burst* são enviadas da seguinte forma: no início, é enviado um único pedido; Assim que a resposta a este primeiro pedido é recebido, todos os pedidos seguintes no *burst* são enviados simultaneamente. Os pedidos concorrentes são enviados em *pipelined calls*[1][2] através de uma ligação persistente existente, ou como pedidos individuais em ligações separadas. A utilização de uma ligação persistente depende do facto do servidor responder ao primeiro pedido com uma resposta que contém no *header* a informação `Connection: close`. Se essa linha existe na resposta, são utilizadas ligações separadas.

As opções especificam os seguintes parâmetros: *N1* é o número total de sessões a gerar, *N2* é o número de pedidos por sessão e *X* é o *think-time* (em segundos) que separa *burst* consecutivos de pedidos. Por exemplo, as opções **--wsess=100,50,10 --burst-len 5** resulta em 100 sessões com um total de 50 pedidos cada. Tendo em consideração que cada *burst* tem um comprimento de 5 pedidos, um total de 10 *bursts* de pedidos seriam gerados por sessão. O *user think-time* entre *bursts* de pedidos seria de 10 segundos. Deve ser salientado que o *user think-time* denota o tempo entre a última resposta de um *burst* de pedidos anterior e o envio do primeiro pedido do *burst* seguinte.

Um teste envolvendo sessões termina assim que o número de pedidos *N1* tenha sido concluído com sucesso, ou tenha falhado. Uma sessão é considerada como falhada se qualquer operação tiver durado mais do que os *timeout* especificados pela opções **--timeout** e **--think-timeout**. Adicionalmente, uma sessão também falha se o servidor devolve uma resposta com o *status code* especificado pela opção --**failure-status**[3].

O número de sessões escolhido neste trabalho visa, por um lado, ser suficientemente elevado de modo a saturar o servidor Apache, mas não o suficiente para gerar filas de espera extensas que conduzam a uma elevada percentagem de pedidos não satisfeitos por *timeout*. No entanto, de modo a despistar o efeito/influência do número de pedidos nos resultados, são realizados dois testes com cargas diferentes: 25 e 50 pedidos por segundo.

A página solicitada é a de teste do servidor Apache do *CentOS*, a qual tem o aspeto apresentado na figura 3.1.

[1] `http://tools.ietf.org/html/rfc7230#section-6.3.2`
[2] `http://www-archive.mozilla.org/projects/netlib/http/pipelining-faq.html`
[3] Neste trabalho não são utilizados os parâmetros **--think-timeout** e --**failure-status**

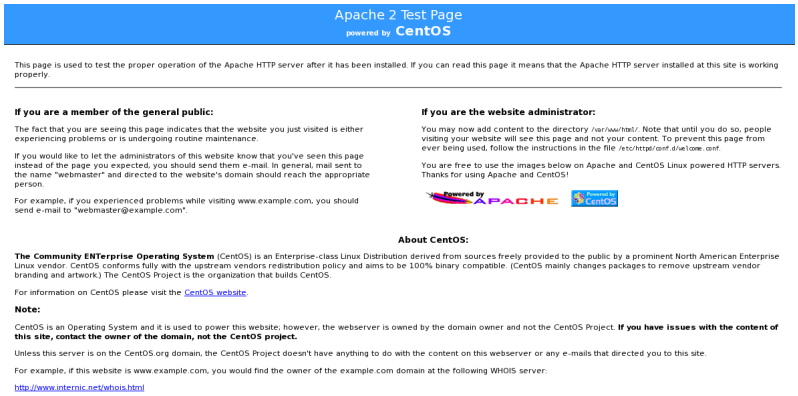


Figura 3.1: Página de teste do servidor HTTP Apache na distribuição Linux CentOS.

A página solicitada é uma página HTML estática com 5043 bytes, já que se pretende que não existam conteúdos dinâmicos, ou acesso a sistemas de gestão de bases de dados (SGBD), os quais poderiam colocar limites/atrasos ao sistema, ou até formar um ponto de congestionamento. A página não deveria igualmente ter dimensões muito reduzidas, já que não permitiria gerar tráfego suficiente, nem ser muito grande, o que criaria tráfego em excesso em cada pedido.

No final de cada execução foi guardado em ficheiro o *output* gerado pelo `httperf`. Os gráficos apresentados no capítulo 4 Apresentação e Discussão de Resultados traduzem os dados registados nesses ficheiros.

Em seguida apresenta-se um exemplo do *output* das estatísticas produzidas pelo `httperf`:

Exemplo de *output* do `httperf`

```
Total: connections 30000 requests 29997 replies 29997 test-duration 299.992 s

Connection rate: 100.0 conn/s (10.0 ms/conn, <=14 concurrent connections)
Connection time [ms]: min 1.4 avg 3.0 max 163.4 median 1.5 stddev 7.3
Connection time [ms]: connect 0.6
Connection length [replies/conn]: 1.000

Request rate: 100.0 req/s (10.0 ms/req)
Request size [B]: 75.0

Reply rate [replies/s]: min 98.8 avg 100.0 max 101.2 stddev 0.3 (60 samples)
Reply time [ms]: response 2.4 transfer 0.0``Apresentação e discussão de resultadosaaa''.
Reply size [B]: header 242.0 content 1010.0 footer 0.0 (total 1252.0)
Reply status: 1xx=0 2xx=29997 3xx=0 4xx=0 5xx=0

CPU time [s]: user 94.31 system 205.26 (user 31.4% system 68.4% total 99.9%)
Net I/O: 129.6 KB/s (1.1*10^6 bps)

Errors: total 3 client-timo 0 socket-timo 0 connrefused 3 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
```

O `httperf` produz seis grupos de estatísticas:

1. Estatísticas globais (`Total`)
2. Resultados relacionados com ligações (`Connection`)
3. Resultados relacionados com os pedidos HTTP (`Request`)
4. Resultados relacionados com as respostas recebidas do servidor (`Reply`)
5. Vários resultados relacionados com o CPU (`CPU`) e Rede (`Net/IO`)
6. Sumário dos erros detetados (`Errors`)

Estatísticas globais (`Total`): Esta secção resume quantas ligações TCP foram iniciadas pelo `httperf`, quantos pedidos foram enviados, quantas respostas foram recebidas e qual a duração total do teste. No exemplo apresentado, foram criadas 30000 ligações, foram enviados 29997 pedidos e foram recebidos 29997 respostas. A duração total do teste foi aproximadamente 5 minutos (300 segundos).

Resultados relacionados com ligações (`Connection`): Esta secção transmite informação relacionada com as ligações TCP geradas pelo `httperf`. A linha `Connection rate` indica que foram iniciadas novas ligações a uma taxa de 100 por segundo. Esta taxa corresponde a um período de 10 ms por ligação. O último número nesta linha indica que no máximo foram abertas 14 ligações num determinado momento.

A primeira linha denominada `Connection time` fornece as estatísticas do "tempo de vida"[1] das ligações com sucesso. O "tempo de vida"de uma ligação é o tempo que decorre entre o inicio de uma ligação TCP e o seu termino, isto é: quando é encerrada. Uma ligação é considerada com sucesso se a mesma teve pelo menos um pedido concluído com sucesso. No exemplo exibido, a linha indica um tempo de vida mínimo (`min`) de 1,4 ms, uma média(`avg`) de 3 ms, um máximo(`max`) de 163,4 ms, uma mediana de 1,5 ms e um desvio padrão dos tempos de vida de 7,3 ms. A mediana do tempo de vida é calculada baseada nas frequências absolutas com 1 segundo de resolução e com um tempo de vida máximo de 100 segundos. Deste modo, a mediana é precisa com a precisão de 1 milissegundo se pelo menos metade das ligações com sucesso tiverem um tempo de vida inferior a 100 segundos.

A estatística seguinte nesta secção é o tempo médio que demorou a estabelecer uma ligação TCP. Só são contabilizadas ligações TCP com sucesso. No exemplo, a segunda linha denominada `Connection time` indica que, em média, demorou 0,6 ms a estabelecer uma ligação.

A última linha nesta secção é denominada `Connection length`. Ela indica o número médio de respostas das ligações que receberam pelo menos uma resposta, isto é: as ligações que falharam antes de obter a primeira resposta não são contabilizadas. Este número pode ser superior a 1 devido a ligações persistentes.

Resultados relacionados com os pedidos HTTP (`Request`): A linha `Request rate` indica a taxa à qual os pedidos HTTP foram enviados e o período a que esta taxa corresponde.

[1] Do inglês, *Lifetime*

No exemplo apresentado, a taxa de pedidos foi de 100 pedidos por segundo, o que corresponde a 10 ms por pedido. Desde que não existam ligações persistentes, os resultados nesta secção são muito semelhantes, ou idênticos, aos resultados da secção `Connection`. No entanto, quando existem ligações persistentes, várias chamadas podem ser realizadas numa única ligação, pelo que os resultados podem ser diferentes.

A linha `Request size` fornece o tamanho médio dos pedidos HTTP em bytes. No exemplo, o tamanho médio do pedido é 75 bytes.

Resultados relacionados com as respostas recebidas do servidor (`Reply`): Para testes mais simples, esta secção é frequentemente a mais interessante já que a linha `Reply rate` fornece várias estatísticas para a taxa de resposta. No exemplo, a taxa mínima de resposta (`min`) foi 98,8 respostas por segundo, a média (`avg`) foi 100 respostas por segundo e a taxa máxima (`max`) foi 101,2 respostas por segundo. O desvio padrão foi 0,3 respostas por segundo. O valor entre parêntesis indica que foram obtidas 60 amostras da taxa de resposta. A versão usada do `httperf` obtém uma amostra a cada 5 segundos. Para obter um desvio padrão significativo é recomendado executar testes suficientemente longos de modo a obter cerca de 50 amostras. Isto corresponde a uma duração de pelo menos 150 segundos. É de recordar que neste trabalho a duração de cada teste foi aproximadamente 3600 segundos.

A linha `Reply time` fornece a informação de quanto tempo o servidor levou a responder e quanto tempo levou a receber a resposta. No exemplo, demorou em média 2,4 ms entre enviar o primeiro byte do pedido e receber o primeiro byte da reposta. O tempo de transferir, ou ler, foi muito curto para ser medido, pelo que é exibido como zero (0). Isto pode suceder quando a reposta cabe num único segmento TCP.

A linha seguinte `Reply size` contém as estatísticas do tamanho médio das respostas. Todos os valores estão em bytes. Esta linha lista especificamente o tamanho médio dos *headers* das repostas, do conteúdo e dos *footers* (o HTTP/1.1 utiliza *footers* para tratar do denominado *chunked transfer encoding*[1]). De modo a ser mais percetível, a média do número total de bytes das respostas é apresentado entre parêntesis. No exemplo, o tamanho médio dos *headers* (`header`) é 242 bytes, a média dos conteúdos (`content`) é 1010 bytes e não existem *footers* (o valor de `footer` é zero). O tamanho médio do total das respostas é 1252 bytes.

A última linha desta secção apresenta as frequências absolutas dos principais códigos recebidos nas respostas do servidor. As respostas estão agregadas de acordo com o algarismo das centenas dos códigos. No exemplo, todas as 29997 respostas tinham o algarismo das centenas dois (2). É admissível estimar que todos os códigos foram `200 OK`, isto é: sucesso. No entanto, a informação das frequências absolutas não é suficientemente detalhada para distinguir os "códigos de estado" com o mesmo código principal[2]. Estes códigos foram explicados na página 9.

[1] `http://www.w3.org/Protocols/HTTP/1.1/draft-ietf-http-v11-spec-01.html`
[2] Do inglês, *Major code*

Resultados relacionados com o CPU (`CPU`) e Rede (`Net/IO`): Esta secção inicia com um sumário da utilização do CPU do cliente. No exemplo, a linha denominada `CPU time` mostra que são despendidos 94,31 segundos de execução em modo utilizador[1] (`user`), 205,26 segundos de execução em modo de sistema[2] (`system`), o que corresponde a 31,4% de execução em modo utilizador e 68,4% em "modo sistema". A utilização total foi 99,9% o que é esperado, tendo em consideração que o `httperf` consume bastante CPU em testes exaustivos. Uma utilização de CPU consideravelmente inferior, significa que existem processos a competir com o `httperf`, o que interfere com os testes.

A linha `Net I/O`fornece o débito médio da rede em kilobytes por segundo (em que um kilobyte corresponde a 1024 bytes) e em megabits por segundo (em que um megabit é 10^6 bits). No exemplo, a utilização média da rede foi de cerca de 129,6 kilobytes por segundo. O número entre parêntesis mostra que isso correspondeu a cerca 1,1 megabits por segundo. A largura de banda é obtida baseada no número de bytes enviados e recebidos nas ligações TCP. Por outras palavras, não tem em conta os *headers*, ou as retransmissões que possam ter ocorrido.

Sumário dos erros detetados (`Errors`): Esta última secção contém as estatísticas dos erros que ocorrem durante um teste. No exemplo, as duas linhas identificadas com `Errors` mostram que houve um total de 3 erros e que todos esses erros são devido ao facto do servidor Apache ter recusado a ligação (`connrefused`). Em seguida apresenta-se a descrição de cada erro.

client-timo: O número de vezes que uma sessão, ligação, ou chamada, falhou devido a um *timeout* do cliente (tal como especificado pelos parâmetros `--timeout` e `--think-timeout`;

socket-timo: O número de vezes que uma ligação TCP falhou com `socket-level timeout` (`ETIMEOUT`);

connrefused: Número de vezes que uma tentativa de ligação TCP falhou com o erro `connection refused by server` (`ECONNREFUSED`);

connreset: Número de vezes que uma ligação TCP falhou devido a `RESET` do servidor. Tipicamente, um `RESET` é recebido quando o cliente tenta enviar dados ao servidor no momento em que o servidor já encerrou o seu lado da ligação;

fd-unavail: Número de vezes que o processo do `httperf` ficou sem descritores de ficheiros. Sempre que este contador for diferente de zero, os resultados não têm qualquer significado, porque o cliente estava sobrecarregado (ver "Escolha do valor do parâmetro **--timeout**" mais adiante).

addrunavail: Número de vezes que o cliente ficou sem portas TCP disponíveis (`EADDRNOTAVAIL`). Este erro nunca deve ocorrer. Se ocorrer, os resultados devem ser descartados.

ftab-full: Número de vezes que a tabela de descritores de ficheiros do sistema está cheia. Tal como no caso do erro `addrunavail`, este erro nunca deve ocorrer. Se ocorrer, os

[1]Do inglês, *user mode*
[2]Do inglês, *system mode*

resultados devem ser descartados.

other: Número de vezes que qualquer outro tipo de erro ocorreu. Sempre que este contador for diferente de zero é necessário avaliar qual a razão. Para auxiliar nesta tarefa, o `httperf` imprime o número do código do erro (`errno`) do primeiro erro que ocorreu durante o teste.

Quando é especificado o parâmetro `--wsess` ou `--wsesslog`, o `httperf` gera e mede sessões em vez de pedidos individuais, bem como estatísticas adicionais, as quais são apresentadas no final do teste. Um exemplo de output é o seguinte:

Exemplo de output adicional do `httperf`

```
Session rate [sess/s]: min 0.00 avg 0.59 max 2.40 stddev 0.37 (240/450)
Session: avg 6.45 connections/session
Session lifetime [s]: 123.9
Session failtime [s]: 58.5
Session length histogram: 4 7 4 ... 3 3 240
```

A linha `Session rate` mostra a taxa mínima e máxima à qual as sessões foram concluídas (baseada em amostras retiradas num intervalo de 5 segundos), bem como também o desvio padrão. Os valores entre parêntesis indicam quantas sessões terminaram com sucesso e quantas foram iniciadas.

A linha `Session` mostra a dimensão média das sessões, medida em ligações (`connections`). No exemplo, foram necessárias em média 6,45 ligações para completar uma sessão.

A linha `Session lifetime` fornece o tempo médio que foi necessário para completar com sucesso uma sessão.

A linha `Session failtime` fornece o tempo médio que demorou até um sessão sem sucesso falhar.

Finalmente, a linha `Session length histogram` fornece as frequências absolutas de respostas recebidas em cada sessão[1][2]. No exemplo, 4 sessões terminaram depois de não receberem qualquer resposta, 7 sessões terminaram após receberem 1 resposta e assim por diante.

Um fator importante é a escolha do valor do parâmetro **--timeout**, já que o equipamento onde é executado o `httperf` tem disponível um conjunto finito de portas; só existem aproximadamente 60000 portas HTTP que podem ser utilizadas num determinado momento.

Tendo em consideração que a maior parte de sistemas `UNIX` demora cerca de um minuto a completar/fechar uma ligação TCP (a abandonar o estado *TIME_ WAIT*), a taxa máxima que um cliente pode suster é cerca de 1000 pedidos por segundo. No entanto, a taxa é frequentemente mais baixa porque provavelmente a máquina antes de ficar sem portas TCP

[1] Os três pontos indicam que existem mais valores mas que foram omitidos, neste exemplo, por questões de espaço.

[2] As frequências absolutas não distinguem entre sessões com sucesso e com falha/erro.

esgota os descritores de ficheiros[1]. Por pré-definição muitos sistemas `UNIX` permitem 1024 descritores de ficheiros por processo. Isto significa que sem precauções extra o `httperf` pode potencialmente utilizar muito rapidamente todos os descritores de ficheiros disponíveis, chegando ao ponto em que não poderia introduzir carga adicional no servidor Apache. Para minorar este problema o `httperf` tem disponível o parâmetro **--timeout** o qual define um valor de *timeout* para todas as comunicações com o servidor. Se o servidor não responder antes do *timeout* expirar, o cliente considera que a sessão correspondente, ligação, ou chamada está "morta", fecha as ligações associadas e incrementa o contador `client-timo`. A única excepção a esta regra é que após o envio de um pedido inteiro ao servidor, o `httperf` permite que o servidor demore algum tempo adicional antes de começar a enviar as respostas. Este aspeto permite que o servidor leve mais algum tempo a processar pedidos que levem mais tempo a processar no servidor. Este tempo adicional é denominado `server think time` e pode ser especificado pelo parâmetro **--think-timeout**. O valor pré-definido de `think time` é zero segundos, pelo que o servidor tem sempre de responder dentro do tempo permitido pelo parâmetro **--think-timeout**. Deve ainda ser salientado que foi devido aos factos anteriormente referidos que foram realizadas as otimizações descritas em 3.2.1.1 na página 49.

Deve ser salientado que, neste trabalho, apenas foram analisadas algumas das estatísticas disponibilizadas pelo `httperf`, as quais serão apresentadas e analisadas no capítulo 4 Apresentação e Discussão de Resultados.

3.2.3.2 *Gateway*

A *Gateway* tem como missão implementar o mecanismo de QoS de acordo com os objetivos definidos e com base na informação recolhida no servidor Apache. A informação obtida no servidor Apache é enviada em intervalos de cerca de 5 segundos para a *Gateway*.

O processo `mst_gateway_server` é executado na *Gateway* e escuta na porta 5000. Após receber a informação do "servidor HTTP Apache modificado", processa e com base na mesma, invoca o *script* `qos.sh` para a definição do controlador. O binário `mst_gateway_server` grava num ficheiro, denominado `output.dat` toda a informação necessária para a análise posterior do comportamento do sistema. No Anexo A Código fonte apresenta-se, na íntegra, o código fonte do binário `mst_gateway_server`.

O binário é invocado da seguinte forma: `./mst_gateway_server p1 p2 p3 p4 p5 p6 p7`. Exemplo: `./mst_gateway_server 60 40 60 0 1200 2400 3600`, em que:

- p1: Percentagem de processos `Premium`: Objetivo do 1º período de 1200 segundos;
- p2: Percentagem de processos `Premium`: Objetivo do 2º período de 1200 segundos;

[1] É utilizado um descritor de ficheiro por cada ligação TCP.

- p3: Percentagem de processos `Premium`: Objetivo do 3º período de 1200 segundos;
- p4: Início do 1º período: 0 segundos ($t_0 = 0s$);
- p5: Início do 2º período: 1200 segundos ($t_1 = t_0 + 1200s$);
- p6: Início do 3º período: 2400 segundos ($t_2 = t_1 + 1200s$);
- p7: Fim do teste: 3600 segundos ($t_3 = t_2 + 1200s$).

Deve ser salientado que foram previstos 3 períodos de 20 minutos de modo a ser possível testar as metodologias "seguimento de referência" e "rejeição de perturbações" (ver 3.2.3.5 Testes. No caso da metodologia "Erro nulo", os valores dos parâmetros p1, p2 e p3 são iguais. O objetivo de processos `Outros` é obtido pelo cálculo: 100 − Objetivo processos `Premium`.

Em seguida, apresenta-se o *script* `qos.sh`. O objetivo deste *script* é implementar o mecanismo de QoS, nomeadamente, a classificação do tráfego, definição e aplicação das classes de serviço.

Gateway - `qos.sh`

```
1  #!/bin/bash
2
3  TC="/sbin/tc"
4
5  NIC="eth0"
6
7  DELAY_PREMIUM=$1
8  DELAY_OUTROS=$2
9  DELAY_UN="ms"
10 NETEM_DELAY_PREMIUM="${DELAY_PREMIUM}${DELAY_UN}"
11 NETEM_DELAY_OUTROS="${DELAY_OUTROS}${DELAY_UN}"
12
13 PORTA="ip dport 80 0xffff"
14
15 DST_PREMIUM="ip dst 192.168.100.101"
16 DST_OUTROS="ip dst 192.168.100.102"
17
18 ${TC} qdisc del dev ${NIC} root
19
20 ${TC} qdisc add dev ${NIC} handle 1: root htb default 13
21
22 ${TC} class add dev ${NIC} parent 1: classid 1:1 htb rate 1000mbit
23
24 ${TC} class add dev ${NIC} parent 1:1 classid 1:11 htb rate 100mbit ceil 100mbit prio 0
25 ${TC} class add dev ${NIC} parent 1:1 classid 1:12 htb rate 100mbit ceil 100mbit prio 0
26 ${TC} class add dev ${NIC} parent 1:1 classid 1:13 htb rate 100mbit ceil 100mbit prio 0
27
28 ${TC} filter add dev ${NIC} parent 1: protocol ip prio 2 u32 match ${DST_PREMIUM} match $
       ↪{PORTA} flowid 1:11
29 ${TC} filter add dev ${NIC} parent 1: protocol ip prio 2 u32 match ${DST_OUTROS} match ${
       ↪PORTA} flowid 1:12
30
31 ${TC} qdisc add dev ${NIC} parent 1:11 handle 10: netem delay ${NETEM_DELAY_PREMIUM}
32 ${TC} qdisc add dev ${NIC} parent 1:12 handle 20: netem delay ${NETEM_DELAY_OUTROS}
```

Até à linha 16 são definidas algumas variáveis para tornar o ficheiro mais legível/simples.

Linha 5 : Define a interface onde o *script* irá aplicar o QoS. É de salientar que o QoS foi

implementado na *interface* eth0 (192.168.100.1) da *Gateway* já que, tal como referido na revisão bibliográfica, o `tc`[1] funciona no controlo de saída (*egress shapping*) e não no de entrada (*ingress shapping*). Na figura 3.3 está representado o sentido da aplicação do QoS (*egress*);

Linha 7 e 8 : Variáveis que contêm o valor dinâmico de atraso a aplicar aos pacotes destinados aos sítios `Premium` e `Outros`, respetivamente;

Linha 9 : Unidade de atraso: milissegundos (ms);

Linha 13 : Porta de destino dos pacotes. Neste caso, porta 80 (HTTP);

Linha 15 e 16 : IP dos *Virtual Host* Apache: 192.168.1.101 (sítio `Premium`) e 192.168.1.102 (sítio `Outros`);

Linha 18 : Elimina, caso exista, alguma disciplina de fila[2] associada à *interface* eth0;

Linha 20 : Atribuí a disciplina de fila `HTB` à *interface* eth0 e atribui-lhe a designação `1:`. Esta designação é apenas um identificador para posterior referência. É, igualmente, definido que todo o tráfego que não seja marcado explicitamente pelas regras/filtros definidos nas linhas 28 e 29, é atribuído à classe de serviço `1:13` (`default`);

Linha 22 : Define a classe de topo: `root`. A classe é identificada como `1:1` e tem como limite de velocidade 1000 mbit/s;

- Linhas 24, 25 e 26 - definição das 3 classes de serviço que eram utilizadas pelos pacotes.
- Classe 1:11: Pacotes marcados como destinados ao sítio `Premium`;
- Classe 1:12 Pacotes marcados como destinados ao sítio `Outros`;
- Classe 1:13 Restantes pacotes. Tal como referido anteriormente, esta classe foi definida como a classe padrão (`default`).

Linha 28 e 29 : Marcação/Classificação dos pacotes destinados ao sítio `Premium` e `Outros`, respetivamente. Os critérios de marcação são os seguintes:

- protocol ip : Protocolo do pacote. Neste caso IP;
- prio 2: prioridade das classes folha[3]. Quanto menor é este valor, maior é a prioridade;
- u32 match ${DST_PREMIUM} match ${PORTA} e ${DST_OUTROS} match ${PORTA}: Pacotes destinados aos *virtual host* do servidor Apache: A classificação (`match`) é feita de acordo com o IP de destino e com a porta 80.

Linhas 31 e 32 : Definição do atraso a aplicar aos pacotes classificados nas linhas 28 e 29 e pertencentes às classes `1:11` e `1:12`

- handle 10: Esta designação é apenas um identificador para referência;
- handle 20: Esta designação é apenas um identificador para referência;
- netem delay: Define o atraso, em milissegundos, a aplicar aos pacotes.

Nota: O `tc` segue a seguinte convenção para as unidades da opção `rate`: kbps significa KB/s

[1] Do inglês, *Traffic control*
[2] Do inglês, *Queue discipline*
[3] Do inglês, *Leaf classes*

(kilobytes/segundo) e kbit significa Kb/s (kilobits/segundo)[1].

Na figura 3.2 apresenta-se o esquema resumido do mecanismo de QoS resultante da aplicação do *script* anterior.

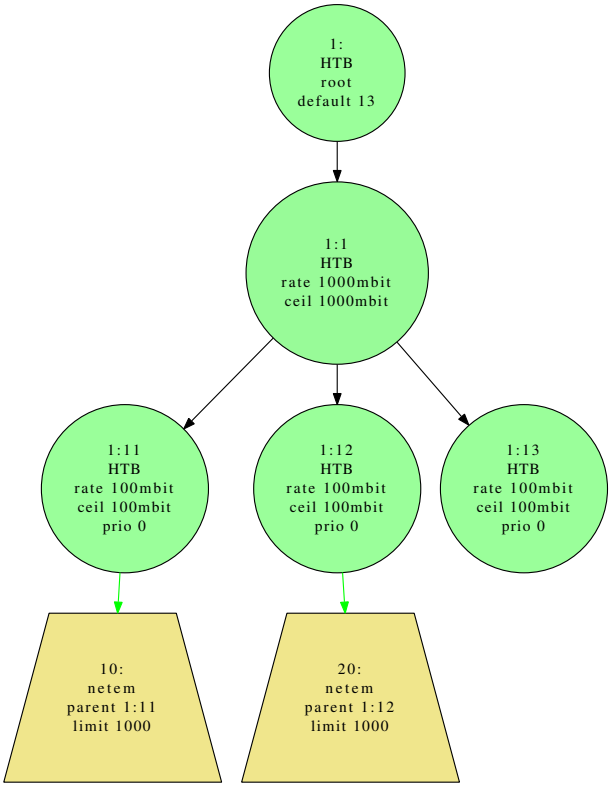


Figura 3.2: Esquema simplicado do mecanismo de QoS da *Gateway*.

Na figura 3.3 é indicado o sentido de aplicação do mecanismo de QoS na *Gateway*. Tal como referido anteriormente, o QoS é aplicado em *egress shaping* no sentido: *interface* eth0 da *Gateway* → Servidor Apache.

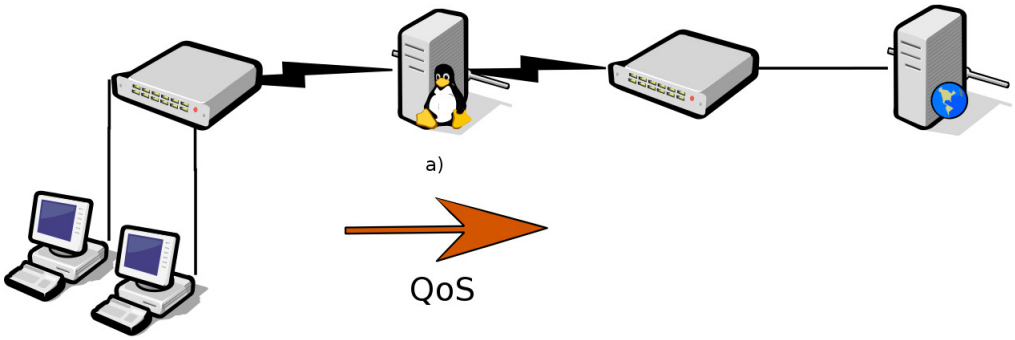


Figura 3.3: Esquema simplificado da infraestrutura e sentido do mecanismo de QoS da *Gateway*.

O mecanismo de QoS implementado utiliza como principais ferramentas o `tc`, o `HTB` e o

[1] `http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm`

`NetEm` com recurso a prioridades (`PRIO`). Os conceitos destas ferramentas são explicados em detalhe na revisão bibliográfica.

3.2.3.3 Controlador Proporcional-Integral-Derivativo(PID)

Para aferir os valores adequados para as constantes do controlador PID são realizados vários testes, com várias combinações de valores das constantes proporcional, integral e diferencial. Algumas das combinações testadas não permitiram a obtenção dos resultados pretendidos[1] tal como, por exemplo:

Constante proporcional	Constante integral	Constante diferencial
$K_p = 100$	$K_i = 0$	$K_d = 0$

A combinação que produziu melhor resultados foi a seguinte:

Constante proporcional	Constante integral	Constante diferencial
$K_p = 5$	$K_i = 3$	$K_d = 8$

Estas constantes foram definidas no ficheiro `server.h`.

Gateway - `server.h` (excerto)

```
...
#define kpPremium 5
#define kiPremium 3
#define kdPremium 8

#define kpOutros 5
#define kiOutros 3
#define kdOutros 8
...
```

3.2.3.4 Servidor HTTP Apache

A classificação dos tipos de sítios `Premium` e `Outros` é definida no ficheiro `def_vhosts.conf`, o qual está no servidor HTTP. O seu conteúdo é:

Servidor Apache - `def_vhosts.conf`

```
# P = Premium ; O = Outro

192.168.100.101 = P

192.168.100.102 = O
```

[1] ver capítulo Apresentação e Discussão de Resultados

A leitura do ficheiro `def_vhosts.conf` é efetuada pela função `void mst_inits()` do ficheiro `mst_ue.c`. Em seguida apresenta-se a referida função.

Servidor Apache - `mst_ue.c` (excerto)

```
void mst_inits() {

    FILE *fp2;
    fp2 = fopen(FILE_IN_DEF_VHOSTS, "r");
    if (fp2 != NULL) {
        //char linha[100], *vhostN, *vhostT;
        char linha[100];
        while (fgets(linha, sizeof linha, fp2) != NULL) {
            if (linha[0] == '#' || linha[0] == '\n') {
                continue;
            }
            if (sscanf(linha, "%32s = %2s", &vhostV[nVHosts].vhost[0],
                    &vhostV[nVHosts].tipo[0]) == 2) {
                vhostV[nVHosts].cont = 0;
                nVHosts++;
            }
        }
        fclose(fp2);
    }

}
```

Tal como referido anteriormente, são definidos dois *virtual host*: `Premium` e `Outros`. A saída do comando `httpd -S` é o seguinte:

Servidor Apache - *Virtual Hosts*

```
VirtualHost configuration:
192.168.100.101:* 192.168.100.101 (/etc/httpd/conf/httpd.conf:243)
192.168.100.102:* 192.168.100.102 (/etc/httpd/conf/httpd.conf:247)
wildcard NameVirtualHosts and _default_ servers:
_default_:443 192.168.100.100 (/etc/httpd/conf.d/ssl.conf:81)
Syntax OK
```

3.2.3.5 Testes

Após a implementação do sistema, procedeu-se à simulação, registo e análise dos resultados do sistema, dos clientes HTTP e do servidor Apache. O mecanismo de QoS é sujeito a várias simulações de carga. Nos clientes são analisados parâmetros tais como: o tempo de resposta dos diferentes sítios, número de pedidos e de respostas de cada sítio Internet. No servidor Apache são analisados parâmetros como a utilização de CPU (%), memória utilizada (KB) e outros considerados relevantes.

São, ainda, conduzidos para cada uma das cargas três metodologias, designadamente:

- Erro nulo(0);
- Seguimento de referência;
- Rejeição de perturbações.

Os aspetos particulares destes três conceitos foram abordados na revisão bibliográfica e a análise detalhada dos resultados obtidos será realizada em 4 Apresentação e Discussão de Resultados.

Tal com referido anteriormente, cada teste foi feito com a duração de aproximadamente 3600 segundos ($\simeq 1hora$). No caso do "Erro 0", o teste tem um período único de 3600 segundos, enquanto no "Seguimento de referência" e "Rejeição de perturbações" existem 3 períodos de 1200 segundos (20 minutos). Tendo em consideração que o servidor Apache envia à *Gateway* informação em intervalos de 5 segundos, em cada teste são recolhidas cerca de 720 amostras.

A *Gateway* produz vários ficheiros de saída. Estes ficheiros são analisados para obter os resultados apresentados em 4 Apresentação e Discussão de Resultados. Sendo admissível que a produção de ficheiros de *output* possa ter originado alguma carga na *Gateway*, devido a operações de leitura/escrita (I/O), foi admitido que não teve influência no sistema e que a verificar-se seria igual para as duas classes de sítios (`Premium` e `Outros`).

No sentido de verificar qual o comportamento do sistema sem o controlador (inativo) são executadas simulações para cada uma das cargas.

3.2.4 Outros

Foi igualmente instalado um servidor de NTP (*Network Time Protocol*[1]) na *Gateway* de modo a que todos os equipamentos pudessem sincronizar a hora do sistema e não haver desfasamento aquando das tarefas agendadas, nomeadamente nos casos do "Seguimento de referência" e da "Rejeição de perturbações".

Deve ser salientado que algum do código na *Gateway*, mais concretamente no controlador (`./mst_gateway_server`), só existe devido à necessidade de proporcionar um ambiente para as metodologias "Seguimento de referência" e "Rejeição de perturbações", no qual o objetivo definido tem de mudar ao longo do tempo. Exemplos:

Gateway - `server.c` (excerto 1)

```
...
tDiffRel1 = atof(argv[4]);
tDiffRel2 = atof(argv[5]);
tDiffRel3 = atof(argv[6]);
tDiffRel4 = atof(argv[7]);
...
```

[1] http://www.ntp.org/

Gateway - `server.c` (excerto 2)

```
...
 while (1) {

       tActual = time(0);
       tDiffAbs = difftime(tActual, tInicio);

       if (tDiffAbs >= tDiffRel4) {
           exit(0);
       } else {
           if (tDiffAbs >= tDiffRel1 && tDiffAbs < tDiffRel2) {
               setPointWorkersPremiumActual_Rel = setPointWorkersPremium1_Rel;
               setPointWorkersOutrosActual_Rel = setPointWorkersOutros1_Rel;
               if (sysIntv == 0) {
                   reset();
                   setPID();
                   sysIntv = 1;
               }
           } else if (tDiffAbs >= tDiffRel2 && tDiffAbs < tDiffRel3) {
               setPointWorkersPremiumActual_Rel = setPointWorkersPremium2_Rel;
               setPointWorkersOutrosActual_Rel = setPointWorkersOutros2_Rel;
               if (sysIntv == 1) {
                   reset();
                   setPID();
                   sysIntv = 2;
               }
           } else if (tDiffAbs >= tDiffRel3 && tDiffAbs < tDiffRel4) {
               setPointWorkersPremiumActual_Rel = setPointWorkersPremium3_Rel;
               setPointWorkersOutrosActual_Rel = setPointWorkersOutros3_Rel;
               if (sysIntv == 2) {
                   reset();
                   setPID();
                   sysIntv = 3;
               }
           }
       }
...
```

4. Apresentação e Discussão de Resultados

Neste capítulo procede-se à apresentação e discussão dos resultados mais relevantes, obtidos pela aplicação da metodologia apresentada no capítulo anterior (3. Equipamentos e Metodologia).

Os resultados estão organizados da seguinte forma:

1. Sistema em cadeia fechada sem controlador
   (a) Sem controlador e sem saturação (20 sessões por segundo (20ss));
   (b) Sem controlador e com saturação (50 sessões por segundo (50ss)).
2. Sistema em cadeia fechada com controlador
   (a) Controlador Proporcional (P);
   (b) Controlador Proporcional-Integral-Derivativo (PID).
      i. Teste 1 (50 sessões por segundo (50ss));
      ii. Teste 2 (25 sessões por segundo (25ss)).

4.1 Sistema de controlo em cadeia fechada sem controlador

Antes de avaliar o comportamento do servidor Apache com o sistema de controlo em cadeia fechada proposto, importa analisar o mesmo com a sua política padrão de resposta a pedidos, isto é: `FIFO`. Neste cenário, o *script* `qos.sh` residente na *Gateway* está desativado, isto é: todas as linhas estão comentadas, pelo que não força qualquer implementação de QoS. Neste contexto, vamos analisar dois cenários:

Sem controlador e sem saturação : A carga dos clientes (20 sessões por segundo, 20ss) não é suficiente para esgotar a capacidade do servidor Apache;

Sem controlador e com saturação : A carga dos clientes (50 sessões por segundo, 50ss) ultrapassa a capacidade do servidor Apache, registando-se assim uma competição pelos recursos disponíveis.

Convém recordar que a capacidade, neste caso concreto, diz respeito ao número máximo de processos-filho que o servidor Apache tem disponível: 256, os quais são configurados no parâmetro `MaxClients`. Quando os pedidos excedem esta capacidade, eles são colocados numa fila de espera, até ser atingido o valor definido no parâmetro `ListenBacklog`.

4.1.1 Sistema de controlo em cadeia fechada sem controlador e sem saturação

Na figura 4.1 apresenta-se a percentagem absoluta de processos-filho HTTP `Ativos`, `Premium` e `Outros` no servidor Apache. É possível constatar que não existe saturação da capacidade do servidor Apache, já que o número de processos `Ativos` ronda habitualmente os 55%, estando o seu valor compreendido entre os 30% e os 80%.

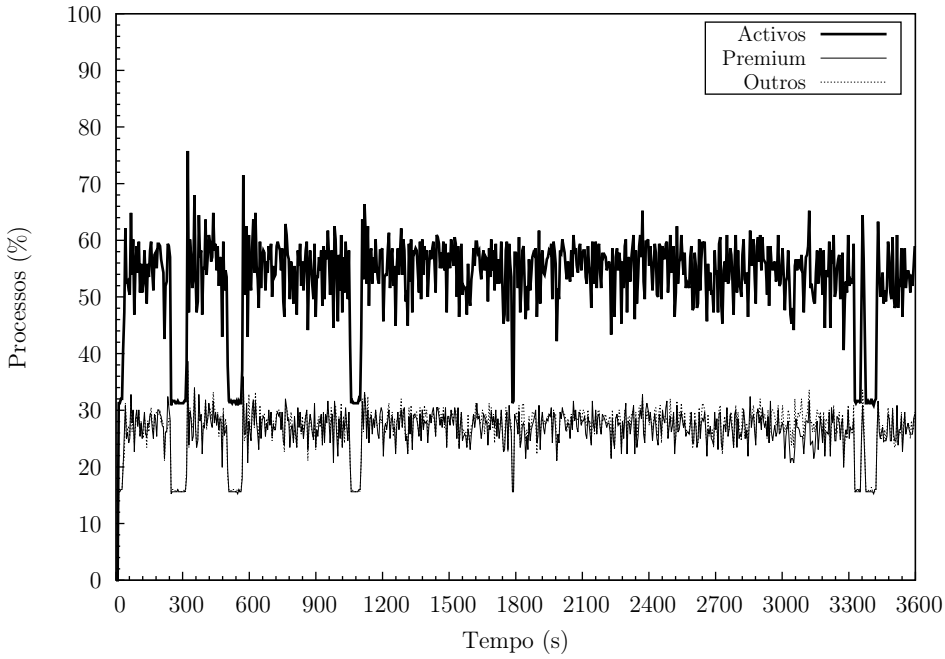


Figura 4.1: Sistema de controlo em cadeia fechada sem controlador e sem saturação. 20ss. Erro nulo. Servidor Apache. Percentagem absoluta de processos.

Analisando as percentagens relativas dos processos-filho HTTP (de agora em diante referidos como processos), constata-se que não existe diferença significativa entre o sítio `Premium` e o sítio `Outros`. O sistema não faz qualquer diferenciação entre os pedidos do cliente HTTP do sítio `Premium` (de agora em diante referido como cliente `Premium`) e os pedidos do cliente HTTP do sítio `Outros` (de agora em diante referido como cliente `Outros`). Este é o comportamento esperado do sistema, já que o servidor Apache tem capacidade para receber/processar os pedidos dos clientes e está configurado para satisfazer os mesmos conforme vão ocorrendo.

As percentagens relativas (PR) são calculadas da seguinte forma:

$$PR_{\text{Processos sítio } \texttt{Premium}}\ (\%) = \frac{\text{Processos sítio } \texttt{Premium}\ (\text{n})}{\text{Processos } \texttt{Ativos}\ (\text{n})} \cdot 100 \tag{4.1}$$

$$PR_{\text{Processos sítio } \texttt{Outros}}\ (\%) = \frac{\text{Processos sítio } \texttt{Outros}\ (\text{n})}{\text{Processos } \texttt{Ativos}\ (\text{n})} \cdot 100 \tag{4.2}$$

Tendo em consideração a natureza estocática dos pedidos, existem instantes em que a percentagem de processos do sítio `Outros` é superior à percentagem de processos do sítio `Premium` e vice-versa.

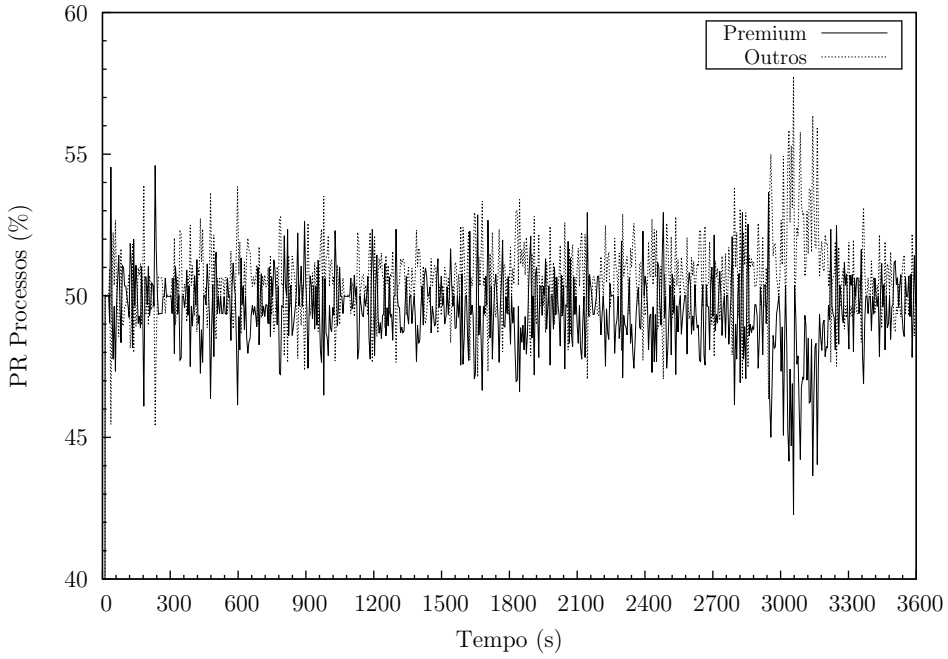


Figura 4.2: Sistema de controlo em cadeia fechada sem controlador e sem saturação. 20ss. Erro nulo. Servidor Apache. Percentagem relativa de processos `Ativos`.

Quer o CPU (figura 4.3), quer a memória (figura 4.4), não foram fatores com influência no resultado. O CPU durante o período de observação esteve cerca de 95% *idle*, ou seja: sem processar. A utilização de memória ronda os 512MB, valor bastante inferior ($\approx 25\%$) à capacidade do servidor Apache: 2GB.

Tal como sucedeu com o CPU e a memória, também o tráfego de rede gerado por estes pedidos não foi fator com influência nos resultados (figura 4.5). O tráfego de rede, recebido (R) e transmitido (T), que passou pela *Gateway* foi sempre inferior a 30Mb/s, valor bastante aquém da capacidade das interfaces de rede: 1Gb/s. Deve ser recordado que a `interface eth0` pertence à rede do servidor Apache (192.168.100.0/24) e que a `interface eth1` pertence à rede dos clientes `Premium` e `Outros`.

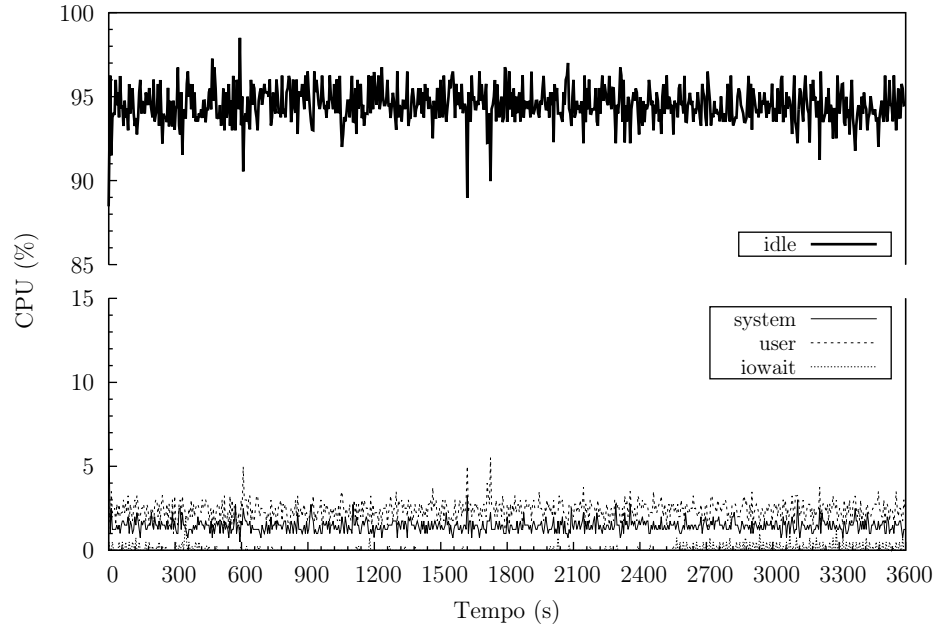


Figura 4.3: Sistema de controlo em cadeia fechada sem controlador e sem saturação. 20ss. Erro nulo. Servidor Apache. CPU.

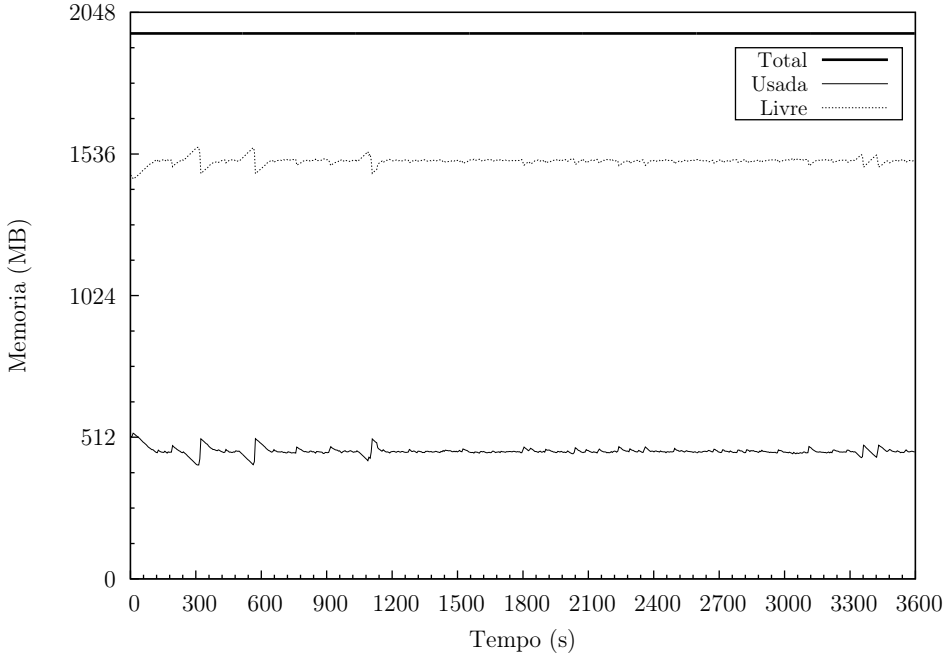


Figura 4.4: Sistema de controlo em cadeia fechada sem controlador e sem saturação. Servidor Apache. 20ss. Erro nulo. Memória.

Em seguida, apresentam-se os resultados obtidos pelo simulador `httperf` nos clientes.

httperf - Cliente `Premium`

```
httperf --hog --timeout=5 --client=0/1 --server=192.168.100.101 --port=80 --uri=/index.
    ↪html --rate=20 --send-buffer=4096 --recv-buffer=16384 --wsess=300000,10,2.000 --
    ↪burst-length=5
^CMaximum connect burst length: 11

Total: connections 75500 requests 726901 replies 723726 test-duration 3601.080 s
```

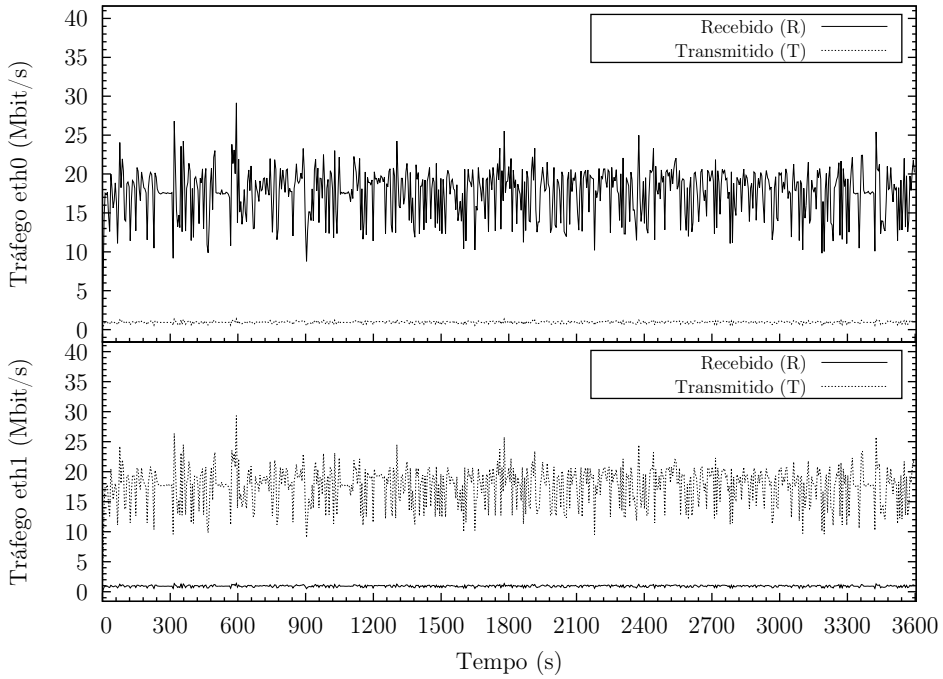


Figura 4.5: Sistema de controlo em cadeia fechada sem controlador e sem saturação. 20ss. Erro nulo. `Gateway`. Tráfego de rede.

```
Connection rate: 20.9 conn/s (48.0 ms/conn, <=126 concurrent connections)
Connection time [ms]: min 2.9 avg 3557.5 max 10145.8 median 3730.5 stddev 834.5
Connection time [ms]: connect 293.3
Connection length [replies/conn]: 9.596

Request rate: 200.7 req/s (5.0 ms/req)
Request size [B]: 78.0

Reply rate [replies/s]: min 108.8 avg 199.9 max 259.0 stddev 21.0 (724 samples)
Reply time [ms]: response 144.4 transfer 79.9
Reply size [B]: header 251.0 content 5043.0 footer 0.0 (total 5294.0)
Reply status: 1xx=0 2xx=723726 3xx=0 4xx=0 5xx=0

CPU time [s]: user 1898.12 system 1720.10 (user 52.4% system 47.5% total 99.9%)
Net I/O: 1048.6 KB/s (8.6*10^6 bps)

Errors: total 3079 client-timo 1 socket-timo 0 connrefused 0 connreset 3078
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 9.60 avg 19.98 max 26.40 stddev 2.16 (72344/72345)
Session: avg 1.04 connections/session
Session lifetime [s]: 3.7
Session failtime [s]: 5.6
Session length histogram: 0 0 1 0 0 0 0 0 0 0 72344
```

httperf - Cliente `Outros`

```
httperf --hog --timeout=5 --client=0/1 --server=192.168.100.102 --port=80 --uri=/index.
    ↪html --rate=20 --send-buffer=4096 --recv-buffer=16384 --wsess=300000,10,2.000 --
    ↪burst-length=5
Maximum connect burst length: 16

Total: connections 75703 requests 727065 replies 723638 test-duration 3600.476 s

Connection rate: 20.9 conn/s (47.8 ms/conn, <=123 concurrent connections)
```

```
Connection time [ms]: min 3.0 avg 3633.8 max 11085.4 median 3815.5 stddev 862.5
Connection time [ms]: connect 293.5
Connection length [replies/conn]: 9.569

Request rate: 200.8 req/s (5.0 ms/req)
Request size [B]: 78.0

Reply rate [replies/s]: min 109.2 avg 199.9 max 265.8 stddev 22.7 (724 samples)
Reply time [ms]: response 153.7 transfer 84.0
Reply size [B]: header 251.0 content 5043.0 footer 0.0 (total 5294.0)
Reply status: 1xx=0 2xx=723638 3xx=0 4xx=0 5xx=0

CPU time [s]: user 1852.63 system 1761.84 (user 51.2% system 48.7% total 99.8%)
Net I/O: 1048.7 KB/s (8.6*10^6 bps)

Errors: total 3295 client-timo 2 socket-timo 0 connrefused 0 connreset 3293
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 9.20 avg 19.98 max 27.80 stddev 2.29 (72328/72330)
Session: avg 1.05 connections/session
Session lifetime [s]: 3.8
Session failtime [s]: 6.9
Session length histogram: 0 0 1 0 0 1 0 0 0 0 72328
```

A taxa de ligações (`Connection rate`), de pedidos (`Request rate`) é muito próximo em ambos os clientes, o que demonstra que os dois têm a mesma capacidade para gerar pedidos/carga no servidor Apache. A taxa de ligações semelhante permite a ambos os clientes manter um número próximo de ligações concorrentes: até 126, no caso do cliente `Premium` e até 123, no caso do cliente `Outros`.

A taxa média de respostas (`Reply rate`) comprova que o servidor Apache não realizou qualquer diferenciação entre os clientes; em ambos casos o valor foi 199,9 repostas/s. O intervalo da taxa de respostas também é muito semelhante. Se tivermos em consideração que a taxa média de pedidos (pedidos/s) foi 200,7 no cliente `Premium` e 200,8 no cliente `Outros`, é demonstrado que o servidor Apache tem, neste caso concreto, capacidade para responder às solicitações dos clientes.

As respostas obtidas pelos clientes foram todas do tipo 2xx, ou seja: "Sucesso", o que indica que a ação foi recebida com sucesso, compreendida e aceite.

Convém chamar a atenção para o facto do parâmetro `wsess` nos clientes ser elevado de modo a que os mesmos não terminem os pedidos durante o período de observação (3600s), o que inviabilizaria os resultados. Assim, é normal que o número total de pedidos seja inferior ao valor estipulado no parâmetro `wsess`: 300000 sessões com 10 pedidos cada.

Outro aspeto a ter em atenção é que, devido à opção mencionada no parágrafo anterior (parâmetro `wsess`), a interrupção dos pedidos nos clientes é manual. Deste modo, no que diz respeito ao tempo de observação, existem alguns milissegundos de diferença entre os dois clientes e alguns segundos de diferença entre as várias metodologias (< 5s). Esta diferença resulta basicamente da combinação de dois fatores: tempo de reação humano e tempo de

reação da aplicação `httperf` ao comando de interrupção (`Control-C`). Contudo, tendo em consideração o tempo de observação: 3600s, julgamos que esta diferença não tem influência significativa nos resultados obtidos pelos clientes. Devemos, igualmente, salientar que a análise dos resultados privilegia sempre os dados resultantes da relação observações/tempo (taxa de ligações: ligações/s; taxa de pedidos: pedidos/s; taxa de respostas: respostas/s), de rácios (taxa de respostas/taxa de pedidos) e de percentagens (taxa de pedidos cliente `Outros`/taxa de pedidos cliente `Premium`), o que torna a influência desta discrepância quase nula.

No que diz respeito a erros, o seu número total é baixo, registando-se valores insignificantes de `timeout` nos clientes `Premium` e `Outros` (1 e 2, respetivamente). Constata-se que a maior parte dos erros foi devido a `connreset`. Este valor diz respeito ao número de vezes que uma ligação falhou devido a um `RESET` do servidor Apache. Tipicamente, um `RESET` sucede quando um cliente tenta enviar dados para o servidor Apache numa altura em que o servidor Apache já encerrou a seu lado da ligação.

Analisando as frequências absolutas regista-se que, em ambos os clientes, quase todas as sessões terminam depois de receberem o décimo pedido.

Em ambos os casos o número de observações foi superior a 720, o que torna os resultados significativos. De acordo com a documentação [30], devem executar-se testes suficientemente longos, de modo a obter pelo menos 50 amostras, para que os dados sejam significativos.

4.1.2 Sistema de controlo em cadeia fechada sem controlador e com saturação

Na figura 4.6 apresenta-se a percentagem absoluta de processos. Ao contrário do que sucede no caso anterior (sem saturação), o número total de processos `Ativos` é quase sempre 100%. Constata-se, assim, a saturação quase permanente do servidor Apache.

Tal como sucede no caso sem saturação, a percentagem relativa de processos do sítio `Premium` e do sítio `Outros`, não evidencia diferença significativa, tal como era expectável (figura 4.7).

Mais uma vez, regista-se a natureza estocática dos pedidos, existindo momentos em que o número de processos do sítio `Outro` é superior aos do sítio `Premium` e vice-versa.

Embora o CPU não seja fator limitativo, ou com influência nos resultados, regista-se uma maior utilização do mesmo (figura 4.8). Durante a maior parte do período de observação o CPU esteve `idle`, entre os 85% e os 90%.

No que diz respeito à memória (figura 4.9), a sua utilização é muito estável rondando os 512 MB, o que representa cerca de 25% da capacidade do servidor Apache. É de recordar

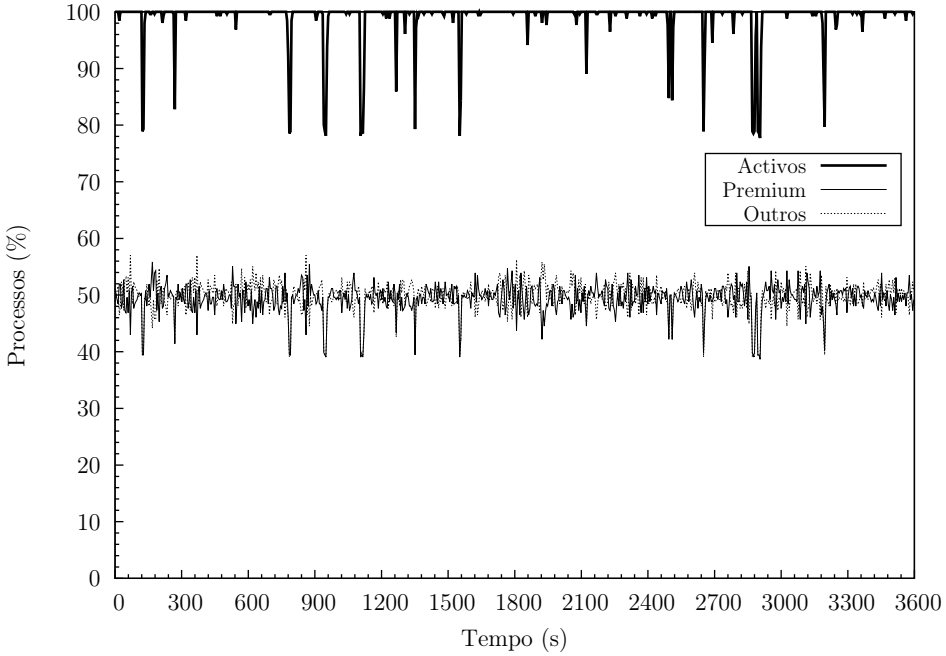


Figura 4.6: Sistema de controlo em cadeia fechada sem controlador e com saturação. 50ss. Erro nulo. Servidor Apache. Percentagem absoluta de processos.

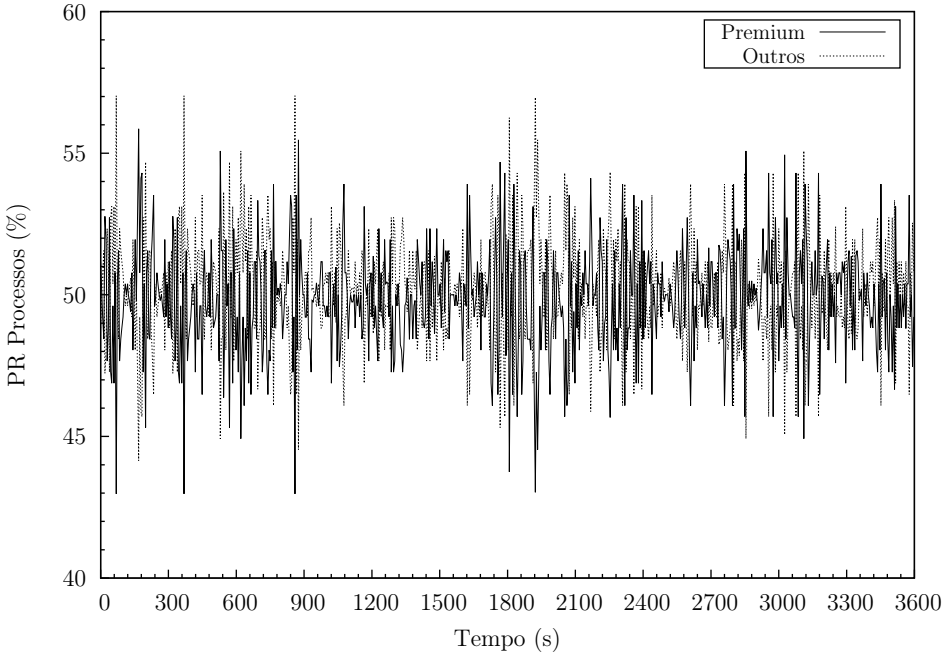


Figura 4.7: Sistema de controlo em cadeia fechada sem controlador e com saturação. 50ss. Erro nulo. Servidor Apache. Percentagem relativa de processos `Ativos`.

que na configuração do servidor Apache o parâmetro `StartServers` é igual ao parâmetro `MaxClients` e `ServerLimit`: 256 (ver quadro 3.3), pelo que o servidor Apache inicia a sua atividade logo com a capacidade máxima de processos. Assim sendo, não existe um aumento no número de processos, o que aumentaria também a memória utilizada.

Neste cenário e como resultado do aumento considerável dos pedidos dos clientes `Outros`

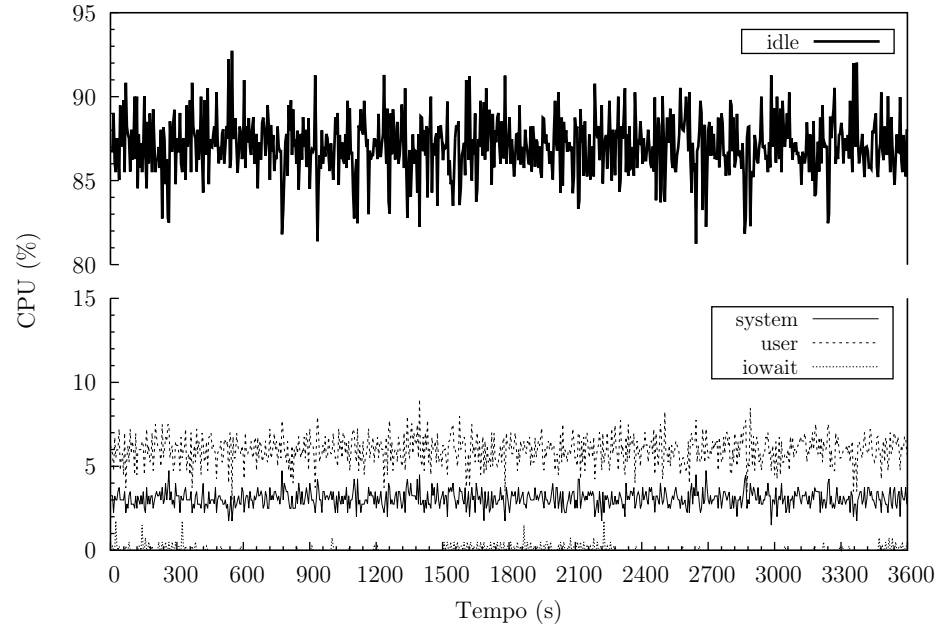


Figura 4.8: Sistema de controlo em cadeia fechada sem controlador e com saturação. 50ss. Erro nulo. Servidor Apache. CPU.

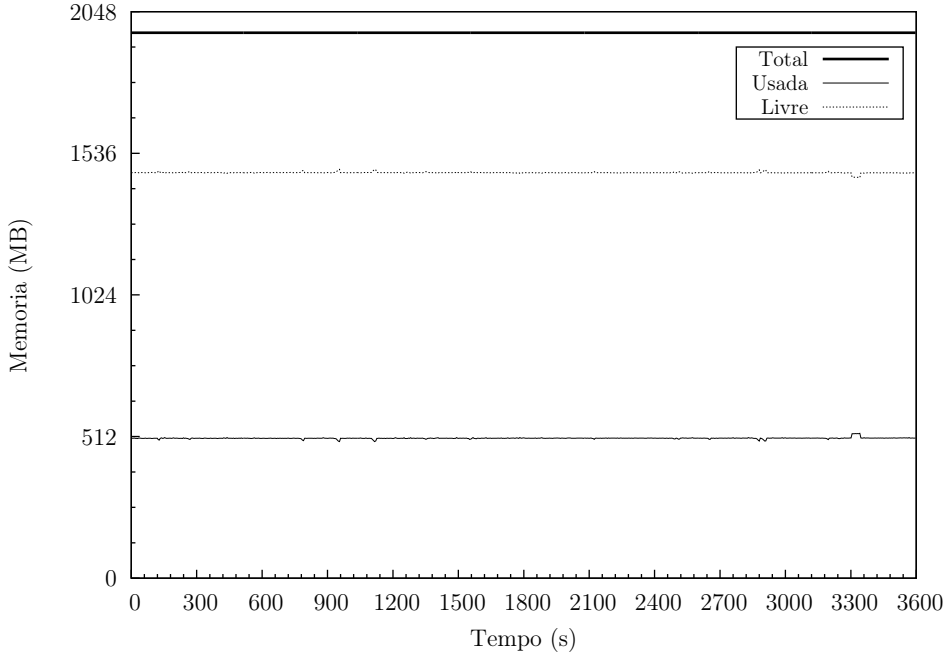


Figura 4.9: Sistema de controlo em cadeia fechada sem controlador e com saturação. 50ss. Erro nulo. Servidor Apache. Memória.

e `Premium` e das respostas do servidor Apache, regista-se um maior tráfego nas interfaces de rede da *Gateway*: entre os 20Mbit/s e os 60Mbit/s (figura 4.10). Estes valores são, no entanto, bem inferiores à capacidade das interfaces de rede: 1Gb/s.

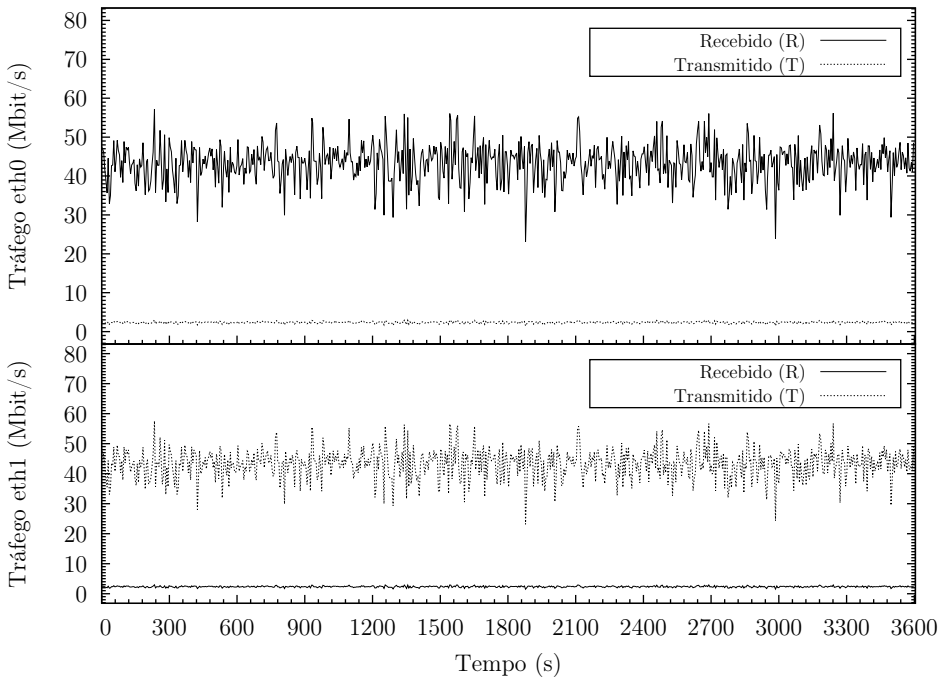


Figura 4.10: Sistema de controlo em cadeia fechada sem controlador e com saturação. 50ss. Erro nulo. *Gateway*. Tráfego de rede.

Tal como era expectável, o aumento, de 20 para 50 sessões por segundo (parâmetro `rate`) no `httperf` aumenta o número total de ligações, sessões, pedidos e respostas. O aumento é praticamente na mesma proporção, isto é: um pouco superior ao dobro. Esta alteração conduz ao aumento do número de ligações concorrentes (até 294, no cliente `Premium` e até 288, no cliente `Outros`), o que se traduz na saturação do servidor Apache.

Esta alteração, produz um aumento significativo do número de erros, em particular ao `timeout` por parte do cliente. No caso do cliente `Premium` de 1 para 4356 e do cliente `Outros`, de 2 para 3791.

Deve ser salientada a capacidade de resposta do servidor Apache, tendo a taxa média de respostas (repostas/s) passado de 199,9 para cerca de 490, ou seja: quase proporcional ao aumento do parâmetro `rate`. Tal sucedeu, porque no cenário sem controlador e sem saturação o número de processos `Ativos` do servidor Apache estava compreendido entre os 50% e os 60%, ou seja: quase metade da sua capacidade. Assim sendo, foi possível a sua quase duplicação.

O aumento expressivo dos erros devido a `timeout` resultou do facto do servidor Apache não ter capacidade imediata de resposta aos pedidos e alguns deles terem eventualmente ficado tempo demais na fila de espera sem terem sido respondidos. Não existe, contudo, informação sobre o tempo que os pedidos estiveram na fila de espera. Convém recordar que o valor de `timeout` é usado quando se estabelece a ligação TCP, quando envia um pedido, quando espera por uma resposta e quando recebe uma resposta. Se a duração de alguma destas atividades num pedido não consegue progredir, no tempo definido, o `httperf` considera que

o pedido morreu, fecha a ligação ou sessão associada e aumenta o contador da estatística `client-timo`.

httperf - Cliente sítio `Premium`

```
httperf --hog --timeout=5 --client=0/1 --server=192.168.100.101 --port=80 --uri=/index.
    ↪html --rate=50 --send-buffer=4096 --recv-buffer=16384 --wsess=300000,10,2.000 --
    ↪burst-length=5
^CMaximum connect burst length: 9

Total: connections 181320 requests 1763609 replies 1759592 test-duration 3603.140 s

Connection rate: 50.2 conn/s (19.9 ms/conn, <=294 concurrent connections)
Connection time [ms]: min 167.6 avg 3987.5 max 13901.5 median 3875.5 stddev 1046.4
Connection time [ms]: connect 353.8
Connection length [replies/conn]: 9.956

Request rate: 488.7 req/s (2.0 ms/req)
Request size [B]: 78.0

Reply rate [replies/s]: min 385.2 avg 487.4 max 648.0 stddev 52.3 (721 samples)
Reply time [ms]: response 180.8 transfer 31.6
Reply size [B]: header 251.0 content 5043.0 footer 0.0 (total 5294.0)
Reply status: 1xx=0 2xx=1759592 3xx=0 4xx=0 5xx=0

CPU time [s]: user 1118.72 system 2487.55 (user 31.0% system 68.9% total 99.9%)
Net I/O: 2557.9 KB/s (21.0*10^6 bps)

Errors: total 5228 client-timo 4356 socket-timo 0 connrefused 0 connreset 872
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 29.80 avg 48.73 max 68.81 stddev 5.86 (175852/180208)
Session: avg 1.00 connections/session
Session lifetime [s]: 4.0
Session failtime [s]: 5.3
Session length histogram: 4315 11 0 4 0 25 0 1 0 0 175852
```

httperf - Cliente sítio `Outros`

```
httperf --hog --timeout=5 --client=0/1 --server=192.168.100.102 --port=80 --uri=/index.
    ↪html --rate=50 --send-buffer=4096 --recv-buffer=16384 --wsess=300000,10,2.000 --
    ↪burst-length=5
^CMaximum connect burst length: 16

Total: connections 181218 requests 1768188 replies 1764784 test-duration 3602.954 s

Connection rate: 50.2 conn/s (19.9 ms/conn, <=288 concurrent connections)
Connection time [ms]: min 2.6 avg 3927.6 max 12727.1 median 3837.5 stddev 997.0
Connection time [ms]: connect 353.7
Connection length [replies/conn]: 9.957

Request rate: 490.1 req/s (2.0 ms/req)
Request size [B]: 78.0

Reply rate [replies/s]: min 395.0 avg 489.2 max 658.6 stddev 51.7 (721 samples)
Reply time [ms]: response 173.8 transfer 31.7
Reply size [B]: header 251.0 content 5043.0 footer 0.0 (total 5294.0)
Reply status: 1xx=0 2xx=1764784 3xx=0 4xx=0 5xx=0

CPU time [s]: user 1142.82 system 2460.28 (user 31.7% system 68.2% total 99.9%)
Net I/O: 2566.2 KB/s (21.0*10^6 bps)

Errors: total 4611 client-timo 3791 socket-timo 0 connrefused 0 connreset 820
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
```

```
Session rate [sess/s]: min 30.00 avg 48.89 max 66.40 stddev 5.68 (176382/180173)
Session: avg 1.00 connections/session
Session lifetime [s]: 3.9
Session failtime [s]: 5.3
Session length histogram: 3725 8 1 2 4 38 3 6 1 3 176382
```

Tendo em consideração os resultados obtidos pelo sistema em cadeia fechada sem controlador (inativo) com e sem saturação, é possível comprovar que o servidor Apache tem o comportamento esperado, isto é, não faz qualquer diferenciação entre os pedidos do cliente `Premium` e do cliente `Outros`. Os pedidos são satisfeitos de acordo com a entrada dos mesmos. Este aspeto é extremamente importante, porque demonstra que a alteração do código fonte do servidor Apache e a inclusão do sistema de controlo em cadeia fechada (inativo) não tem influência significativa no comportamento do mesmo. Estamos, deste modo, em condições de verificar qual o comportamento do servidor Apache com o sistema de controlo em cadeia fechada ativo, sem o receio da interferência das alterações produzidas ao nível do código fonte.

4.2 Sistema de controlo em cadeia fechada com controlador

4.2.1 Controlador Proporcional (P)

Para verificar qual o comportamento do sistema com um controlador proporcional, são definidos as seguintes constantes: $k_p = 100, k_i = 0, k_d = 0$. Neste cenário, a taxa de sessões é de 50 por segundo e cada sessão efetua 50 pedidos, já que se pretende avaliar o comportamento do servidor Apache com saturação. Recordemos que, neste trabalho, se pretende verificar o comportamento do sistema de controlo em cadeia quando existe competição intensiva pelos recursos do servidor Apache.

Em seguida vamos avaliar o comportamento do servidor Apache com o objetivo "Erro nulo (0)". O objetivo (*setpoint*) definido é: 80% da capacidade do servidor Apache, expressa em processos `Ativos`, a processar pedidos do sítio `Premium` e os remanescentes 20% a processar pedidos do sítio `Outros`. Recordemos, novamente, que nos estamos a referir a percentagens relativas, isto é: relativas ao número de processos `Ativos`.

A percentagem absoluta de processos (figura 4.11) permite constatar que o objetivo de saturação quase nunca é atingido sendo, no entanto, a percentagem de processos `Ativos` muito elevada. A razão deste comportamento do sistema será explicado em detalhe, quando analisarmos o atraso aplicado aos pacotes destinados aos dois sítios.

O sistema nunca estabiliza, sendo a oscilação total: entre os 0% e os 100%. Este aspeto é ainda mais percetível na figura 4.12.

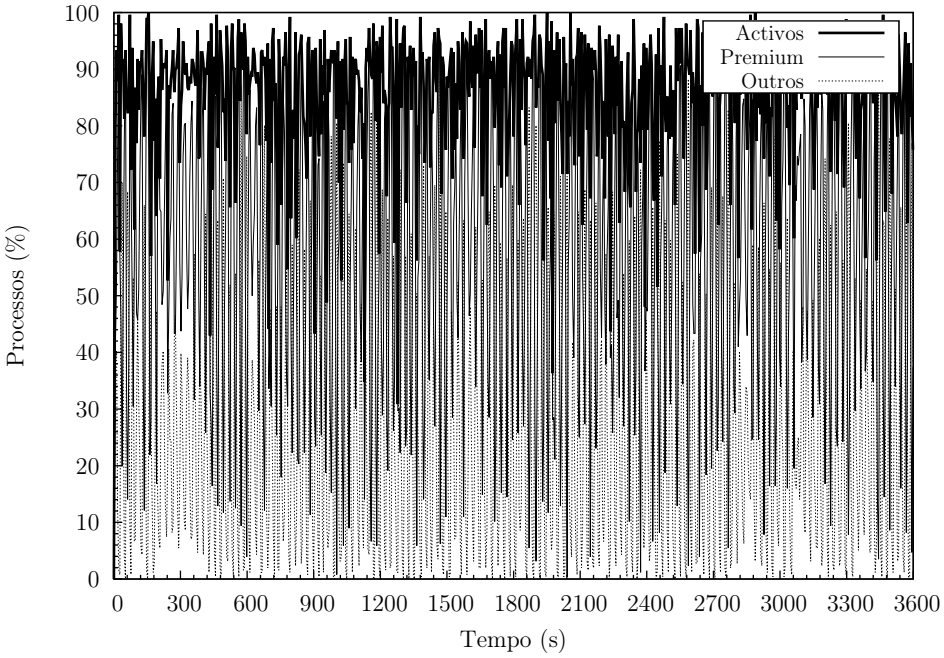


Figura 4.11: Sistema de controlo em cadeia fechada com controlador proporcional (k_p=100). 50ss. Erro nulo. Servidor Apache. Percentagem absoluta de processos.

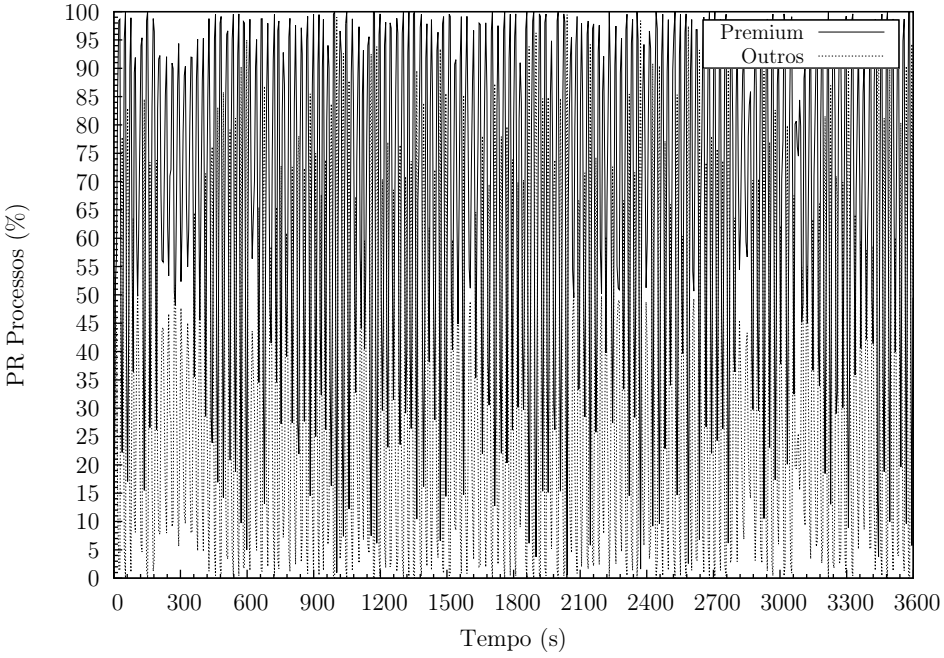


Figura 4.12: Sistema de controlo em cadeia fechada com controlador proporcional (k_p=100). 50ss. Erro nulo. Servidor Apache. Percentagem relativa de processos `Ativos`.

Como resultado da oscilação das percentagens e da não estabilização do sistema nos objetivos definidos (80% `Premium`; 20% `Outros`), o erro (sendo $erro = objetivo - valor\ medido$)

é sempre elevado, atingindo em alguns casos os 80% (figura 4.13). Estamos, assim, na presença de oscilações duráveis com elevada amplitude.

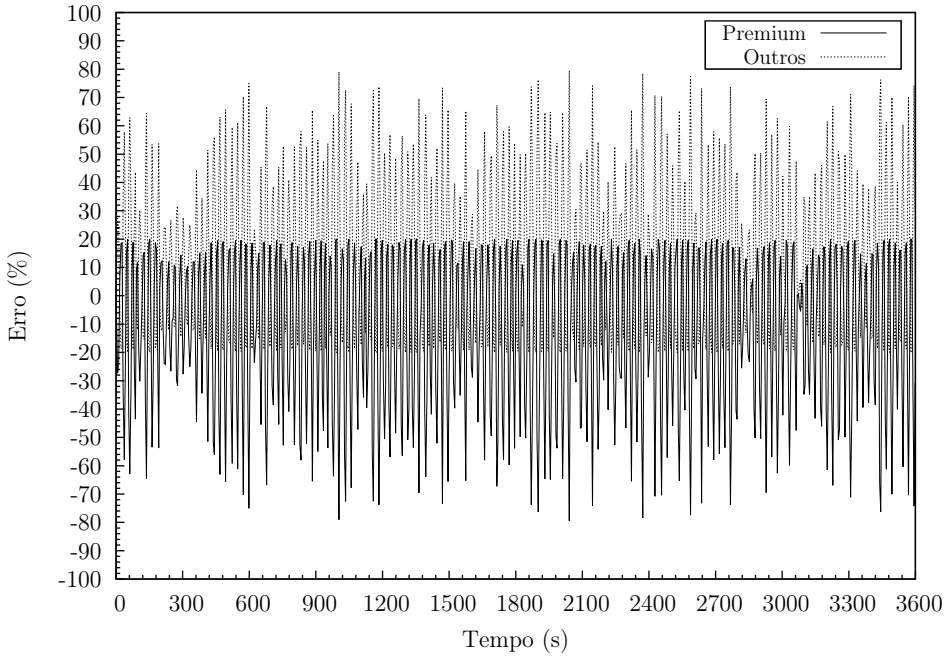


Figura 4.13: Sistema de controlo em cadeia fechada com controlador proporcional (k_p=100). 50ss. Erro nulo. Servidor Apache. Erro.

A razão pela qual o sistema não estabiliza está relacionada com o valor da constante do termo proporcional (k_p) escolhido. O seu valor é propositadamente bastante elevado para se constatar que o sistema não estabiliza. Os resultados apresentados estão de acordo com o que foi anteriormente referido no ponto 2.4.4 da revisão bibliográfica, isto é, quando se aumenta bastante o valor de k_p, o sistema pode oscilar permanentemente, nunca estabilizando de acordo com os objetivos definidos [51].

No arranque do sistema e conforme os pedidos vão chegando, o sistema deteta que não estão a ser cumpridos os objetivos estabelecidos, razão pela qual começa a impor atraso aos pacotes destinados ao sítio `Outros`, mas também aos pacotes destinados ao sítio `Premium` (figura 4.14). Os atrasos atingem, no caso do sítio `Outros`, inclusivamente os 5000ms, valor máximo de atraso definido no código (ver A.1 *Gateway*: `server.h`).

Os elevados atrasos impostos aos pacotes de ambos os clientes condicionam claramente o seu desempenho, mas de forma mais acentuada no cliente `Outros`. Este aspeto é claramente visível no indicador `timeout`, conforme será visto em pormenor na análise dos clientes. O facto de serem aplicados atrasos a todos os pacotes, leva a que por vezes o servidor Apache não tenha a sua capacidade total ocupada. Esta é a razão pela qual, mesmo com a carga imposta no sistema, a capacidade do servidor Apache não é esgotada. É, igualmente, por esta razão que o sistema nunca estabiliza e está alternadamente a impor valores elevados de atraso aos pacotes destinados ao dois sítios. Convém, recordar que o atraso a aplicar os pacotes resulta da atividade do transdutor do sistema, isto é: da conversão das percentagens

medidas e do erro apurado, em atrasos a aplicar aos pacotes.

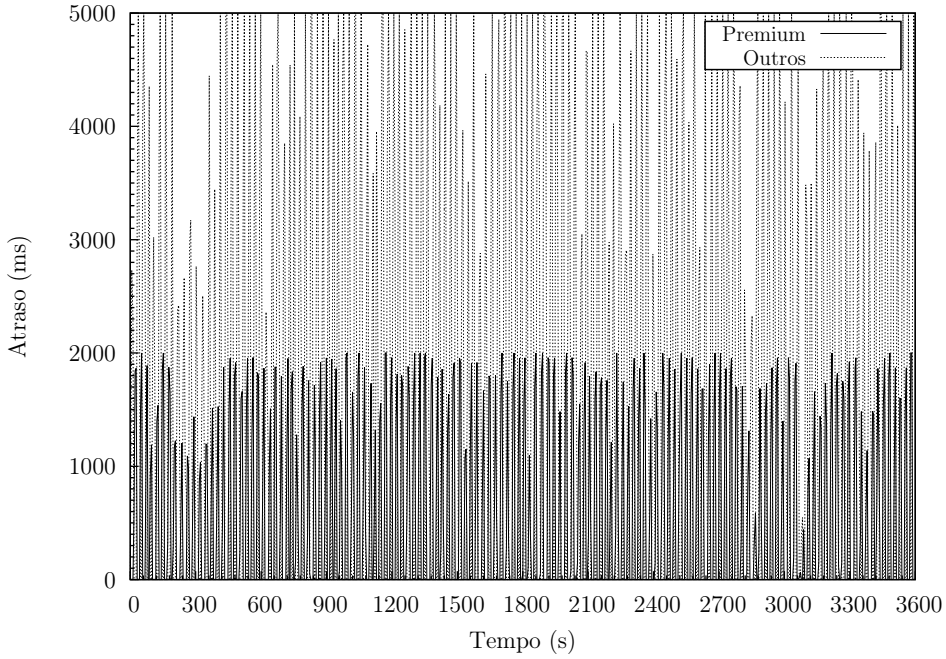


Figura 4.14: Sistema de controlo em cadeia fechada com controlador proporcional (k_p=100). 50ss. Erro nulo. *Gateway*. Atraso.

Tal como sucede nos casos anteriores, nem o CPU (4.15), nem a memória (4.16) são fatores com influência dos resultados. O processador está 95% do tempo `idle` e a memória utilizada ronda os 512MB ($\approx$ 25% da capacidade do servidor Apache).

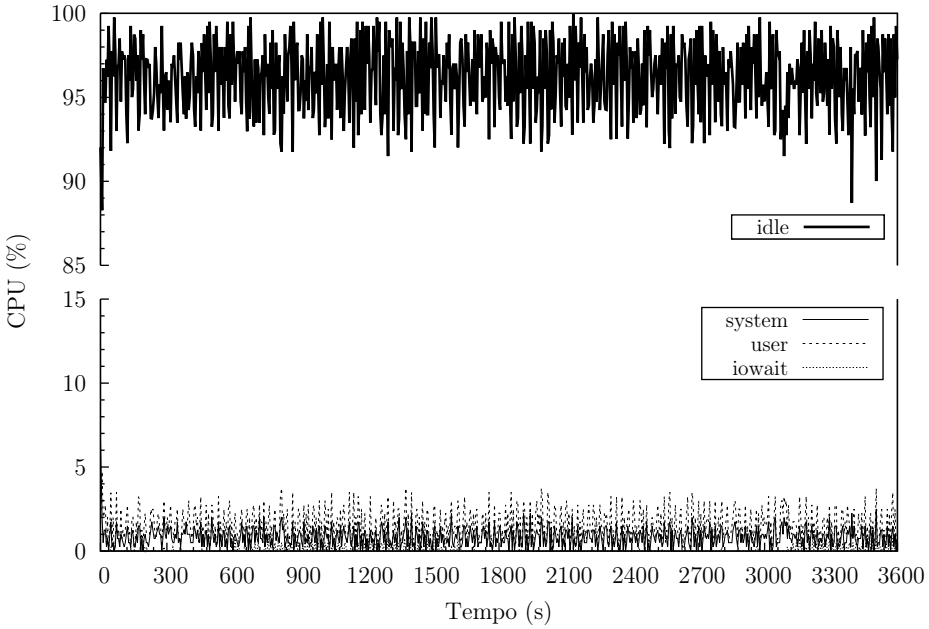


Figura 4.15: Sistema de controlo em cadeia fechada com controlador proporcional (k_p=100). 50ss. Erro nulo. Servidor Apache. CPU.

Devido ao facto de, de um modo geral, o servidor Apache quase nunca estar saturado, exibindo até em algumas situações uma baixa percentagem de processos `Ativos` ($\approx$ 30%),

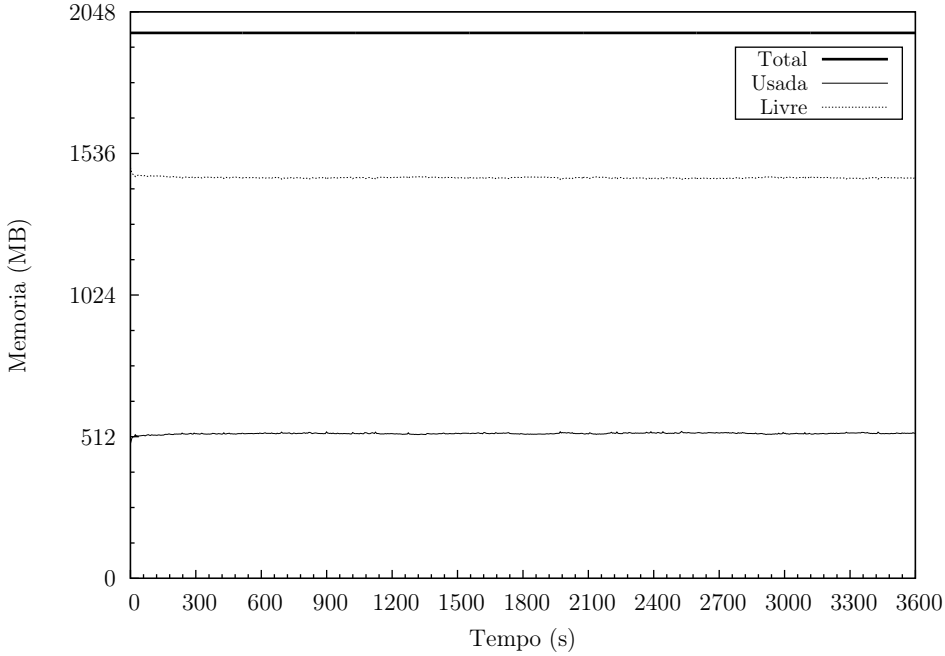


Figura 4.16: Sistema de controlo em cadeia fechada com controlador proporcional (k_p=100). 50ss. Erro nulo. Servidor Apache. Memória.

o tráfego de rede (figura 4.17) exibe um comportamento mais semelhante ao sistema sem controlador e sem saturação, do que ao sistema sem controlador e com saturação. Como é lógico, neste sistema, o tráfego de rede, salvo algum fator externo (que não se regista) resulta do volume de informação a transferir dos pedidos dos clientes e das respostas do servidor Apache.

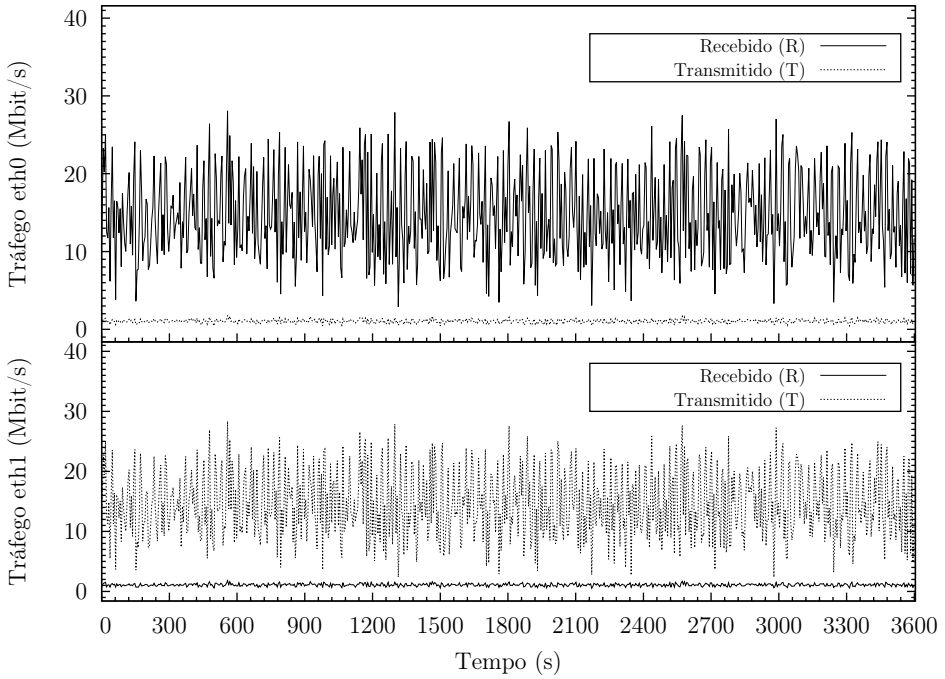


Figura 4.17: Sistema de controlo em cadeia fechada com controlador proporcional (k_p=100). 50ss. Erro nulo. *Gateway*. Tráfego de rede.

Com este controlador proporcional, constata-se o seguinte:

- A taxa de ligações dos dois clientes é superior à registado no caso do sistema sem controlador (com e sem saturação);
- A taxa de pedidos no cliente `Premium` é um pouco superior ao sistema sem controlador e sem saturação, mas bastante inferior (menos de metade) ao sistema sem controlador e com saturação. No caso do cliente `Outros` a taxa de pedidos é 19,7% relativamente ao sistema sem controlador e com saturação e 45,7% relativamente ao sistema sem controlador e sem saturação;
- A taxa média de respostas no cliente `Premium` é um pouco superior ao sistema sem controlador e sem saturação, mas bastante inferior (menos de metade) ao sistema sem controlador e com saturação, o que acompanha o registado na taxa de pedidos. No caso do cliente `Outros` a taxa média de respostas é 13,5% relativamente ao sistema sem controlador e com saturação e 60,5% relativamente ao sistema sem controlador e sem saturação;

Estes resultados indicam que nos dois clientes muitas sessões não conseguem realizar o número de pedidos para os quais estão configurados e/ou não recebem respostas do servidor Apache. Este aspeto é confirmado se tivermos em consideração a linha das frequências absolutas das sessões (`Session length histogram`). Recordemos que a linha `Session length histogram` fornece o número de respostas recebidas em cada sessão. No cliente `Premium`, 142581 sessões terminam após não receberem qualquer resposta, 3533 sessões terminam após receberem 1 resposta e assim por diante. No cliente `Outros`, 155761 sessões terminam após não receberem qualquer resposta, 3975 sessões terminam após receberem 1 resposta e assim por diante. Nos casos analisados anteriormente, estes valores são consideravelmente inferiores.

O cliente `Outros` é o mais penalizado, tal como esperado. O sua taxa de ligações foi 93,5% relativamente ao cliente `Premium`, mas a sua taxa média de respostas só representa 38,21%. Este resultado é o esperado se tivermos em consideração o atraso imposto aos pacotes destinados ao sítio `Outros`: até 5000ms (figura 4.14).

Todos as respostas são do tipo 2xx (Sucesso), ou seja: "Sucesso", o que indica que a ação foi recebida com sucesso, compreendida e aceite.

Embora o número de erros seja algo superior no cliente `Premium`, esta diferença justifica-se também pelo maior número de ligações, sessões e pedidos realizado por este cliente. Convém salientar, que o facto de se privilegiar os pedidos do cliente `Premium`, não lhe garante o serviço, isto é: perante um grande afluxo de pedidos, alguns podem sofrer algum atraso, ou até serem descartados. Neste caso, esse aspeto ainda é mais notório, porque o comportamento oscilante do sistema leva a que, por vezes, o sistema também imponha atrasos significativos nos pacotes do cliente `Premium`.

httperf - Cliente sítio `Premium`

```
httperf --hog --timeout=5 --client=0/1 --server=192.168.100.101 --port=80 --uri=/index.
    ↪html --rate=50 --send-buffer=4096 --recv-buffer=16384 --wsess=300000,50,2.000 --
    ↪burst-length=5
Maximum connect burst length: 139

Total: connections 217236 requests 784622 replies 623345 test-duration 3599.788 s

Connection rate: 60.3 conn/s (16.6 ms/conn, <=694 concurrent connections)
Connection time [ms]: min 1355.8 avg 13260.5 max 41863.0 median 12421.5 stddev 5203.5
Connection time [ms]: connect 1624.0
Connection length [replies/conn]: 15.554

Request rate: 218.0 req/s (4.6 ms/req)
Request size [B]: 81.0

Reply rate [replies/s]: min 0.0 avg 173.2 max 554.8 stddev 156.2 (720 samples)
Reply time [ms]: response 397.7 transfer 86.5
Reply size [B]: header 251.0 content 5042.0 footer 0.0 (total 5293.0)
Reply status: 1xx=0 2xx=623345 3xx=0 4xx=0 5xx=0

CPU time [s]: user 447.54 system 3148.74 (user 12.4% system 87.5% total 99.9%)
Net I/O: 912.5 KB/s (7.5*10^6 bps)

Errors: total 216339 client-timo 179093 socket-timo 0 connrefused 0 connreset 37246
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 0.13 max 11.80 stddev 0.65 (481/179574)
Session: avg 1.99 connections/session
Session lifetime [s]: 34.9
Session failtime [s]: 8.7
Session length histogram: 142581 3533 725 397 64 2767 328 12 68 12 4536 384 11 113 33
    ↪6344 367 29 84 81 5781 425 79 138 85 5630 317 77 90 66 1876 89 17 43 24 1170 99 16
    ↪22 10 404 11 4 4 7 127 6 3 3 1 481
```

httperf - Cliente sítio `Outros`

```
httperf --hog --timeout=5 --client=0/1 --server=192.168.100.102 --port=80 --uri=/index.
    ↪html --rate=50 --send-buffer=4096 --recv-buffer=16384 --wsess=300000,50,2.000 --
    ↪burst-length=5
Maximum connect burst length: 99

Total: connections 202986 requests 330575 replies 237689 test-duration 3599.646 s

Connection rate: 56.4 conn/s (17.7 ms/conn, <=551 concurrent connections)
Connection time [ms]: min 3404.8 avg 9964.1 max 27696.0 median 9672.5 stddev 2978.9
Connection time [ms]: connect 1799.3
Connection length [replies/conn]: 9.777

Request rate: 91.8 req/s (10.9 ms/req)
Request size [B]: 79.0

Reply rate [replies/s]: min 0.0 avg 66.1 max 406.4 stddev 90.3 (719 samples)
Reply time [ms]: response 419.1 transfer 96.0
Reply size [B]: header 251.0 content 5042.0 footer 0.0 (total 5293.0)
Reply status: 1xx=0 2xx=237689 3xx=0 4xx=0 5xx=0

CPU time [s]: user 493.66 system 3084.08 (user 13.7% system 85.7% total 99.4%)
Net I/O: 348.5 KB/s (2.9*10^6 bps)

Errors: total 202729 client-timo 179726 socket-timo 0 connrefused 0 connreset 23003
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 0.00 max 0.00 stddev 0.00 (0/179726)
```

```
Session: avg 0.00 connections/session
Session lifetime [s]: 0.0
Session failtime [s]: 6.9
Session length histogram: 155761 3975 230 345 14 4429 259 23 130 14 4117 360 18 259 19
    ↪7614 218 15 44 48 1501 47 4 3 3 213 12 3 4 0 38 0 0 0 0 2 0 0 1 0 1 0 0 0 0 2
```

Tendo em consideração os resultados obtidos, podemos concluir que o sistema com este controlador tem um comportamento ainda pior do que sem controlador.

Embora a perceção de QoS dos serviços em "tempo real", por parte dos utilizadores, seja subjetiva e esteja relacionada/condicionada por vários aspetos (ex.: atraso, *jitter*, perda de pacotes, *etc.*) [43], os atrasos provocados nos pacotes e os valores obtidos nas ligações, respostas, erros e estatísticas de sessão, certamente se traduziriam numa experiência de navegação lenta e desagradável por parte dos utilizadores. Esta impressão, por parte dos utilizadores, resultaria do facto de muitos pedidos não serem respondidos, dando a sensação do sítio estar indisponível/com problemas.

Esta constatação reforça a noção que deve ser prestada muita atenção à introdução deste tipo de alterações nos sistemas em produção porque pode não se melhorar o seu desempenho, podendo até piorar. Como é natural esta situação é indesejável.

Devido ao desempenho/comportamento insatisfatório do sistema, não se considera relevante realizar os testes de "Seguimento de referência" e "Rejeição de perturbações".

Deve-se mencionar que foram realizados vários testes, com diferentes valores da constante proporcional, mas que também produziram resultados insatisfatórios. Para valores muito baixos de k_p (< 2), o sistema não consegue fazer a diferenciação pretendida entre o cliente `Premium` e o cliente `Outros`, enquanto que para valores elevados k_p (> 10), o sistema apresenta oscilações duráveis com elevada amplitude.

4.2.2 Controlador Proporcional-Integral-Derivativo (PID)

4.2.2.1 Teste 1 - 50 sessões por segundo

Em seguida, analisa-se o comportamento do sistema fechado com um controlador proporcional-integral-derivativo (PID) e as constantes definidas do seguinte modo: $k_p = 5, k_i = 3, k_d = 8$. A taxa de sessões é de 50 por segundo (50ss) e cada sessão efetua 50 pedidos.

Erro nulo

Para verificar se o sistema consegue realizar a diferenciação de serviço pretendida, ser robusto e estável, vamos observar o seu comportamento com o objetivo "Erro nulo (0)". Tal como sucedeu no controlador proporcional, o objetivo (*setpoint*) definido é: 80% da capacidade

do servidor Apache, expressa em processos `Ativos`, a processar pedidos do sítio `Premium` e os remanescentes 20% a processar pedidos do sítio `Outros`.

A percentagem de processos `Ativos` é muito elevado, estando quase sempre compreendida entre os 90% e os 100% (figura 4.18). É possível observar que existe uma clara diferenciação entre o cliente do sítio `Premium` e o cliente do sítio `Outros`.

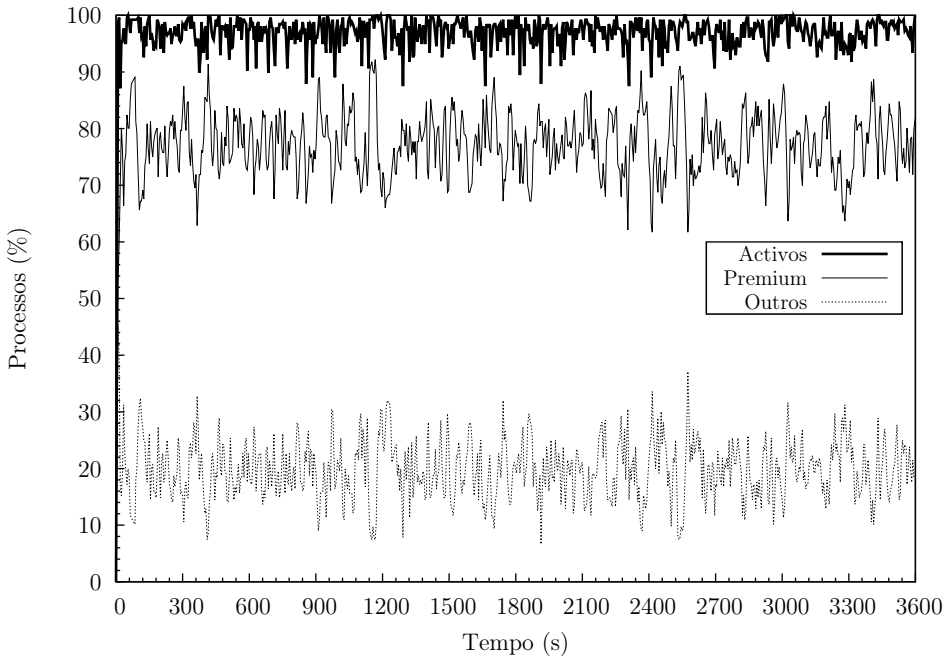


Figura 4.18: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 50ss. Erro nulo. Servidor Apache. Percentagem absoluta de processos.

Na figura 4.19 constatamos, tal como já era possível observar na figura 4.18, que o objetivo definido foi atingido. A percentagem de processos a executar pedidos do sítio `Premium` é tendencialmente 80% e do sítio `Outros` é tendencialmente 20%. A natureza estocástica dos pedidos HTTP não permite atingir, na perfeição, o objetivo definido, pelo que este comportamento do sistema não está relacionado com o seu incorreto funcionamento.

No que diz respeito ao erro (figura 4.20), o mesmo está quase sempre compreendido entre os -10% e os +10%, o que é bastante bom tendo em consideração a carga imposta ao servidor Apache e a variabilidade dos pedidos.

O sistema consegue ajustar dinamicamente o atraso nos pacotes de modo a atingir o objetivo definido (figura 4.21). Tal como esperado, é quase sempre imposto exclusivamente atraso ao pacotes do cliente `Outros`. Existem dois momentos em que é imposto também atraso aos pacotes do cliente `Premium`, mas tal facto não está associado a uma efetiva necessidade de atraso nesses pacotes. Convém recordar que no código do sistema foram implementadas algumas funcionalidades para comportar as metodologias "Seguimento de referência" e "Rejeição de perturbações". Nessas metodologias, o período de observação de 3600s é dividido

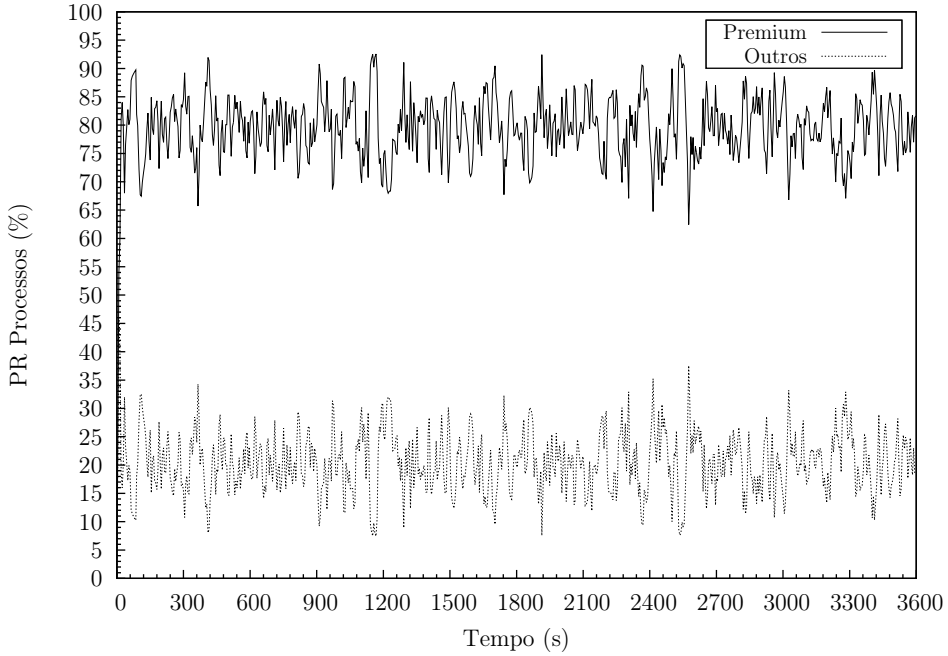


Figura 4.19: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 50ss. Erro nulo. Servidor Apache. Percentagem relativa de processos `Ativos`.

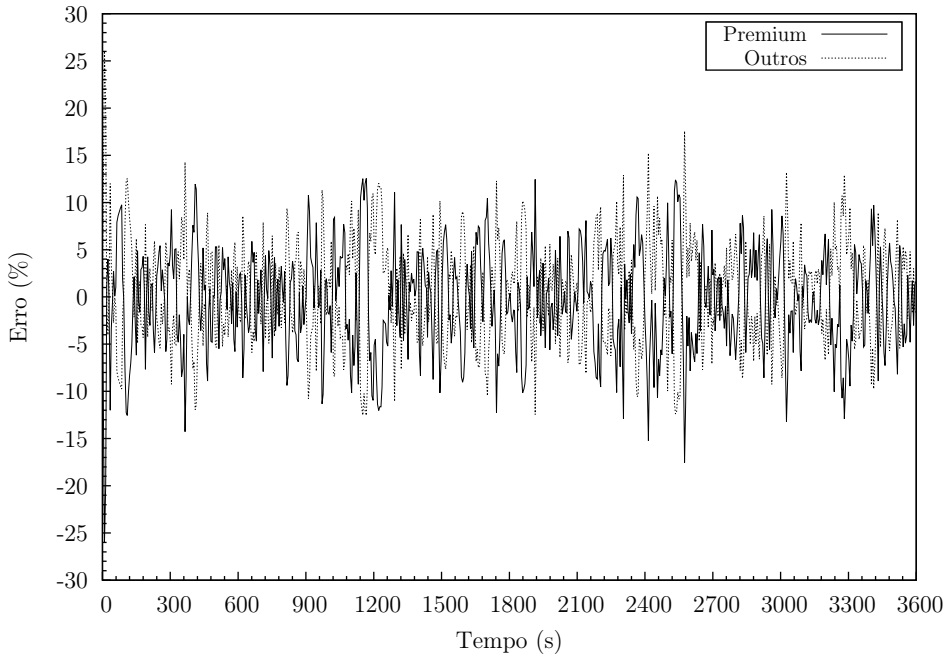


Figura 4.20: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 50ss. Erro nulo. Servidor Apache. Erro.

em três intervalos de 1200s. No final destes intervalos é realizado um *reset* do controlador PID (Listagem 4.1) de modo a termos um sistema sem memória/histórico.

Esta situação, normalmente indesejada, pode ser encarada como uma perturbação do sistema, mas que atestada a robustez do mesmo, já que ele reage de forma rápida e correta, impondo novamente os atrasos necessários com vista a obtenção dos objetivos traçados.

Listagem 4.1: Gateway - server.c (função reset)

```
void reset() {
   pOutros_Abs = 0;
   pPremium_Rel = 0;
   pOutros_Rel = 0;
   erroPremium_Rel = 0;
   erroOutros_Rel = 0;
   delayPremium = 0;
   delayOutros = 0;
   pidPremium.termoP = 0;
   pidPremium.termoI = 0;
   pidPremium.termoD = 0;
   pidPremium.termoTotal = 0;
   pidOutros.termoP = 0;
   pidOutros.termoI = 0;
   pidOutros.termoD = 0;
   pidOutros.termoTotal = 0;
}
```

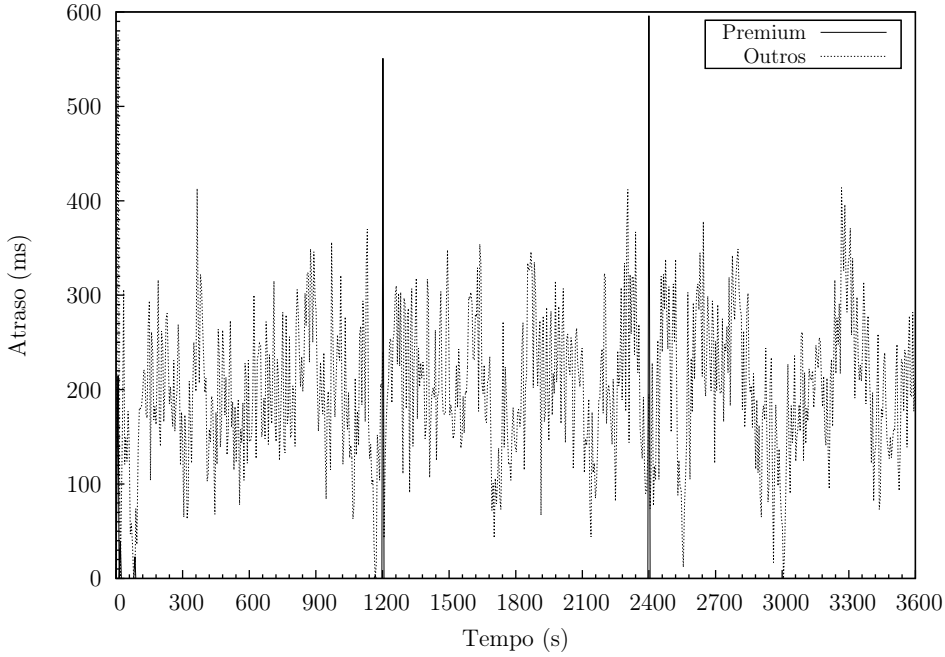


Figura 4.21: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 50ss. Erro nulo. *Gateway*. Atraso.

Mais uma vez, nem o CPU (figura 4.22), nem a memória (figura 4.23) influenciam nos resultados.

No que diz respeito ao tráfego de rede, o mesmo esteve compreendido entre os 10Mbit/s e os 35Mbit/s, mas mais frequentemente entre os 15Mbit/s e os 30Mbit/s.

Novamente, o tráfego é quase sempre inferior a 40Mbit/s (figura 4.24), sendo bastante inferior à capacidade das interfaces de rede: 1Gbit/s, pelo que não teve influência nos resultados obtidos.

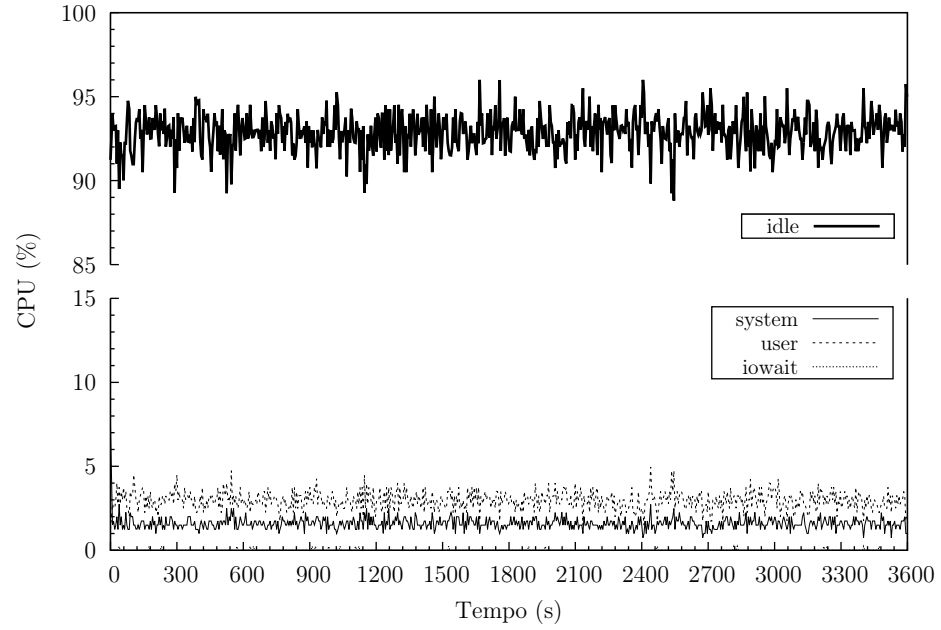


Figura 4.22: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 50ss. Erro nulo. Servidor Apache. CPU.

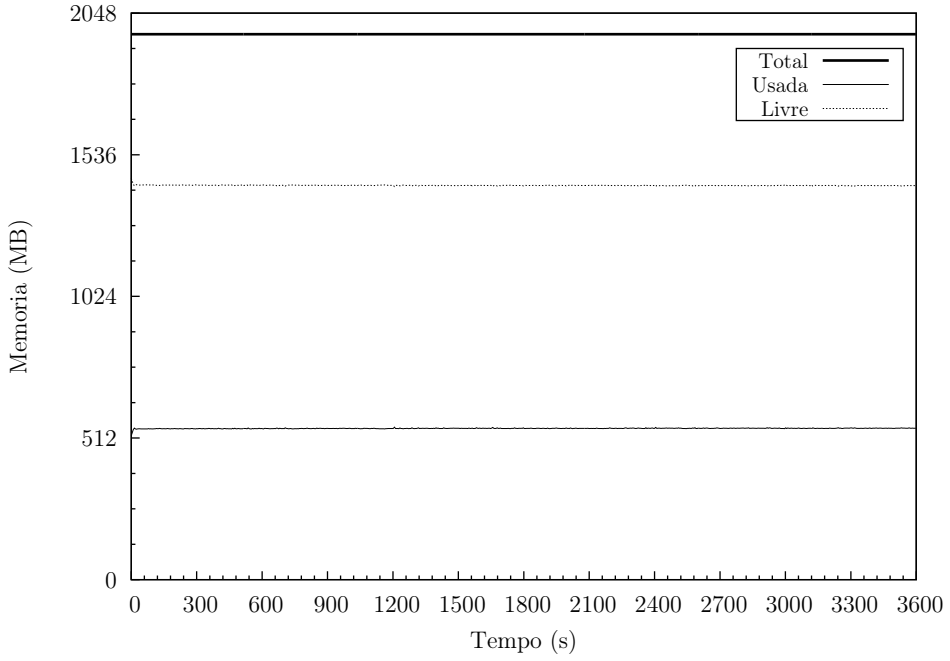


Figura 4.23: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 50ss. Erro nulo. Servidor Apache. Memória.

Importa, agora, conhecer os resultados obtidos pelo cliente `Premium` e pelo cliente `Outros`.

A taxa de ligações, nos dois clientes, é superior ao sistema sem controlador e sem saturação, mas semelhante ao sistema sem controlador com saturação. Contudo, no que diz respeito à taxa de pedidos e à taxa de respostas o sistema exibe um comportamento que permite atestar a ação do controlador:

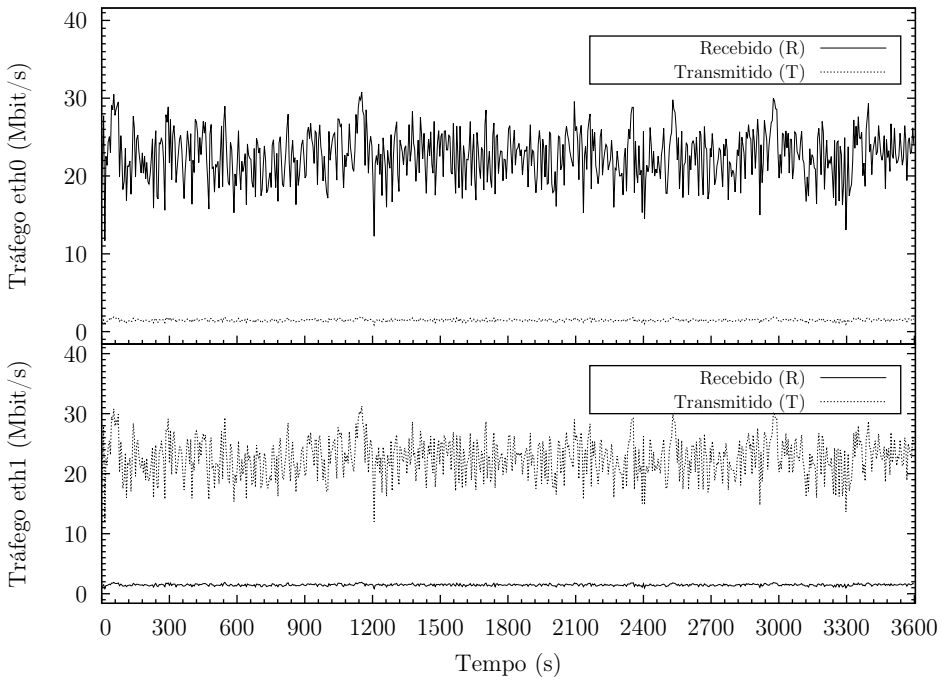


Figura 4.24: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 50ss. Erro nulo. *Gateway*. Tráfego de rede.

- No cliente `Premium` a taxa de pedidos e respostas é superior ao sistema sem controlador e sem saturação, mas inferior ao sistema sem controlador com saturação;
- No cliente `Outros` a taxa de pedidos e respostas é inferior às duas situações, isto é: ao sistema sem controlador e sem saturação e, ao sistema sem controlador com saturação.

É assim perceptível a ação do sistema, em particular sobre o cliente `Outros`, mais concretamente através da imposição de atrasos aos seus pacotes, conforme constatado na figura 4.21.

Não houve diferença significativa na taxa de ligações (lig/s) entre o cliente `Premium` e o cliente `Outros`: 50,8 e 53,4, respetivamente. No entanto, tendo em consideração o que é referido anteriormente, já se registou uma diferença expressiva na taxa de pedidos (ped/s): 439,9 no cliente `Premium` e 84,4 no cliente `Outros`. Esta diferença justifica-se se tivermos em consideração os valores obtidos na relação repostas por ligação (resp./lig.). No caso do cliente `Premium` este valor foi de 46,963, enquanto no cliente `Outros` foi apenas 16,840. Assim sendo, o cliente `Premium` fez, em média, mais pedidos e obteve mais respostas.

Como resultado de uma maior taxa de pedidos do cliente `Premium` e deste ser privilegiado relativamente ao cliente `Outros`, também obteve uma maior taxa média de respostas (resp/s): 426,4 contra 61,8 no cliente `Outros`.

As respostas obtidas pelos clientes foram todas do tipo 2xx, ou seja: "Sucesso", o que indica que a ação foi recebida com sucesso, compreendida e aceite.

Não se registam erros devido à falta de descritores de ficheiros (`fd-unavail`), falta de portas TCP disponíveis (`addrunvail`), ou exaustão da tabela de descritores de ficheiros do sistema (`ftab-full`). Este aspeto é muito importante, pois caso algum deste contadores fosse diferente de zero revelava incapacidade do cliente em criar e manter a carga de pedidos aos servidor Apache, o que inviabilizaria os resultados obtidos. Não existem, igualmente, outro tipo de erros (`other`), o que justificaria avaliar a sua razão.

httperf - Cliente sítio `Premium`

```
httperf --hog --timeout=5 --client=0/1 --server=192.168.100.101 --port=80 --uri=/index.
    ↪html --rate=50 --send-buffer=4096 --recv-buffer=16384 --wsess=300000,50,2.000 --
    ↪burst-length=5
Maximum connect burst length: 33

Total: connections 182727 requests 1582198 replies 1533622 test-duration 3596.784 s

Connection rate: 50.8 conn/s (19.7 ms/conn, <=532 concurrent connections)
Connection time [ms]: min 1261.3 avg 25668.6 max 36226.5 median 26447.5 stddev 4549.9
Connection time [ms]: connect 1374.4
Connection length [replies/conn]: 46.963

Request rate: 439.9 req/s (2.3 ms/req)
Request size [B]: 78.0

Reply rate [replies/s]: min 183.4 avg 426.4 max 647.0 stddev 57.4 (719 samples)
Reply time [ms]: response 188.5 transfer 45.8
Reply size [B]: header 251.0 content 5042.0 footer 0.0 (total 5293.0)
Reply status: 1xx=0 2xx=1533622 3xx=0 4xx=0 5xx=0

CPU time [s]: user 469.68 system 3123.18 (user 13.1% system 86.8% total 99.9%)
Net I/O: 2237.9 KB/s (18.3*10^6 bps)

Errors: total 152501 client-timo 149614 socket-timo 0 connrefused 0 connreset 2887
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 8.28 max 17.80 stddev 2.55 (29768/179382)
Session: avg 1.04 connections/session
Session lifetime [s]: 27.2
Session failtime [s]: 5.6
Session length histogram: 148064 5 0 1 1 129 1 0 0 0 173 0 0 0 0 143 0 0 0 0 152 0 0 0 0
    ↪178 0 0 0 0 205 0 0 0 0 201 0 0 0 0 186 0 0 0 0 175 0 0 0 0 29768
```

httperf - Cliente sítio `Outros`

```
httperf --hog --timeout=5 --client=0/1 --server=192.168.100.102 --port=80 --uri=/index.
    ↪html --rate=50 --send-buffer=4096 --recv-buffer=16384 --wsess=300000,50,2.000 --
    ↪burst-length=5
Maximum connect burst length: 18

Total: connections 192070 requests 303601 replies 222206 test-duration 3596.597 s

Connection rate: 53.4 conn/s (18.7 ms/conn, <=453 concurrent connections)
Connection time [ms]: min 3337.6 avg 17476.1 max 51521.3 median 15461.5 stddev 7689.0
Connection time [ms]: connect 1580.4
Connection length [replies/conn]: 16.840

Request rate: 84.4 req/s (11.8 ms/req)
Request size [B]: 79.0

Reply rate [replies/s]: min 17.2 avg 61.8 max 179.6 stddev 20.7 (719 samples)
Reply time [ms]: response 811.7 transfer 148.3
Reply size [B]: header 251.0 content 5042.0 footer 0.0 (total 5293.0)
```

```
Reply status: 1xx=0 2xx=222206 3xx=0 4xx=0 5xx=0

CPU time [s]: user 432.55 system 3133.00 (user 12.0% system 87.1% total 99.1%)
Net I/O: 326.0 KB/s (2.7*10^6 bps)

Errors: total 190948 client-timo 178709 socket-timo 0 connrefused 0 connreset 12239
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 0.21 max 2.60 stddev 0.36 (773/179482)
Session: avg 1.29 connections/session
Session lifetime [s]: 37.9
Session failtime [s]: 6.7
Session length histogram: 167387 527 12 87 34 1467 111 8 30 2 2653 53 0 3 4 2074 29 0 2 2
    ↪ 1617 14 0 4 0 979 12 0 0 0 593 4 0 3 0 450 3 0 0 0 307 1 0 0 0 236 0 1 0 0 773
```

Tendo em consideração o bom desempenho do sistema implementado e do comportamento do controlador, vamos avaliar o seu funcionamento no que diz respeito à "Rejeição de perturbações" e "Seguimento de referência".

Rejeição de perturbações

Na figura 4.25 está representada a percentagem absoluta de processos quando existe uma perturbação no instante t_{1200} e a reposição das condições iniciais no instante t_{2400}. A perturbação consiste na introdução de pedidos do cliente `Outros`. O objetivo é verificar como reage o sistema ao aparecimento de uma elevada carga de pedidos do cliente `Outros`, quando este só estava a processar pedidos do cliente `Premium` e subitamente os pedidos do cliente `Outros` desaparecem e o sistema volta a ter exclusivamente pedidos do cliente `Premium`.

Como é natural, nos intervalos t_0 até t_{1200} e perto t_{2400} até t_{3600} existe uma coincidência total entre a percentagem de processos `Ativos` e processos `Premium`, já que o servidor Apache só está a processar pedidos do cliente `Premium`. Deve ser salientado que nestes intervalos o sistema tem o controlador ativo, pelo que tenta manter a percentagem relativa de processos `Premium` perto dos 80% (objetivo), mesmo que não existam pedidos do cliente `Outros`. Essa é a razão pela qual o número de processos `Ativos` não se aproxima dos 100%, com exceção no arranque do sistema. Quando começam a surgir pedidos do cliente `Outros`, a percentagem de processos `Ativos` é geralmente superior aos 95%.

A percentagem relativa de processos `Ativos` (figura 4.26) permite notar que após o início dos pedidos do cliente `Outros` o sistema conseguiu adaptar-se rapidamente de modo a manter o objetivo traçado: 80% da capacidade do servidor Apache, expressa em processos `Ativos`, a processar pedidos do sítio `Premium` e os remanescentes 20% a processar pedidos do sítio `Outros`. Existe um momento inicial, após a perturbação, em que a percentagem dos processos `Premium` desce até cerca de 40%, mas recupera rapidamente. Esta redução resulta do aumento do atraso imposto aos pacotes destinados ao sítio `Premium`, conforme veremos adiante (figura 4.28). O aumento do atraso deve-se ao facto da percentagem relativa dos processos `Premium` ser superior aos 80%, o que não é crítico enquanto o servidor Apache só

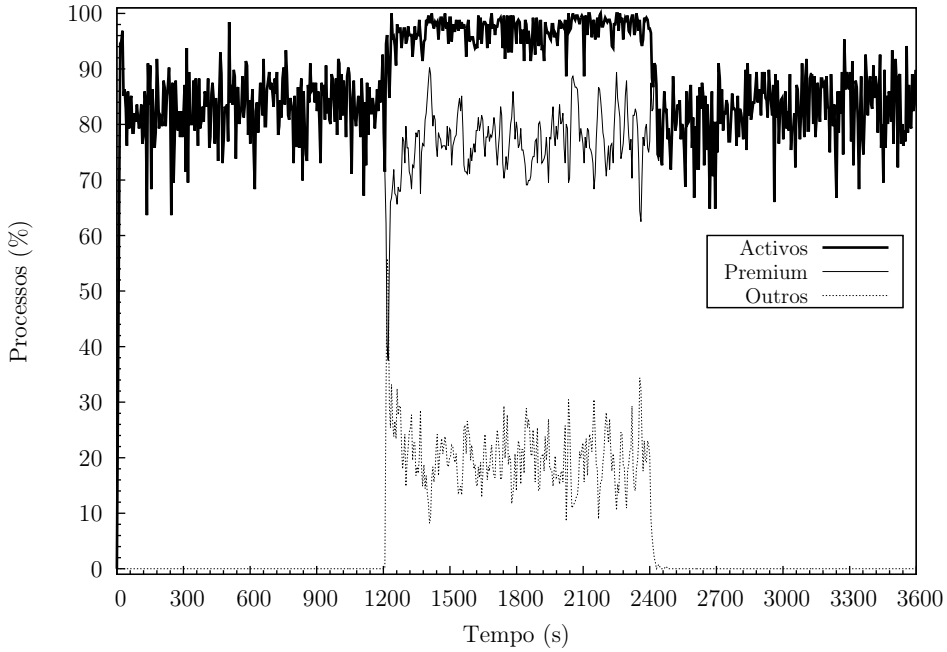


Figura 4.25: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 50ss. Rejeição de perturbações. Servidor Apache. Percentagem absoluta de processos.

está a processar os seus pedidos, mas o é após existirem pedidos para o sítio `Outros`. O sistema tem de forçar o cumprimento dos objetivos, mesmo que isso implique o aumento temporário do atraso dos pacotes destinados ao sítio `Premium`.

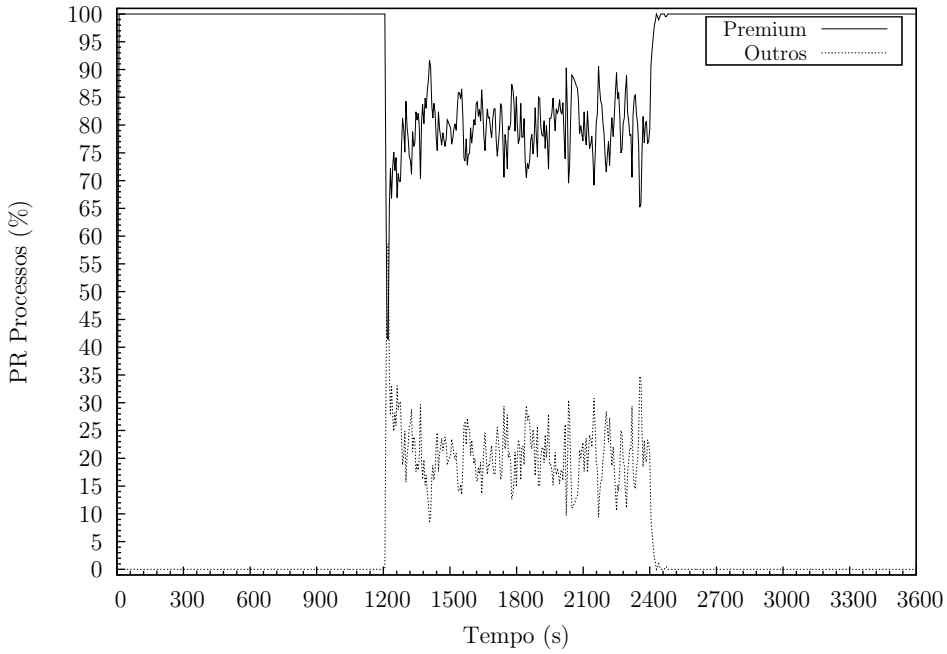


Figura 4.26: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 50ss. Rejeição de perturbações. Servidor Apache. Percentagem relativa de processos `Ativos`.

Nos intervalos t_0 a t_{1200} e perto t_{2400} a t_{3600} regista-se um erro de +20% e -20% relativamente ao objetivo do sítios `Premium` e `Outros`, respetivamente (figura 4.27). Contudo, este erro não resulta do incorreto funcionamento do sistema, mas do facto de não existir, nesses intervalos, pedidos destinados ao sítio `Outros`. Nos segundos imediatamente após o instante t_{1200} regista-se um erro positivo do processos do sítio `Outros` e negativo do processos do sítio `Premium`. Este erro resulta da ação do sistema ao detetar que, no instante t_{1200}, existe uma percentagem relativa superior à definida para os processos do sítio `Premium` e uma percentagem relativa inferior à definida para os processos do sítio `Outros`, o que força o sistema a impor um atraso aos pacotes destinados ao sítio `Premium` nesse instante (figura 4.28).

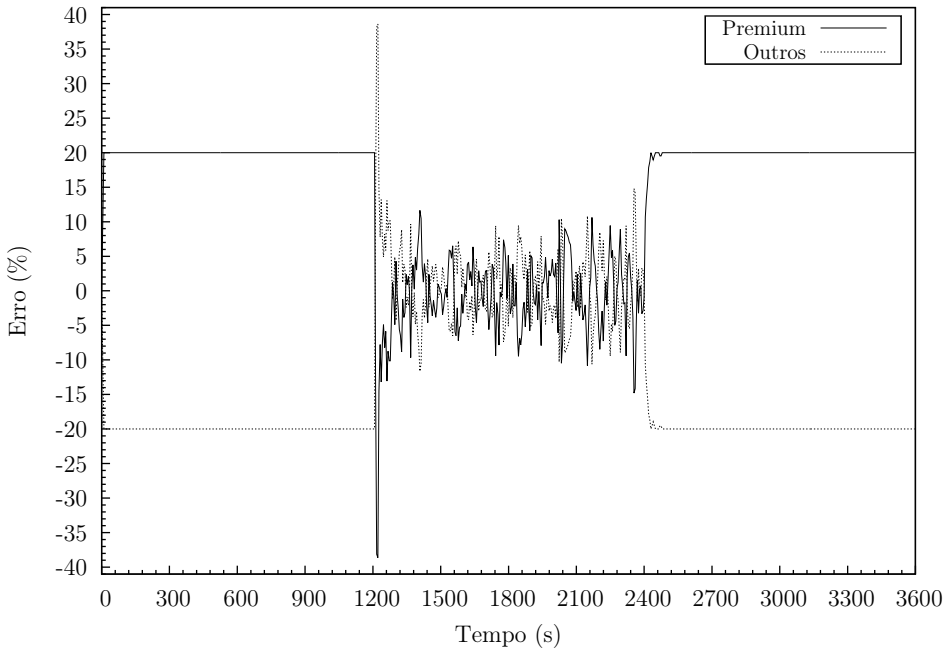


Figura 4.27: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 50ss. Rejeição de perturbações. Servidor Apache. Erro.

Se atentarmos ao atraso imposto aos pacotes verificamos que:

- Até ao instante t_{1200} é imposto um atraso de cerca de 400ms aos pacotes destinados ao sítio `Premium` (figura 4.28). O intuito do sistema é forçar o cumprimento do objetivo de 80% de processos HTTP `Ativos` a processar pedidos do sítio `Premium` e impedir que os mesmos monopolizem o servidor Apache. Caso se registasse essa saturação com pedidos `Premium`, o sistema podia levar muito tempo a estabilizar após o aparecimento dos primeiros pedidos do cliente `Outros`. Conforme foi constatado anteriormente, o sistema consegue atingir os objetivos definidos muito rapidamente;
- No instante t_{1200} é imposto um atraso de quase 1000ms ao pacotes destinados ao sítio `Premium`, devido às razões explicadas anteriormente;
- Entre o instante t_{1200} e o instante t_{2400} é exclusivamente imposto limite aos pacotes destinados ao sítio `Outros`, sendo o atraso variável. Mais uma vez se constata o comportamento

dinâmico do sistema;

- Após o instante t_{2400} e devido ao facto de não existirem novamente pedidos destinados ao sítio `Outros`, só são impostos atrasos aos pacotes destinados ao sítio `Premium`, sendo este atraso maior no instante t_{2400}. Este maior atraso permite ao sistema atingir mais rapidamente o objetivo definido.

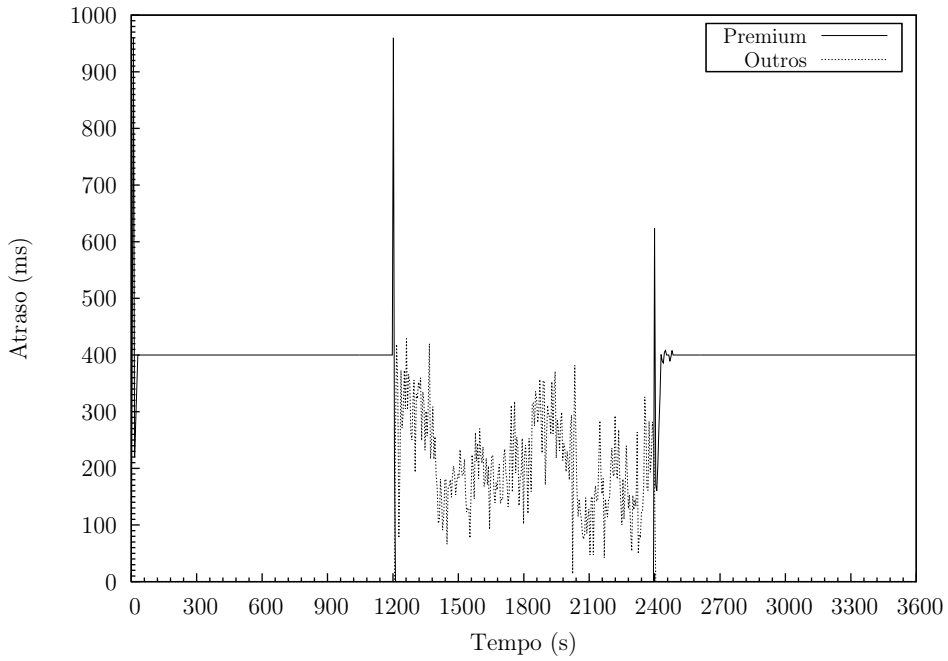


Figura 4.28: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 50ss. Rejeição de perturbações. *Gateway*. Atraso.

Tal como em situações anteriores, nem o CPU (figura 4.29) nem a memória (figura 4.30) foram fatores com influência no sistema.

Tal como esperado, no que diz respeito ao tráfego de rede, existem três zonas distintas (figura 4.31), as quais coincidem com os intervalos em que só existem pedidos destinados ao sítio `Premium` (instante t_0 a instante t_{1200} e instante t_{2400} a instante t_{3600}) e o intervalo em que os pedidos destinados ao dois sítios (instante t_{1200} a instante t_{2400}) coexistem .

Como é lógico, neste sistema, o tráfego de rede, salvo algum fator externo (que não se regista) resulta do volume de informação a transferir dos pedidos dos clientes e das respostas do servidor Apache. Assim sendo, é natural que o tráfego com os dois clientes ativos, o que se traduz em mais pedidos e respostas, se traduza num aumento no tráfego de rede.

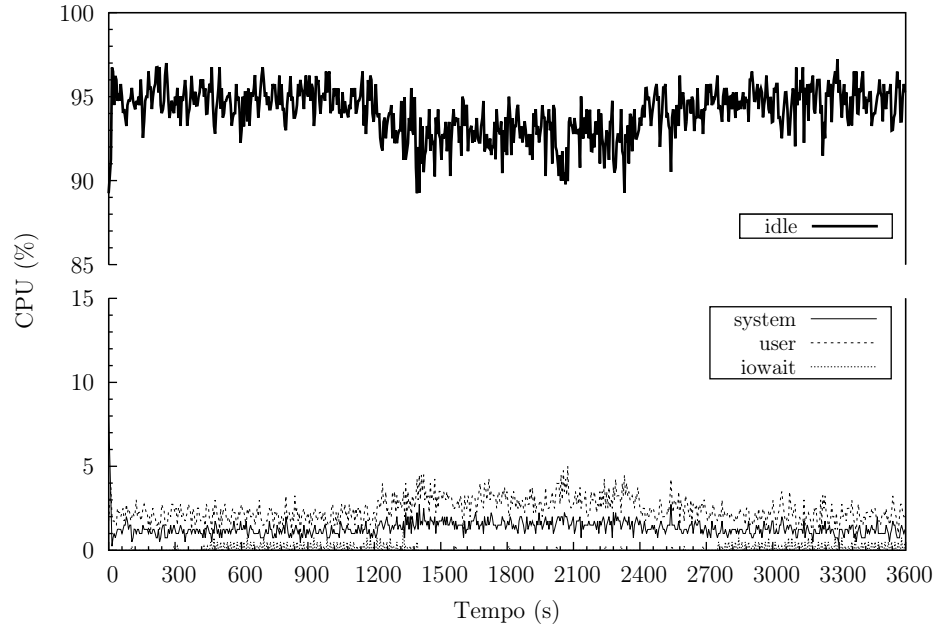


Figura 4.29: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 50ss. Rejeição de perturbações. Servidor Apache. CPU.

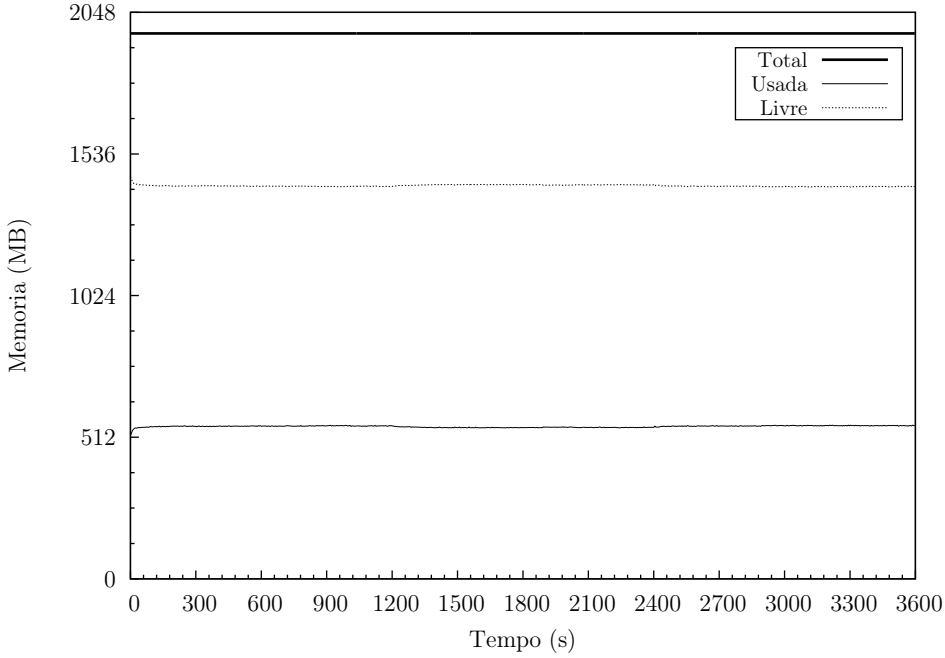


Figura 4.30: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 50ss. Rejeição de perturbações. Servidor Apache. Memória.

No caso do cliente `Premium`, não é possível fazer uma análise comparativa relativamente às observações anteriores já que assentam em pressupostos distintos, nomeadamente a coexistência permanente de pedidos dos dois clientes. Será realizada uma análise comparativa da "Rejeição de perturbações" entre as 50 sessões por segundo (50ss) e as 25 sessões por segundo (25ss, pág. 115).

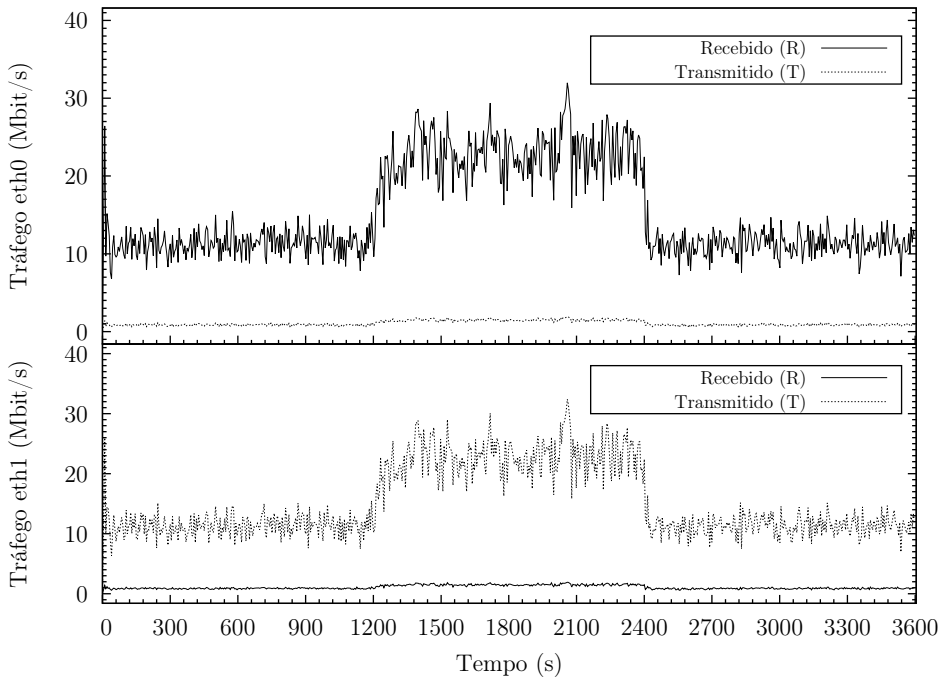


Figura 4.31: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 50ss. Rejeição de perturbações. *Gateway*. Tráfego de rede.

No que diz respeito ao cliente `Outros` é possível fazer uma análise comparativa já que no período em que efetuou pedidos, os mesmos coexistiam com os pedidos do cliente `Premium` e com o mesmo objetivo.

É possível notar que, neste caso, a taxa de ligações (lig/s) do cliente `Outros` é muito semelhante ao "Erro nulo (0)": 53,3 e 53,4, respetivamente. Também a taxa de pedidos (ped./s) é bastante semelhante: 86,1 contra 84,4 no "Erro nulo (0)", bem como a taxa média de respostas (resp./s): 64,2 contra 61,8. É, assim, validado o comportamento do sistema durante o período em que os pedidos coexistiam, já que se registou uma carga semelhante aos outros cenários por parte do cliente `Outros`.

Todas as respostas foram do tipo 2xx, ou seja: "Sucesso", o que indica que a ação foi recebida com sucesso, compreendida e aceite.

Não se registam erros devido à falta de descritores de ficheiros (`fd-unavail`), falta de portas TCP disponíveis (`addrunvail`), ou exaustão da tabela de descritores de ficheiros do sistema (`ftab-full`).

httperf - Cliente sítio `Premium`

```
httperf --hog --timeout=5 --client=0/1 --server=192.168.100.101 --port=80 --uri=/index.
    ↪html --rate=50 --send-buffer=4096 --recv-buffer=16384 --wsess=300000,50,2.000 --
    ↪burst-length=5
Maximum connect burst length: 54

Total: connections 240449 requests 1217417 replies 1019398 test-duration 3598.004 s

Connection rate: 66.8 conn/s (15.0 ms/conn, <=734 concurrent connections)
```

```
Connection time [ms]: min 2842.8 avg 14554.7 max 52152.2 median 12025.5 stddev 7063.4
Connection time [ms]: connect 1426.7
Connection length [replies/conn]: 13.801

Request rate: 338.4 req/s (3.0 ms/req)
Request size [B]: 82.0

Reply rate [replies/s]: min 81.2 avg 283.4 max 620.4 stddev 111.9 (719 samples)
Reply time [ms]: response 819.6 transfer 154.4
Reply size [B]: header 250.0 content 5042.0 footer 0.0 (total 5292.0)
Reply status: 1xx=0 2xx=1019398 3xx=0 4xx=0 5xx=0

CPU time [s]: user 362.27 system 3232.37 (user 10.1% system 89.8% total 99.9%)
Net I/O: 1492.0 KB/s (12.2*10^6 bps)

Errors: total 229479 client-timo 168931 socket-timo 0 connrefused 0 connreset 60548
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 2.87 max 17.80 stddev 4.13 (10325/179256)
Session: avg 1.14 connections/session
Session lifetime [s]: 28.3
Session failtime [s]: 10.2
Session length histogram: 125697 4760 61 1788 164 11381 818 35 273 19 7605 475 11 227 49
    ↪5713 335 15 155 13 3565 182 3 76 10 2247 113 3 48 3 1327 82 1 35 5 759 42 0 18 2
    ↪460 25 0 7 0 297 22 0 4 1 10325
```

httperf - Cliente sítio `Outros`

```
httperf --hog --timeout=5 --client=0/1 --server=192.168.100.102 --port=80 --uri=/index.
    ↪html --rate=50 --send-buffer=4096 --recv-buffer=16384 --wsess=300000,50,2.000 --
    ↪burst-length=5
Maximum connect burst length: 13

Total: connections 63957 requests 103348 replies 77006 test-duration 1199.889 s

Connection rate: 53.3 conn/s (18.8 ms/conn, <=481 concurrent connections)
Connection time [ms]: min 4849.3 avg 18188.7 max 50247.3 median 16319.5 stddev 7950.6
Connection time [ms]: connect 1591.9
Connection length [replies/conn]: 17.682

Request rate: 86.1 req/s (11.6 ms/req)
Request size [B]: 79.0

Reply rate [replies/s]: min 24.2 avg 64.2 max 240.2 stddev 22.5 (239 samples)
Reply time [ms]: response 813.3 transfer 149.1
Reply size [B]: header 251.0 content 5043.0 footer 0.0 (total 5294.0)
Reply status: 1xx=0 2xx=77006 3xx=0 4xx=0 5xx=0

CPU time [s]: user 145.66 system 1043.32 (user 12.1% system 87.0% total 99.1%)
Net I/O: 338.5 KB/s (2.8*10^6 bps)

Errors: total 63315 client-timo 59353 socket-timo 0 connrefused 0 connreset 3962
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 0.25 max 2.20 stddev 0.37 (296/59649)
Session: avg 1.27 connections/session
Session lifetime [s]: 38.4
Session failtime [s]: 6.7
Session length histogram: 55674 143 2 68 18 467 23 0 9 1 736 25 0 1 0 728 4 0 2 0 492 4 0
    ↪ 0 0 399 2 0 0 0 220 2 0 0 0 134 1 0 0 0 107 2 0 0 0 89 0 0 0 0 296
```

Seguimento de Referência

Neste caso, o objetivo é variável, isto é: em determinados intervalos o objetivo é 80% da capacidade do servidor Apache, expressa em processos `Ativos`, a processar pedidos do sítio `Premium` e os remanescentes 20% a processar pedidos do sítio `Outros` (instante t_0 a instante t_{1200} e instante t_{2400} a instante t_{3600}) e noutro intervalo o objetivo é 60% da capacidade do servidor Apache a processar pedidos do sítio `Premium` e os remanescentes 40% a processar pedidos do sítio `Outros` (instante t_{1200} a instante t_{2400}).

Tal como em casos anteriores, constata-se a tendência para a saturação do servidor Apache (figura 4.32). A percentagem absoluta de processos `Ativos` está quase sempre compreendida entre os 90% e os 100%.

Tal como na "Rejeição de perturbações", é possível observar três zonas distintas, estando as mesmas relacionadas com o objetivo definido: 80%/20% e 60%/40%.

É possível notar que, mais uma vez, o sistema funcionou como pretendido, ou seja: cumpre os objetivos definidos, é rápido a reagir às alterações (Figuras 4.32 e 4.33) e é estável. À semelhança da "Rejeição de perturbações", perto do instante t_{1200} as percentagens denotam grande diferença relativamente ao pretendido, mas mais uma vez corresponde ao instante em que o sistema está a adaptar-se à alteração dos objetivos, mais concretamente à redução das percentagens relativas do cliente `Premium` e aumento das percentagens relativas do cliente `Outros`.

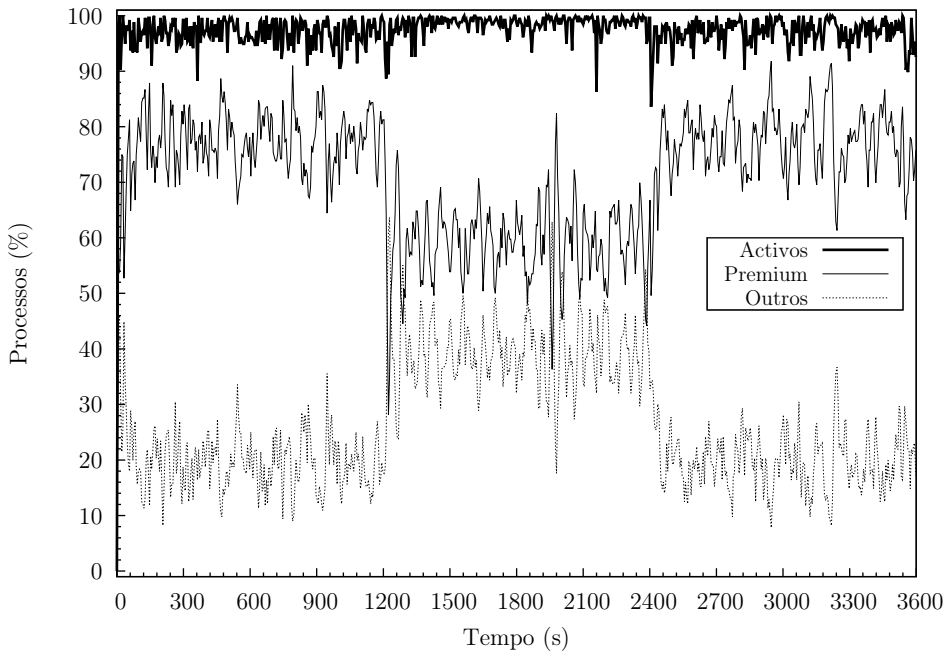


Figura 4.32: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 50ss. Seguimento de Referência. Servidor Apache. Percentagem absoluta de processos.

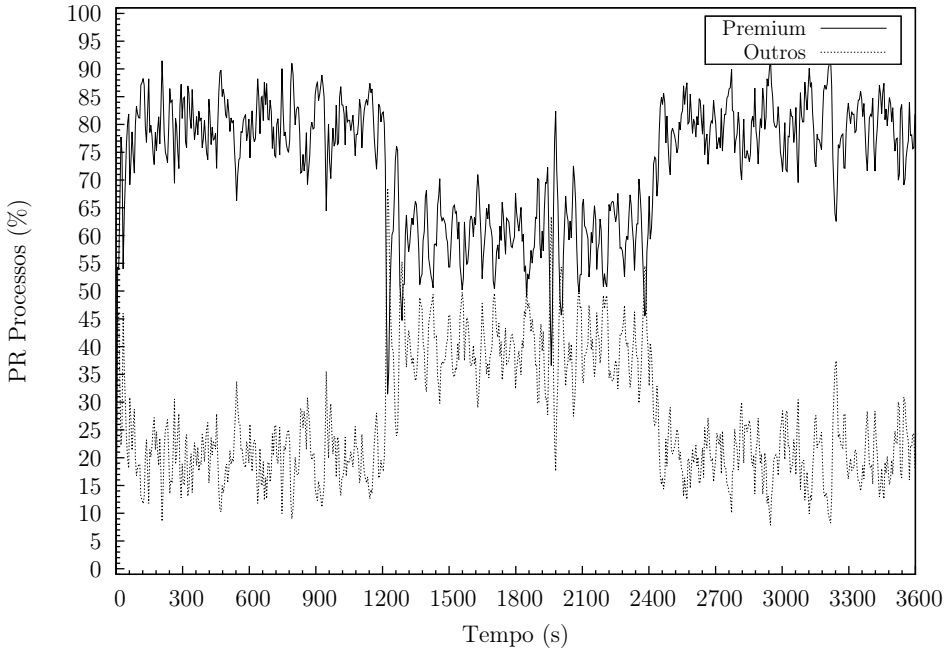


Figura 4.33: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 50ss. Seguimento de Referência. Servidor Apache. Percentagem relativa de processos `Ativos`.

De um modo geral, o erro das percentagens dos processos (figura 4.34) está compreendido num intervalo aceitável: -15% e +15%. Existem quatro excepções:

1. Instante t_{1200}: Adaptação do sistema aos novos objetivos 60%/40%;
2. Perto do instante t_{2000}: Sem razão específica. Resulta da natureza estocática dos pedidos HTTP;
3. Instante t_{2400}: Adaptação do sistema aos novos objetivos 80%/20%;
4. Perto do instante t_{3200}: Sem razão específica. Resulta da natureza estocática dos pedidos HTTP.

Conforme se pode comprovar na figura 4.35 entre o instante t_{1200} e o instante t_{2400} são aplicados pontualmente pequenos atrasos aos pacotes destinados ao sítio `Premium`, facto que não sucede fora deste intervalo. Este situação pode ser explicada, pelo facto do objetivo do sítio `Premium` não diferir muito do objetivo do sítio `Outros`, 60% e 40% respetivamente. Devido à natureza variável dos pedidos, bem como ao facto de uma sessão poder fazer mais de um pedido (o que ocupa um processo durante um período de tempo não determinado), o sistema necessitou de aplicar atrasos aos pacotes destinados aos dois sítios.

Mais uma vez, nem o CPU (figura 4.36), nem a memória (figura 4.37) foram fatores com influência nos resultados.

No que diz respeito ao tráfego de rede, não existem zonas distintas no período de observação (figura 4.38). O tráfego de rede esteve compreendido entre os 10Mbit/s e os 35Mbit/s com maior frequência entre os 15Mbit/s e os 30Mbit/s, pelo que a alteração dos objetivos durante

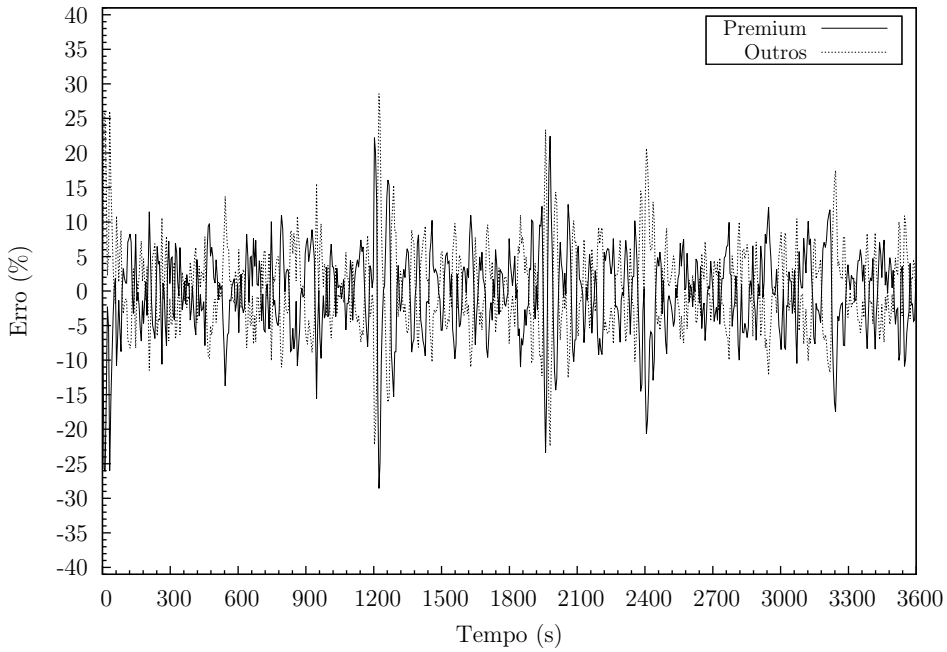


Figura 4.34: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 50ss. Seguimento de Referência. Servidor Apache. Erro.

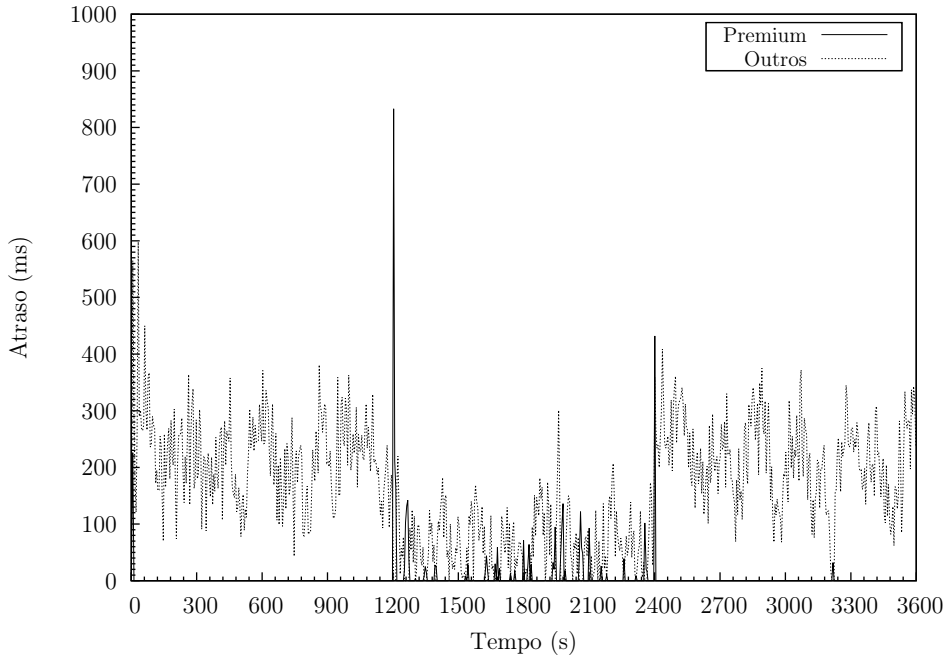


Figura 4.35: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 50ss. Seguimento de Referência. *Gateway*. Atraso.

o período de observação ("Seguimento de referência") não teve influência neste aspeto.

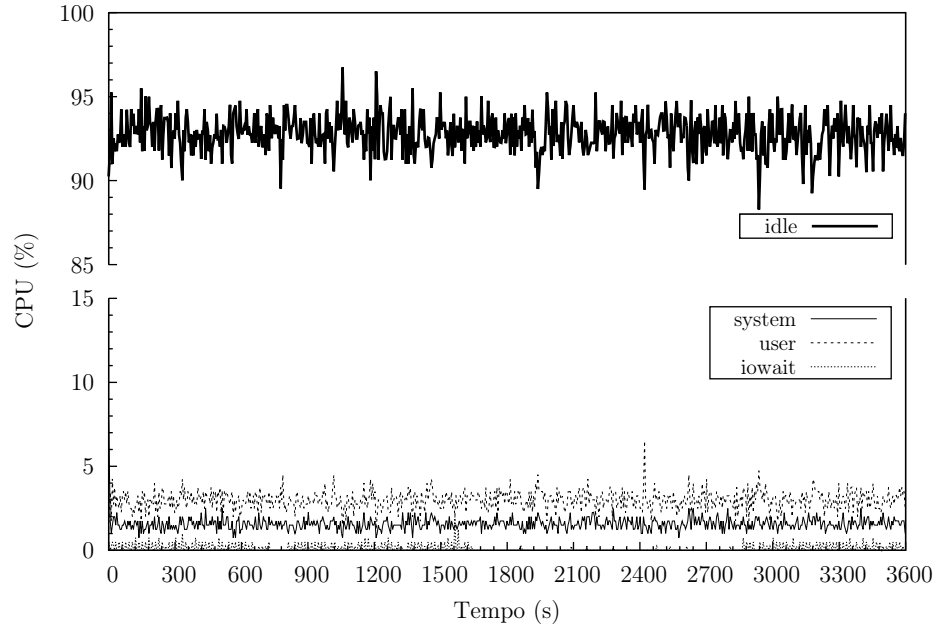


Figura 4.36: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 50ss. Seguimento de Referência. Servidor Apache. CPU.

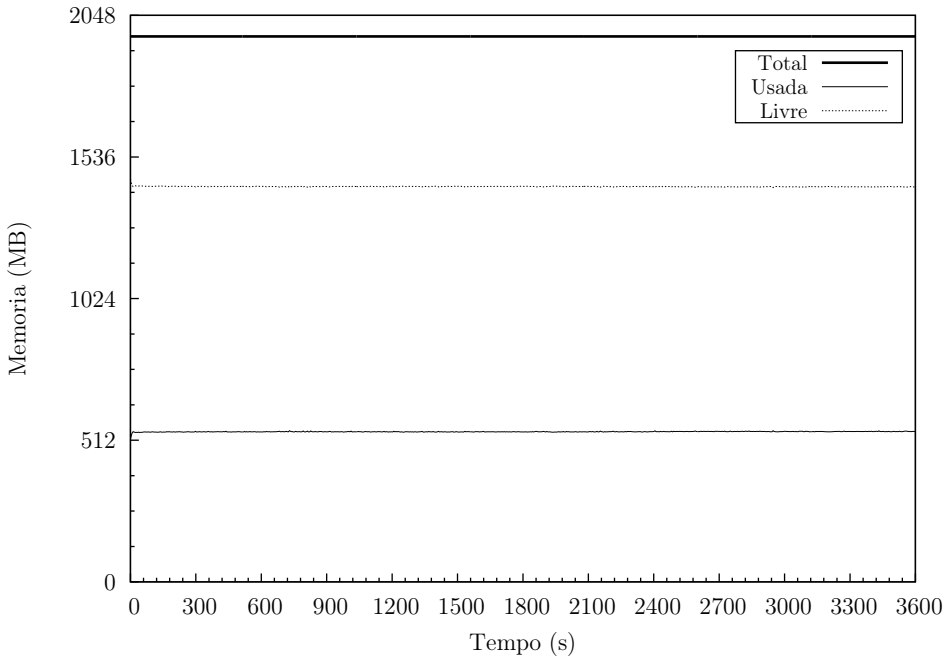


Figura 4.37: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 50ss. Seguimento de Referência. Servidor Apache. Memória.

O cliente `Outros` tem uma taxa de ligações superior ao cliente `Premium` (104,1%), mas apresenta uma taxa de pedidos e uma taxa média de repostas bem inferior, 30,3% e 26,1%, respetivamente. Consultando a linha das frequências absolutas das sessões, verificamos que, tal como esperado, os resultados obtidos pelo cliente `Outros` foram bastante piores do que pelo cliente `Premium`, existindo um maior número de sessões a terminar após um número

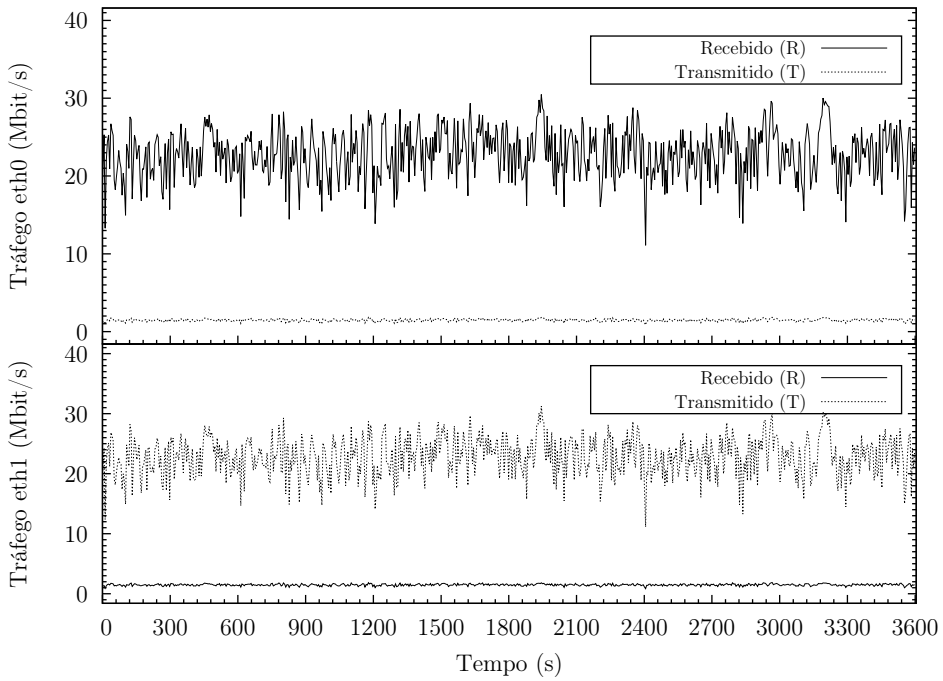


Figura 4.38: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 50ss. Seguimento de Referência. *Gateway*. Tráfego de rede.

reduzido de pedidos. Todas as respostas foram do tipo 2xx, ou seja: "Sucesso", o que indica que a ação foi recebida com sucesso, compreendida e aceite.

Não se registam erros devido à falta de descritores de ficheiros (`fd-unavail`), falta de portas TCP disponíveis (`addrunvail`), ou exaustão da tabela de descritores de ficheiros do sistema (`ftab-full`).

httperf - Cliente sítio `Premium`

```
httperf --hog --timeout=5 --client=0/1 --server=192.168.100.101 --port=80 --uri=/index.
    ↪html --rate=50 --send-buffer=4096 --recv-buffer=16384 --wsess=300000,50,2.000 --
    ↪burst-length=5
Maximum connect burst length: 24

Total: connections 183001 requests 1451924 replies 1403491 test-duration 3600.229 s

Connection rate: 50.8 conn/s (19.7 ms/conn, <=536 concurrent connections)
Connection time [ms]: min 2258.0 avg 25774.3 max 41308.4 median 26628.5 stddev 4711.5
Connection time [ms]: connect 1387.1
Connection length [replies/conn]: 46.677

Request rate: 403.3 req/s (2.5 ms/req)
Request size [B]: 78.0

Reply rate [replies/s]: min 101.8 avg 389.9 max 628.8 stddev 74.2 (720 samples)
Reply time [ms]: response 197.3 transfer 46.8
Reply size [B]: header 251.0 content 5043.0 footer 0.0 (total 5294.0)
Reply status: 1xx=0 2xx=1403491 3xx=0 4xx=0 5xx=0

CPU time [s]: user 477.52 system 3119.36 (user 13.3% system 86.6% total 99.9%)
Net I/O: 2046.2 KB/s (16.8*10^6 bps)

Errors: total 155457 client-timo 152468 socket-timo 0 connrefused 0 connreset 2989
```

```
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 7.52 max 19.60 stddev 2.60 (27076/179544)
Session: avg 1.04 connections/session
Session lifetime [s]: 27.4
Session failtime [s]: 5.6
Session length histogram: 150716 20 1 0 1 140 7 0 0 0 190 3 0 0 0 173 1 0 0 0 210 2 0 0 0
    ↪ 194 2 1 0 0 190 1 0 0 0 179 5 0 0 0 227 4 0 0 0 200 1 0 0 0 27076
```

httperf - Cliente sítio `Outros`

```
httperf --hog --timeout=5 --client=0/1 --server=192.168.100.102 --port=80 --uri=/index.
    ↪html --rate=50 --send-buffer=4096 --recv-buffer=16384 --wsess=300000,50,2.000 --
    ↪burst-length=5
Maximum connect burst length: 31

Total: connections 190277 requests 440374 replies 365907 test-duration 3600.070 s

Connection rate: 52.9 conn/s (18.9 ms/conn, <=516 concurrent connections)
Connection time [ms]: min 3510.7 avg 20889.5 max 53386.9 median 19369.5 stddev 8652.3
Connection time [ms]: connect 1553.9
Connection length [replies/conn]: 25.509

Request rate: 122.3 req/s (8.2 ms/req)
Request size [B]: 79.0

Reply rate [replies/s]: min 20.6 avg 101.6 max 306.0 stddev 62.6 (720 samples)
Reply time [ms]: response 556.5 transfer 105.3
Reply size [B]: header 251.0 content 5042.0 footer 0.0 (total 5293.0)
Reply status: 1xx=0 2xx=365907 3xx=0 4xx=0 5xx=0

CPU time [s]: user 450.96 system 3060.80 (user 12.5% system 85.0% total 97.5%)
Net I/O: 534.9 KB/s (4.4*10^6 bps)

Errors: total 185987 client-timo 175714 socket-timo 0 connrefused 0 connreset 10273
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 1.10 max 8.20 stddev 1.52 (3952/179666)
Session: avg 1.11 connections/session
Session lifetime [s]: 31.9
Session failtime [s]: 6.5
Session length histogram: 166304 335 14 54 27 1150 49 1 14 0 2024 25 0 1 0 1543 34 0 1 0
    ↪1370 9 0 1 0 866 5 0 0 0 693 4 0 0 0 522 1 0 0 0 359 2 0 0 0 304 2 0 0 0 3952
```

4.2.2.2 Teste 2 - 25 sessões por segundo

Vamos, em seguida, analisar o comportamento do sistema com uma carga inferior, isto é: uma menor taxa de sessões. Neste caso concreto são iniciadas 25 sessões por segundo (25ss), sendo realizados 50 pedidos por sessão.

Erro nulo

Não se registam diferenças significativas no comportamento do sistema, no que diz respeito à percentagem absoluta de processos e relativa de processos `Ativos`, entre com os clientes a iniciarem sessões a uma taxa de 50ss (figuras 4.18 e 4.19) ou 25ss (figuras 4.39 e 4.40).

Esta situação demonstra que o sistema funciona de forma semelhante e com o mesmo bom desempenho nas duas situações.

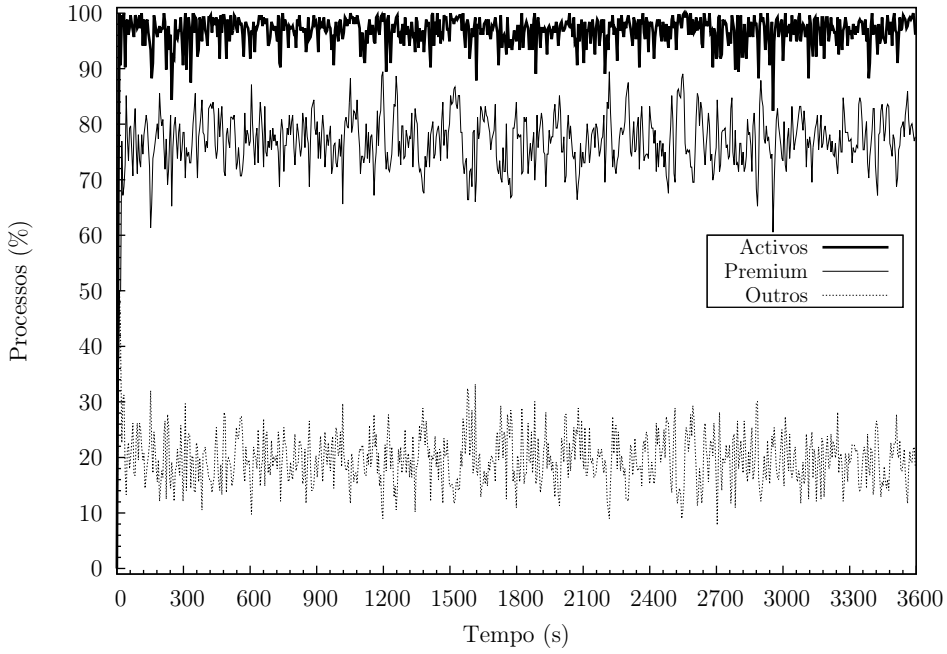


Figura 4.39: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 25ss. Erro nulo. Servidor Apache. Percentagem absoluta de processos.

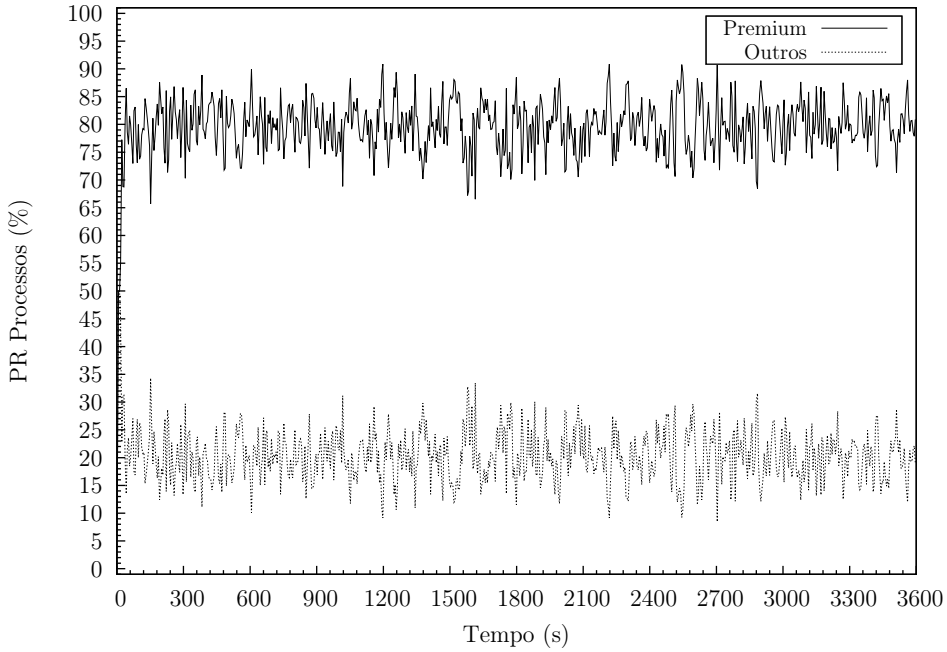


Figura 4.40: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 25ss. Erro nulo. Servidor Apache. Percentagem relativa de processos `Ativos`.

No que diz respeito ao erro (figura 4.41), o mesmo está geralmente compreendido entre os -10% e os +10%, tal como sucedeu nas 50ss, o que dada a natureza dos pedidos HTTP e a carga exercida sobre o servidor Apache é aceitável.

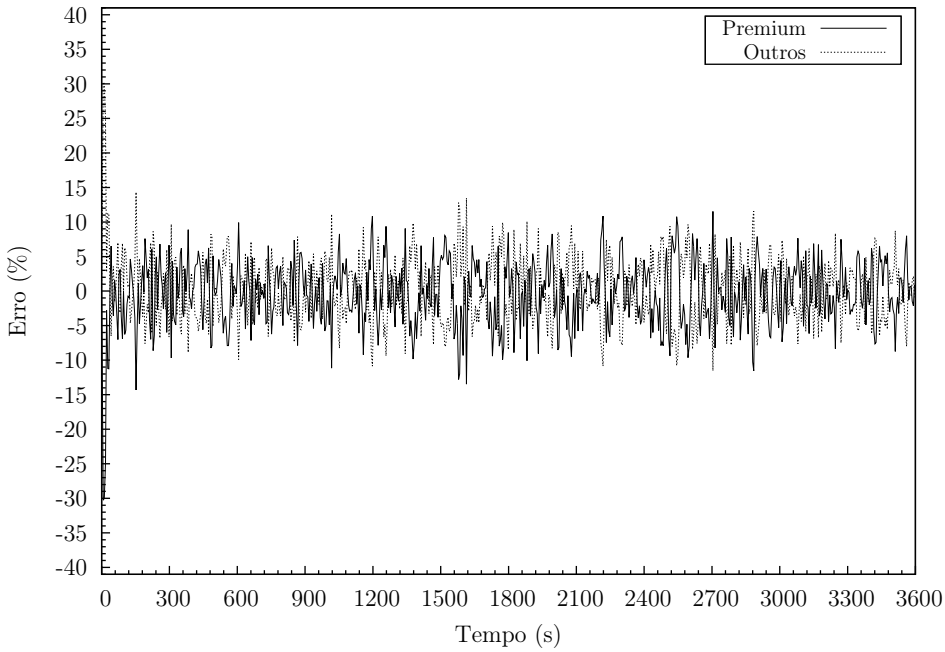


Figura 4.41: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 25ss. Erro nulo. Servidor Apache. Erro.

Com exceção do momento inicial (cerca de 10s), o sistema só necessitou de aplicar atraso aos pacotes destinados ao sítio `Outros` (figura 4.42). Este atraso foi quase sempre inferior a 500ms sendo, no entanto, superior ao registado no caso das 50ss. Esta situação pode ser explicada pelo facto de não existir uma carga tão grande do cliente `Premium`, facto este que resulta de uma menor taxa de sessões. Este acontecimento é compreensível se atendermos a que o cliente `Outros` utiliza os recursos remanescentes do cliente `Premium`. Assim sendo, se o cliente `Premium` exerce menor carga comparativamente às 50ss, o cliente `Outros` tenta exercer maior carga. Cabe ao sistema de controlo travar esta tendência, através da introdução de atraso nos pacotes do cliente `Outros`.

Tal como em situações anteriores, nem o CPU (figura 4.43) nem a memória (figura 4.44) foram fatores com influência no desempenho do sistema.

No que diz respeito ao tráfego de rede (figura 4.45), o mesmo esteve compreendido entre os 10Mbit/s e os 35Mbit/s, mas com maior frequência entre os 15Mbit/s e os 30Mbit/s, sendo também neste aspeto semelhante às 50ss.

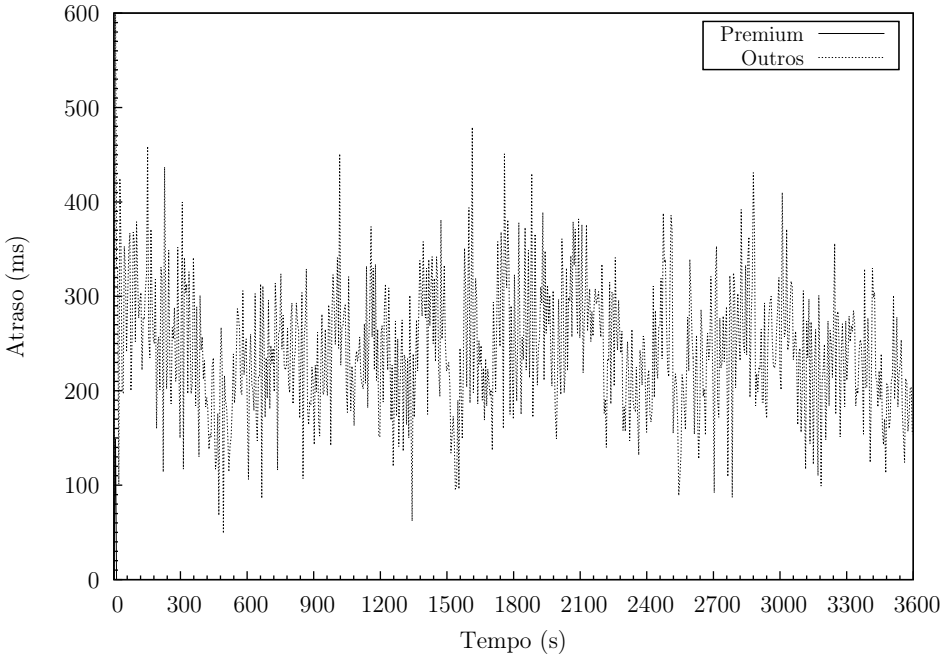


Figura 4.42: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 25ss. Erro nulo. *Gateway*. Atraso.

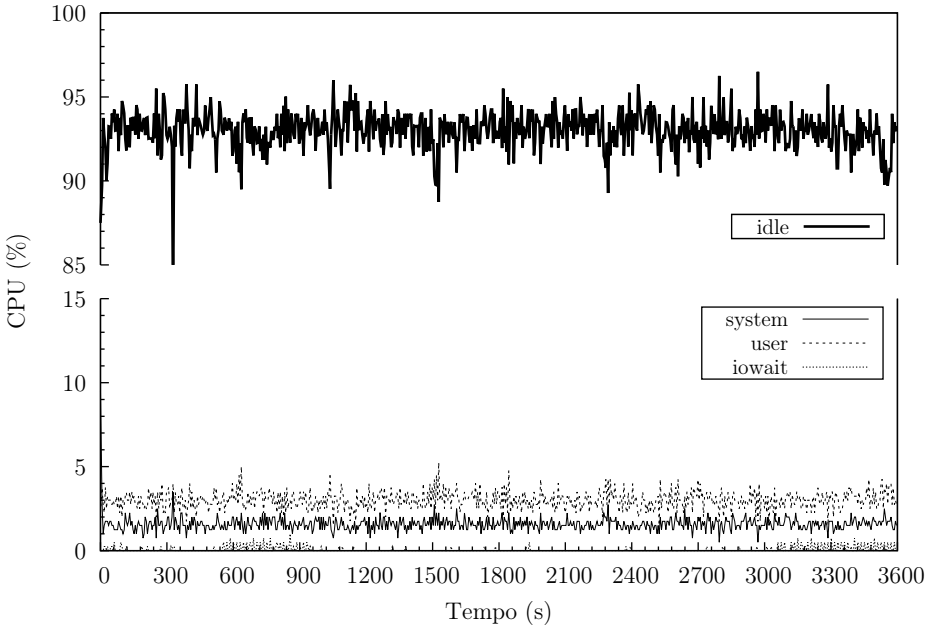


Figura 4.43: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 25ss. Erro nulo. Servidor Apache. CPU.

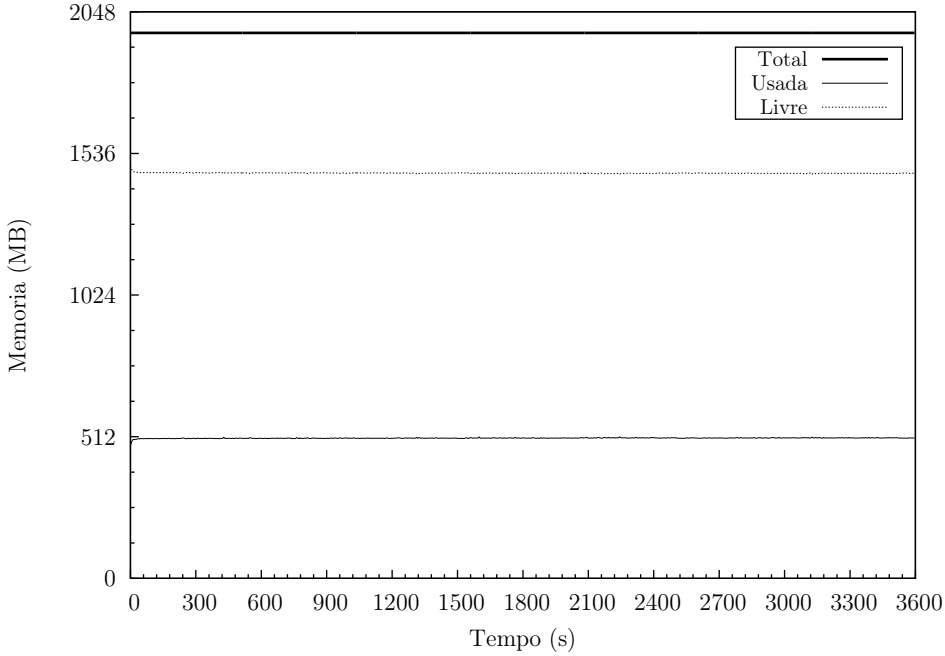


Figura 4.44: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 25ss. Erro nulo. Servidor Apache. Memória.

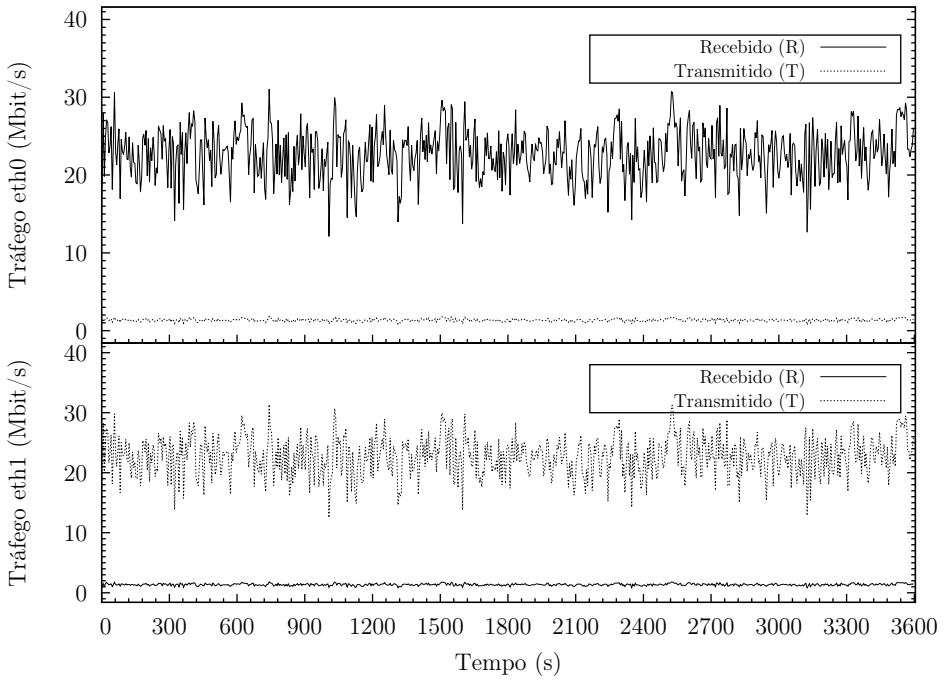


Figura 4.45: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 25ss. Erro nulo. *Gateway*. Tráfego de rede.

O cliente `Outros` foi, tal como esperado, o mais penalizado. A taxa de ligações foi 112,3% relativamente ao cliente `Premium`, mas a taxa de pedidos só foi 18,0%.

Convém frisar que no sistema implementado não existe relação direta entre o número de ligações e o número de pedidos. Basta um número razoável de ligações encerradas prematuramente, isto é: sem completarem todos os pedidos (ou até sem conseguirem fazer qualquer pedido), para que a um número elevado de ligações não corresponda um número elevado de pedidos e de respostas. Neste caso concreto, o cliente `Outros` apresenta uma maior taxa de ligações: 29,2 contra 26,0 do cliente `Premium`, mas um menor número de ligações simultâneas/concorrentes: 328 contra 387. Este aspeto permite constatar que o cliente `Outros` abre mais ligações por segundo porque não as consegue manter tanto tempo como o cliente `Premium`. Este aspeto pode ser constatado pelo resultado da variável `Connection time`, a qual fornece as estatísticas do tempo de vida[1] das ligações com sucesso, isto é: as ligações que obtiveram pelo menos uma resposta. No caso do cliente `Premium` a média foi 25462,7ms e no caso do cliente `Outros` foi 16194,7ms, ou seja: 63,6%. Tendo em consideração o que foi referido, é no cliente `Outros`, onde se regista um maior número de sessões encerradas após um baixo número de pedidos.

Sempre que é encerrada, o simulador `httperf` uma ligação abre outra. Este aspeto pode ser validado se observarmos as frequências absolutas de sessões.

Outro aspeto que é possível analisar é o rácio `taxa de respostas/taxa de pedidos`. No caso do cliente `Premium` esta relação foi 98,2% (433,3/441,3), enquanto do cliente `Outros` foi 76,3% (60,7/79,6), ou seja: uma menor taxa de resposta do servidor Apache ao cliente `Outros` o que evidencia, mais uma vez, uma diferenciação, pela negativa, deste cliente.

Comparando com as 50ss, no caso do cliente `Premium`:

- A taxa de ligações é consideravelmente inferior (26,0 contra 50,8 nas 50ss), o que se traduz num menor número de ligações concorrentes (até 387 contra até 532 nas 50ss). Este aspeto está ligado, como seria de esperar, à menor carga do cliente relativamente às 50ss;
- A taxa de pedidos é semelhante (441,3 contra 439,9 nas 50ss), bem como a taxa média de respostas (433,3 contra 426,4 nas 50ss), logo o rácio `taxa de respostas/taxa de pedidos` é muito semelhante, 98,2% e 96,9% nas 25ss e 50ss, respetivamente. Regista-se, assim, que quase todos os pedidos do cliente `Premium` foram respondidos pelo servidor Apache;
- Os resultados obtidos nas frequências absolutas de sessões é bastante melhor do que no caso das 50ss. Nas 25ss, 57793 sessões terminam após não receberem qualquer resposta, 4 sessões terminam após receberem 1 resposta e assim por diante. No caso das 50ss, 148064 sessões terminam após não receberem qualquer resposta, 5 sessões terminam após receberem 1 resposta e assim por diante.

[1] Tempo que decorre entre o início da ligação TCP e o seu encerramento.

Comparando com as 50ss, no caso do cliente `Outros`:

- A taxa de ligações é pouco mais de metade: 54,7% (29,2 contra 53,4 nas 50ss), o que se traduz num menor número de ligações concorrentes (328 contra 453 nas 50ss). Tal como no cliente `Premium` este aspeto está ligado, à menor carga do cliente relativamente às 50ss;
- A taxa de pedidos é semelhante (79,6 contra 84,4 nas 50ss) bem como a taxa média de respostas (60,7 e 61,8 nas 50ss), logo o rácio `taxa de respostas/taxa de pedidos` foi muito semelhante, 76,3% e 73,2%, respetivamente.
- Nas frequências absolutas das sessões verificamos que o número de sessões que foram encerradas depois de nenhum pedido recebido é menos de metade: 45,5% (76088/167387). Assim sendo, o cliente `Outros` não necessitou de abrir tantas ligações ao servidor Apache para efetuar praticamente o mesmo número de pedidos e obter número semelhante de respostas: 105104 ligações, 286421 pedidos, 218541 respostas nas 25ss e 192070 ligações, 303601 pedidos, 222206 respostas nas 50ss.

As respostas obtidas pelos clientes foram todas do tipo 2xx, ou seja: "Sucesso", o que indica que a ação foi recebida com sucesso, compreendida e aceite.

Não se registam erros devido à falta de descritores de ficheiros (`fd-unavail`), falta de portas TCP disponíveis (`addrunvail`), ou exaustão da tabela de descritores de ficheiros do sistema (`ftab-full`).

httperf - Cliente sítio `Premium`

```
httperf --hog --timeout=5 --client=0/1 --server=192.168.100.101 --port=80 --uri=/index.
    ↪html --rate=25 --send-buffer=4096 --recv-buffer=16384 --wsess=300000,50,2.000 --
    ↪burst-length=5
Maximum connect burst length: 31

Total: connections 93452 requests 1587941 replies 1558987 test-duration 3598.644 s

Connection rate: 26.0 conn/s (38.5 ms/conn, <=387 concurrent connections)
Connection time [ms]: min 1058.9 avg 25462.7 max 35016.6 median 26585.5 stddev 5013.1
Connection time [ms]: connect 1219.1
Connection length [replies/conn]: 46.133

Request rate: 441.3 req/s (2.3 ms/req)
Request size [B]: 78.0

Reply rate [replies/s]: min 222.8 avg 433.3 max 612.6 stddev 54.5 (719 samples)
Reply time [ms]: response 194.5 transfer 48.1
Reply size [B]: header 250.0 content 5043.0 footer 0.0 (total 5293.0)
Reply status: 1xx=0 2xx=1558987 3xx=0 4xx=0 5xx=0

CPU time [s]: user 670.74 system 2924.72 (user 18.6% system 81.3% total 99.9%)
Net I/O: 2273.4 KB/s (18.6*10^6 bps)

Errors: total 62815 client-timo 59330 socket-timo 0 connrefused 0 connreset 3485
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 8.42 max 23.20 stddev 2.48 (30302/89632)
Session: avg 1.06 connections/session
Session lifetime [s]: 27.4
Session failtime [s]: 6.0
Session length histogram: 57793 4 1 2 0 123 1 0 0 1 180 0 0 0 0 174 0 0 0 0 186 0 0 0 0
```

```
    ↪158 0 0 0 0 198 0 0 0 0 153 0 0 0 0 173 0 0 0 0 183 0 0 0 0 30302
```

httperf - Cliente sítio `Outros`

```
httperf --hog --timeout=5 --client=0/1 --server=192.168.100.102 --port=80 --uri=/index.
    ↪html --rate=25 --send-buffer=4096 --recv-buffer=16384 --wsess=300000,50,2.000 --
    ↪burst-length=5
Maximum connect burst length: 16

Total: connections 105104 requests 286421 replies 218541 test-duration 3598.470 s

Connection rate: 29.2 conn/s (34.2 ms/conn, <=328 concurrent connections)
Connection time [ms]: min 3980.7 avg 16194.7 max 55852.8 median 14741.5 stddev 6773.9
Connection time [ms]: connect 1483.3
Connection length [replies/conn]: 13.833

Request rate: 79.6 req/s (12.6 ms/req)
Request size [B]: 81.0

Reply rate [replies/s]: min 22.2 avg 60.7 max 224.8 stddev 18.0 (719 samples)
Reply time [ms]: response 971.7 transfer 171.6
Reply size [B]: header 251.0 content 5042.0 footer 0.0 (total 5293.0)
Reply status: 1xx=0 2xx=218541 3xx=0 4xx=0 5xx=0

CPU time [s]: user 688.57 system 2893.91 (user 19.1% system 80.4% total 99.6%)
Net I/O: 320.3 KB/s (2.6*10^6 bps)

Errors: total 104491 client-timo 89349 socket-timo 0 connrefused 0 connreset 15142
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 0.11 max 1.60 stddev 0.20 (405/89754)
Session: avg 1.67 connections/session
Session lifetime [s]: 43.7
Session failtime [s]: 8.4
Session length histogram: 76088 671 25 107 61 2096 137 4 19 6 3388 80 4 8 5 2286 38 0 5 6
    ↪ 1707 22 0 3 2 1057 9 1 3 1 626 6 0 0 0 429 3 0 0 0 276 1 0 0 2 163 4 0 0 0 405
```

Rejeição de perturbações

No que concerne à percentagem absoluta de processos HTTP (figura 4.46) e percentagem relativa de processos `Ativos` (figura 4.47), é possível notar que o sistema regista um comportamento semelhante ao do teste das 50ss.

É possível constatar que:

- Nos intervalos t_0 a t_{1200} e perto t_{2400} a t_{3600} existe uma coincidência total entre a percentagem de processos `Ativos` e processos `Premium`, já que o servidor Apache só está a processar pedidos do cliente `Premium`;
- Após o aparecimento dos pedidos do cliente `Outros`, o sistema conseguiu rapidamente se adaptar, de modo a manter o objetivo traçado: 80% da capacidade do servidor Apache, expressa em processos `Ativos`, a processar pedidos do sítio `Premium` e os remanescentes 20% a processar pedidos do sítio `Outros`;
- Existe um momento inicial, após a perturbação, no qual a percentagem dos processos `Premium` desce até cerca de 40%, mas recupera rapidamente.

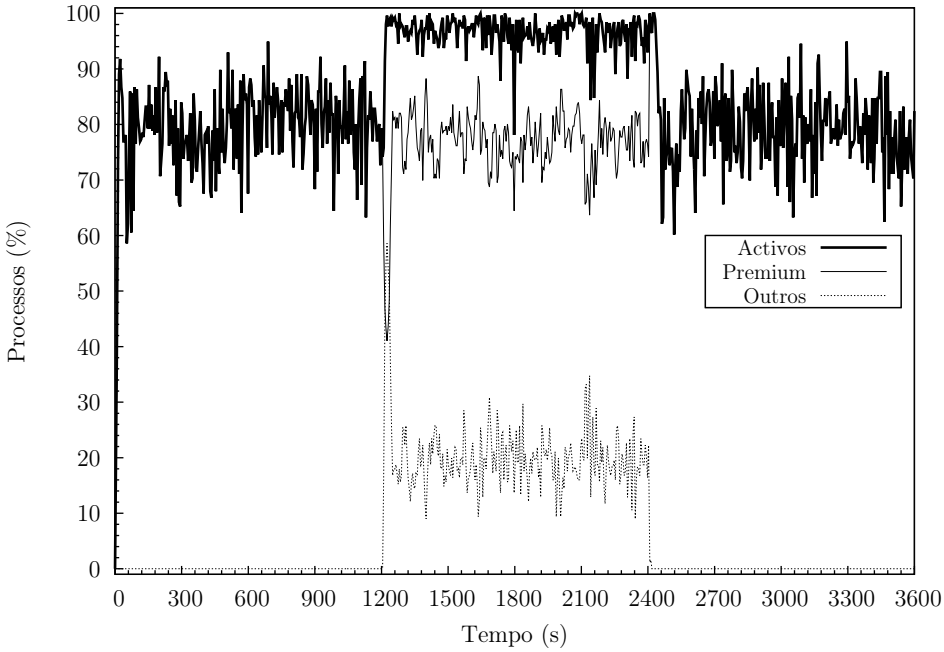


Figura 4.46: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 25ss. Rejeição de perturbações. Servidor Apache. Percentagem absoluta de processos.

Observa-se, assim, que o sistema, no que diz repeito ao seu comportamento global para a "Rejeição de perturbações", não foi influenciado pela taxa de sessões.

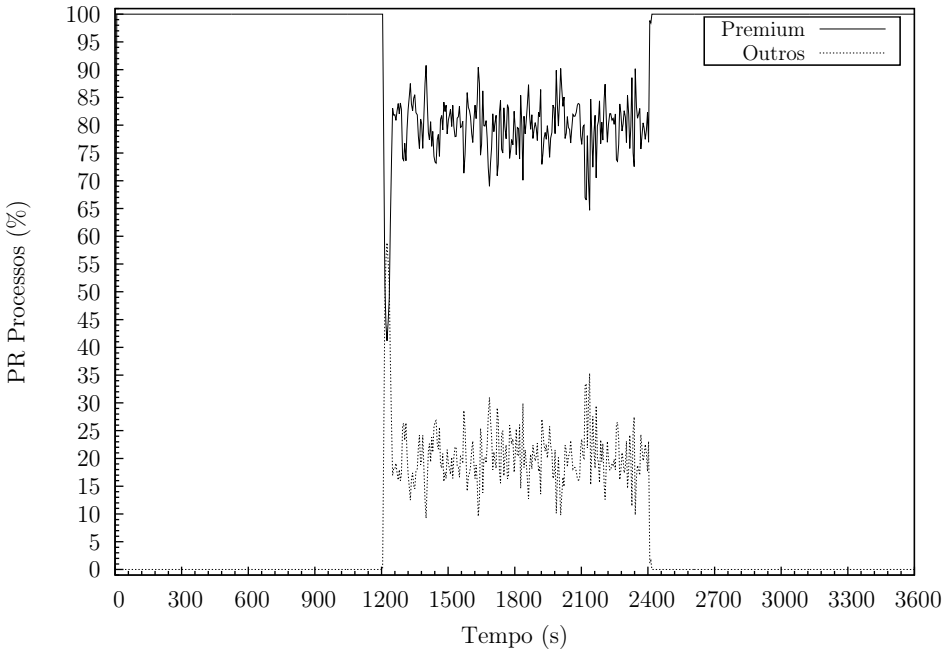


Figura 4.47: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 25ss. Rejeição de perturbações. Servidor Apache. Percentagem relativa de processos `Ativos`.

Tal como sucedeu nas 50ss, nos intervalos, t_0 a t_{1200} e perto t_{2400} a t_{3600}, regista-se um erro entre +20% e -20% relativamente ao objetivo do sítios `Premium` e `Outros`, repetivamente

(figura 4.48). Contudo, mais uma vez, este erro não é devido ao incorreto funcionamento do sistema, mas o resultado de nesses intervalos não existirem pedidos destinados ao sítio `Outros`. Nos segundos imediatamente após ao instante t_{1200} regista-se um erro positivo dos processos do sítio `Outros` e negativo dos processos do sítio `Premium`. Este erro resulta da ação do sistema ao detetar que no instante t_{1200} existe uma percentagem relativa superior à definida para os processos do sítio `Premium` e uma percentagem relativa inferior à definida para os processos do sítio `Outros`, o que induz o sistema a impor, nesse instante, um atraso aos pacotes destinados ao sítio `Premium` (figura 4.49).

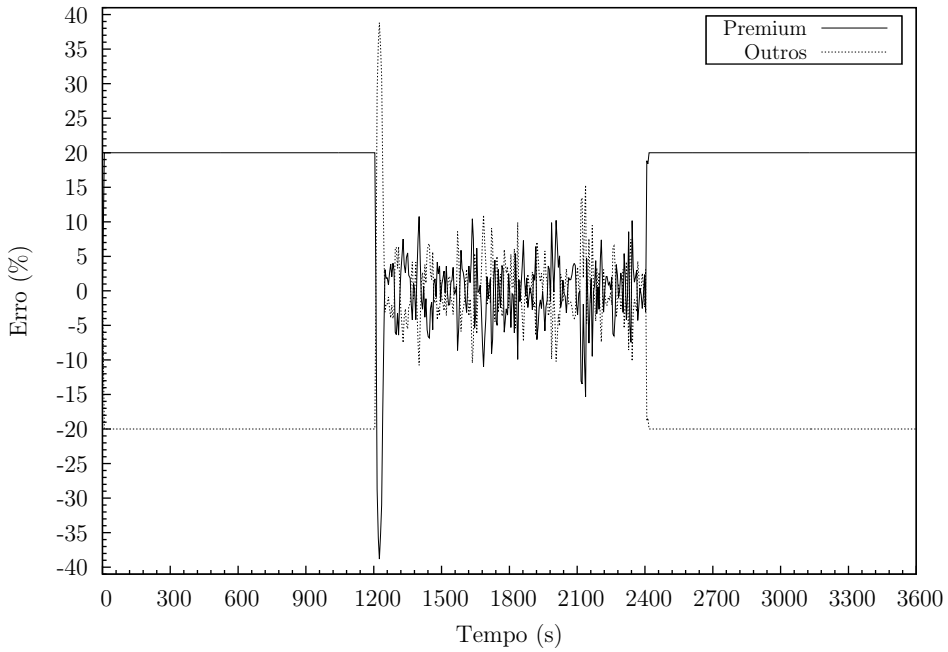


Figura 4.48: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 25ss. Rejeição de perturbações. Servidor Apache. Erro.

No que diz respeito ao atraso imposto pelo sistema aos pacotes (figura 4.49), regista-se o seguinte:

- Até ao instante t_{1210} é imposto um atraso de cerca de 400ms aos pacotes destinados ao sítio `Premium` (figura 4.49). O intuito do sistema é forçar o cumprimento do objetivo de 80% de processos a processar pedidos do sítio `Premium`;
- Entre o instante t_{1200} e o instante t_{2408} é unicamente imposto atraso aos pacotes do sítio `Outros`, sendo este atraso variável. Como é natural, está associado às percentagens de processos a processar pedidos do sítio `Premium` e do sítio `Outros`;
- Após o instante t_{2408} e como resultado do facto de desaparecerem pedidos destinados ao sítio `Outros`, só é imposto atraso aos pacotes destinados ao sítio `Premium`, sendo este atraso maior no instante t_{2400}. Este maior atraso permite ao sistema atingir mais rapidamente o objetivo definido.

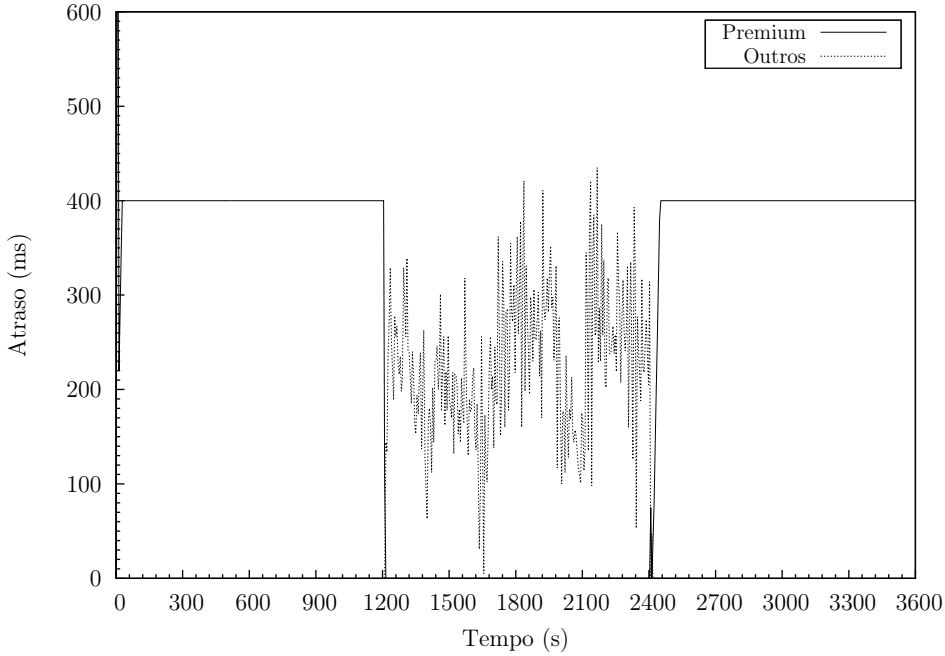


Figura 4.49: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 25ss. Rejeição de perturbações. *Gateway*. Atraso.

À semelhança de situações anteriores, nem o CPU (figura 4.50) nem a memória (figura 4.51) foram fatores com influência nos resultados do sistema.

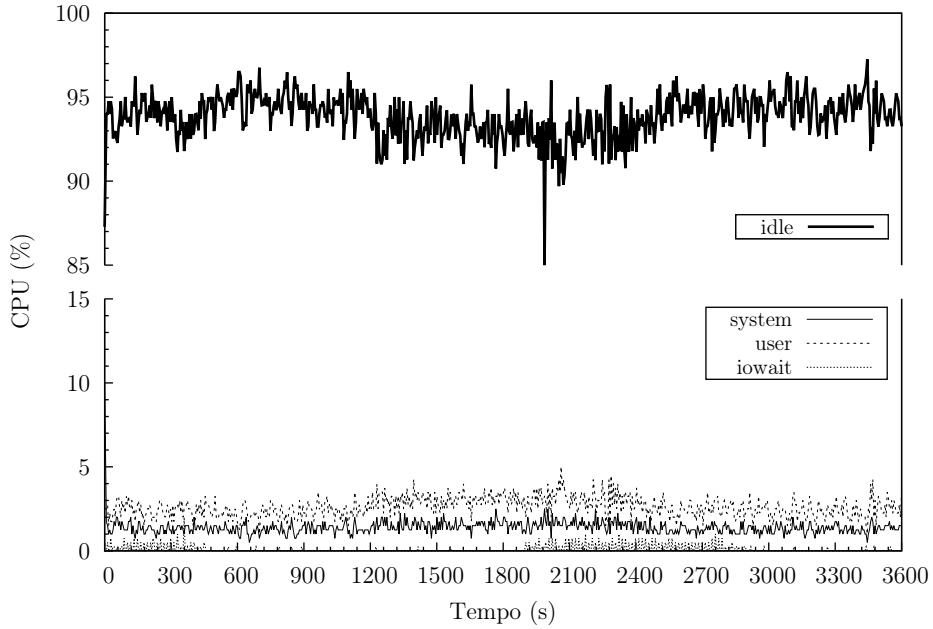


Figura 4.50: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 25ss. Rejeição de perturbações. Servidor Apache. CPU.

O tráfego de rede teve igualmente, neste caso, um comportamento semelhante às 50ss, isto é: existem três zonas distintas (figura 4.52), as quais coincidem com os intervalos em que só

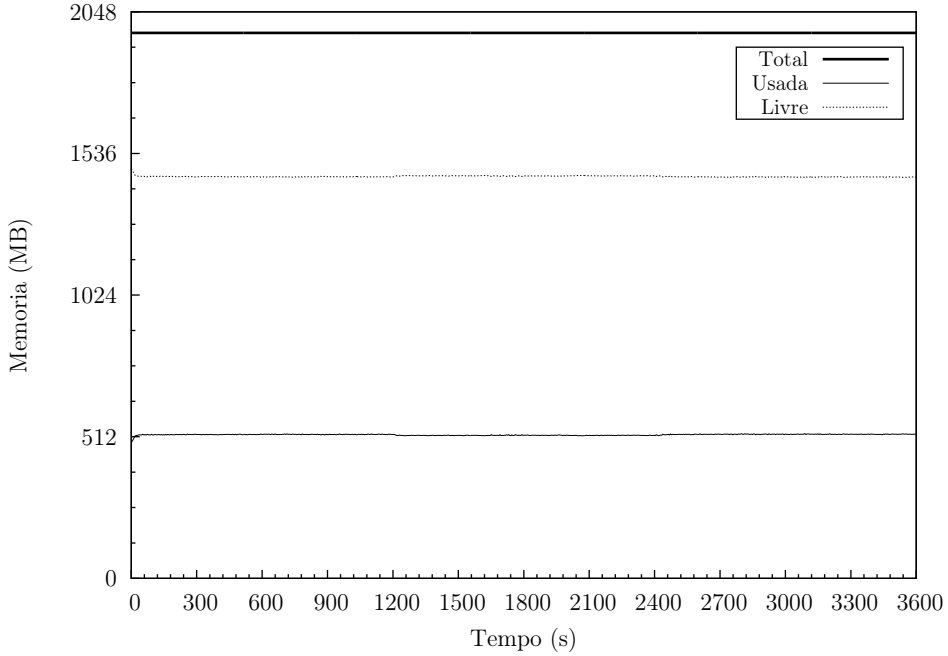


Figura 4.51: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 25ss. Rejeição de perturbações. Servidor Apache. Memória.

existem pedidos destinados ao sítio `Premium` (instante t_0 a instante t_{1200} e instante t_{2400} a instante t_{3600}) e o intervalo em que os pedidos destinados ao dois sítios coexistem (instante t_{1200} a t_{2400}).

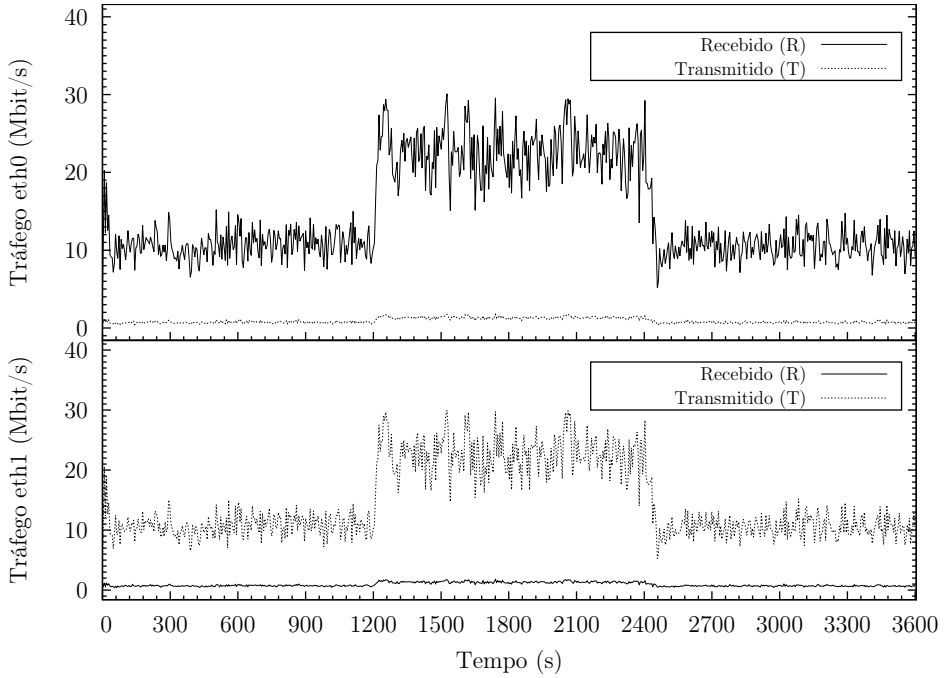


Figura 4.52: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 25ss. Rejeição de perturbações. *Gateway*. Tráfego de rede.

No caso do cliente `Premium` e comparando com os resultados obtidos nas 50ss (pág. 96), é possível constatar o seguinte:

- A taxa de ligações (lig/s) é consideravelmente inferior: 43,0 contra 66,8 nas 50ss, o que se traduz num menor número de ligações concorrentes: 579 contra 734 nas 50ss. Este aspeto está ligado, como seria de esperar, à menor carga do cliente relativamente às 50ss;
- A taxa de pedidos (ped./s) é semelhante: 343,8 contra 338,4 nas 50ss, bem como a taxa média de respostas (resp./s): 302,9 contra 283,4 nas 50ss, logo o rácio `taxa de respostas/taxa de pedidos` é muito semelhante, 88,1% e 83,7% nas 25ss e 50ss, respetivamente. Regista-se, assim, que a maioria dos pedidos do cliente `Premium` foi respondida pelo servidor Apache;
- Os resultados obtidos nas frequências absolutas de sessões é bastante melhor do que no caso das 50ss. Nas 25ss, 44559 sessões terminam após não receberem qualquer resposta, 1790 sessões terminam após receberem 1 resposta e assim por diante. No caso as 50ss, 125697 sessões terminam após não receberem qualquer resposta, 4760 sessões terminam após receberem 1 resposta e assim por diante.

No que diz respeito ao cliente `Outros`, comparando com as 50ss, sucedem vários factos que devem ser analisados:

- A taxa de ligações (lig/s) é pouco mais de metade: 54,2% (28,9 contra 53,3 nas 50ss), o que se traduz num menor número de ligações concorrentes (318 contra 481 nas 50ss). Tal como no cliente `Premium`, este aspeto está ligado, à menor carga do cliente relativamente às 50ss;
- A taxa de pedidos (ped./s) é semelhante: 83,7 contra 86,1 nas 50ss, bem como a taxa média de respostas (resp./s)123 65,0 e 64,2 nas 50ss, logo o rácio `taxa de respostas/taxa de pedidos` foi muito semelhante, 77,7% e 74,6%, respetivamente.
- Nas frequências absolutas de sessões verificamos que o número de sessões que são encerradas depois de nenhum pedido recebido é menos de metade: 45,8% (25475/55674). Assim sendo, o cliente `Outros` não necessitou de abrir tantas ligações as servidor Apache para efetuar praticamente o mesmo número de pedidos e obter número semelhante de respostas: 34693 ligações, 100509 pedidos, 78070 respostas nas 25ss e 63957 ligações, 103348 pedidos, 77006 respostas nas 50ss.

É possível notar que, neste caso, a taxa de ligações do cliente `Outros` é muito semelhante ao "Erro nulo (0)": 28,9 e 29,2, respetivamente. Também a taxa de pedidos é semelhante: 83,7 neste caso e 79,6 no "Erro nulo (0)", bem como a taxa média de respostas: 65,0 e 60,7 no "Erro nulo (0)". É, assim, validado o comportamento do sistema durante o período em que os pedidos coexistiam, já que se registou uma capacidade semelhante aos outros cenários por parte do cliente `Outros`.

Todas as respostas foram do tipo 2xx, ou seja: "Sucesso", o que indica que a ação foi recebida com sucesso, compreendida e aceite.

Não se registam erros devido à falta de descritores de ficheiros (`fd-unavail`), falta de portas TCP disponíveis (`addrunvail`), ou exaustão da tabela de descritores de ficheiros do sistema (`ftab-full`).

Podemos afirmar que, tal como seria de esperar, o sistema apresenta um melhor comportamento do que no caso das 50ss, sendo este aspeto particularmente visível se atentarmos ao rácio `taxa de respostas/taxa de pedidos` e às frequências absolutas de sessões. A explicação para este facto reside na menor carga exercida pelo cliente `Premium`, o que levou a que o servidor Apache tivesse mais recursos disponíveis para processar os pedidos. Ainda assim, constata-se a ação do controlador na diferenciação, pela positiva, do cliente `Premium`.

Julgamos que este ligeiro melhor comportamento do cliente `Outros` esteja relacionado com o ajustamento do sistema ao aparecimento dos seus pedidos no instante t_{1200}. Após esse instante e durante cerca de 100s é aplicado atraso aos pacotes do cliente `Premium` (figura 4.49). Existe igualmente perto desse instante uma percentagem superior ao objetivo de processos `Outros` (figura 4.48).

As respostas obtidas pelos clientes foram todas do tipo 2xx, ou seja: "Sucesso", o que indica que a ação foi recebida com sucesso, compreendida e aceite.

Não se registam erros devido à falta de descritores de ficheiros (`fd-unavail`), falta de portas TCP disponíveis (`addrunvail`), ou exaustão da tabela de descritores de ficheiros do sistema (`ftab-full`).

httperf - Cliente sítio `Premium`

```
httperf --hog --timeout=5 --client=0/1 --server=192.168.100.101 --port=80 --uri=/index.
    ↪html --rate=25 --send-buffer=4096 --recv-buffer=16384 --wsess=300000,50,2.000 --
    ↪burst-length=5
Maximum connect burst length: 73

Total: connections 153230 requests 1224126 replies 1078271 test-duration 3599.467 s

Connection rate: 43.0 conn/s (23.2 ms/conn, <=579 concurrent connections)
Connection time [ms]: min 2489.8 avg 14436.0 max 53948.0 median 12179.5 stddev 6997.9
Connection time [ms]: connect 1243.7
Connection length [replies/conn]: 13.924

Request rate: 343.8 req/s (2.9 ms/req)
Request size [B]: 82.0

Reply rate [replies/s]: min 107.6 avg 302.9 max 600.0 stddev 101.1 (712 samples)
Reply time [ms]: response 812.2 transfer 152.1
Reply size [B]: header 251.0 content 5042.0 footer 0.0 (total 5293.0)
Reply status: 1xx=0 2xx=1078271 3xx=0 4xx=0 5xx=0

CPU time [s]: user 477.76 system 3079.78 (user 13.4% system 86.5% total 99.9%)
Net I/O: 1593.4 KB/s (13.1*10^6 bps)

Errors: total 141030 client-timo 76812 socket-timo 0 connrefused 0 connreset 64218
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 3.28 max 15.60 stddev 3.87 (11686/88498)
Session: avg 1.55 connections/session
```

```
Session lifetime [s]: 32.5
Session failtime [s]: 15.2
Session length histogram: 44559 1790 80 670 54 6132 721 5 93 14 6360 534 13 137 21 4386
    ↪327 13 120 16 3189 259 7 76 10 2229 189 10 59 6 1648 123 2 36 7 1177 75 2 34 3 874
    ↪51 0 18 4 613 49 2 11 4 11686
```

httperf - Cliente sítio `Outros`

```
httperf --hog --timeout=5 --client=0/1 --server=192.168.100.102 --port=80 --uri=/index.
    ↪html --rate=25 --send-buffer=4096 --recv-buffer=16384 --wsess=300000,50,2.000 --
    ↪burst-length=5
Maximum connect burst length: 18

Total: connections 34693 requests 100509 replies 78070 test-duration 1200.446 s

Connection rate: 28.9 conn/s (34.6 ms/conn, <=318 concurrent connections)
Connection time [ms]: min 4953.8 avg 16887.1 max 51643.5 median 15215.5 stddev 7378.8
Connection time [ms]: connect 1459.0
Connection length [replies/conn]: 15.533

Request rate: 83.7 req/s (11.9 ms/req)
Request size [B]: 80.0

Reply rate [replies/s]: min 23.6 avg 65.0 max 239.4 stddev 28.7 (240 samples)
Reply time [ms]: response 879.8 transfer 160.7
Reply size [B]: header 251.0 content 5042.0 footer 0.0 (total 5293.0)
Reply status: 1xx=0 2xx=78070 3xx=0 4xx=0 5xx=0

CPU time [s]: user 224.94 system 951.04 (user 18.7% system 79.2% total 98.0%)
Net I/O: 342.8 KB/s (2.8*10^6 bps)

Errors: total 34215 client-timo 29534 socket-timo 0 connrefused 0 connreset 4681
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 0.22 max 7.80 stddev 0.61 (269/29803)
Session: avg 1.46 connections/session
Session lifetime [s]: 39.3
Session failtime [s]: 8.2
Session length histogram: 25475 239 8 24 21 594 53 1 11 1 930 32 1 2 0 694 11 0 2 0 517
    ↪21 0 1 0 315 6 0 0 0 204 1 0 0 0 155 5 0 0 1 110 3 0 1 0 93 0 1 1 0 269
```

Seguimento de Referência

Mais uma vez e tal como sucedeu nas 50ss, o objetivo é variável, isto é: em determinados intervalos (instante t_0 a instante t_{1200} e instante t_{2400} a instante t_{3600}) o objetivo é 80% da capacidade do servidor Apache, expressa em processos-filho `Ativos`, a processar pedidos do sítio `Premium` e os remanescentes 20% a processar pedidos do sítio `Outros` e noutro intervalo (instante t_{1200} a instante t_{2400}) o objetivo é 60% da capacidade do servidor Apache a processar pedidos do sítio `Premium` e os remanescentes 40% a processar pedidos do sítio `Outros` .

Contrariamente ao que sucedeu no caso do "Seguimento de Referência" nas 50ss, perto do instante t_{1200}, as percentagens absolutas não têm grande diferença relativamente ao pretendido (figura 4.53). Este instante corresponde ao momento em que o sistema se está a adaptar aos novos objetivos. Tendo em consideração que a carga exercida pelos clientes é menor, é natural que o sistema se adapte mais rapidamente e não existam *outliers* nesses instantes.

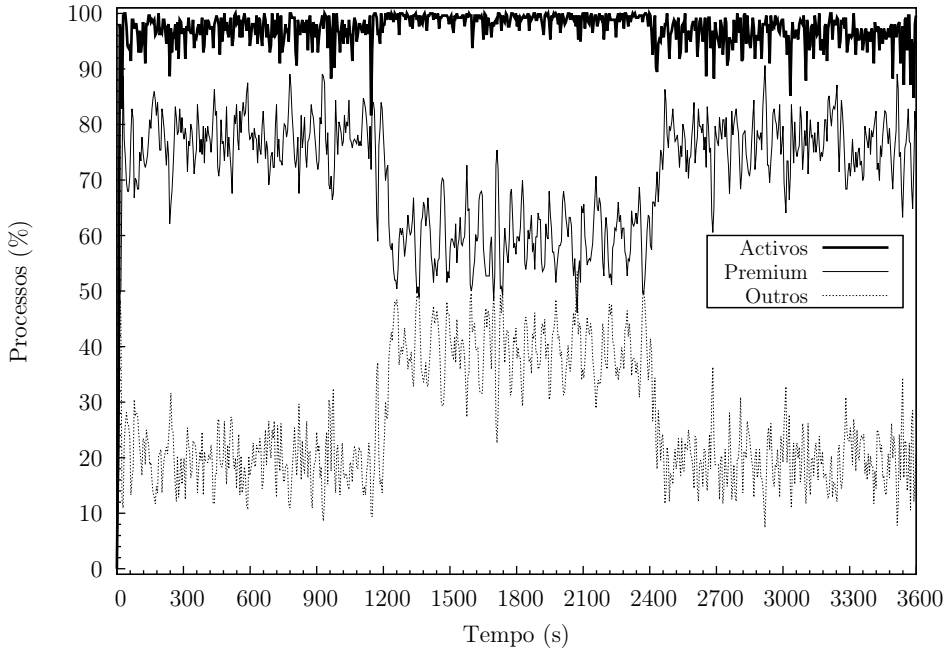


Figura 4.53: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 25ss. Seguimento de Referência. Servidor Apache. Percentagem absoluta de processos.

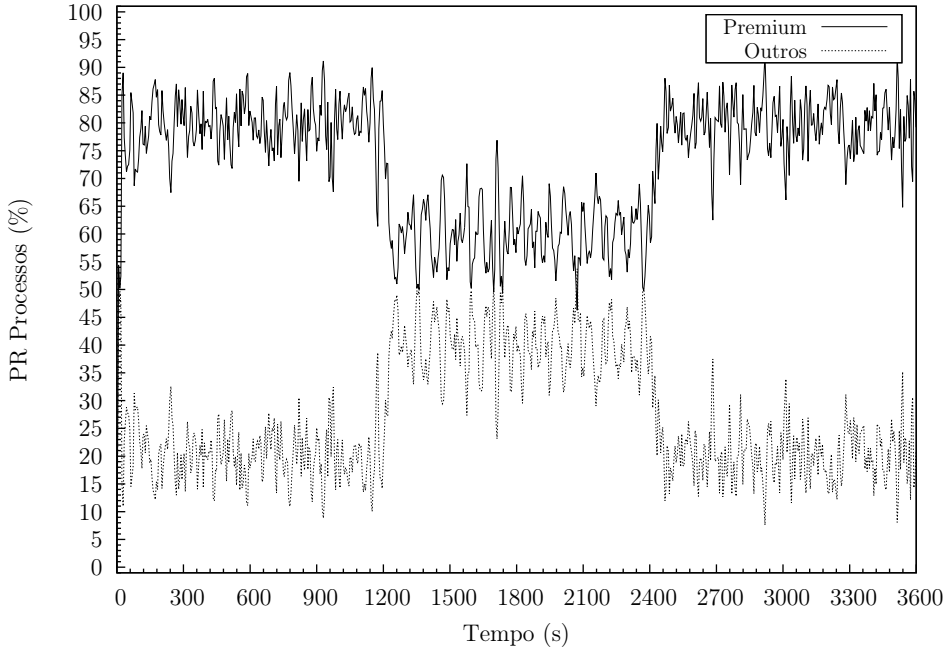


Figura 4.54: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 25ss. Seguimento de Referência. Servidor Apache. Percentagem relativa de processos `Ativos`.

De um modo geral, o erro das percentagens dos processos (figura 4.55) está compreendido entre intervalos aceitáveis: -15% e +15%. Existem quatro exceções:

1. Instante t_{1200}: Adaptação do sistema aos novos objetivos 60%/40%;
2. Perto do instante t_{1800}: Sem razão específica. Resulta da natureza estocática dos pedidos HTTP;

3. Instante t_{2400}: Adaptação do sistema aos novos objetivos 80%/20%;
4. Perto do instante t_{2700}: Sem razão específica. Resulta da natureza estocática dos pedidos HTTP.

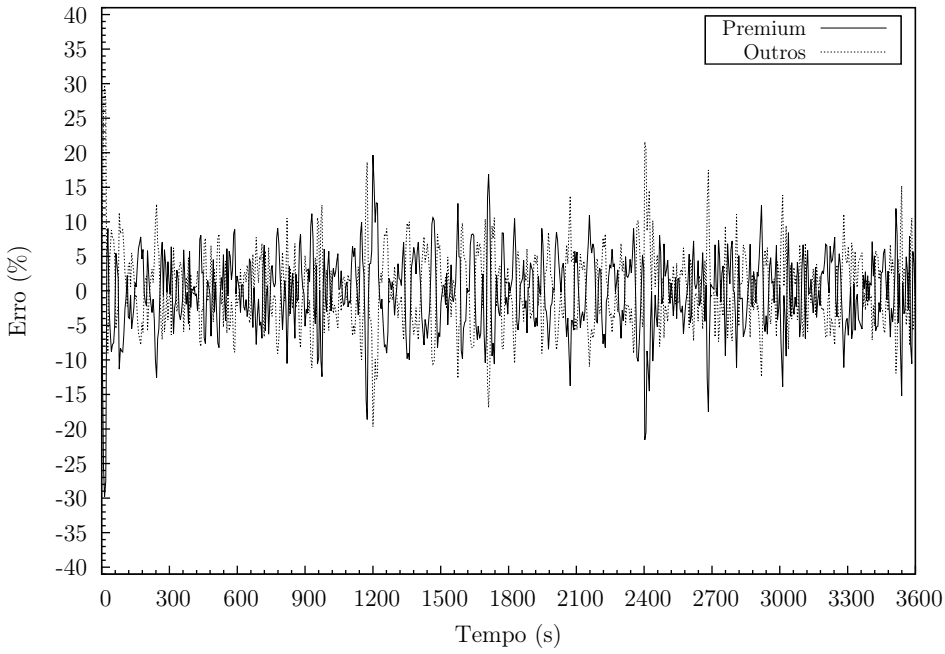


Figura 4.55: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 25ss. Seguimento de Referência. Servidor Apache. Erro.

Conforme se pode comprovar na figura 4.56 entre o instante t_{1200} e o instante t_{2400} são aplicados pontualmente pequenos atrasos aos pacotes destinados ao sítio `Premium`, facto que não sucede fora deste intervalo. Este situação pode ser explicada, pelo facto do objetivo dos processos `Premium` não diferir muito do objetivo dos processos `Outros`, 60% e 40%, respetivamente.

Mais uma vez, nem o CPU (figura 4.57), nem a memória (figura 4.58) foram fatores com influência nos resultados.

No que diz respeito ao tráfego de rede (figura 4.59) e tal como sucedeu nas 50ss, não existem zonas distintas no período de observação; o seu valor está compreendido entre os 10Mbit/s e os 35Mbit/s, mas com maior frequência entre os 15Mbit/s e os 30Mbit/s, sendo também neste aspeto semelhante à carga de 50ss, pelo que a alteração de objetivos ("Seguimento de referência") e do número de sessões por segundo (50ss para 25ss) não teve influência no comportamento da rede. Isto sucede porque, mesmo com essas alterações, a *Gateway* esteve constantemente a encaminhar pacotes com pedidos dos clientes e respostas do servidor Apache (geralmente saturado de pedidos), o que conforme referimos anteriormente é o fator causador do tráfego de rede.

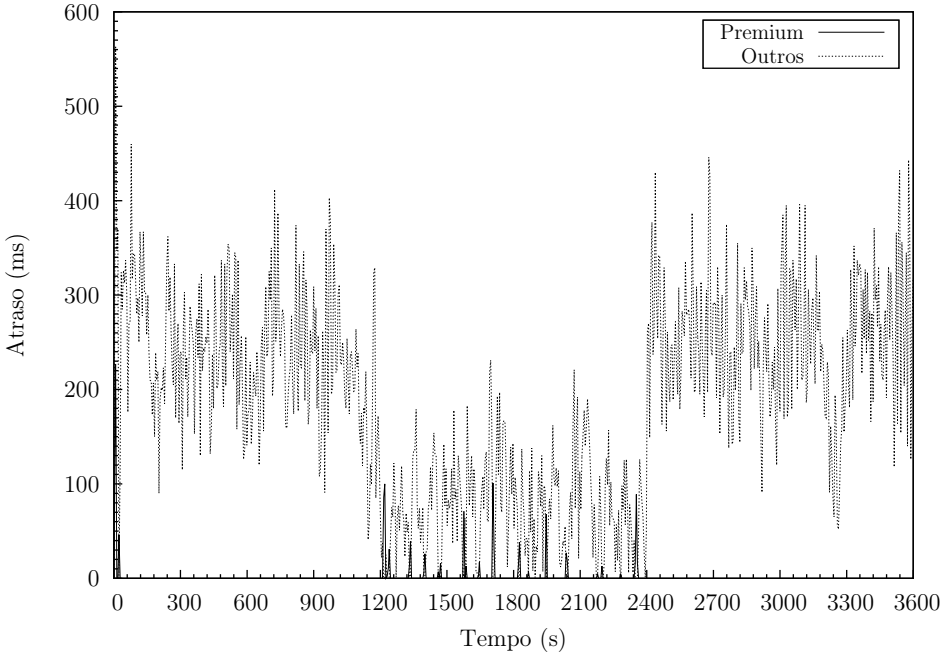


Figura 4.56: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 25ss. Seguimento de Referência. *Gateway*. Atraso.

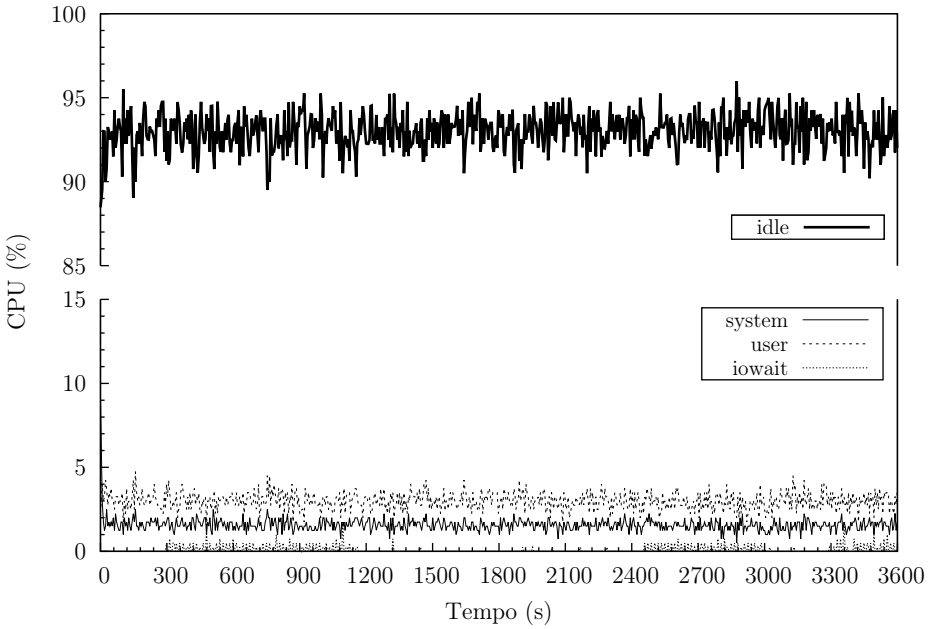


Figura 4.57: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 25ss. Seguimento de Referência. Servidor Apache. CPU.

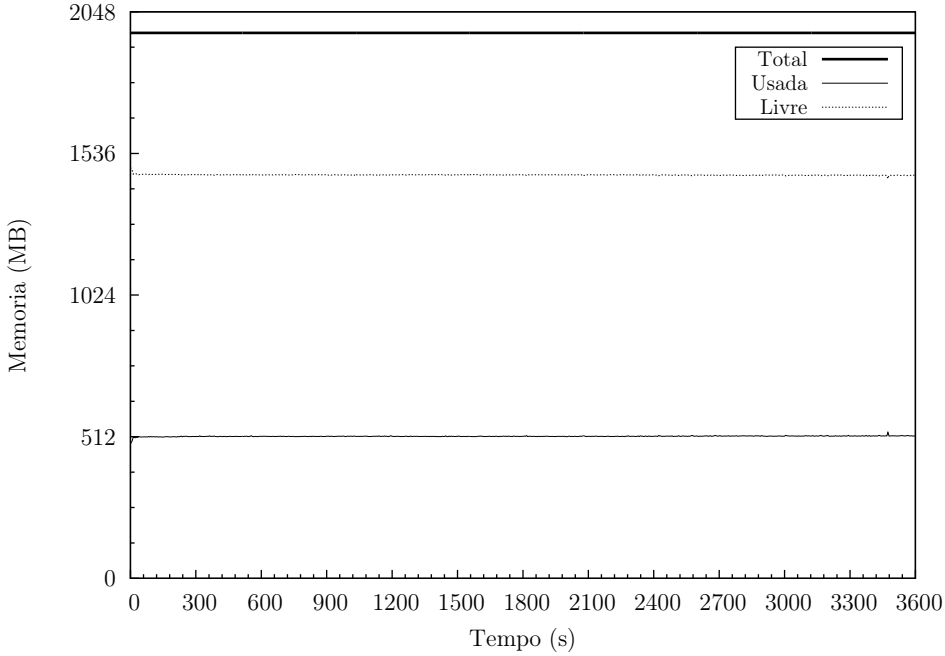


Figura 4.58: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 25ss. Seguimento de Referência. Servidor Apache. Memória.

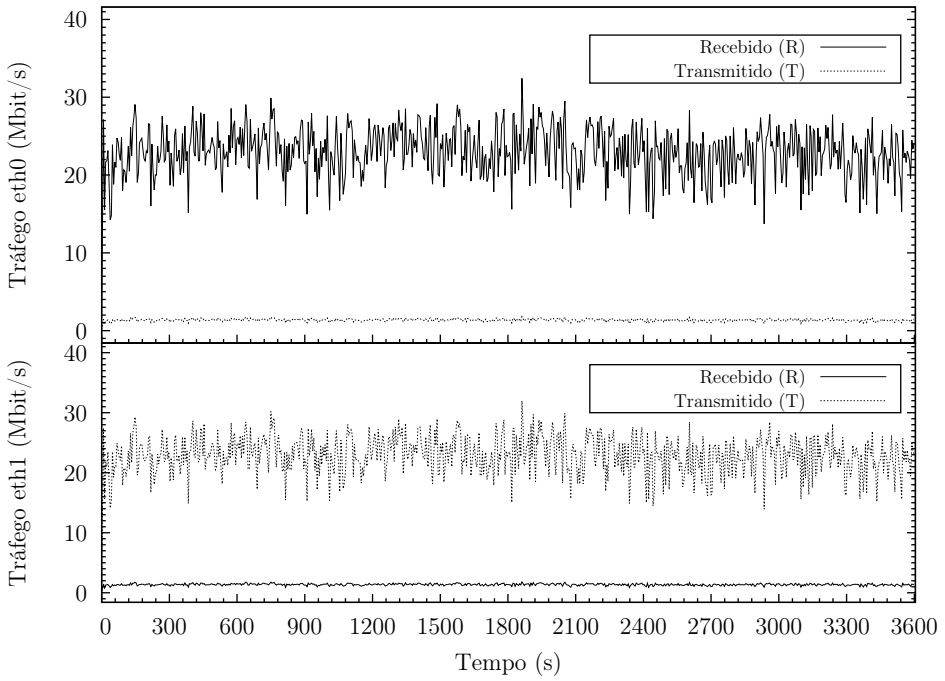


Figura 4.59: Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo ($k_p = 5, k_i = 3, k_d = 8$). 25ss. Seguimento de Referência. *Gateway*. Tráfego de rede.

O cliente `Outros` tem uma taxa de ligações superior ao cliente `Premium` (110,9%), mas apresenta uma taxa de pedidos e uma taxa média de repostas bem inferior, 29,0% e 25,3%, respetivamente. Mais uma vez, o cliente `Outros` abre mais ligações por segundo porque não as consegue manter tanto tempo como o cliente `Premium`. Este aspeto pode ser constatado pelo resultado da variável `Connection time`. No caso do cliente `Premium` a média foi 25747,5ms e no caso do cliente `Outros` foi 18944,8ms, ou seja: 73,6%. Tendo em consideração o que foi referido, é no cliente `Outros`, que se regista um maior número de sessões encerradas após um baixo número de pedidos. Consultando as frequências absolutas de sessões, verificamos que, tal como esperado, os resultados obtidos pelo cliente `Outros` foram bastante piores do que os do cliente `Premium`, existindo um maior número de sessões a terminar após um número reduzido de pedidos.

httperf - Cliente sítio `Premium`

```
httperf --hog --timeout=5 --client=0/1 --server=192.168.100.101 --port=80 --uri=/index.
    ↪html --rate=25 --send-buffer=4096 --recv-buffer=16384 --wsess=300000,50,2.000 --
    ↪burst-length=5
Maximum connect burst length: 30

Total: connections 92677 requests 1458637 replies 1428401 test-duration 3598.472 s

Connection rate: 25.8 conn/s (38.8 ms/conn, <=384 concurrent connections)
Connection time [ms]: min 2103.7 avg 25747.5 max 35886.9 median 26627.5 stddev 4754.6
Connection time [ms]: connect 1253.6
Connection length [replies/conn]: 46.718

Request rate: 405.3 req/s (2.5 ms/req)
Request size [B]: 78.0

Reply rate [replies/s]: min 182.4 avg 396.8 max 593.8 stddev 65.4 (719 samples)
Reply time [ms]: response 196.7 transfer 47.2
Reply size [B]: header 250.0 content 5043.0 footer 0.0 (total 5293.0)
Reply status: 1xx=0 2xx=1428401 3xx=0 4xx=0 5xx=0

CPU time [s]: user 687.17 system 2907.86 (user 19.1% system 80.8% total 99.9%)
Net I/O: 2083.1 KB/s (17.1*10^6 bps)

Errors: total 64495 client-timo 61780 socket-timo 0 connrefused 0 connreset 2715
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 7.74 max 16.00 stddev 2.40 (27856/89636)
Session: avg 1.05 connections/session
Session lifetime [s]: 27.5
Session failtime [s]: 6.0
Session length histogram: 60557 4 0 0 1 139 3 0 1 0 108 1 0 0 0 121 1 0 1 0 140 0 0 0 0
    ↪137 0 0 0 0 156 0 0 0 0 0 155 0 0 0 0 0 121 0 0 0 0 0 134 0 0 0 0 0 27856
```

httperf - Cliente sítio `Outros`

```
httperf --hog --timeout=5 --client=0/1 --server=192.168.100.102 --port=80 --uri=/index.
    ↪html --rate=25 --send-buffer=4096 --recv-buffer=16384 --wsess=300000,50,2.000 --
    ↪burst-length=5
^CMaximum connect burst length: 16

Total: connections 103051 requests 423042 replies 360370 test-duration 3598.280 s

Connection rate: 28.6 conn/s (34.9 ms/conn, <=319 concurrent connections)
Connection time [ms]: min 3218.9 avg 18944.8 max 52653.4 median 16644.5 stddev 8412.5
Connection time [ms]: connect 1440.8
```

```
Connection length [replies/conn]: 21.368

Request rate: 117.6 req/s (8.5 ms/req)
Request size [B]: 79.0

Reply rate [replies/s]: min 19.0 avg 100.2 max 270.8 stddev 61.6 (719 samples)
Reply time [ms]: response 631.6 transfer 117.2
Reply size [B]: header 251.0 content 5042.0 footer 0.0 (total 5293.0)
Reply status: 1xx=0 2xx=360370 3xx=0 4xx=0 5xx=0

CPU time [s]: user 677.60 system 2893.82 (user 18.8% system 80.4% total 99.3%)
Net I/O: 526.9 KB/s (4.3*10^6 bps)

Errors: total 99199 client-timo 86106 socket-timo 0 connrefused 0 connreset 13093
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 1.01 max 6.00 stddev 1.47 (3627/89733)
Session: avg 1.21 connections/session
Session lifetime [s]: 33.2
Session failtime [s]: 8.1
Session length histogram: 75061 493 8 69 32 1638 132 4 16 2 2604 77 1 3 1 1832 39 1 4 1
    ↪1453 24 3 1 1 918 11 0 2 0 603 3 0 0 0 479 5 0 0 0 307 1 0 0 0 276 0 0 0 1 3627
```

5. Conclusões e Trabalho Futuro

Ao longo desta dissertação são abordadas questões relativas aos servidores HTTP, com particular ênfase no servidor HTTP Apache, sistemas de controlo e qualidade de serviço. São analisadas as principais características e especificidades do servidor HTTP Apache, sendo proposto e implementado um sistema de controlo em cadeia fechada. O objetivo principal do sistema proposto é realizar a diferenciação de serviço num servidor sujeito a um elevado número de ligações e pedidos, no qual existe competição intensiva pelos recursos, permitindo melhorar a QoS do sítio *Premium*.

Em primeiro lugar, é definido e implementado um sistema original de controlo em cadeia fechada de QoS, aplicado ao servidor HTTP Apache. Em segundo lugar, é demonstrada a robustez, estabilidade e capacidade para gerir os recursos de forma dinâmica do sistema. Por fim, é comprovada a sua capacidade de realizar a diferenciação de serviço pretendida entre as duas classes de sítios: *Premium* e *Outros*. Esta qualidade é comprovada para o *Erro nulo*, *Rejeição de perturbações* e *Seguimento de referência com erro nulo*.

Existem, no entanto, várias linhas de trabalho em aberto as quais, julgamos, permitirão ampliar a aplicabilidade do sistema e facilitar a sua integração com o servidor HTTP Apache. Como principais linhas de trabalho futuro podemos mencionar:

- Modificar o sistema de modo a ser possível alterar, em tempo de execução, a percentagem de processos destinados ao sítio `Premium`. Para o efeito, tem de existir capacidade em interagir e atualizar dinamicamente o sistema de controlo. Foi constatado neste trabalho, através do *Seguimento de referência*, que o sistema implementado é capaz de se ajustar dinamicamente a novos objetivos;
- Implementar a funcionalidade de alterar, em tempo de execução, alguns dos parâmetros do servidor Apache, nomeadamente `KeepAlive` e `MaxClients`, o que permitirá em situações pontuais ajustar e melhorar o desempenho do sistema;
- Transformar o que neste momento é um *patch* do código fonte do servidor HTTP Apache num módulo Apache, o que torná o sistema mais eficaz e flexível;
- Melhorar a gestão da fila de espera. Este objetivo pode ser atingido definindo uma fila de espera dinâmica (de comprimento variável) e registo do tempo que os pedidos aguardam serviço;

- Determinar uma metodologia para afinação das constantes do controlador do sistema (k_p, k_i, k_d);
- Testar o sistema proposto com páginas PHP (conteúdo dinâmico) e com Sistemas de Gestão de Bases de Dados (SGBD);
- Implementar e testar um sistema semelhante para o `MPM Worker`, já que nem sempre é utilizado o PHP nos sítios de Internet;
- Testar o sistema com outras versões do servidor HTTP Apache, nomeadamente a versão mais recente: 2.4;
- Equacionar a alteração deste sistema para comportar duas vertentes: a priorização de clientes privilegiados e/ou de sítios privilegiados. Neste trabalho foi apenas abordada a vertente sítios privilegiados.

Em suma, o sistema de controlo em cadeia fechada proposto nesta dissertação constitui um contributo válido para a definição de mecanismos de diferenciação de serviço em servidores HTTP Apache. Podemos considerar que os objetivos estabelecidos foram alcançados. Além de possíveis melhorias na implementação deste sistema existem, ainda, várias linhas de trabalho em aberto, representando a sua implementação um desafio aliciante.

Referências Bibliográficas

[1] T. F. Abdelzaher e N. Bhatti, "Web server QoS management by adaptive content delivery," in *In International Workshop on Quality of Service*, 1999.

[2] T. F. Abdelzaher e K. G. Shin, "Qos provisioning with qcontracts in web and multimedia servers," in *Real-Time Systems Symposium, 1999. Proceedings. The 20th IEEE.* IEEE, 1999, p. 44–53.

[3] Apache Software Foundation, "Apache Core Features," http://httpd.apache.org/docs/current/mod/core.html, acedido 10 agosto 2015.

[4] Apache Software Foundation, "Apache MPM," http://httpd.apache.org/docs/2.2/mpm.html, acedido 10 agosto 2015.

[5] Apache Software Foundation, "Apache MPM Common Directives," http://httpd.apache.org/docs/2.2/mod/mpm_common.html, acedido 10 agosto 2015.

[6] Apache Software Foundation, "Apache MPM prefork," http://httpd.apache.org/docs/2.2/mod/prefork.html, acedido 10 agosto 2015.

[7] Apache Software Foundation, "Apache MPM worker," http://httpd.apache.org/docs/2.2/mod/worker.html, acedido 10 agosto 2015.

[8] Apache Software Foundation, "Apache Performance Tuning," http://httpd.apache.org/docs/2.2/misc/perf-tuning.html, acedido 10 agosto 2015.

[9] Apache Software Foundation, "Apache Portable Runtime (APR)," http://apr.apache.org/, acedido 10 agosto 2015.

[10] Apache Software Foundation, "Apache Portable Runtime (APR) - Projectos," http://apr.apache.org/projects.html, acedido 10 agosto 2015.

[11] Apache Software Foundation, "Apache Virtual Host documentation," http://httpd.apache.org/docs/2.2/vhosts/, acedido 10 agosto 2015.

[12] Apache Software Foundation, "O projecto Servidor HTTP Apache," http://httpd.apache.org/ABOUT_APACHE.html, acedido 10 agosto 2015.

[13] D. Balan e D. Potorac, "Extended Linux HTB Queuing Discipline Implementations," *International Journal of Information Studies*, vol. 2, no. 2, 2010.

[14] Y. Benita, "Analysis of the HTB Queuing Discipline," *Linux journal*, no. 131, p. 22–27, 2005.

[15] N. Bhatti e R. Friedrich, "Web server support for tiered services," *Network, IEEE*, vol. 13, no. 5, p. 64–71, 1999.

[16] J. M. Blanquer *et al.*, "QoS for internet services: done right," in *Proceedings of the 11th workshop on ACM SIGOPS European workshop*. ACM, 2004, p. 8.

[17] M. A. Brown, "Traffic Control HOWTO," http://linux-ip.net/articles/Traffic-Control-HOWTO/index.html, acedido 10 agosto 2015.

[18] P. Buchbinder, "mod_qos," http://opensource.adnovum.ch/mod_qos/, acedido 10 agosto 2015.

[19] CERT, "Code Red Worm Exploiting Buffer Overflow In IIS Indexing Service DLL," http://www.cert.org/historical/advisories/ca-2001-19.cfm, acedido 10 agosto 2015.

[20] CERT, "Nimda Worm," http://www.cert.org/historical/advisories/ca-2001-26.cfm, acedido 10 agosto 2015.

[21] K. H. Chan e X. Chu, "Design of a Fuzzy PI Controller to Guarantee Proportional Delay Differentiation on Web Servers," in *Proceedings of the 3rd International Conference on Algorithmic Aspects in Information and Management*, ser. AAIM '07. Springer-Verlag, 2007, p. 389–398.

[22] K. H. Chan e X. Chu, "Using Fuzzy PI Controller to Provide QoS on Web Servers," *Jornal*, 2007.

[23] P. Dash, "Bandwidth throttling with netem network emulation," http://www.opensourceforu.com/2012/06/bandwidth-throttling-netem-network-emulation, acedido 10 agosto 2015.

[24] M. Devera e D. Cohen, "HTB Linux queuing discipline manual - user guide," http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm, acedido 10 agosto 2015.

[25] Y. Diao, G. Neha, J. L. Hellerstein, S. Parekh, e D. M. Tilbury, "Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache Web server," in *In Proceedings of the Network Operations and Management Symposium 2002*, 2002, p. 219–234.

[26] J. Doyle *et al.*, *Feedback Control Theory*. Macmillan Publishing Co., 1990.

[27] L. Eggert e J. Heidemann, "Application-level differentiated services for Web servers," *World Wide Web*, vol. 2, no. 3, p. 133–142, 1999.

[28] J. Hellerstein *et al.*, *Feedback Control of Computing Systems*. John Wiley & Sons IEE Press, 2004.

[29] S. Hemminger, "Network Emulation with NetEm," in *LCA 2005, Australia's 6th national Linux conference (linux.conf.au)*, M. Pool, Ed., Linux Australia. Linux Australia, 2005.

[30] Hewlett-Packard Research Laboratories, "httperf," http://www.hpl.hp.com/research/linux/httperf/, acedido 10 agosto 2015.

[31] X. Huang, "UsageQoS: Estimating the QoS of Web services through online user communities," *ACM Transactions on the Web (TWEB)*, vol. 8, no. 1, p. 1, 2013.

[32] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. Schroeder, J. Spaans, e P. Larroy, "Linux advanced routing & traffic control," in *Ottawa Linux Symposium*, 2002, p. 213.

[33] IANA, "IANA - HTTP Status Codes," http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml, acedido 10 agosto 2015.

[34] M. D. A. M. J. Almedia, P. Cao, J. A. M. D. A. Manikutty, e P. Cao, "Providing differentiated levels of service in web content hosting," in *Proc. First Workshop Internet Server Performance*, 1998.

[35] N. Kew, *The Apache Modules Book: Application Development with Apache (Prentice Hall Open Source Software Development Series)*. Prentice Hall PTR, 2007.

[36] A. Kleen e M. Kerrisk, "Linux Programmer's Manual," http://www.kernel.org/doc/man-pages/online/pages/man7/tcp.7.html, acedido 10 agosto 2015.

[37] E. Lindegren, "Preparing the Apache HTTP Server for Feedback Control Application," Department of Automatic Control, Lund University, Sweden, Master's Thesis, 2008.

[38] Linux Foundation, "iproute2," http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2, acedido 10 agosto 2015.

[39] Linux Foundation, "netem," http://www.linuxfoundation.org/collaborate/workgroups/networking/netem, acedido 10 agosto 2015.

[40] M. Loudini e S. Rezig, "Enhancing Web Server Relative Delay Services by an Integrated SA-fuzzy Logic Controller," *Int. J. Web Eng. Technol.*, vol. 8, no. 1, p. 27–57, Mar. 2013.

[41] C. Lu, Y. Lu, T. F. Abdelzaher, J. A. Stankovic, e S. H. Son, "Feedback Control Architecture and Design Methodology for Service Delay Guarantees in Web Servers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 9, p. 1014–1027, 2006.

[42] Netcraft, "Netcraft - February 2015 Web Server Survey," http://news.netcraft.com/archives/2015/02/24/february-2015-web-server-survey.html, acedido 10 agosto 2015.

[43] K. I. Park, *QoS in Packet Networks*. Springer Science & Business Media, 2005.

[44] X. Peng, B. Chen, Y. Yu, e W. Zhao, "Self-Tuning of Software Systems Through Goal-based Feedback Loop Control," in *Proceedings of the 2010 18th IEEE International Requirements Engineering Conference*, ser. RE '10. IEEE Computer Society, 2010, p. 104–107.

[45] P. Pivoňka, "Comparative analysis of fuzzy PI/PD/PID controller based on classical PID controller approach," in *Proceedings of the 2002 IEEE International Conference on Fuzzy systems*, 2002, p. 541–546.

[46] A. Robertsson, B. Wittenmark, e M. Kihl, "Analysis and Design of Admission Control in Web-Server Systems," in *In American Control Conference (ACC)*, 2003.

[47] M. A. Serhani, Y. Atif, e A. Benharref, "Towards an Adaptive QoS-driven Monitoring of Cloud SaaS," *Int. J. Grid Util. Comput.*, vol. 5, no. 4, p. 263–277, 2014.

[48] A. Tanenbaum e D. Wetherall, *Computer Networks*, 5th ed. Prentice Hall, 2011.

[49] w3techs, "w3techs - Overview Programming Language," http://w3techs.com/technologies/overview/programming_language/all, acedido 10 agosto 2015.

[50] M. Welsh e D. Culler, "Adaptive Overload Control for Busy Internet Servers," in *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, ser. USITS'03. Berkeley, CA, USA: USENIX Association, 2003.

[51] T. Wescott, "PID without a PhD," *Embedded Systems Programming*, p. 86–108, out 2000.

[52] T. Wescott, *Applied Control Theory for Embedded Systems*. Elsevier, 2006.

[53] World Wide Web Consortium, "W3C HTTP Response," http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html, acedido 10 agosto 2015.

Anexos

A. Código fonte

A.1 *Gateway*

Gateway - server.h

```
2  #ifndef SERVER_H
3  #define SERVER_H

5  #ifdef __cplusplus
6  extern "C" {
7  #endif

9  #ifdef __cplusplus
10 }
11 #endif

13 #endif /* SERVER_H */

15 typedef struct {
16     float kp, ki, kd;
17     float termoP, termoI, termoD, termoTotal;
18     float iState;
19     float dState;
20     float iMax, iMin;
21 } PID;

23 typedef struct {
24     float eth0rv, eth0tv;
25     float eth1rv, eth1tv;
26 } NETFLOW;

28 #define MAXCON 5 // Numero maximo de pedidos de ligacao que sao aceites
29 #define BUFFSIZE 128 // Tamanho do buffer associado a informacao a receber
30 #define PORTA 5000 //Porta em que o serviço irá escutar

32 // Parametros do Controlador PID do site "Premium"
33 #define kpPremium 5
34 #define kiPremium 3
35 #define kdPremium 8

37 // Parametros do Controlador PID do site "Outros"
38 #define kpOutros 5
39 #define kiOutros 3
40 #define kdOutros 8

42 //Delay (ms)
43 #define delayMIN 0
44 #define delayMAX 5000
45 #define delayMAXABS 5000

47 // Ficheiros de input e output
48 #define FILE_IN_NET "/proc/net/dev"
49 #define FILE_OUT_LOG "/mst/output.dat"
```

Gateway - server.c

```
1  #include <stdio.h>
2  #include <sys/socket.h>
3  #include <arpa/inet.h>
4  #include <stdlib.h>
5  #include <string.h>
6  #include <unistd.h>
```

```
7  #include <netinet/in.h>
8  #include <time.h>
9  #include "server.h"

11 /*
12  * Prototipos das funcoes
13  */

15 // TCP socket
16 void erro(char *mess);
17 void recebeInfoApache(int sock);
18 int validaInfoApache(char c[]);
19 void trataInfoApache(char c[]);
20 char* subs(int i);

22 // Feedback control
23 void setPID();
24 void controladorPID(char *infoP[]);
25 void reset();
26 int actualizaControladorPID(PID *pid, float erro, int pWP);

28 // Info Rede
29 void trafegoRedeGW(NETFLOW *n);

31 // Debug e Log
32 void logEcran();
33 void logFicheiro(char log[]);

35 /*
36  * Variaveis globais
37  */

39 int sysIntv;

41 float setPointBaseline;
42 float setPointPremium1_Rel, setPointPremium2_Rel, setPointPremium3_Rel;
43 float setPointOutros1_Rel, setPointOutros2_Rel, setPointOutros3_Rel;
44 float setPointPremiumActual_Rel, setPointOutrosActual_Rel;

46 float pTotal_Abs, pPremium_Abs, pOutros_Abs, pPremium_Rel, pOutros_Rel;
47 int delayPremium, delayOutros;
48 float erroPremium_Rel, erroOutros_Rel;
49 PID pidPremium, pidOutros;
50 NETFLOW nf;

52 time_t tInicio, tActual;
53 double tDiffAbs;
54 double tDiffRel1, tDiffRel2, tDiffRel3, tDiffRel4;

56 /*
57  * Codigo
58  */

60 void erro(char *mess) {

62     perror(mess);
63     exit(1);

65 }

67 int main(int argc, char *argv[]) {

69     int serversock, clientsock;
70     struct sockaddr_in servidor, cliente;

72     if ((serversock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
73         erro("Failed to create socket");
74     }
75     memset(&servidor, 0, sizeof (servidor));
76     servidor.sin_family = AF_INET;
77     servidor.sin_addr.s_addr = htonl(INADDR_ANY);
78     servidor.sin_port = htons(PORTA);

80     if (bind(serversock, (struct sockaddr *) & servidor, sizeof (servidor)) < 0) {
81         erro("Não foi possivel fazer bind ao socket do servidor");
82     }
83     if (listen(serversock, MAXCON) < 0) {
84         erro("Não foi possivel escutar(listen) no socket do servidor");
85     }

87     setPointPremiumActual_Rel = setPointPremium1_Rel = atof(argv[1]);
88     setPointPremium2_Rel = atof(argv[2]);
89     setPointPremium3_Rel = atof(argv[3]);

91     setPointOutrosActual_Rel = setPointOutros1_Rel = 100.0 - setPointPremium1_Rel;
```

```
92      setPointOutros2_Rel = 100.0 - setPointPremium2_Rel;
93      setPointOutros3_Rel = 100.0 - setPointPremium3_Rel;

95      tDiffRel1 = atof(argv[4]);
96      tDiffRel2 = atof(argv[5]);
97      tDiffRel3 = atof(argv[6]);
98      tDiffRel4 = atof(argv[7]);

100     tInicio = time(0);

102     sysIntv = 0;

104     logFicheiro("tDiffAbs, pTotal_Abs, pPremium_Abs, pOutros_Abs, pPremium_Rel, pOutros_Rel, erroPremium_Rel,
            ↪erroOutros_Rel, pcpu_t, pcpu_us, pcpu_ni, pcpu_sy, pcpu_id, pcpu_wa, pcpu_hirq, pcpu_sirq,
            ↪pcpu_steal, totalmem, usedmem, realfreemem, delayPremium, delayOutros, nf.eth0rv, nf.eth0tv, nf.
            ↪eth1rv, nf.eth1tv, pidPremium.kp, pidPremium.ki, pidPremium.kd, pidPremium.iState, pidPremium.
            ↪dState, pidOutros.kp, pidOutros.ki, pidOutros.kd, pidOutros.iState, pidOutros.dState,
            ↪setPointPremiumActual_Rel, setPointOutrosActual_Rel");

106     while (1) {

108         tActual = time(0);
109         tDiffAbs = difftime(tActual, tInicio);

111         if (tDiffAbs >= tDiffRel4) {
112             exit(0);
113         } else {
114             if (tDiffAbs >= tDiffRel1 && tDiffAbs < tDiffRel2) {
115                 setPointPremiumActual_Rel = setPointPremium1_Rel;
116                 setPointOutrosActual_Rel = setPointOutros1_Rel;
117                 if (sysIntv == 0) {
118                     reset();
119                     setPID();
120                     sysIntv = 1;
121                 }
122             } else if (tDiffAbs >= tDiffRel2 && tDiffAbs < tDiffRel3) {
123                 setPointPremiumActual_Rel = setPointPremium2_Rel;
124                 setPointOutrosActual_Rel = setPointOutros2_Rel;
125                 if (sysIntv == 1) {
126                     reset();
127                     setPID();
128                     sysIntv = 2;
129                 }
130             } else if (tDiffAbs >= tDiffRel3 && tDiffAbs < tDiffRel4) {
131                 setPointPremiumActual_Rel = setPointPremium3_Rel;
132                 setPointOutrosActual_Rel = setPointOutros3_Rel;
133                 if (sysIntv == 2) {
134                     reset();
135                     setPID();
136                     sysIntv = 3;
137                 }
138             }
139         }

141         unsigned int clientlen = sizeof (cliente);
142         if ((clientsock = accept(serversock, (struct sockaddr *) & cliente, &clientlen)) < 0) {
143             erro("Impossivel aceitar ligacao cliente");
144         }
145         fprintf(stdout, "Cliente ligado: %s\n", inet_ntoa(cliente.sin_addr));
146         recebeInfoApache(clientsock);
147     }

149 }

151 void recebeInfoApache(int sock) {

153     char infoApache[BUFFSIZE];

155     int received = -1;

157     // Recebe mensagem do cliente - Apache
158     if ((received = recv(sock, infoApache, BUFFSIZE - 1, 0)) < 0) {
159         erro(" Impossivel receber dados do cliente");
160     }

162     // Trata mensagem do cliente
163     trataInfoApache(infoApache);

165     close(sock);

167     memset((void*) & infoApache, 0, sizeof (char) * BUFFSIZE);

169 }

171 void trataInfoApache(char c[]) {
```

```
173     int i;
174     int ct = 0;
175     char tmp[sizeof (char) * BUFFSIZE];
176     int nParams = 14; // %pTotal(n=1), %pPremium(n=1), CPU(n=8), Memoria(n=4)
177     char *info[nParams];

179     strcpy(tmp, c);

181     ct = validaInfoApache(tmp);
182     if (ct == nParams) {
183         info[i = 0] = strtok(c, " ,\n\0");
184         //printf("pWA = %s\n", info[0]);
185         for (i = 1; i < nParams; i++) {
186             info[i] = strtok(NULL, " ,\n\0");
187             char *s = subs(i);
188             //printf("%s = %s\n", s, info[i]);
189             free(s);
190         }
191         controladorPID(info);
192     } else {
193         printf("Numero argumentos inválidos. n=%d", ct);
194     }

196 }

198 int validaInfoApache(char c[]) {

200     int ct = 0;
201     char tmp[100];
202     char *p;
203     int s;

205     strcpy(tmp, c);
206     s = sizeof (strtok(tmp, ","));
207     if (s > 0) { //Nao foi passada uma string vazia, ex. 'Enter'
208         p = strtok(c, ",");
209         while (p != NULL) {
210             ct++;
211             p = strtok(NULL, ",");
212         }
213     } else {
214         ct = 1; //Nota: 'Enter' -> \0'
215     }
216     return ct;

218 }

220 void setPID() {

222     pidPremium.kp = kpPremium;
223     pidPremium.ki = kiPremium;
224     pidPremium.kd = kdPremium;
225     pidPremium.iState = 0;
226     pidPremium.iMin = -100;
227     pidPremium.iMax = 100;
228     pidPremium.dState = 0;

230     pidOutros.kp = kpOutros;
231     pidOutros.ki = kiOutros;
232     pidOutros.kd = kdOutros;
233     pidOutros.iState = 0;
234     pidOutros.iMin = -100;
235     pidOutros.iMax = 100;
236     pidOutros.dState = 0;

238 }

240 void controladorPID(char *infoP[]) { //Controlador

242     float pcpu_t, pcpu_us, pcpu_ni, pcpu_sy, pcpu_id, pcpu_wa, pcpu_hirq, pcpu_sirq, pcpu_steal;
243     float totalmem, usedmem, realfreemem; //Notaint

245     char log[256];
246     char cQoS[32];

248     pTotal_Abs = atof(infoP[0]);
249     pPremium_Abs = atof(infoP[1]);
250     pcpu_t = atof(infoP[2]);
251     pcpu_us = atof(infoP[3]);
252     pcpu_ni = atof(infoP[4]);
253     pcpu_sy = atof(infoP[5]);
254     pcpu_id = atof(infoP[6]);
255     pcpu_wa = atof(infoP[7]);
256     pcpu_hirq = atof(infoP[8]);
```

```
257     pcpu_sirq = atof(infoP[9]);
258     pcpu_steal = atof(infoP[10]);
259     totalmem = atof(infoP[11]);
260     usedmem = atof(infoP[12]);
261     realfreemem = atof(infoP[13]);

263     if (pTotal_Abs != 0) {
264         pOutros_Abs = pTotal_Abs - pPremium_Abs;
265         pPremium_Rel = (pPremium_Abs / pTotal_Abs) * 100;
266         pOutros_Rel = (pOutros_Abs / pTotal_Abs) * 100;

268         erroPremium_Rel = pPremium_Rel - setPointPremiumActual_Rel;
269         delayPremium = actualizaControladorPID(&pidPremium, erroPremium_Rel, pPremium_Rel);
270         if (delayPremium < delayMIN) {
271             delayPremium = delayMIN;
272         } else if (delayPremium > delayMAX) {
273             delayPremium = delayMAX;
274         }

276         erroOutros_Rel = pOutros_Rel - setPointOutrosActual_Rel;
277         delayOutros = actualizaControladorPID(&pidOutros, erroOutros_Rel, pOutros_Rel);
278         if (delayOutros < delayMIN) {
279             delayOutros = delayMIN;
280         } else if (delayOutros > delayMAX) {
281             delayOutros = delayMAX;
282         }
283     } else {
284         reset();
285     }

287     trafegoRedeGW(&nf);

289     sprintf(log, "%f,%2.2f,%2.2f,%2.2f,%2.2f,%2.2f,%2.2f,%2.2f,%2.2f,%2.2f,%2.2f,%2.2f,%2.2f,%2.2f,%2.2f
            ↪,%2.2f,%2.2f,%2.2f,%2.2f,%d,%d,%8.1f,%8.1f,%8.1f,%8.1f,%2.2f,%2.2f,%2.2f,%2.2f,%2.2f,%2.2f,%2.2f
            ↪,%2.2f,%2.2f,%2.2f,%2.2f,%2.2f", tDiffAbs, pTotal_Abs, pPremium_Abs, pOutros_Abs, pPremium_Rel,
            ↪pOutros_Rel, erroPremium_Rel, erroOutros_Rel, pcpu_t, pcpu_us, pcpu_ni, pcpu_sy, pcpu_id, pcpu_wa,
            ↪pcpu_hirq, pcpu_sirq, pcpu_steal, totalmem, usedmem, realfreemem, delayPremium, delayOutros, nf.
            ↪eth0rv, nf.eth0tv, nf.eth1rv, nf.eth1tv, pidPremium.kp, pidPremium.ki, pidPremium.kd, pidPremium.
            ↪iState, pidPremium.dState, pidOutros.kp, pidOutros.ki, pidOutros.kd, pidOutros.iState, pidOutros.
            ↪dState, setPointPremiumActual_Rel, setPointOutrosActual_Rel);

291     // LOG - Ecra
292     logEcran();

294     //QoS
295     sprintf(cQoS, ". ./qos.sh %d %d", delayPremium, delayOutros);
296     system(cQoS);

298     // LOG - Ficheiro
299     logFicheiro(log);

301 }

303 int actualizaControladorPID(PID *pid, float erro, int pProc) {

305     //termoP
306     pid->termoP = pid->kp * erro;

308     //termoI
309     pid->iState += erro;
310     if (pid->iState > pid->iMax) {
311         pid->iState = pid->iMax;
312     } else if (pid->iState < pid->iMin) {
313         pid->iState = pid->iMin;
314     }
315     pid->termoI = pid->ki * pid->iState;

317     //termoD
318     pid->termoD = pid->kd * (pProc - pid->dState);
319     pid->dState = pProc;

321     pid->termoTotal = pid->termoP + pid->termoI + pid->termoD;

323     return (int) pid->termoTotal;

325 }

327 void reset() {
328     pOutros_Abs = 0;
329     pPremium_Rel = 0;
330     pOutros_Rel = 0;
331     erroPremium_Rel = 0;
332     erroOutros_Rel = 0;
333     delayPremium = 0;
334     delayOutros = 0;
```

```
335     pidPremium.termoP = 0;
336     pidPremium.termoI = 0;
337     pidPremium.termoD = 0;
338     pidPremium.termoTotal = 0;
339     pidOutros.termoP = 0;
340     pidOutros.termoI = 0;
341     pidOutros.termoD = 0;
342     pidOutros.termoTotal = 0;
343 }

345 void trafegoRedeGW(NETFLOW *n) {

347     FILE *fileNET;
348     int l;
349     int linhaDim = 1000;
350     char linha[linhaDim];
351     unsigned long eth0ri, eth0ti, eth0rf, eth0tf;
352     unsigned long eth1ri, eth1ti, eth1rf, eth1tf;

354     fileNET = fopen(FILE_IN_NET, "r");

356     for (l = 1; l <= 3; l++) {
357         fgets(linha, linhaDim, fileNET);
358     }
359     fgets(linha, linhaDim, fileNET);
360     sscanf(linha, "%*s %lu %*s %*s %*s %*s %*s %*s %*s %lu", &eth0ri, &eth0ti);
361     fgets(linha, linhaDim, fileNET);
362     sscanf(linha, "%*s %lu %*s %*s %*s %*s %*s %*s %*s %lu", &eth1ri, &eth1ti);

364     fclose(fileNET);

366     sleep(2);

368     fileNET = fopen(FILE_IN_NET, "r");

370     for (l = 1; l <= 3; l++) {
371         fgets(linha, linhaDim, fileNET);
372     }
373     fgets(linha, linhaDim, fileNET);
374     sscanf(linha, "%*s %lu %*s %*s %*s %*s %*s %*s %*s %lu", &eth0rf, &eth0tf);
375     fgets(linha, linhaDim, fileNET);
376     sscanf(linha, "%*s %lu %*s %*s %*s %*s %*s %*s %*s %lu", &eth1rf, &eth1tf);

378     fclose(fileNET);

380     n->eth0rv = (float) ((eth0rf - eth0ri) * 8) / (float) (2 * 1024);
381     n->eth0tv = (float) ((eth0tf - eth0ti) * 8) / (float) (2 * 1024);
382     n->eth1rv = (float) ((eth1rf - eth1ri) * 8) / (float) (2 * 1024);
383     n->eth1tv = (float) ((eth1tf - eth1ti) * 8) / (float) (2 * 1024);

385 }

387 void logEcran() {

389     printf("\n");
390     printf("|-Tempo: %2.2f s\n", tDiffAbs);
391     printf("|\n");
392     printf("|-pTotal_Abs:\t %2.2f\n", pTotal_Abs);
393     printf("|\n");
394     printf("|-Premium\n");
395     printf(" |---Setpoint:%2.2f\n", setPointPremiumActual_Rel);
396     printf(" |---pPremium_Rel: %2.2f\n", pPremium_Rel);
397     printf(" |---Erro:\t %2.2f\n", erroPremium_Rel);
398     printf(" |---TermoP:\t %2.2f\n", pidPremium.termoP);
399     printf(" |---TermoI:\t %2.2f\n", pidPremium.termoI);
400     printf(" |---TermoD:\t %2.2f\n", pidPremium.termoD);
401     printf(" |---TermoPID:%2.2f\n", pidPremium.termoTotal);
402     printf(" |---Delay:\t %d\n", delayPremium);
403     printf("|-Outros\n");
404     printf(" |---Setpoint:%2.2f\n", setPointOutrosActual_Rel);
405     printf(" |---pOutros_Rel: %2.2f\n", pOutros_Rel);
406     printf(" |---Erro:\t %2.2f\n", erroOutros_Rel);
407     printf(" |---TermoP:\t %2.2f\n", pidOutros.termoP);
408     printf(" |---TermoI:\t %2.2f\n", pidOutros.termoI);
409     printf(" |---TermoD:\t %2.2f\n", pidOutros.termoD);
410     printf(" |---TermoPID:%2.2f\n", pidOutros.termoTotal);
411     printf(" |---Delay:\t %d\n", delayOutros);
412     printf("\n----------------------------------\n\n");

414 }

416 void logFicheiro(char log[]) {

418     FILE *fp;
419     fp = fopen(FILE_OUT_LOG, "a");
```

```
420     fprintf(fp, log);
421     fprintf(fp, "\n");
422     fclose(fp);

424 }

426 char* subs(int i) {
427     char *s;
428     s = (char *) malloc(20 * sizeof (char));

430     switch (i) {
431         case 1:
432             strcpy(s, "pPremium");
433             break;
434         case 2:
435             strcpy(s, "pcpu_t");
436             break;
437         case 3:
438             strcpy(s, "pcpu_us");
439             break;
440         case 4:
441             strcpy(s, "pcpu_ni");
442             break;
443         case 5:
444             strcpy(s, "pcpu_sy");
445             break;
446         case 6:
447             strcpy(s, "pcpu_id");
448             break;
449         case 7:
450             strcpy(s, "pcpu_wa");
451             break;
452         case 8:
453             strcpy(s, "pcpu_hirq");
454             break;
455         case 9:
456             strcpy(s, "pcpu_sirq");
457             break;
458         case 10:
459             strcpy(s, "pcpu_steal");
460             break;
461         case 11:
462             strcpy(s, "totalmem");
463             break;
464         case 12:
465             strcpy(s, "usedmem");
466             break;
467         case 13:
468             strcpy(s, "realfreemem");
469             break;
470     }
471     return s;
472 }
```

,

A.2 Servidor HTTP Apache

Servidor Apache - `httpd.conf`

```
1  ServerTokens OS
2  ServerRoot "/etc/httpd"
3  PidFile run/httpd.pid
4  Timeout 5
5  KeepAlive On
6  MaxKeepAliveRequests 0
7  KeepAliveTimeout 3
8  <IfModule prefork.c>
9  StartServers 256
10 MinSpareServers 5
11 MaxSpareServers 10
12 ServerLimit 256
13 MaxClients 256
14 MaxRequestsPerChild 0
15 </IfModule>
16 <IfModule worker.c>
17 StartServers 4
```

```
MaxClients 300
MinSpareThreads 25
MaxSpareThreads 75
ThreadsPerChild 25
MaxRequestsPerChild 0
</IfModule>
Listen 80
LoadModule auth_basic_module modules/mod_auth_basic.so
LoadModule auth_digest_module modules/mod_auth_digest.so
LoadModule authn_file_module modules/mod_authn_file.so
LoadModule authn_alias_module modules/mod_authn_alias.so
LoadModule authn_anon_module modules/mod_authn_anon.so
LoadModule authn_dbm_module modules/mod_authn_dbm.so
LoadModule authn_default_module modules/mod_authn_default.so
LoadModule authz_host_module modules/mod_authz_host.so
LoadModule authz_user_module modules/mod_authz_user.so
LoadModule authz_owner_module modules/mod_authz_owner.so
LoadModule authz_groupfile_module modules/mod_authz_groupfile.so
LoadModule authz_dbm_module modules/mod_authz_dbm.so
LoadModule authz_default_module modules/mod_authz_default.so
LoadModule ldap_module modules/mod_ldap.so
LoadModule authnz_ldap_module modules/mod_authnz_ldap.so
LoadModule include_module modules/mod_include.so
LoadModule log_config_module modules/mod_log_config.so
LoadModule logio_module modules/mod_logio.so
LoadModule env_module modules/mod_env.so
LoadModule ext_filter_module modules/mod_ext_filter.so
LoadModule mime_magic_module modules/mod_mime_magic.so
LoadModule expires_module modules/mod_expires.so
LoadModule deflate_module modules/mod_deflate.so
LoadModule headers_module modules/mod_headers.so
LoadModule usertrack_module modules/mod_usertrack.so
LoadModule setenvif_module modules/mod_setenvif.so
LoadModule mime_module modules/mod_mime.so
LoadModule dav_module modules/mod_dav.so
LoadModule status_module modules/mod_status.so
LoadModule autoindex_module modules/mod_autoindex.so
LoadModule info_module modules/mod_info.so
LoadModule dav_fs_module modules/mod_dav_fs.so
LoadModule vhost_alias_module modules/mod_vhost_alias.so
LoadModule negotiation_module modules/mod_negotiation.so
LoadModule dir_module modules/mod_dir.so
LoadModule actions_module modules/mod_actions.so
LoadModule speling_module modules/mod_speling.so
LoadModule userdir_module modules/mod_userdir.so
LoadModule alias_module modules/mod_alias.so
LoadModule rewrite_module modules/mod_rewrite.so
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_balancer_module modules/mod_proxy_balancer.so
LoadModule proxy_ftp_module modules/mod_proxy_ftp.so
LoadModule proxy_http_module modules/mod_proxy_http.so
LoadModule proxy_connect_module modules/mod_proxy_connect.so
LoadModule cache_module modules/mod_cache.so
LoadModule suexec_module modules/mod_suexec.so
LoadModule disk_cache_module modules/mod_disk_cache.so
LoadModule file_cache_module modules/mod_file_cache.so
LoadModule mem_cache_module modules/mod_mem_cache.so
LoadModule cgi_module modules/mod_cgi.so
LoadModule version_module modules/mod_version.so
Include conf.d/*.conf
ExtendedStatus On
User apache
Group apache
ServerAdmin root@localhost
UseCanonicalName Off
DocumentRoot "/var/www/html"
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>
<Directory "/var/www/html">
    Options Indexes FollowSymLinks
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
<IfModule mod_userdir.c>
    #
    # UserDir is disabled by default since it can confirm the presence
    # of a username on the system (depending on home directory
    # permissions).
    #
    UserDir disable
    #
    # To enable requests to /~user/ to serve the user's public_html
```

```
    # directory, remove the "UserDir disable" line above, and uncomment
    # the following line instead:
    #
    #UserDir public_html
</IfModule>
DirectoryIndex index.html index.html.var
AccessFileName .htaccess
<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
</Files>
TypesConfig /etc/mime.types
DefaultType text/plain
<IfModule mod_mime_magic.c>
    MIMEMagicFile conf/magic
</IfModule>
HostnameLookups Off
LogLevel warn
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
ServerSignature On
Alias /icons/ "/var/www/icons/"
<Directory "/var/www/icons">
    Options Indexes MultiViews
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
<IfModule mod_dav_fs.c>
    # Location of the WebDAV lock database.
    DAVLockDB /var/lib/dav/lockdb
</IfModule>
ScriptAlias /cgi-bin/ "/var/www/cgi-bin/"
<Directory "/var/www/cgi-bin">
    AllowOverride None
    Options None
    Order allow,deny
    Allow from all
</Directory>
IndexOptions FancyIndexing VersionSort NameWidth=* HTMLTable
AddIconByEncoding (CMP,/icons/compressed.gif) x-compress x-gzip
AddIconByType (TXT,/icons/text.gif) text/*
AddIconByType (IMG,/icons/image2.gif) image/*
AddIconByType (SND,/icons/sound2.gif) audio/*
AddIconByType (VID,/icons/movie.gif) video/*
AddIcon /icons/binary.gif .bin .exe
AddIcon /icons/binhex.gif .hqx
AddIcon /icons/tar.gif .tar
AddIcon /icons/world2.gif .wrl .wrl.gz .vrml .vrm .iv
AddIcon /icons/compressed.gif .Z .z .tgz .gz .zip
AddIcon /icons/a.gif .ps .ai .eps
AddIcon /icons/layout.gif .html .shtml .htm .pdf
AddIcon /icons/text.gif .txt
AddIcon /icons/c.gif .c
AddIcon /icons/p.gif .pl .py
AddIcon /icons/f.gif .for
AddIcon /icons/dvi.gif .dvi
AddIcon /icons/uuencoded.gif .uu
AddIcon /icons/script.gif .conf .sh .shar .csh .ksh .tcl
AddIcon /icons/tex.gif .tex
AddIcon /icons/bomb.gif core
AddIcon /icons/back.gif ..
AddIcon /icons/hand.right.gif README
AddIcon /icons/folder.gif ^^DIRECTORY^^
AddIcon /icons/blank.gif ^^BLANKICON^^
DefaultIcon /icons/unknown.gif
ReadmeName README.html
HeaderName HEADER.html
IndexIgnore .??* *~ *# HEADER* README* RCS CVS *,v *,t
AddLanguage ca .ca
AddLanguage cs .cz .cs
AddLanguage da .dk
AddLanguage de .de
AddLanguage el .el
AddLanguage en .en
AddLanguage eo .eo
AddLanguage es .es
AddLanguage et .et
AddLanguage fr .fr
AddLanguage he .he
AddLanguage hr .hr
AddLanguage it .it
AddLanguage ja .ja
```

```
AddLanguage ko .ko
AddLanguage ltz .ltz
AddLanguage nl .nl
AddLanguage nn .nn
AddLanguage no .no
AddLanguage pl .po
AddLanguage pt .pt
AddLanguage pt-BR .pt-br
AddLanguage ru .ru
AddLanguage sv .sv
AddLanguage zh-CN .zh-cn
AddLanguage zh-TW .zh-tw
LanguagePriority en ca cs da de el eo es et fr he hr it ja ko ltz nl nn no pl pt pt-BR ru sv zh-CN zh-TW
ForceLanguagePriority Prefer Fallback
AddDefaultCharset UTF-8
AddType application/x-compress .Z
AddType application/x-gzip .gz .tgz
AddHandler type-map var
AddType text/html .shtml
AddOutputFilter INCLUDES .shtml
Alias /error/ "/var/www/error/"
<IfModule mod_negotiation.c>
<IfModule mod_include.c>
    <Directory "/var/www/error">
        AllowOverride None
        Options IncludesNoExec
        AddOutputFilter Includes html
        AddHandler type-map var
        Order allow,deny
        Allow from all
        LanguagePriority en es de fr
        ForceLanguagePriority Prefer Fallback
    </Directory>
</IfModule>
</IfModule>
BrowserMatch "Mozilla/2" nokeepalive
BrowserMatch "MSIE 4\.0b2;" nokeepalive downgrade-1.0 force-response-1.0
BrowserMatch "RealPlayer 4\.0" force-response-1.0
BrowserMatch "Java/1\.0" force-response-1.0
BrowserMatch "JDK/1\.0" force-response-1.0
BrowserMatch "Microsoft Data Access Internet Publishing Provider" redirect-carefully
BrowserMatch "MS FrontPage" redirect-carefully
BrowserMatch "^WebDrive" redirect-carefully
BrowserMatch "^WebDAVFS/1.[0123]" redirect-carefully
BrowserMatch "^gnome-vfs/1.0" redirect-carefully
BrowserMatch "^XML Spy" redirect-carefully
BrowserMatch "^Dreamweaver-WebDAV-SCM1" redirect-carefully
<Location /server-status>
    SetHandler server-status
    Order deny,allow
    Deny from all
    Allow from 127.0.0.1
    Allow from 192.168.1.0/24
    Allow from 192.168.100.0/24
</Location>
ServerName 192.168.100.100
<VirtualHost 192.168.100.101>
    ServerName 192.168.100.101
    DocumentRoot /var/www/html/site1
</VirtualHost>
<VirtualHost 192.168.100.102>
    ServerName 192.168.100.102
    DocumentRoot /var/www/html/site2
</VirtualHost>
```

Servidor Apache - `scoreboard.h`

```
/* Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements. See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
 * The ASF licenses this file to You under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License. You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
```

```
17  /**
18   * @file scoreboard.h
19   * @brief Apache scoreboard library
20   */

22  #ifndef APACHE_SCOREBOARD_H
23  #define APACHE_SCOREBOARD_H

25  #ifdef __cplusplus
26  extern "C" {
27  #endif

29  #ifdef HAVE_SYS_TIMES_H
30  #include <sys/time.h>
31  #include <sys/times.h>
32  #elif defined(TPF)
33  #include <time.h>
34  #endif

36  #include "ap_config.h"
37  #include "apr_hooks.h"
38  #include "apr_thread_proc.h"
39  #include "apr_portable.h"
40  #include "apr_shm.h"
41  #include "apr_optional.h"

43  /* Scoreboard file, if there is one */
44  #ifndef DEFAULT_SCOREBOARD
45  #define DEFAULT_SCOREBOARD "logs/apache_runtime_status"
46  #endif

48  /* Mestrado */
49  #ifndef MST_UE
50  #define MST_UE
51  #endif

53  /* Scoreboard info on a process is, for now, kept very brief ---
54   * just status value and pid (the latter so that the caretaker process
55   * can properly update the scoreboard when a process dies). We may want
56   * to eventually add a separate set of long_score structures which would
57   * give, for each process, the number of requests serviced, and info on
58   * the current, or most recent, request.
59   *
60   * Status values:
61   */

63  #define SERVER_DEAD 0
64  #define SERVER_STARTING 1 /* Server Starting up */
65  #define SERVER_READY 2  /* Waiting for connection (or accept() lock) */
66  #define SERVER_BUSY_READ 3 /* Reading a client request */
67  #define SERVER_BUSY_WRITE 4 /* Processing a client request */
68  #define SERVER_BUSY_KEEPALIVE 5 /* Waiting for more requests via keepalive */
69  #define SERVER_BUSY_LOG 6 /* Logging the request */
70  #define SERVER_BUSY_DNS 7 /* Looking up a hostname */
71  #define SERVER_CLOSING 8 /* Closing the connection */
72  #define SERVER_GRACEFUL 9 /* server is gracefully finishing request */
73  #define SERVER_IDLE_KILL 10 /* Server is cleaning up idle children. */
74  #define SERVER_NUM_STATUS 11 /* number of status settings */

76  /* Type used for generation indicies. Startup and every restart cause a
77   * new generation of children to be spawned. Children within the same
78   * generation share the same configuration information -- pointers to stuff
79   * created at config time in the parent are valid across children. However,
80   * this can't work effectively with non-forked architectures. So while the
81   * arrays in the scoreboard never change between the parent and forked
82   * children, so they do not require shm storage, the contents of the shm
83   * may contain no pointers.
84   */
85  typedef int ap_generation_t;

87  /* Is the scoreboard shared between processes or not?
88   * Set by the MPM when the scoreboard is created.
89   */
90  typedef enum {
91      SB_NOT_SHARED = 1,
92      SB_SHARED = 2
93  } ap_scoreboard_e;

95  #define SB_WORKING 0 /* The server is busy and the child is useful. */
96  #define SB_IDLE_DIE 1 /* The server is idle and the child is superfluous. */
97                        /* The child should check for this and exit gracefully. */

99  /* stuff which is worker specific */
100 /************************WARNING*************************************/
101 /* These are things that are used by mod_status. Do not put anything */
```

```
102 /* in here that you cannot live without. This structure will not */
103 /* be available if mod_status is not loaded. */
104 /************************************************************/
105 typedef struct worker_score worker_score;

107 struct worker_score {
108     int thread_num;
109 #if APR_HAS_THREADS
110     apr_os_thread_t tid;
111 #endif
112     /* With some MPMs (e.g., worker), a worker_score can represent
113      * a thread in a terminating process which is no longer
114      * represented by the corresponding process_score. These MPMs
115      * should set pid and generation fields in the worker_score.
116      */
117     pid_t pid;
118     ap_generation_t generation;
119     unsigned char status;
120     unsigned long access_count;
121     apr_off_t bytes_served;
122     unsigned long my_access_count;
123     apr_off_t my_bytes_served;
124     apr_off_t conn_bytes;
125     unsigned short conn_count;
126     apr_time_t start_time;
127     apr_time_t stop_time;
128 #ifdef HAVE_TIMES
129     struct tms times;
130 #endif
131     apr_time_t last_used;
132     char client[32];  /* Keep 'em small... */
133     char request[64];  /* We just want an idea... */
134     char vhost[32]; /* What virtual host is being accessed? */
135 };

137 typedef struct {
138     int server_limit;
139     int thread_limit;
140     ap_scoreboard_e sb_type;
141     ap_generation_t running_generation; /* the generation of children which
142                                          * should still be serving requests.
143                                          */
144     apr_time_t restart_time;
145     int lb_limit;
146 } global_score;

148 /* stuff which the parent generally writes and the children rarely read */
149 typedef struct process_score process_score;
150 struct process_score{
151     pid_t pid;
152     ap_generation_t generation; /* generation of this child */
153     ap_scoreboard_e sb_type;
154     int quiescing; /* the process whose pid is stored above is
155                             * going down gracefully
156                             */
157 };

159 /* stuff which is lb specific */
160 typedef struct lb_score lb_score;
161 struct lb_score{
162     /* TODO: make a real stuct from this */
163     unsigned char data[1024];
164 };

166 /* Scoreboard is now in 'local' memory, since it isn't updated once created,
167  * even in forked architectures. Child created-processes (non-fork) will
168  * set up these indicies into the (possibly relocated) shmem records.
169  */
170 typedef struct {
171     global_score *global;
172     process_score *parent;
173     worker_score **servers;
174     lb_score *balancers;
175 } scoreboard;

177 typedef struct ap_sb_handle_t ap_sb_handle_t;

179 AP_DECLARE(int) ap_exists_scoreboard_image(void);
180 AP_DECLARE(void) ap_increment_counts(ap_sb_handle_t *sbh, request_rec *r);

182 int ap_create_scoreboard(apr_pool_t *p, ap_scoreboard_e t);
183 apr_status_t ap_reopen_scoreboard(apr_pool_t *p, apr_shm_t **shm, int detached);
184 void ap_init_scoreboard(void *shared_score);
185 AP_DECLARE(int) ap_calc_scoreboard_size(void);
186 apr_status_t ap_cleanup_scoreboard(void *d);
```

```
188 AP_DECLARE(void) ap_create_sb_handle(ap_sb_handle_t **new_sbh, apr_pool_t *p,
189                                      int child_num, int thread_num);

191 int find_child_by_pid(apr_proc_t *pid);
192 AP_DECLARE(int) ap_update_child_status(ap_sb_handle_t *sbh, int status, request_rec *r);
193 AP_DECLARE(int) ap_update_child_status_from_indexes(int child_num, int thread_num,
194                                                     int status, request_rec *r);
195 void ap_time_process_request(ap_sb_handle_t *sbh, int status);

197 AP_DECLARE(worker_score *) ap_get_scoreboard_worker(int x, int y);
198 AP_DECLARE(process_score *) ap_get_scoreboard_process(int x);
199 AP_DECLARE(global_score *) ap_get_scoreboard_global(void);
200 AP_DECLARE(lb_score *) ap_get_scoreboard_lb(int lb_num);

202 AP_DECLARE_DATA extern scoreboard *ap_scoreboard_image;
203 AP_DECLARE_DATA extern const char *ap_scoreboard_fname;
204 AP_DECLARE_DATA extern int ap_extended_status;

206 AP_DECLARE_DATA extern ap_generation_t volatile ap_my_generation;

208 /* Hooks */
209 /**
210   * Hook for post scoreboard creation, pre mpm.
211   * @param p Apache pool to allocate from.
212   * @param sb_type
213   * @ingroup hooks
214   * @return OK or DECLINE on success; anything else is a error
215   */
216 AP_DECLARE_HOOK(int, pre_mpm, (apr_pool_t *p, ap_scoreboard_e sb_type))

218 /**
219   * proxy load balancer
220   * @return the number of load balancer workers.
221   */
222 APR_DECLARE_OPTIONAL_FN(int, ap_proxy_lb_workers,
223                         (void));

225 /* for time_process_request() in http_main.c */
226 #define START_PREQUEST 1
227 #define STOP_PREQUEST 2

229 #ifdef __cplusplus
230 }
231 #endif

233 #endif /* !APACHE_SCOREBOARD_H */
```

Servidor HTTP - `scoreboard.c`

```
1  /* Licensed to the Apache Software Foundation (ASF) under one or more
2   * contributor license agreements. See the NOTICE file distributed with
3   * this work for additional information regarding copyright ownership.
4   * The ASF licenses this file to You under the Apache License, Version 2.0
5   * (the "License"); you may not use this file except in compliance with
6   * the License. You may obtain a copy of the License at
7   *
8   * http://www.apache.org/licenses/LICENSE-2.0
9   *
10  * Unless required by applicable law or agreed to in writing, software
11  * distributed under the License is distributed on an "AS IS" BASIS,
12  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13  * See the License for the specific language governing permissions and
14  * limitations under the License.
15  */

17 #include "apr.h"
18 #include "apr_strings.h"
19 #include "apr_portable.h"
20 #include "apr_lib.h"

22 #define APR_WANT_STRFUNC
23 #include "apr_want.h"

25 #if APR_HAVE_SYS_TYPES_H
26 #include <sys/types.h>
27 #endif

29 #include "ap_config.h"
30 #include "httpd.h"
31 #include "http_log.h"
32 #include "http_main.h"
33 #include "http_core.h"
```

```
34  #include "http_config.h"
35  #include "ap_mpm.h"

37  #include "mpm.h"
38  #include "scoreboard.h"

40  //Mestrado

42  #ifdef MST_UE
43  #include "mst_ue.c"
44  #endif

46  AP_DECLARE_DATA scoreboard *ap_scoreboard_image = NULL;
47  AP_DECLARE_DATA const char *ap_scoreboard_fname = NULL;
48  AP_DECLARE_DATA int ap_extended_status = 0;

50  #if APR_HAS_SHARED_MEMORY

52  #include "apr_shm.h"

54  #ifndef WIN32
55  static /* but must be exported to mpm_winnt */
56  #endif
57          apr_shm_t *ap_scoreboard_shm = NULL;

59  #endif

61  APR_HOOK_STRUCT(
62      APR_HOOK_LINK(pre_mpm)
63  )

65  AP_IMPLEMENT_HOOK_RUN_ALL(int,pre_mpm,
66                            (apr_pool_t *p, ap_scoreboard_e sb_type),
67                            (p, sb_type),OK,DECLINED)

69  static APR_OPTIONAL_FN_TYPE(ap_proxy_lb_workers)
70                                *proxy_lb_workers;

72  struct ap_sb_handle_t {
73      int child_num;
74      int thread_num;
75  };

77  static int server_limit, thread_limit, lb_limit;
78  static apr_size_t scoreboard_size;

80  /*
81   * ToDo:
82   * This function should be renamed to cleanup_shared
83   * and it should handle cleaning up a scoreboard shared
84   * between processes using any form of IPC (file, shared memory
85   * segment, etc.). Leave it as is now because it is being used
86   * by various MPMs.
87   */
88  static apr_status_t ap_cleanup_shared_mem(void *d)
89  {
90  #if APR_HAS_SHARED_MEMORY
91      free(ap_scoreboard_image);
92      ap_scoreboard_image = NULL;
93      apr_shm_destroy(ap_scoreboard_shm);
94  #endif
95      return APR_SUCCESS;
96  }

98  AP_DECLARE(int) ap_calc_scoreboard_size(void)
99  {
100     ap_mpm_query(AP_MPMQ_HARD_LIMIT_THREADS, &thread_limit);
101     ap_mpm_query(AP_MPMQ_HARD_LIMIT_DAEMONS, &server_limit);

103     if (!proxy_lb_workers)
104         proxy_lb_workers = APR_RETRIEVE_OPTIONAL_FN(ap_proxy_lb_workers);
105     if (proxy_lb_workers)
106         lb_limit = proxy_lb_workers();
107     else
108         lb_limit = 0;

110     scoreboard_size = sizeof(global_score);
111     scoreboard_size += sizeof(process_score) * server_limit;
112     scoreboard_size += sizeof(worker_score) * server_limit * thread_limit;
113     if (lb_limit)
114         scoreboard_size += sizeof(lb_score) * lb_limit;

116     return scoreboard_size;
117 }
```

```
119 void ap_init_scoreboard(void *shared_score)
120 {
121     char *more_storage;
122     int i;
123
124     ap_calc_scoreboard_size();
125     ap_scoreboard_image =
126         calloc(1, sizeof(scoreboard) + server_limit * sizeof(worker_score *) +
127                server_limit * lb_limit * sizeof(lb_score *));
128     more_storage = shared_score;
129     ap_scoreboard_image->global = (global_score *)more_storage;
130     more_storage += sizeof(global_score);
131     ap_scoreboard_image->parent = (process_score *)more_storage;
132     more_storage += sizeof(process_score) * server_limit;
133     ap_scoreboard_image->servers =
134         (worker_score **)((char*)ap_scoreboard_image + sizeof(scoreboard));
135     for (i = 0; i < server_limit; i++) {
136         ap_scoreboard_image->servers[i] = (worker_score *)more_storage;
137         more_storage += thread_limit * sizeof(worker_score);
138     }
139     if (lb_limit) {
140         ap_scoreboard_image->balancers = (lb_score *)more_storage;
141         more_storage += lb_limit * sizeof(lb_score);
142     }
143     ap_assert(more_storage == (char*)shared_score + scoreboard_size);
144     ap_scoreboard_image->global->server_limit = server_limit;
145     ap_scoreboard_image->global->thread_limit = thread_limit;
146     ap_scoreboard_image->global->lb_limit = lb_limit;
147 }
148
149 /**
150  * Create a name-based scoreboard in the given pool using the
151  * given filename.
152  */
153 static apr_status_t create_namebased_scoreboard(apr_pool_t *pool,
154                                                 const char *fname)
155 {
156 #if APR_HAS_SHARED_MEMORY
157     apr_status_t rv;
158
159     /* The shared memory file must not exist before we create the
160      * segment. */
161     apr_shm_remove(fname, pool); /* ignore errors */
162
163     rv = apr_shm_create(&ap_scoreboard_shm, scoreboard_size, fname, pool);
164     if (rv != APR_SUCCESS) {
165         ap_log_error(APLOG_MARK, APLOG_CRIT, rv, NULL,
166                      "unable to create scoreboard \"%s\" "
167                      "(name-based shared memory failure)", fname);
168         return rv;
169     }
170 #endif /* APR_HAS_SHARED_MEMORY */
171     return APR_SUCCESS;
172 }
173
174 /* ToDo: This function should be made to handle setting up
175  * a scoreboard shared between processes using any IPC technique,
176  * not just a shared memory segment
177  */
178 static apr_status_t open_scoreboard(apr_pool_t *pconf)
179 {
180 #if APR_HAS_SHARED_MEMORY
181     apr_status_t rv;
182     char *fname = NULL;
183     apr_pool_t *global_pool;
184
185     /* We don't want to have to recreate the scoreboard after
186      * restarts, so we'll create a global pool and never clean it.
187      */
188     rv = apr_pool_create(&global_pool, NULL);
189     if (rv != APR_SUCCESS) {
190         ap_log_error(APLOG_MARK, APLOG_CRIT, rv, NULL,
191                      "Fatal error: unable to create global pool "
192                      "for use with by the scoreboard");
193         return rv;
194     }
195
196     /* The config says to create a name-based shmem */
197     if (ap_scoreboard_fname) {
198         /* make sure it's an absolute pathname */
199         fname = ap_server_root_relative(pconf, ap_scoreboard_fname);
200         if (!fname) {
201             ap_log_error(APLOG_MARK, APLOG_CRIT, APR_EBADPATH, NULL,
202                          "Fatal error: Invalid Scoreboard path %s",
203                          ap_scoreboard_fname);
```

```
204             return APR_EBADPATH;
205         }
206         return create_namebased_scoreboard(global_pool, fname);
207     }
208     else { /* config didn't specify, we get to choose shmem type */
209         rv = apr_shm_create(&ap_scoreboard_shm, scoreboard_size, NULL,
210                             global_pool); /* anonymous shared memory */
211         if ((rv != APR_SUCCESS) && (rv != APR_ENOTIMPL)) {
212             ap_log_error(APLOG_MARK, APLOG_CRIT, rv, NULL,
213                          "Unable to create scoreboard "
214                          "(anonymous shared memory failure)");
215             return rv;
216         }
217         /* Make up a filename and do name-based shmem */
218         else if (rv == APR_ENOTIMPL) {
219             /* Make sure it's an absolute pathname */
220             ap_scoreboard_fname = DEFAULT_SCOREBOARD;
221             fname = ap_server_root_relative(pconf, ap_scoreboard_fname);

223             return create_namebased_scoreboard(global_pool, fname);
224         }
225     }
226 #endif /* APR_HAS_SHARED_MEMORY */
227     return APR_SUCCESS;
228 }

230 /* If detach is non-zero, this is a seperate child process,
231  * if zero, it is a forked child.
232  */
233 apr_status_t ap_reopen_scoreboard(apr_pool_t *p, apr_shm_t **shm, int detached)
234 {
235 #if APR_HAS_SHARED_MEMORY
236     if (!detached) {
237         return APR_SUCCESS;
238     }
239     if (apr_shm_size_get(ap_scoreboard_shm) < scoreboard_size) {
240         ap_log_error(APLOG_MARK, APLOG_CRIT, 0, NULL,
241                      "Fatal error: shared scoreboard too small for child!");
242         apr_shm_detach(ap_scoreboard_shm);
243         ap_scoreboard_shm = NULL;
244         return APR_EINVAL;
245     }
246     /* everything will be cleared shortly */
247     if (*shm) {
248         *shm = ap_scoreboard_shm;
249     }
250 #endif
251     return APR_SUCCESS;
252 }

254 apr_status_t ap_cleanup_scoreboard(void *d)
255 {
256     if (ap_scoreboard_image == NULL) {
257         return APR_SUCCESS;
258     }
259     if (ap_scoreboard_image->global->sb_type == SB_SHARED) {
260         ap_cleanup_shared_mem(NULL);
261     }
262     else {
263         free(ap_scoreboard_image->global);
264         free(ap_scoreboard_image);
265         ap_scoreboard_image = NULL;
266     }
267     return APR_SUCCESS;
268 }

270 /* Create or reinit an existing scoreboard. The MPM can control whether
271  * the scoreboard is shared across multiple processes or not
272  */
273 int ap_create_scoreboard(apr_pool_t *p, ap_scoreboard_e sb_type)
274 {
275     int running_gen = 0;
276     int i;
277 #if APR_HAS_SHARED_MEMORY
278     apr_status_t rv;
279 #endif

281     if (ap_scoreboard_image) {
282         running_gen = ap_scoreboard_image->global->running_generation;
283         ap_scoreboard_image->global->restart_time = apr_time_now();
284         memset(ap_scoreboard_image->parent, 0,
285                sizeof(process_score) * server_limit);
286         for (i = 0; i < server_limit; i++) {
287             memset(ap_scoreboard_image->servers[i], 0,
288                    sizeof(worker_score) * thread_limit);
```

```
289         }
290         /* Clean up the lb workers data */
291         if (lb_limit) {
292             memset(ap_scoreboard_image->balancers, 0,
293                    sizeof(lb_score) * lb_limit);
294         }
295         return OK;
296     }

298     ap_calc_scoreboard_size();
299 #if APR_HAS_SHARED_MEMORY
300     if (sb_type == SB_SHARED) {
301         void *sb_shared;
302         rv = open_scoreboard(p);
303         if (rv || !(sb_shared = apr_shm_baseaddr_get(ap_scoreboard_shm))) {
304             return HTTP_INTERNAL_SERVER_ERROR;
305         }
306         memset(sb_shared, 0, scoreboard_size);
307         ap_init_scoreboard(sb_shared);
308     }
309     else
310 #endif
311     {
312         /* A simple malloc will suffice */
313         void *sb_mem = calloc(1, scoreboard_size);
314         if (sb_mem == NULL) {
315             ap_log_error(APLOG_MARK, APLOG_CRIT, 0, NULL,
316                          "(%d)%s: cannot allocate scoreboard",
317                          errno, strerror(errno));
318             return HTTP_INTERNAL_SERVER_ERROR;
319         }
320         ap_init_scoreboard(sb_mem);
321     }

323 #ifdef MST_UE
324          mst_ue(server_limit);
325 #endif

327     ap_scoreboard_image->global->sb_type = sb_type;
328     ap_scoreboard_image->global->running_generation = running_gen;
329     ap_scoreboard_image->global->restart_time = apr_time_now();

331     apr_pool_cleanup_register(p, NULL, ap_cleanup_scoreboard, apr_pool_cleanup_null);

333     return OK;
334 }

336 /* Routines called to deal with the scoreboard image
337  * --- note that we do *not* need write locks, since update_child_status
338  * only updates a *single* record in place, and only one process writes to
339  * a given scoreboard slot at a time (either the child process owning that
340  * slot, or the parent, noting that the child has died).
341  *
342  * As a final note --- setting the score entry to getpid() is always safe,
343  * since when the parent is writing an entry, it's only noting SERVER_DEAD
344  * anyway.
345  */

347 AP_DECLARE(int) ap_exists_scoreboard_image(void)
348 {
349     return (ap_scoreboard_image ? 1 : 0);
350 }

352 AP_DECLARE(void) ap_increment_counts(ap_sb_handle_t *sb, request_rec *r)
353 {
354     worker_score *ws;

356     if (!sb)
357         return;

359     ws = &ap_scoreboard_image->servers[sb->child_num][sb->thread_num];

361 #ifdef HAVE_TIMES
362     times(&ws->times);
363 #endif
364     ws->access_count++;
365     ws->my_access_count++;
366     ws->conn_count++;
367     ws->bytes_served += r->bytes_sent;
368     ws->my_bytes_served += r->bytes_sent;
369     ws->conn_bytes += r->bytes_sent;
370 }

372 int find_child_by_pid(apr_proc_t *pid)
373 {
```

```
374     int i;
375     int max_daemons_limit;

377     ap_mpm_query(AP_MPMQ_MAX_DAEMONS, &max_daemons_limit);

379     for (i = 0; i < max_daemons_limit; ++i) {
380         if (ap_scoreboard_image->parent[i].pid == pid->pid) {
381             return i;
382         }
383     }

385     return -1;
386 }

388 AP_DECLARE(void) ap_create_sb_handle(ap_sb_handle_t **new_sbh, apr_pool_t *p,
389                                      int child_num, int thread_num)
390 {
391     *new_sbh = (ap_sb_handle_t *)apr_palloc(p, sizeof(ap_sb_handle_t));
392     (*new_sbh)->child_num = child_num;
393     (*new_sbh)->thread_num = thread_num;
394 }

396 AP_DECLARE(int) ap_update_child_status_from_indexes(int child_num,
397                                                     int thread_num,
398                                                     int status,
399                                                     request_rec *r)
400 {
401     int old_status;
402     worker_score *ws;
403     process_score *ps;

405     if (child_num < 0) {
406         return -1;
407     }

409     ws = &ap_scoreboard_image->servers[child_num][thread_num];
410     old_status = ws->status;
411     ws->status = status;

413     ps = &ap_scoreboard_image->parent[child_num];

415     if (status == SERVER_READY
416         && old_status == SERVER_STARTING) {
417         ws->thread_num = child_num * thread_limit + thread_num;
418         ps->generation = ap_my_generation;
419     }

421     if (ap_extended_status) {
422         ws->last_used = apr_time_now();
423         if (status == SERVER_READY || status == SERVER_DEAD) {
424             /*
425              * Reset individual counters
426              */
427             if (status == SERVER_DEAD) {
428                 ws->my_access_count = 0L;
429                 ws->my_bytes_served = 0L;
430             }
431             ws->conn_count = 0;
432             ws->conn_bytes = 0;
433         }
434         if (r) {
435             conn_rec *c = r->connection;
436             apr_cpystrn(ws->client, ap_get_remote_host(c, r->per_dir_config,
437                         REMOTE_NOLOOKUP, NULL), sizeof(ws->client));
438             if (r->the_request == NULL) {
439                 apr_cpystrn(ws->request, "NULL", sizeof(ws->request));
440             } else if (r->parsed_uri.password == NULL) {
441                 apr_cpystrn(ws->request, r->the_request, sizeof(ws->request));
442             } else {
443                 /* Don't reveal the password in the server-status view */
444                 apr_cpystrn(ws->request, apr_pstrcat(r->pool, r->method, " ",
445                             apr_uri_unparse(r->pool, &r->parsed_uri,
446                             APR_URI_UNP_OMITPASSWORD),
447                             r->assbackwards ? NULL : " ", r->protocol, NULL),
448                             sizeof(ws->request));
449             }
450             apr_cpystrn(ws->vhost, r->server->server_hostname,
451                         sizeof(ws->vhost));
452         }
453     }

455     return old_status;
456 }

458 AP_DECLARE(int) ap_update_child_status(ap_sb_handle_t *sbh, int status,
```

```
459                                     request_rec *r)
460 {
461     if (!sbh)
462         return -1;
463 
464     return ap_update_child_status_from_indexes(sbh->child_num, sbh->thread_num,
465                                                status, r);
466 }
467 
468 void ap_time_process_request(ap_sb_handle_t *sbh, int status)
469 {
470     worker_score *ws;
471 
472     if (!sbh)
473         return;
474 
475     if (sbh->child_num < 0) {
476         return;
477     }
478 
479     ws = &ap_scoreboard_image->servers[sbh->child_num][sbh->thread_num];
480 
481     if (status == START_PREQUEST) {
482         ws->start_time = apr_time_now();
483     }
484     else if (status == STOP_PREQUEST) {
485         ws->stop_time = apr_time_now();
486     }
487 }
488 
489 AP_DECLARE(worker_score *) ap_get_scoreboard_worker(int x, int y)
490 {
491     if (((x < 0) || (server_limit < x)) ||
492         ((y < 0) || (thread_limit < y))) {
493         return(NULL); /* Out of range */
494     }
495     return &ap_scoreboard_image->servers[x][y];
496 }
497 
498 AP_DECLARE(process_score *) ap_get_scoreboard_process(int x)
499 {
500     if ((x < 0) || (server_limit < x)) {
501         return(NULL); /* Out of range */
502     }
503     return &ap_scoreboard_image->parent[x];
504 }
505 
506 AP_DECLARE(global_score *) ap_get_scoreboard_global()
507 {
508     return ap_scoreboard_image->global;
509 }
510 
511 AP_DECLARE(lb_score *) ap_get_scoreboard_lb(int lb_num)
512 {
513     if (((lb_num < 0) || (lb_limit < lb_num))) {
514         return(NULL); /* Out of range */
515     }
516     return &ap_scoreboard_image->balancers[lb_num];
517 }
```

Servidor Apache - `mst_ue.h`

```
1  #include "apr.h"
2  #include "apr_lib.h"
3  #include "apr_portable.h"
4  #include "apr_strings.h"
5  #include "apr_want.h"
6  #include "http_config.h"
7  #include "http_core.h"
8  #include "httpd.h"
9  #include "http_log.h"
10 #include "http_main.h"
11 #include "http_protocol.h"
12 #include "scoreboard.h"
13 
14 #include <errno.h>
15 #include <fcntl.h>
16 #include <pthread.h>
17 #include <stdio.h>
18 #include <stdlib.h>
19 #include <string.h>
20 #include <syslog.h>
21 #include <arpa/inet.h>
```

```
22 #include <netinet/in.h>
23 #include <sys/socket.h>
24 #include <sys/stat.h>
25 #include <sys/time.h>
26 #include <sys/types.h>
27 #include <time.h>
28 #include <unistd.h>

30 #define FILE_IN_DEF_VHOSTS "/mst/def_vhosts.conf"
31 #define FILE_IN_MEMINFO "/proc/meminfo"
32 #define FILE_IN_PROC_STAT "/proc/stat"

34 #ifndef HARD_SERVER_LIMIT
35 #define HARD_SERVER_LIMIT 256
36 #endif

38 #define VHOSTVECTOR_SIZE 255

40 #define WAIT_DAEMON 3 // 3s de espera entre ciclos do daemon
41 #define WAIT_READ_PROC_STAT 2 // 2s de espera entre leituras do ficheiro /proc/stat

43 //Defs para enviar para a gateway
44 #define BUFFSIZE2 512
45 #define GTW_IP "192.168.100.1"
46 #define GTW_PORTA "5000"

48 /*
49 ---------------- Defines para debug - extra ------------------------
50 #define FILE_OUT 0
51 #define FILE_OUT_SCOREBOARDDETALHES "/mst/scoreboardDetalhes.log"
52 #define FILE_OUT_SCOREBOARDSTATS "/mst/scoreboardStats.log"
53 #define FILE_OUT_WORKERSTIPO "/mst/workersTipo.log"
54 #define FILE_OUT_DEBUG "/mst/debug.log"
55 #define FILE_IN_PROC_LOADAVG "/proc/loadavg"
56 #define FILE_OUT_CPU_MEM "/mst/cpu_mem.log"
57 #define FILE_IN_DEF_GERAL "/mst/def_geral.conf"
58 ------------------------------------------------------------------------------
59 */

61 struct vhostCont {
62    char vhost[32];
63    char tipo[2];
64    int cont;
65 };

68 /*
69  Prototipos das funcoes
70  */

72 void erro(char *mess);
73 int mst_ue(int sl);
74 void mst_inits();
75 void infoApache(struct vhostCont v[]);
76 void mst_reset_scoreboard_info(struct vhostCont v[]);
77 int enviaInfo(char info[]);
```

Servidor Apache - `mst_ue.c`

```
1  #include "mst_ue.h"

3  int nVHosts = 0; //Numero virtual hosts
4  int nProcessosActivos;
5  int nProcessosPremium;
6  int nProcessosTotal;
7  float pProcessosActivos, pProcessosPremium;
8  struct vhostCont vhostV[VHOSTVECTOR_SIZE];

11 void erro(char *mess) {
12     perror(mess);
13     exit(1);
14 }

16 int mst_ue(int sl) {

18     int pid;

20     nProcessosTotal = sl;

22     pid = fork();
23     if (!pid) {
24         mst_inits();

26         while (1) {
```

```
27              mst_reset_scoreboard_info(vhostV);
28              infoApache(vhostV);
29              sleep(WAIT_DAEMON);
30          }
31          exit(0);
32      }
33      return 0;
34  }

36  void mst_inits() {

38      FILE *fp2;
39      fp2 = fopen(FILE_IN_DEF_VHOSTS, "r");
40      if (fp2 != NULL) {
41          char linha[100];
42          while (fgets(linha, sizeof linha, fp2) != NULL) {
43              if (linha[0] == '#' || linha[0] == '\n') {
44                  continue;
45              }
46              if (sscanf(linha, "%32s = %2s", &vhostV[nVHosts].vhost[0],
47                      &vhostV[nVHosts].tipo[0]) == 2) {
48                  vhostV[nVHosts].cont = 0;
49                  nVHosts++;
50              }
51          }
52          fclose(fp2);
53      }

55  }

57  void infoApache(struct vhostCont v[]) {

60      char info[512];

62      int i, j, k;

64      int totalmem, freemem, realfreemem, buffers, cached, usedmem;

66      unsigned cpu_us_i, cpu_ni_i, cpu_sy_i, cpu_wa_i, cpu_hirq_i, cpu_sirq_i,
67              cpu_steal_i, cpu_us_f, cpu_ni_f, cpu_sy_f, cpu_wa_f, cpu_hirq_f,
68              cpu_sirq_f, cpu_steal_f, usage;
69      unsigned long cpu_id_i, cpu_id_f, idle, total;
70      float pcpu_t, pcpu_us, pcpu_ni, pcpu_sy, pcpu_id, pcpu_wa, pcpu_hirq,
71              pcpu_sirq, pcpu_steal;
72      char name_1[4];
73      char name_2[4];

77      for (i = 0; i < HARD_SERVER_LIMIT; ++i) {

79          worker_score *ws;
80          int estado;
81          char *cliente, *pedido, *vhst;

83          ws = &ap_scoreboard_image->servers[i][0];

85          estado = ws->status;
86          cliente = ws->client;
87          pedido = ws->request;
88          vhst = ws->vhost;

90          if (estado >= 3 && estado <= 7) {

92              nProcessosActivos++;

94              for (j = 0; j < nVHosts; j++) {
95                  if (strcmp(v[j].vhost, vhst) == 0)
96                      v[j].cont++;
97              }
98          }
99      }

101     for (k = 0; i < nVHosts; i++) {
102         if (strcmp(v[k].tipo, "P") == 0) {
103             nProcessosPremium += v[k].cont;
104         }
105     }

107      //CPU

109     FILE *fp1;
110     fp1 = fopen(FILE_IN_PROC_STAT, "r");
111     fscanf(fp1, "%s %u %u %u %lu %u %u %u %u", name_1, &cpu_us_i, &cpu_ni_i, &cpu_sy_i, &cpu_id_i, &cpu_wa_i,
            ↪&cpu_hirq_i, &cpu_sirq_i, &cpu_steal_i);
112     fclose(fp1);
```

```
114     sleep(WAIT_READ_PROC_STAT);

116     FILE *fp2;
117     fp2 = fopen(FILE_IN_PROC_STAT, "r");
118     fscanf(fp2, "%s %u %u %u %lu %u %u %u %u", name_2, &cpu_us_f, &cpu_ni_f, &cpu_sy_f, &cpu_id_f, &cpu_wa_f,
            ↪&cpu_hirq_f, &cpu_sirq_f, &cpu_steal_f);
119     fclose(fp2);

121     //Memoria

123     FILE *fp3;
124     fp3 = fopen(FILE_IN_MEMINFO, "r");
125     fscanf(fp3, "%*s %d %*s %*s %d %*s %*s %d %*s %*s %d", &totalmem, &freemem, &buffers, &cached);
126     fclose(fp3);

128     realfreemem = freemem + buffers + cached;
129     usedmem = totalmem - realfreemem;

131     //Calculos

133     usage = cpu_us_f - cpu_us_i + cpu_ni_f - cpu_ni_i + cpu_sy_f - cpu_sy_i
134            + cpu_hirq_f - cpu_hirq_i + cpu_sirq_f - cpu_sirq_i + cpu_steal_f
135            - cpu_steal_i;
136     idle = cpu_id_f - cpu_id_i + cpu_wa_f - cpu_wa_i;
137     total = usage + idle;

139     pcpu_t = (float) usage * 100 / (float) total;
140     pcpu_us = (float) (cpu_us_f - cpu_us_i) * 100 / (float) total;
141     pcpu_ni = (float) (cpu_ni_f - cpu_ni_i) * 100 / (float) total;
142     pcpu_sy = (float) (cpu_sy_f - cpu_sy_i) * 100 / (float) total;
143     pcpu_id = (float) (cpu_id_f - cpu_id_i) * 100 / (float) total;
144     pcpu_wa = (float) (cpu_wa_f - cpu_wa_i) * 100 / (float) total;
145     pcpu_hirq = (float) (cpu_hirq_f - cpu_hirq_i) * 100 / (float) total;
146     pcpu_sirq = (float) (cpu_sirq_f - cpu_sirq_i) * 100 / (float) total;
147     pcpu_steal = (float) (cpu_steal_f - cpu_steal_i) * 100 / (float) total;

149     //Obtem percentagem dos processos activos e processos premium

151     pProcessosActivos = (nProcessosActivos / nProcessosTotal) * 100;
152     pProcessosPremium = (nProcessosPremium / nProcessosTotal) * 100;

154     sprintf(info, "%2.2f,%2.2f,%2.2f,%2.2f,%2.2f,%2.2f,%2.2f,%2.2f,%2.2f,%2.2f,%2.2f,%d,%d,%d",
            ↪pProcessosActivos, pProcessosPremium, pcpu_t, pcpu_us, pcpu_ni, pcpu_sy, pcpu_id, pcpu_wa,
            ↪pcpu_hirq, pcpu_sirq, pcpu_steal, totalmem, usedmem, realfreemem);

156     //Envia para a Gateway a info
157     enviaInfo(info);

159 }

161 void mst_reset_scoreboard_info(struct vhostCont v[]) {

163     int i;

165     for (i = 0; i < VHOSTVECTOR_SIZE; i++) {

167         v[i].cont = 0;
168     }

170     nProcessosActivos = 0;
171 }

173 int enviaInfo(char info[]) {

175     int sock;
176     struct sockaddr_in echoserver;
177     unsigned int echolen;

180     if ((sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
181         erro("Failed to create socket");
182     }

184     memset(&echoserver, 0, sizeof (echoserver));
185     echoserver.sin_family = AF_INET;
186     echoserver.sin_addr.s_addr = inet_addr(GTW_IP);
187     echoserver.sin_port = htons(atoi(GTW_PORTA));

189     if (connect(sock,
190             (struct sockaddr *) &echoserver,
191             sizeof (echoserver)) < 0) {
192         erro("Failed to connect with server");
193     }

195     echolen = strlen(info);
196     if (send(sock, info, echolen, 0) != echolen) {
```

```
197         erro("Mismatch in number of sent bytes");
198     }

200     close(sock);

202 }
```

B. Quadros resumo dos clientes HTTP

B.1 Sistema de controlo em cadeia fechada sem controlador

Parâmetro	Sistema sem controlador sem saturação	
	Premium	Outros
Taxa de ligações (a)	20,9	20,9
Taxa média de sessões (b)	19,98	19,98
Taxa de pedidos (c)	200,7	200,8
Taxa média de respostas (d)	199,9	199,9
Respostas por ligação	9,596	9,569
Ligações por sessão	1,04	1,05
Ligações concorrentes	≤ 126	≤ 123

(a) ligações/s, (b) sessões/s, (c) pedidos/s, (d) respostas/s

Quadro B.1: Quadro resumo dos clientes HTTP. Sistema de controlo em cadeia fechada sem controlador e sem saturação. 20ss. Erro nulo.

Parâmetro	Sistema sem controlador com saturação	
	Premium	Outros
Taxa de ligações (a)	50,2	50,2
Taxa média de sessões (b)	48,73	48,89
Taxa de pedidos (c)	488,7	490,1
Taxa média de respostas (d)	487,4	489,2
Respostas por ligação	9,956	9,957
Ligações por sessão	1,00	1,00
Ligações concorrentes	≤ 294	≤ 288

(a) ligações/s, (b) sessões/s, (c) pedidos/s, (d) respostas/s

Quadro B.2: Quadro resumo dos clientes HTTP. Sistema de controlo em cadeia fechada sem controlador e com saturação. 50ss. Erro nulo.

B.2 Sistema de controlo em cadeia fechada com controlador

Parâmetro	Sistema com controlador P	
	Premium	Outros
Taxa de ligações (a)	60,3	56,4
Taxa média de sessões (b)	0,13	0,00
Taxa de pedidos (c)	218,0	91,8
Taxa média de respostas (d)	173,2	66,1
Respostas por ligação	15,554	9,777
Ligações por sessão	1,99	0,00
Ligações concorrentes	≤ 694	≤ 551

(a) ligações/s, (b) sessões/s, (c) pedidos/s, (d) respostas/s

Quadro B.3: Quadro resumo dos clientes HTTP. Sistema de controlo em cadeia fechada com controlador proporcional (k_p=100). 50 ss. Erro nulo.

Parâmetro	Sistema com controlador PID					
	Erro nulo		Rej. Perturb.		Seg. Ref.	
	Premium	Outros	Premium	Outros	Premium	Outros
Taxa de ligações (a)	50,8	53,4	66,8	53,3	50,8	52,9
Taxa média de sessões (b)	8,28	0,21	2,87	0,25	7,52	1,10
Taxa de pedidos (c)	439,9	84,4	338,4	86,1	403,3	122,3
Taxa média de respostas (d)	426,4	61,8	283,4	64,2	389,9	101,6
Respostas por ligação	46,963	16,840	13,801	17,682	46,677	25,509
Ligações por sessão	1,04	1,29	1,14	1,27	1,04	1,11
Ligações concorrentes	≤ 532	≤ 453	≤ 734	≤ 481	≤ 536	≤ 516

(a) ligações/s, (b) sessões/s, (c) pedidos/s, (d) respostas/s

Quadro B.4: Quadro resumo dos clientes HTTP. Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo (k_p=5,k_i=3,k_d=8). 50ss.

Parâmetro	Sistema com controlador PID					
	Erro nulo		Rej. Perturb.		Seg. Ref.	
	Premium	Outros	Premium	Outros	Premium	Outros
Taxa de ligações (a)	26,0	29,2	43,0	28,9	25,8	28,6
Taxa média de sessões (b)	8,42	0,11	3,28	0,22	7,74	1,01
Taxa de pedidos (c)	441,3	79,6	343,8	83,7	405,3	117,6
Taxa média de respostas (d)	433,3	60,7	302,9	65,0	396,8	100,2
Respostas por ligação	46,133	13,833	13,924	15,533	46,718	21,368
Ligações por sessão	1,06	1,67	1,55	1,46 1,05	1,21	
Ligações concorrentes	≤ 387	≤ 328	≤ 579	≤ 318	≤ 384	≤ 319

(a) ligações/s, (b) sessões/s, (c) pedidos/s, (d) respostas/s

Quadro B.5: Quadro resumo dos clientes HTTP. Sistema de controlo em cadeia fechada com controlador proporcional-integral-derivativo (k_p=5,k_i=3,k_d=8). 25ss.