Nuno Miguel Prata Eliseu

Licenciado em Ciências de Engenharia

Electrotécnica e de Computadores

Redes de Telecomunicações com baixo atraso

Dissertação para Obtenção do Grau de Mestre em

Engenharia Electrotécnica e de Computadores

Orientador: Paulo da Costa Luís da Fonseca Pinto, Professor Catedrático, FCT-UNL

Júri

Presidente: Prof. Doutor João Almeida das Rosas

Arguentes: Prof. Doutor Luís Filipe Lourenço Bernardo

Vogais: Prof. Doutor Paulo da Costa Luís da Fonseca Pinto (Orientador)

Setembro de 2017

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Redes de Telecomunicações com baixo atraso

Agradecimentos

Gostaria de começar por agradecer ao meu orientador, Sr. professor Paulo da Costa Luís da Fonseca Pinto, por me ter aceite como seu orientando e por ter me orientado durante estes meses. Gostaria de agradecer a todos os meus professores que me orientaram ao longo destes 5 anos.

Gostava de agradecer à minha família pelo apoio que me deram ao longo destes 17 anos de escolaridade. E especialmente à minha mãe por ter estado sempre presente, mesmo nos momentos mais difíceis. Gosto muito dela e quero que saiba o quanto ela significa para mim.

Obrigado ao meu tio Alberto Carlos pelo apoio e confiança nas minhas competências e agradeço muito os percursos profissionais escolhidos para o meu futuro.

Queria deixar um obrigado aos meus amigos por terem feito parte do meu percurso académico e por termos passado bons momentos juntos.

Gostaria de agradecer ao meu colega Ailton Luz por me ter ajudado durante grande parte da tese e me ter orientado no NS-3. Foi das pessoas mais prestáveis durante todo o decorrer da tese.

Por fim, um especial agradecimento à minha namorada, por todos os momentos que passamos e por todos os dias que estamos juntos. Quero-lhe agradecer muito por me ter suportado durante a reta final do curso e por me ter ajudado sempre que precisava. Foi quem me apoiou em qualquer decisão.

Muito Obrigado.

Resumo

Todos os *routers* possuem *buffers* com uma grande capacidade de armazenamento, sendo possível uma fila de pacotes muito grande. Estes buffers foram desenhados para suportar o comportamento do TCP. Devido a este factor, existe uma crescente avidez por encontrar estratégias de enviar e receber mais dados, sem respeito para com outras aplicações, nem respeito pela própria rede.

Sendo assim, deverá haver uma mudança para que as redes não permitissem tal desrespeito. Essa mudança seria feita através de descartes de pacotes de *hosts* que estariam a cima dos limites da rede. Teria de haver ajuste das aplicações conforme o estado da rede para que, em vez das redes serem construídas para determinados protocolos, seriam os protocolos que teriam de se adaptar às redes.

Logo, a solução para este problema é um protocolo de congestão que se adapte ao estado da rede, sem que se saiba explicitamente o estado actual da rede. Pelo facto de a rede descartar pacotes dependendo do ritmo de transmissão de cada um dos seus utilizadores, cada utilizador tem que saber se está a transmitir no ritmo certo ou demasiado rápido.

Deste problema pode sair uma solução que pode acabar com a diferença de desempenhos de aplicações dentro de uma rede. Ou seja, se uma aplicação de transferência de ficheiros estivesse a ocupar grande parte da largura de banda da rede e uma aplicação web se juntasse, teria de haver um ajuste por ambas as partes para que nenhuma das duas aplicações sofresse com a utilização conjunta da rede.

Palavras-chave: Redes; Controlo de Congestão; Protocolo de Transmissão; Gestão de envio de pacotes; Redes com baixo atraso.

Abstract

All routers have buffers with big storage capacity, allowing for large buffering queues for packets. This kind of routers was designed to support the TCP behaviour towards congestion. A side effect has been the construction of greed applications that are aggressive in sending and receiving data, without respecting other applications, neither their own networks, and even taking advantage of the behaviour of the other applications.

As a solution, there must be a change, inside networks, for them not allowing such abuses to happen. That change would happen with the introduction of more packet drops from those who are abusing on the network. Applications would have to change their behaviour according to the network state meaning that instead of networks being built specifically for some protocols, it would be the protocols that would need to adapt to the networks.

The solution for this problem is a congestion control protocol that adapts to the network state, without directly knowing the actual state. Because the network would discard packets depending of the transmission rate of every host, everyone has to know if it is transmitting at the right rate or too fast.

From this problem could come out a solution that could end the performance difference of applications inside the network. As an example, if a file transfer application was occupying almost all the network bandwidth and a web application would join in, a change in both applications would take place, so none of them would suffer from the other one behaviour.

Keywords: Networks; Congestion Control; Transmission protocol; Packet sending management; Low delay networks.

Índice Geral

Índice de Tabelas

Índice de Figuras

Abreviaturas e siglas

ACK	*Acknowledge*
AIAD	*Additive Increase Additive Decrease*
AIMD	*Additive Increase Multiplicative Decrease*
AQM	*Active Queue Management*
ARED	*Adaptative RED*
CoDel	*Controll Delay*
DNS	*Domain Name System*
FCCP	*Fast Congestion Control Protocol*
FEC	*Forward Error Correction*
MIMD	*Multiplicative Increase Multiplicative Decrease*
NACK	*Not Acknowledge*
P2P	*Peer to Peer*
PID	*Propotional Integral and Differential Controller*
QUIC	*Quick UDP Internet Connection*
RAP	*Rate Adaptation Protocol*
RED	*Random Early Detection*
RTT	*Round Trip Time*
SDN	*Software Defined Network*

TCP	*Transport Control Protocol*
TLP	*Tail Loss Probe*
UDP	*User Datagram Protocol*
UDT	*User Data Transfer*
VoIP	*Voice over IP*

1 Introdução

1.1 Enquadramento

Hoje em dia os *routers* são constituídos por *buffers* sobredimensionados, o que permite a acumulação de um grande número de pacotes. Isto acontece porque esses *buffers* são desenhados para um melhor desempenho do TCP, para evitar perdas. Estas perdas são a razão do atraso no desempenho do controlo de congestão, pois provocam uma imediata diminuição do débito. De certa forma, se houver uma diminuição nas perdas, existe um melhor desempenho nas transferências. Sendo assim a rede está desenhada de acordo com o modelo do TCP e não para um melhor desempenho da própria rede.

Actualmente existem aplicações que abrem múltiplas ligações TCP para ter o melhor desempenho possível. Estas aplicações aproveitam-se da filosofia subjacente aos algoritmos de controlo do TCP, que reduz o ritmo em caso de congestão, de maneira a que sejam mais rápidas que as aplicações atualmente existentes na rede. Desta forma é criada uma filosofia de ganancia entre as aplicações fazendo com que aquela que enganar melhor a rede é a que vai transferir mais rápido e melhor. O que está subjacente é a ideia de que ter 30 conexões a transferir a um ritmo mínimo, mas constante, poder ser melhor do que ter uma conexão a transferir a um ritmo variável (umas vezes mais rápido, outras vezes mais lento).

Para além destes factos, os *buffers* que se encontram nos *routers* são o que tornam o equipamento caro, por esse motivo, se o tamanho destes *buffers* for reduzido, o preço destes mesmos equipamentos também se reduz. Mas para tal tem de ser criado um protocolo novo para as aplicações, com um controlo de congestão diferente dos protocolos já existentes. Tem também de ser criado um protocolo novo para os routers, que saiba gerir

esta nova realidade de *buffers* mais pequenos. Em vez de evitar as perdas de pacotes, através de *buffers* de grandes dimensões, reduz-se os seus tamanhos e assume-se as perdas dos pacotes, ajustando o valor de transmissão. Estas perdas podem ser, ou não, utilizadas para ajustar o ritmo de transmissão por parte das aplicações.

Se os equipamentos fossem mudados com base em *buffers* de tamanho reduzido, todo o comportamento relacionado com o descartar pacotes da rede teria de ser alterado. Na parte dos *routers* existe o modo como os pacotes podem ser descartados e de quem (*hosts*, fluxos, etc.) eles devem ser descartados. No primeiro aspeto, existem alternativas como o RED ou o sistema *drop tail*. No segundo aspeto o *router* pode identificar a origem e descartar em função dela baseado num valor relacionado com a capacidade da rede. Assim, a rede ganha controlo sobre o tráfego e deixa de ser a aplicação que gere isso.

Com as redes a ganhar controlo do tráfego em vez das aplicações, as aplicações têm de se adaptar ao novo modelo de congestão: reagir às perdas o melhor possível sem que seja comprometido o desempenho da aplicação.

1.2 Objectivos

O objectivo desta dissertação foi o de desenvolver um sistema de controlo de congestão que se adaptasse à rede. A rede possui um algoritmo de descarte de pacotes justo com base no comportamento de cada *host* na rede. Com isto pretende-se que exista uma nova forma de controlar o tráfego que passa por uma rede que não esteja desenhada de acordo com o protocolo de transporte.

Pretende-se que este modelo de controlo de congestão, na camada de transporte, seja: mais realista do que o TCP em presença de aplicações gananciosas, devolvendo o controlo à rede; que tenha uma semântica de fiabilidade de entrega de pacotes sendo, portanto, diferente do UDP, em que são enviados pacotes sem ter interesse sobre a sua recepção; e consiga adaptar-se às condições da rede quando esta reordena os recursos da rede penalizando aplicações que estão a transmitir acima do que devem.

1.3 Organização da Dissertação

Capítulo 1 – Introdução:

Breve explicação do que vai ser elaborado nesta dissertação. Inclui um breve enquadramento do tema que vai ser abordado e quais os objectivos que se pretendem realizar.

Capítulo 2 – Estado da Arte:

Apresentação das tecnologias em desenvolvimento neste momento. Quais os estudos mais relevantes para esta dissertação e quais as que merecem mais atenção. É apresentado o modelo do TCP e algumas variações. É apresentado o modelo QUIC como exemplo de uma nova tecnologia apresentada pela Google.

Capítulo 3 – Descrição do Projecto:

Nesta secção é apresentado o objectivo e o que consiste o projecto desenvolvido nos capítulos seguintes. É feita uma pequena introdução da rede e o porquê do estudo de redes com muitos descartes de pacotes. Este é o ponto inicial para todo o projecto.

Capítulo 4 – Descrição e testes da Rede:

São apresentados todos os testes feitos, com a descrição do que cada um representa. É descrito em pormenor a rede usada e no que consiste cada parte. São feitos 3 testes distintos para comparar resultados de cada um.

Capítulo 5 – Descrição e teste de aplicações:

Aqui são apresentadas as diferentes aplicações criadas e são apresentados todos os testes efectuados. Cada teste é comparado a um *stream* UDP com ritmo idêntico ao da aplicação em causa.

Capítulo 6 – Conclusões:

Neste capítulo são feitas todas as conclusões relativas ao estudo do modelo apresentado nesta dissertação. É dito se os objectivos foram cumpridos e qual a razão para tais resultados. Por fim é retratado o trabalho que deve ser efectuado em trabalhos futuros. Também retrata a possibilidade de desenvolver todo este trabalho em redes SDN, algo que ainda não foi possível fazer.

2 Estado da Arte

Com o desenvolvimento das redes de telecomunicações, muitos são os estudos para desenvolver protocolos que satisfaçam a procura de maneira a que haja a melhor qualidade possível durante o maior tempo possível. Alguns estudos são realizados para procurar maneiras de responder à procura quando a rede não tem muitas capacidades para responder. Outros têm como obejctivo tornar o envio e recepção o mais rápido possível com o menos falhas possível.

Este capítulo tem como objectivo introduzir as tecnologias existentes nos tempos de hoje e as que as primeiras a serem usadas e mostrar quais as suas vantagens e desvantagens. Com isto, é pretendido encontrar qual o melhor caminho a seguir para resolver o problema apresentado neste estudo.

2.1 Protocolos de Rede

2.1.1 Drop Tail

O algoritmo de controlo de congestão *Drop Tail* foi dos primeiros a ser implementado em *buffers* da rede pela sua simplicidade de implementação. Consiste em armazenar pacotes enquanto houver espaço para tal e, no momento em que a rede fica congestionada e as filas de espera ficam sobrelotadas, são descartados pacotes por não terem espaço para serem armazenados.

É um algoritmo um quanto primitivo pelo facto de só haver prevenção de congestão quando o mal já está cometido. Só quando os *buffers* estão saturados é que ocorre descar-

tes de pacotes, não sendo nada feito até aí para prevenir a congestão. Se o problema continuar por algum tempo, não existe nenhum mecanismo que alívia a rede da congestão, tornando o problema persistente e só com a redução das transmissões por parte dos *hosts* é que o problema se resolve.

Este problema pode comprometer a Internet dos dias de hoje pelo facto de o tráfego ser composto, essencialmente, por rajadas de pacotes, grandes quantidades de pacotes que chegam aos dispositivos da rede. Este tráfego ao chegar a um *buffer* saturado, tem a grande parte dos pacotes, ou a sua totalidade, descartados, causando uma grande redução na taxa de transferência da rede.

Outro problema que ocorre com o uso de um mecanismo como o *Drop Tail* é a falta de justiça. Se um *host* transferir uma grande quantidade de pacotes em rajada, com uma taxa de envio muito superior aos restantes *hosts*, vai ser ele quem vai ocupar a maior parte do espaço no *buffer*; ou seja, será ele o grande causador do congestionamento na rede. Por esse motivo os novos algoritmos de controlo de congestionamento são desenhados de forma a evitar a monopolização da rede.

Um outro problema que pode ocorrer com o algoritmo *Drop Tail* é o facto de apenas os mesmos *flows* estarem a sofrer descartes e os maiores responsáveis pelo congestionamento terem "sorte" pelos pacotes descartados.

Estes são problemas apontados pelo RFC 7567 [15] onde também descreve, como solução para o *Drop Tail*, dois mecanismos já existentes. Estes mecanismos têm as suas vantagens e desvantagens em termos de implementação.

O primeiro mecanismo que visa resolver o problema de justiça no algoritmo *Drop Tail* é o *Random Drop*. Este mecanismo consiste em descartar um pacote já armazenado na fila de espera, de forma aleatória, para que um pacote recém-chegado ao *router* seja armazenado. Desta forma, os *hosts* que possuem mais pacotes no *buffer*, têm mais probabilidade de terem um dos seus pacotes descartados. A única questão contraditória deste mecanismo é o elevado poder computacional necessário para ser implementado.

O segundo mecanismo criado para resolver os problemas do *Drop Tail* é o algoritmo *Head Drop*. Este é idêntico ao *Random Drop*, com a diferença de que o pacote descartado não é aleatório, mas sim o primeiro pacote do *buffer*. A vantagem deste mecanismo é a simplicidade de implementação, resolvendo, como no *Random Drop*, o problema de monopolização do *buffer*.

Por muito que melhore o funcionamento do *Drop Tail*, o algoritmo em si não tem grandes vantagens face a alguns mecanismos desenhados para resolver o problema de

congestionamento na rede. Nas redes de hoje em dia, os pacotes descartados são o suficiente para indicar a congestão aos *hosts*. A solução que deve ser tomada para prevenir a congestão, ao invés de remediar, será o descartar de pacotes, ou a marcação de pacotes antes de ocorrer uma saturação na rede. Dessa forma os *hosts* notificados terão tempo para ajustar as suas taxas de envio. Sistemas como este estão a ser implementados e é parte dos objectivos desta tese: criar uma rede com notificação antecipada para prevenir uma congestão.

Sistemas como este são categorizados como sistemas AQM (*Active Queue Management*) e permitem que os dispositivos da rede controlem quantos e quando é que um pacote deve ser descartado.

2.1.2 Active Queue Management

Para resolver os problemas apresentados pelo *Drop Tail* foi desenvolvido um sistema que toma certas medidas para prevenir a congestão na rede. Foi desenvolvido então o *Active Queue Management* (AQM) [15].

No início da Internet a rede era usada para uma grande variedade de tráfego, para diferentes protocolos e aplicações [16]. Nos dias de hoje, é necessária uma latência reduzida para satisfazer as necessidades das aplicações com baixo atraso do tipo VoIP, *Stream*, *Live TV*, *Videogames*, etc.

De acordo com o RFC 7567 [15], um dos maiores responsáveis pelo aumento do atraso na transferência de pacotes é o aumento do tamanho dos *buffers*. O acumular de pacotes deve-se ao facto de o ritmo de chegada ser maior do que o ritmo de saída. O tamanho excessivo dos *buffers* leva a um atraso indesejado, reduzindo o desempenho das aplicações.

Características

O objectivo de um sistema AQM é de reduzir a latência significativamente, em toda a Internet, dando possibilidade aos dispositivos da rede de controlar o tamanho da fila de espera ou o tempo que um pacote permanece no *buffer*. Sendo possível distinguir dois tipos de algoritmos, dentro do AQM: "*queue management*" e "*scheduling*". O primeiro algoritmo tem como tarefa gerir o tamanho dos *buffers* ao marcar ou descartar pacotes quando necessário. O "*scheduling*" tem como função programar o envio de certos pacotes e tem como principal tarefa alocar largura de banda na rede.

O AQM foi criado como forma de substituir sistemas como o *Drop Tail* e tem como objectivo resolver os seguintes problemas:

- Congestão nas filas de espera
- Evitar que os *buffers* sejam monopolizados
- Maximizar a largura de banda utilizada na rede
- Reduzir o atraso de transmissão de pacotes
- Diminuir o descarte de pacotes

São exemplo de AQM's algoritmos como o CoDel (Controlled Delay Management), RED (*Random Early Detection*), ARED (Adaptive RED), etc.

Múltiplos Buffers

Uma das grandes características do AQM é a criação de mecanismos para recepção de rajadas de pacotes sem haver descarte, mas para isso é necessário ter o total controlo do tamanho das filas de espera. É necessário a divisão de filas para cada fluxo de transferência de forma a que o atraso seja o mais baixo possível. Dessa forma o AQM, combinado com mecanismos de "*scheduling*", divide o tráfico da rede em múltiplas filas de espera.

Isto deve-se ao facto de haver tráfego completamente diferente no mesmo protocolo de transporte. Dois fluxos de TCP podem ter um comportamento completamente diferente só por possuírem RTT's diferentes [17]. Dessa forma, é necessário um sistema que se adapte ou que seja rigoroso a todos estes diferente fluxos de dados para que não haja prejudicados injustamente.

Os algoritmos AQM, ao possuírem divisão de *buffers*, estão a prevenir que haja um "abuso" da rede por parte de aplicações sem controlo de congestão ou com sistema propositados para transferirem mais do que estão permitidos.

Explicit Congestion Notification (ECN)

Uma das formas de prevenir o congestionamento da rede é enviar pacotes de aviso aos causadores da saturação. Esses avisos são os *Explicit Congestion Notification* (ECN), e consistem em assinalar alguns pacotes com informação relevante ao destino do pacote para reduzir o seu ritmo de transferência. Essa medida serve para diminuir a quantidade de pacotes descartados e assim reduzir a congestão do dispositivo de rede.

Tipos de tráfegos

O trabalho do TCP em reduzir o ritmo de transferência sempre que ocorre congestionamento é um dos elementos chave para o sucesso da Internet. O problema que surge, ao longo do tempo, são os métodos criados para contornar o controlo de congestionamento do TCP. Um desses métodos consiste em criar múltiplas conexões TCP em paralelo e terminar cada ligação após o *Slow Start*. Após uma ligação ser desligada, é criada uma nova, dessa forma o débito da aplicação é superior por não haver fase de

controlo de congestão do TCP. Resumindo: existem muitos tipos de algoritmos de transferência, uns que colaboram com as medidas de controlo de congestão e outros não. Desta forma o AQM classificou dois tipos de tráfegos correntes na rede.

Um dos tráfegos classificados foram os fluxos TCP-*friendly*, por outras palavras: fluxos "amigos" do TCP, ou seja, são algoritmos que respeitam os mecanismos de recuperação da rede do TCP e não transmitem mais do que o TCP. São os fluxos que se adaptam à recepção de notificações de congestionamento, seja descartes de pacotes, seja ECN's.

O outro tipo de tráfego classificado foram os fluxos não responsivos. São caracterizados por não responderem a notificações de congestionamento. Exemplos deste tipo de fluxos são as aplicações que usam *User Datagram Protocol* (UDP) [18]. São algoritmos simples de envio de pacotes, sem qualquer preocupação pelo tráfego na rede. São insensíveis a qualquer tipo de notificação de congestionamento.

Como conclusão, é necessário que haja uma protecção por parte da rede para combater as aplicações insensíveis a descartes ou notificações e haver um trabalho mútuo entre a rede e as aplicações com controlos de congestionamento implementados, para resolver casos de congestionamento que ocorram em certos dispositivos da rede.

2.2 Controlo de Congestão

2.2.1 AIMD

O AIMD é a abreviatura para *Additive Increase Multiplicative Decrease* e foi primeiramente implementado no controlo de congestão do TCP. Este mecanismo consiste em aumentos aditivos da janela de congestão do TCP, e decrementos multiplicativos quando ocorrem falhas na transmissão. De acordo com [19], quantos mais fluxos de tráfego houver na rede com sistemas de AIMD, mais facilmente a rede converge para um valor estável sem congestão.

No caso do TCP, pode-se considerar como um sistema AIMD o mecanismo que ocorre após o valor da janela de congestão ultrapassar o valor de *threshold*. Nesse momento, a janela de congestão é aumentada unitariamente até que uma falha na transmissão ocorra. Essa falha de transmissão pode ser declarada tanto pelo descartar de pacotes, como por notificações à fonte de que a rede está a saturar: no caso do TCP é por descarte de pacotes que a origem é notificada. Ao ocorrer uma falha a janela é reduzida de uma forma multiplicativa. Por exemplo, no caso do TCP *Reno*, é reduzida em metade do valor.

A diferença entre os sistemas AIMD é os valores usados para o incremento e o decremento. Por exemplo, o trabalho [20] retrata um sistema AIMD geral para vários valores de incrementos e decrementos.

Existem variações do AIMD que tendem a ser variações idênticas, mas com mecanismos diferentes. É o caso do *Multiplicative Increase Multiplicative Decrease* (MIMD), que consiste em incrementos e decrementos multiplicativos, e o caso do *Additive Increase Additive Decrease* (AIAD), que consiste em incrementos e decrementos aditivos. A particularidade destes mecanismos é que não tendem em convergir para um valor estável.

2.2.2 Window Based

O sistema de controlo de congestão do TCP é composto por uma janela de transferência cujo tamanho é ajustado conforme são recebidos pacotes correctamente pelo destino. Este tamanho representa o número de pacotes que são transferidos num período de tempo em que não é recebido nenhum reconhecimento (*Acknowledge*). Quanto maior for o tamanho da janela mais pacotes são enviados antes que o primeiro pacote seja confirmado pelo destino.

No caso do XCP [21], a noção de janela é idêntica à do TCP, a maneira como o valor da janela se altera é que difere do TCP. É através de uma variável no cabeçalho de *feedback* que a origem faz um pedido de aumento do tamanho da janela. Em caso de resposta positiva, no ACK recebido, a janela é aumentada; em caso de resposta negativa a janela é reduzida:

$$cwnd = \max\left(cwnd + H\_feedback, s\right) \quad (1)$$

onde *cwnd* é o tamanho da janela, *s* é o tamanho dos pacotes e *H_feedback* é o *feedback* recebido do destino dos dados.

O valor do aumento desejado é calculado através do valor do ritmo de transferência desejado (*r*), do valor actual do *round trip time* (*rtt*) e o valor actual do tamanho da janela (*cwnd*):

$$H\_feedback = r \times rtt - cwnd$$

dividido pelo número de pacotes na janela de congestão actual. Se for possível, em termo de largura de banda, a origem ajusta o seu valor de ritmo depois de apenas um RTT.

2.2.3 Rate Based

Para algoritmos de controlo de congestão, só agora é que estão a aparecer alternativas aos sistemas de janela. Exemplos desses algoritmos são os controlos de congestão do UDT e do RAP.

No caso do UDT (*UDP-based Data Transfer protocol*) é utilizado um sistema de controlo de congestão baseado em controlo de ritmo, mas também possui um sistema de controlo de fluxo com um controlo de janela. No caso do controlo de ritmo, ele actualiza o ritmo de transferência a cada RTT. De acordo com [4], em redes com tráfegos muito rápidos, gerar pacotes *Acknowledge* por cada pacote de dados enviado é algo que atrasa o processo. Por esse motivo é utilizado um sistema de ACK selectivo através do tempo. Cada pacote ACK é gerado num tempo fixo, dessa forma, quanto maior o ritmo de transferência, menor o número de ACK's enviados, em termos relativos.

Tal como XCP, o UDT tem ajustes de ritmo que são feitos através de *feedback* vindo do *host* destino. Quando o *feedback* é positivo o ritmo é aumentado e quando o *feedback* é negativo o ritmo é decrementado. Mais pormenores do protocolo UDT serão tratados na secção 2.4.3.

Em relação ao RAP (*Rate Adaptation Protocol*) [22], se ocorrer algum sinal de congestão, o ritmo de transferência é diminuído. Se não houver nenhum sinal de que a rede está saturada o ritmo de transferência é aumentado. Estes aumentos e decrementos são feitos com base num sistema AIMD, ou seja, sempre que o ritmo é aumentado é de forma incremental, se houver um decréscimo de ritmo é de forma multiplicativa. Só ocorre decréscimos sempre que é detectada uma perda. Existem dois mecanismos para detectar uma perda: *timeouts* e falhas entre pacotes recebidos.

Para cada pacote enviado são guardadas todas as informações de envio do pacote (numero de sequência, tempo de saída, ritmo de transferência e *flags* do estado). Esta informação guardada é chamada de *transmission history*. Os pacotes em memória são depois utilizados sempre que um novo pacote é transmitido para estimar o *timeout* desse novo pacote.

O mecanismo de ajuste de ritmo (S) é feito através de ajustamentos no *inter-packet-gap* (IPG). Cada ajustamento de ritmo é feito através de ajustamentos por escala:

$$S_{i+1} = S_i + \alpha \tag{2}$$

onde α representa o incremento. Para o ritmo ser aumentado, o IPG tem de ser aumentado:

$$IPG_{i+1} = \frac{IPG_i \times C}{IPG_i + C} \tag{3}$$

onde vem que o valor S_i é mudado com base no valor de IPG:

$$S_i = \frac{PacketSize}{IPG_i} \tag{4}$$

através destes valores é calculado o valor de α:

$$\alpha = S_{i+1} - S_i = \frac{PacketSize}{C} \tag{5}$$

No caso de um decremento no ritmo, como é um decremento multiplicativo, é uma redução de metade do valor actual:

$$S_{i+1} = \beta S_i \qquad IPG_{i+1} = \frac{IPG_i}{\beta} \qquad \beta = 0.5 \tag{6}$$

2.3 Variações TCP

2.3.1 TCP

Slow Start e Congestion Avoidance

Os mecanismos de *Slow Start* e *Congestion Avoidance* são os dois elementos do TCP que controlam a quantidade de pacotes enviados, na rede, pelo algoritmo TCP. Controlam o tamanho da janela de congestão, o tamanho máximo que o protocolo TCP espera enviar sem ter recebido nenhum ACK, mas também o tamanho da janela de recepção, que representa o tamanho máximo que o destino espera receber. Estes elementos também são os responsáveis pela definição do *threshold*, que é o valor que define se o controlo de dados é feito pelo algoritmo *Slow Start* ou pelo *Congestion Avoidance*.

Estas variáveis têm de ser definidas através de uma grande carga na rede para o algoritmo saber qual o limite antes da congestão. Esse mecanismo é activado através do *Slow Start*. Sendo o valor do *threshold* definido com um valor aleatório alto, mas reduzido à medida que a congestão aumenta.

O mecanismo *Slow Start* é utilizado somente quando a janela de congestão tem um valor menor que o *threshold*. No momento em que a janela é maior que o *threshold* é a *Congestion Avoidance* que entra em acção. Quando os valores são iguais, tanto é utilizado um como o outro, dependendo da preferência.

O aumento da janela, durante o *Slow Start*, é multiplicativo usando-se o fator dois, ou seja, é sempre aumentado o valor anterior para o dobro. No momento em que o valor

é superior ao do *threshold*, a janela é aumentada de forma somatória, com incrementos unitários.

Quando ocorre uma falha, ou a origem não recebe um ACK da janela enviada, o *threshold* é definido para metade do valor da janela de transferência. A janela é reiniciada com o valor unitário e todo o processo repete-se. [11]

Fast Retransmit e Fast Recovery

O *Fast Retransmit* e o *Fast Recovery* foram criados para recuperar algumas falhas que ocorrem durante a transmissão de pacotes por parte do TCP. Eles complementam-se um ao outro. Enquanto o *Fast Retransmit* baseia-se na recepção de *Acknowledge* duplos de pacotes que foram perdidos, o *Fast Recovery* assegura o reenvio destes pacotes detectados.

Para ser mais concreto, o *Fast Retransmit* tem como objectivo detectar se ocorrem falhas na transmissão através de múltiplos ACK's duplicados que chegam ao controlo de congestão da origem. Como o TCP não consegue detectar se a chegada destes sinais duplicados é causada por falhas de pacotes ou somente a reordenação de segmentos, o algoritmo espera por um pequeno número de ACK's duplicados para serem recebidos. No momento que são recebidos três duplicados é accionado o *Fast Recovery*.

Este *Fast Recovery* é o responsável por assegurar que os excertos de dados perdidos são reenviados. A única diferença para o TCP *Tahoe* é que, em vez de ser accionado o mecanismo de *Slow Start*, o *Fast Recovery* acciona o mecanismo de *Congestion Control*. Esta grande diferença permite que haja uma maior convergência para o valor óptimo de envio e consequentemente um maior *throughput*. [11]

Outros algoritmos

Nos dias que correm existem muitos géneros de TCP's com diferentes protocolos de controlo de congestão. Isto faz com que a *Internet* esteja a evoluir de um tráfico homogéneo para um tráfico mais heterogéneo, em termos de controlos de congestão. É possível reparar na listagem de algoritmos TCP existentes nos diferentes sistemas operativos na tabela seguinte. [6]

Operating Systems	TCP algorithms
Windows family	RENO [2], and CTCP [8]
Linux family	RENO, BIC [12], CUBIC [13], HSTCP [14], HTCP [15], HYBLA [16], ILLINOIS [17], LP [18], STCP [19], VEGAS [20], VENO [21], WESTWOOD+ [22], and YEAH [23]

Tabela 1. Lista de diferentes algoritmos TCP.

Grande parte dos controlos de congestão existentes actualmente já não são RENO, o que indica que outros protocolos como o CUBIC [7], CTCP [8], DCCP [9] e SCTP [10], que foram inicialmente desenhados para competirem com o RENO mantendo uma relação amigável para com o seu tráfego, já não têm de se comportar como se comportavam. Por esse motivo é necessário haver protocolos que não se preocupem com o desempenho de outros algoritmos, mas sim unicamente com o seu desempenho [6].

2.3.2 TCP Reno e TCP New-Reno

Após o aparecimento do TCP, seguiu-se o aparecimento de protocolos de transmissão derivados do TCP *Tahoe* (um dos primeiros algoritmos implementados) como por exemplo o TCP *Reno*, uma das primeiras derivações do TCP criadas [12]. Tendo sofrido mais tarde adaptações para dar lugar ao TCP *New-Reno*. Os fundamentos destes algoritmos são bastante semelhantes ao protocolo inicial do TCP, tendo sido feitos alguns ajustes em termos da janela de congestão.

TCP Reno

No caso do algoritmo *Reno*, o que difere do *Tahoe* é o facto de implementar o sistema de *Fast Recovery*. Enquanto no *Tahoe* se reinicia a janela sempre que se recebem três vezes um *Acknowledge* repetido, no caso do *Reno* a janela não é reiniciada. Como neste caso o tamanho da janela é aproximado do tamanho ideal de envio, o algoritmo *Reno* define o valor da janela igual ao valor do *threshold*. Dessa forma o débito do algoritmo é muito superior ao do *Tahoe*, pois não é definido com o valor de janela unitário.

Tudo o resto no algoritmo é idêntico ao *Tahoe*. Sempre que ocorre um *timeout* a janela é reiniciada, tal como no *Tahoe*. A única diferença é mesmo quando ocorrem três ACK repetidos.

TCP New-Reno

As diferenças entre o protocolo *Reno* e o protocolo *New-Reno* ainda são menores do que do *Reno* para o *Tahoe*. Neste caso a diferença fica só na maneira como o *New-Reno* lida com os *Acknowledges* duplicados após o primeiro ACK duplo. Enquanto o *Reno*, simplesmente reduz a sua janela para o valor de *threshold*, o protocolo *New-Reno* vai aumentando a janela de congestão à medida que novos ACK's chegam à origem. Dessa forma, um *Acknowledge* duplicado já não implica ter que reduzir a janela enquanto se procede o *Fast Retransmit*, mas são enviados os pacotes seguintes dos *Acknowledges* recebidos a seguir ao ACK duplo.

2.3.3 TCP Vegas

Retransmissão

Segundo [2], o TCP Vegas tem como objectivo minimizar as perdas do TCP e isso é feito através de uma decisão mais temporal na retransmissão de pacotes perdidos. Para isso ele divide-se em duas partes fundamentais. A primeira é idêntica ao método já utilizado pelo TCP que consiste em calcular o RTT através do tempo entre um pacote enviado e um ACK recebido desse mesmo pacote. Depois decide retransmitir dependendo destas duas situações: ao receber dois ACK's repetidos, com um RTT superior ao RTT anterior calculado em situação normal, então há retransmissão; quando um pacote não repetido de um ACK é recebido, se for o primeiro ou o segundo após uma retransmissão, então é feito o mesmo cálculo dos RTT's e, no caso de ser maior, então é retransmitido.

Em segundo lugar, o TCP Vegas, ao detectar perdas mais rapidamente do que o TCP original, só baixa a sua janela de transmissão se o pacote retransmitido foi enviado primeiramente antes do último reajuste da janela de transmissão. Sendo assim, se for recebido um ACK duplicado após um decréscimo da janela e a retransmissão em causa for de um pacote enviado antes da nova janela, então não ocorre um novo ajuste, mas se o pacote em causa for enviado após a nova janela de transmissão, então haverá um novo ajuste.

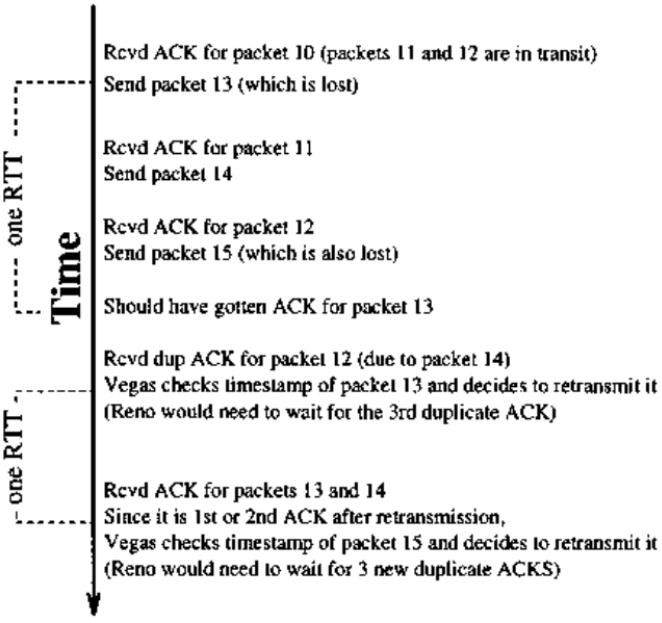


Figura 2.1: Exemplo de mecanismo de retransmissão do Vegas [2]

Controlo de Congestão

A proposta de controlo de congestão é semelhante à proposta apresentada por Wang e Crowcroft's Tri-S scheme [referência]. Este sistema consiste em, a cada RTT, aumentar a janela de transferência em um segmento e comparar as taxas de transferências, antes e depois do aumento. Se a diferença entre as duas taxas for menor do que a metade da taxa adquirida quando a janela era só um segmento, então a janela é diminuída por uma unidade.

O que difere entre o Tri-S e o Vegas é a forma como é calculada a taxa de transferência. Enquanto no sistema de Wang e Crowcroft, a taxa é calculada dividindo o número de *bits* pelo RTT, o TCP Vegas elabora esse cálculo através da comparação da taxa de transferência medida (R_{act}) com a taxa de transferência esperada (R_{exp}).

$$R_{exp} = \frac{WindowSize}{BaseRTT} \quad (1)$$

Em que $BaseRTT$ é o menor RTT e é, geralmente, calculado no início de cada transmissão pelo facto de não haver congestão.

$$Diff = R_{exp} - R_{act} \quad (2)$$

Em seguida, como $Diff$ é positivo ou zero por definição porque $R_{act} > R_{exp}$, é substituído o $BaseRTT$ pelo RTT actual. São depois definidos dois *thresholds*, em que $\alpha < \beta$. Quando $Diff < \alpha$ então a janela de transferência é aumentada linearmente dentro do próximo RTT. Quando $Diff > \beta$ a janela de transferência é diminuída linearmente no próximo RTT. Isto significa que o α é responsável pelo incremento da janela de transferência e o β o responsável pelo decremento da janela.

Novo mecanismo de Slow-Start

O *Slow-Start* do Vegas consiste em aumentar a janela de transferência exponencialmente, tal como o TCP original, mas com a pequena diferença de que, em vez de fazer essa alteração a cada ACK recebido pelo *host* de origem, a alteração é feita cada vez que o RTT é alterado. Desta forma dá tempo para que a taxa de transferência seja analisada com cuidado para se tomar decisões nesse tempo de espera. O aumento da janela do Vegas é exponencial, tal como o do TCP original (1, 2, 4, 8, 16...).

No momento em que a taxa de transferência actual é menor que a taxa de transferência esperada, então o Vegas passa de modo *Slow-Start* para modo Controlo de Congestão.

2.3.4 Fast TCP

Arquitectura

A arquitectura do Fast TCP separa o controlo de congestão do TCP em quatro componentes: *data control* que determina que pacotes devem ser enviados; *window control* que determina quantos pacotes devem ser transmitidos; *burstiness control* que determina quando deve haver transmissão ou não; componente de estimação que fornece informação aos outros componentes para que estes tomem decisões.

Mais concretamente, as estimativas são calculadas a cada pacote enviado, e são computadas duas informações – um multibit que indica o atraso da fila de espera e um bit que indica se houve perda ou não – e através destas informações os outros três componentes tomam decisões sobre o envio de pacotes.

Falando em pormenor do *data control*, este tem a tarefa de transmitir pacotes de um conjunto de três tipos de pacotes. Este conjunto está dividido em pacotes novos, ainda não transmitidos, pacotes em que os ACK's ainda não foram recebidos e pacotes em que foram recebidos NACK's.

O módulo *window control*, como já foi antes dito, é o responsável pela regulação do envio de pacotes num determinado RTT. Este módulo é abordado com mais pormenor na próxima parte deste subcapítulo.

Quanto ao *burstiness control*, ele tem a tarefa de amortizar e suavizar o envio de pacotes de maneira a que o envio seja o mais fluído possível. São implementados dois mecanismos, um mecanismo de relógio para envio individual de pacotes e um mecanismo de aumento gradual da janela de envio através de pequenos incrementos.

Algoritmo do Window Control

O sistema TCP Fast adapta a sua janela de congestão periodicamente dependendo da média do RTT e do atraso da fila de pacotes. Este modelo de congestão tanto reage a perdas de pacotes como a atrasos das filas de pacotes.

$$Window(w) = \min\left\{2w, (1-\gamma)w + \gamma\left(\frac{baseRTT}{RTT}w + \alpha\right)\right\} \quad (10)$$

Na equação (10), $\gamma \in [0,1]$, $baseRTT$ é o RTT mínimo calculado e α é um parâmetro próprio do protocolo que determina o número total de pacotes armazenados em fila. Sempre que se detectar perdas de pacotes a janela de transferência é reduzida para metade da janela anterior e assim o protocolo entra em modo de recuperação de perdas.

2.4 Variações UDP

2.4.1 UDP

O protocolo UDP é, juntamente com o TCP, o protocolo mais usado na Internet para a camada de Transporte, mas ao contrário do TCP, não estabelece uma conexão entre os dois *hosts*. É um protocolo estabelecido para que sejam as aplicações a fazer o seu próprio protocolo de congestionamento. Não é feito mais que o necessário, ou seja, enviar pacotes e receber, podendo detectar falhas, sendo as aplicações a decidir o que se faz com essa informação.

UDP tem como significado *User Datagram Protocol*, e é usado por aplicações para serem enviados pacotes encapsulados em pacotes IP sem que seja necessário estabelecer-se qualquer tipo de conexão entre a origem e o destino, sendo possível conexões na camada da aplicação.

O cabeçalho de um pacote de UDP é constituído por 8 *bytes*, divididos em 4 conjuntos de 2 *bytes*. O primeiro conjunto identifica o porto da origem, de onde foi enviado o pacote e tem como intuito identificar a aplicação que enviou o pacote, para depois ser usado, por exemplo, para enviar a resposta após recepção. Seguidamente vem o porto do destino. Serve para identificar a aplicação a que se destina o pacote de dados. Depois vem o campo que indica qual o tamanho total do pacote enviado, juntamente com o cabeçalho e a zona para os dados. O tamanho mínimo dos pacotes UDP é de 8 *bytes* que representa somente o tamanho do cabeçalho e o valor máximo é de 65515 *bytes*, sendo o limite de um pacote IP.

No cabeçalho do UDP existe um último campo, o quarto, que é responsável por garantir o mínimo de confiabilidade no envio dos pacotes. Esse campo é o *Checksum*, é opcional e tem como objectivo saber se houve uma falha no pacote durante o envio do mesmo. O algoritmo para o seu cálculo consiste em adicionar um zero no fim dos dados se o tamanho do pacote for um número ímpar, em seguida somar todos os conjuntos de 16 *bits* em complementos para um e pegar no resultado de todos os complementos de todas as somas efectuadas. No destino, ao efectuar o mesmo cálculo, já com o *checksum* incluído, é verificado se o valor é zero. Se não for, é assumido que houve um problema na transmissão do pacote.

Como já foi dito antes, o UDP só envia os pacotes da origem para o destino e por esse motivo não possui controlo de congestão nem reenvio após falha. Este detalhe pode ser considerado uma vantagem para comunicações de respostas curtas entre o cliente e o servidor como o caso do protocolo DNS.

2.4.2 QUIC

QUIC (*Quick UDP Internet Connection*) é um protocolo desenvolvido pela Google com o objectivo de reduzir a latência na camada de Transporte de uma conexão. Para isso é desenvolvido em cima do UDP com influência no controlo de congestão do TCP. É composto por uma mistura tanto do UDP como do TCP. A sua conexão entre a origem e o destino é mais idêntica à do UDP, com inspiração no *handshake* do TLS[x] e não tanto na conexão *3-way handshake* do TCP. A vantagem de não ter uma conexão igual à do TCP é que torna o sistema mais rápido. Estas conexões, do QUIC, são sempre encriptadas e autenticadas.

De acordo com a Google, este protocolo funciona melhor em ambientes com grandes RTT's, ou seja, com grande latência de envio de pacotes. Ainda é afirmado que, para além dos cenários de congestão, o QUIC é tão bom quanto o TCP na maioria dos outros cenários da rede. Isto é explicado pelo facto de este protocolo tentar resolver os problemas mais comuns do TCP: 3-*way handshake* do TCP; um segmento TCP só consegue enviar um/a único/a pedido/resposta HTTP 1.1; uma transferência de um item de um servidor tem de ser sempre iniciado pelo cliente mesmo que o servidor já saiba que esse item tem de ser descarregado; quando um pacote é perdido durante a transferência, o destinatário tem de esperar pelo reenvio do pacote para poder prosseguir.

Mecanismos TCP implementados no QUIC

Tal como o TCP o QUIC também usa ACK para saber se um pacote foi correctamente enviado e recebido pelo destinatário. Quanto ao controlo de congestionamento, o QUIC reimplementa o controlo do TCP Cubic com mecanismos adicionais. É usado, também, um temporizador de retransmissão que, cada vez que um segmento não é confirmado com um ACK, quando expira é retransmitido esse segmento.

O QUIC separa dois mecanismos principais: *Slow Start* e *Congestion Avoidance*. Enquanto o *Slow Start* tem início sempre que uma nova conexão começa e a janela de transferência tem um aumento exponencial. O *Congestion Avoidance* tem início sempre que há perda de pelo menos um pacote, ou a não recepção de um ACK, e nesse momento entra em funcionamento o mecanismo *Fast Retransmit* que executa um ajuste da janela linear.

O *Fast Retransmit* tem como objectivo fazer com que o sistema retransmite devido a *timeouts* (RTO's), pelo facto de ser pior para a velocidade de transferência entrar nesse estado. Este mecanismo é accionado quando a origem recebe três ACK's duplicados. Quando então é accionado, o sistema define a janela de congestão e o *threshold* do

Slow Start para um determinado valor, dependendo de onde se encontrava a janela no momento da falha. De acordo com a Google, 99% dos casos de pacotes perdidos é reconhecido pelo sistema por ACK's duplicados.

O mecanismo *Tail Loss Probe* (TLP) entra em funcionamento quando o último segmento de uma transmissão falha. Isto acontece pelo facto de ser necessário receber um segmento à frente para se poder enviar ACK's duplicados e assim identificar quais os pacotes que falharam. Para isso, a origem envia dois TLP's, antes de expirar o temporizador do RTO, contendo o último pacote não confirmado com ACK. Dessa forma o receptor acciona o mecanismo de *Fast Recovery*.

Após um *RTO*, é definida a janela de transferência para um *Maximum Segment Size* (MSS), que é o valor máximo que a janela pode chegar e a transferência prossegue com uma nova fase de *Slow Start*.

Melhoramentos do QUIC

O QUIC apresenta os seguintes melhoramentos:

As conexões são estabelecidas mais rapidamente que no TCP, entre o utilizador e o servidor. Enquanto o TCP necessita de 3 RTT's para que uma conexão seja estabelecida, no QUIC, no máximo, é necessário 1 RTT, quando o cliente é desconhecido, e 0 RTT's, quando o utilizador é conhecido. Sempre que um cliente desconhecido se tenta conectar a um servidor (HELO), o servidor envia um *Reject* (REJ), com os detalhes da sua configuração para que, da próxima que o cliente entre em comunicação com o servidor, o cliente envie um CHLO já com as configurações próprias do servidor e então ser aceite.

São usados 64-bits como identificador de uma conexão, que são gerados aleatoriamente pelo cliente. Por utilizar um único identificador para uma conexão é possível ligar múltiplos feixes (*streams*) de dados a uma só ligação, sendo mais fácil para sincronizar os pacotes enviados. Ainda facilita na mobilidade da conexão, ou seja, uma conexão pode-se manter ligada mesmo quando o IP é diferente.

Existe uma distinção entre pacotes retransmitidos e novos, algo que não se verifica no TCP. O motivo de tal acontecer é devido ao facto de cada segmento ter um número próprio, incrementado monotonamente sempre que é enviado um pacote, novo ou reenviado. Assim é mais fácil identificar qual é o segmento confirmado por um ACK, tendo vantagens no cálculo do RTT de transmissão.

Em vez de utilizar SACK's para confirmar a chegada de pacotes ao destino, como é feito no TCP, o QUIC utiliza NACK's para confirmar falha de pacotes no destino. Sendo assim evita-se ser enviado um grande número de pacotes ACK's de volta à origem. Este mecanismo torna-se vantajoso em cenários de janelas de grandes dimensões.

O QUIC utiliza um sistema de *Forward Error Correction* (FEC) que através de informação redundante nos pacotes é possível recuperar uma falha de um pacote entre vários outros pacotes.

Enquanto que o TCP envia pacotes o mais rápido que lhe é possível, o QUIC tem um ajuste de ritmo para envio de pacotes, evitando rajadas de pacotes que podem muito bem congestionar a rede e causar perdas. Esta solução é boa para redes de baixa velocidade, mas já não é tão ideal para as redes de alta velocidade.

2.4.3 UDT

O UDT é um protocolo que tanto pode ser usado para *streams* de dados como para troca de dados de maneira fiável, como é o caso de trocas de mensagens. O UDT encontra-se na camada a cima do UDP e é através do *socket* UDT que a aplicação troca pacotes. Dessa forma é criado quase como um túnel entre a aplicação e o UDP através do UDT.

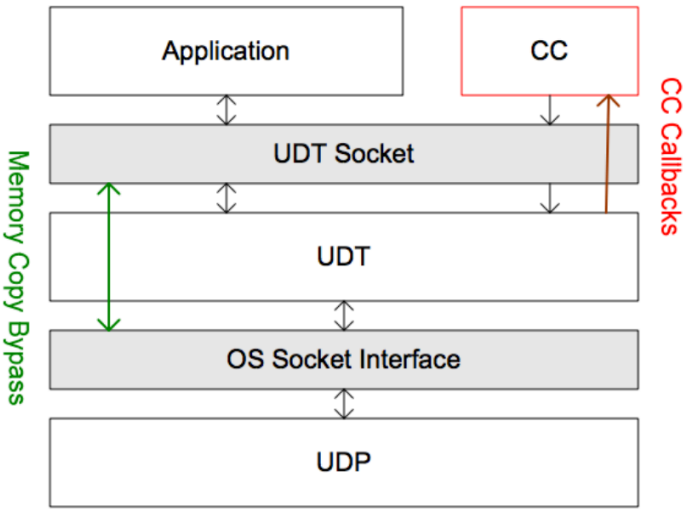


Figura 2.2: Arquitectura de camadas do UDT.

UDT utiliza controlo de congestão com *rate-based* e *window-based* para controlo de fluxo, para regular o tráfego que é enviado. O receptor é o responsável por desencadear todos os eventos de controlo e os seus mecanismos associados.

Estrutura dos pacotes UDT

O UDT é constituído por dois tipos de pacotes, um tipo de dados e um tipo de controlo. Os pacotes de dados são constituídos por um número de sequência do pacote, um número de sequência para as mensagens e o tempo relativo ao início de uma conexão.

Existem dois tipos de números de sequência. O número de sequência destinado aos pacotes de dados é o número que indica o tamanho do pacote que vai ser enviado (idêntico à sequencia do número de bytes que existe no TCP). O número de sequência destinado às mensagens indica para que aplicação de mensagens é destinado o pacote em questão. As mensagens são constituídas por vários pacotes e são necessários campos que indiquem quando é o início ou o quando é o fim da mensagem. O campo "FF" indica o limite da mensagem transmitida: 10 – primeiro pacote; 01 – ultimo pacote; 11 – único pacote. O campo "O" indica a necessidade de um determinado pacote ser entregue em ordem.

O primeiro bit de cada pacote indica se é um pacote de dados ou um pacote de controlo. Se o pacote for de dados, então este *bit* é colocado a "0", se for de controlo então o valor é "1".

Existem 7 tipos de pacotes de controlo: *handshake*; ACK; ACK2; NAK; *keep-alive*; *shutdown*; *drop request*. Em que o tipo é definido por 14 *bits* a contar desde o segundo *bit* do cabeçalho do pacote de controlo.

Inicio e fim de uma Conexão

Existem dois tipos de conexões no UDT: uma é tradicional onde um cliente e um servidor comunicam, a outra é uma conexão em modo *rendezvous*. A primeira conexão é iniciada pelo cliente que envia um pacote *handshake* ao servidor. Desde esse momento são enviados pacotes *handshake* ao servidor de x em x tempo até que o cliente receba uma resposta do servidor e a partir daí a conexão está estabelecida.

Os pacotes de *handshake* são constituídos por 5 tipos de informação associada ao cabeçalho do pacote: a versão do UDT, o tipo de *socket* que é usado (SOCK_STREAM ou SOCK_DGRAM), o número de sequência inicial gerado aleatoriamente, o tamanho máximo do pacote e o tamanho máximo da janela de envio.

Existe ainda o tipo de conexão *rendezvous* que tem como objectivo serem conectados dois utilizadores sem haver necessidade para servidores nem clientes. Ambos os

clientes iniciam a tentativa de conexão e o primeiro a receber o pedido envia uma resposta e é iniciada a conexão entre ambos os utilizadores.

Fiabilidade na comunicação

É usado um sistema como no TCP para identificar uma perda de pacote ou uma situação de congestão na rede. Esse sistema é uma mensagem de ACK entre o destino e a origem para indicar a chegada de um pacote, ou n pacotes.

Este processo de confirmar a chegada de pacotes no destino é um processo lento de se fazer, logo, o que o UDT faz é iniciar temporizadores para o envio de ACK's. Desta forma sempre que o temporizador é disparado é enviado um ACK, desde de que não haja falhas nos envios de pacotes. Isto significa que à medida que a velocidade de transferência aumenta, menor é a taxa de envio de ACK's por pacote recebido.

Existe também um pacote ACK2 que é criado sempre que a origem recebe um ACK do destino. Serve para confirmar a chegada de um ACK, desta forma é possível também calcular o RTT da linha, pelo facto de serem pacotes com tamanho reduzido.

Para transmitir uma perda de pacote, em vez de ser usado o sistema SACK do TCP, são usados pacotes NAK que são transmitidos sempre que é detectada a perda de pacote. Tem como função fazer reagir a origem para se adaptar à perda, o mais rápido possível.

Controlo de Congestão

O algoritmo de controlo de congestão *rate-based*, é classificado como AIMD, mais concretamente um sistema DAIMD, que funciona como um AIMD, mas com decrementos na função de incremento, ou seja, cada incremento que o ritmo sofre é menor do que o incremento anterior.

Mais concretamente, o algoritmo de congestão, desde de que não receba nenhuma resposta de perda de pacotes ou aumento do tempo de envio de pacotes, o ritmo de transferência (x) será aumentado através de uma função $\alpha(x)$ (Figura 2.3).

$$x \leftarrow x + \alpha(x) \tag{4}$$

A função $\alpha(x)$ aproxima-se de 0 à medida que x aumenta, ou seja, $\lim_{x \to +\infty} \alpha(x) = 0$. Sempre que existe uma resposta negativa, de perda ou de atraso, o ritmo de envio de pacotes é diminuído por um factor de β, onde ($0 < \beta < 1$):

$$x \leftarrow (1 - \beta) \cdot x \tag{5}$$

O UDT define que a função $\alpha(x)$ tem de ser grande quando $x = 0$ e diminuir rapidamente para convergir para a estabilidade o mais rapidamente possível, para evitar oscilações:

$$\alpha(x) = 10^{\lceil \log(L - C(x)) \rceil - \tau} \cdot \frac{1500}{S} \cdot \frac{1}{SYN} \tag{6}$$

Onde SYN é igual a 0.01 segundos e representa o intervalo de tempo de sincronização, x representa as unidades de pacotes por unidade de tempo, L é a capacidade da ligação medida em bits por segundo, S é o tamanho dos pacotes do UDT em bytes, τ é um parâmetro específico do protocolo e $C(x)$ é a função que converte a unidade do ritmo actual para bits por segundo.

Este controlo de congestão só é activado quando a origem recebe o seu primeiro pacote de *feedback* negativo, como resposta a uma perda ou atraso de pacotes, e aí o algoritmo deixa de ser o *Slow-Start*.

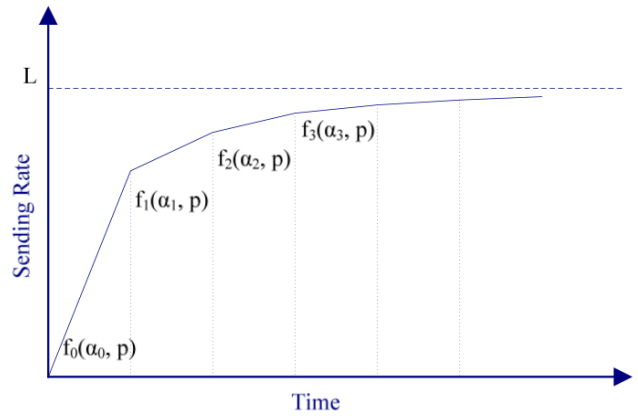


Figura 2.3: Incrementos de transferências de pacotes no UDT [4]

3 Descrição do Projecto

Como já foi dito anteriormente no capítulo 1, os *routers* possuem *buffers* de grandes dimensões para servirem as necessidades de descarte do TCP. Para além do preço de algoritmia existente em cada *router,* o preço das memórias também encarece muito o equipamento de roteamento. Quanto maior for o tamanho do *buffer*, maior o valor de compra do equipamento, pelo facto deste ser um dos componente mais dispendioso de todo o *router*. Para além do preço, o sobredimensionamento leva a que haja um maior atraso dos pacotes entre o envio e a recepção, prejudicando os *hosts*. Quando um pacote é colocado numa fila de espera de grandes dimensões, como acontece actualmente, este vai demorar muito mais tempo a ser transmitido pelo *router* em causa. Através da criação de sistemas AQM [15], foi desenvolvida a hipótese de reduzir o tamanho dos *buffers* e consequentemente uma redução no atraso dos pacotes. Isto também permitiu que o preço dos equipamentos seja reduzido.

Ao haver *buffers* muito grandes, menos pacotes são descartados pelos *routers* e assim o sistema de janela do TCP é reiniciado menos vezes. Os *buffers* têm grandes dimensões, logo, é possível armazenar uma grande quantidade de pacotes. Assim existe

uma melhor taxa de transferência por parte do protocolo e consequentemente um melhor desempenho por parte do TCP.

Através deste desenvolvimento da rede é possível perceber que esta não é feita para ter um grande desempenho de envio, mas sim permitir que o protocolo TCP tenha o melhor desempenho possível, pelo facto de haver poucas perdas de pacotes. Sendo assim, no desenho actual das redes, estas são desenhadas à volta do protocolo de transporte, em vez de ser o protocolo de transporte a ser desenhado à volta das redes. Porém, se as redes fossem desenhadas independentemente do transporte, haveria menos ganância em descobrir maneiras de transmitir melhor e mais rápido, sem haver preocupação pela rede.

Esta ganância envolve a criação de várias conexões TCP para que haja uma maior taxa de transferência, independentemente do valor das janelas de transmissão do protocolo.

Ao serem criadas redes independentes dos protocolos de transporte (UDP e TCP), existe a necessidade de criar novos protocolos de transporte que se adaptem às novas condições da rede. Estes protocolos podem estar conectados directamente com uma aplicação e ser essa mesma aplicação a criar esse mesmo protocolo de transporte, oferecendo total liberdade de implementação. Por esse motivo juntou-se a aplicação e o transporte numa só camada, formando somente a camada aplicação.

Assim sendo, esta tese está dividida em duas partes: a rede e a aplicação. Tendo como objectivo construir um protótipo de rede, que preencha a necessidade de ser independente dos protocolos de transporte, e por fim criar várias aplicações que tenham um comportamento adaptativo à rede desenvolvida.

Mais concretamente, os objectivos que se pretendem alcançar no final deste trabalho são:

1. Uma rede com *buffers* de dimensão reduzida com maior número de perdas por parte de tráfego prejudicial à rede.
2. Três aplicações com diferentes tipos de controlo de congestão (controlo de ritmo e de janela).

3.1 Rede

A maneira mais usual de estruturar uma rede é constituir três camadas: Core, Distribuição e Acesso (Figura 2.3). Esta diferenciação de camadas serve para distinguir o tráfego que passa na rede. Normalmente é criado um protocolo distinto para cada *layer*

da rede. No caso da rede desenvolvida neste documento as camadas têm o mesmo protocolo de descarte, de armazenamento e de controlo de congestão, para facilitar o desenvolvimento do algoritmo.

Uma das necessidades fundamentais para uma rede AQM [15] é a existência de um sistema que identifique, antes de ocorrer uma congestão, quando é que os *buffers* de um *router* estão a ficar saturados. Essa identificação tanto pode ser feita por estimativa da ocupação actual do *buffer*, como por cálculo de custos da entrada de pacotes, ou mesmo por sistemas de controlo. Sendo assim, a identificação antecipada serve para prejudicar aqueles que estão a transmitir com um ritmo maior do que é permitido, ou seja, aqueles que estão a causar a saturação nos *buffers*. Desta forma pretende-se reverter o estado iminente de congestão o mais rapidamente possível.

Para haver uma melhor prevenção de congestão é necessário separar as diferentes entradas de um *router* em fluxos e criar *buffers* para cada um dos fluxos. Esse é um dos pontos discutidos pelo AQM [15]: a criação de múltiplos *buffers*. Dessa forma, se houver a identificação de um fluxo prejudicial à rede, estando a transmitir mais do que seria suposto, é possível separá-lo dos restantes fluxos, entrando assim num estado de quarentena.

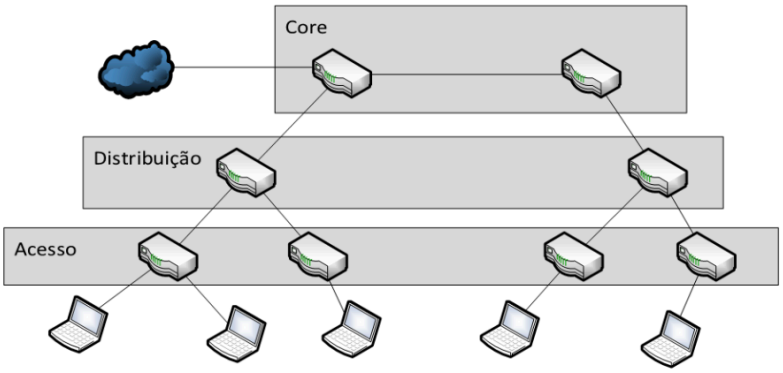


Figura 2.3: Rede hierárquica.

3.1.1 Identificação de congestão

Para identificar a congestão existente na rede foram selecionados 2 mecanismos de controlo, um mecanismo PI (*Proportional Integral Controller*) [23] e um mecanismo PID (*Proportional Integral and Differential Controller*) [24]. Foram realizados testes, num trabalho de dissertação anterior a este, a ambos os mecanismos, sendo que o que obteve melhores resultados foi o controlador PID. Por esse motivo, foi escolhido este mecanismo como o sistema detector de congestão.

$$pid(k) = pid(k - 1) + a \times e(k) - b \times e(k - 1) + c \times e(k - 2) \quad (7)$$

A função (7) é a função que define o controlador usado nesta rede. É esta a função que identifica se a rede está congestionada ou não. Sendo que $e(k) = d(k) - target$, onde $target$ é um valor de referência e $d(k)$ é o valor instantâneo de tempo de permanência na rede.

O controlador tem o seguinte comportamento: quando o valor de $d(k)$ for maior ou igual ao valor do $target$ e a variação do valor do controlador PID for crescente, é detectada congestão na rede e o sistema entra na fase do controlo de congestão.

3.1.2 Separação de fluxos

A rede desenvolvida tem como objectivo separar os tráfegos que transmitem com ritmos elevados dos tráfegos que transmitem com ritmos adequados. Este tipo de tráfego é chamado de tráfego prejudicial e é causado por *hosts* que transmitem mais dados do que é possível processar pelo *router* naquele momento. Esta separação é feita através do mecanismo PID, descrito na secção anterior. Quando é identificado o fluxo que provocou o congestionamento da rede, este fluxo é separado do *buffer* principal, com o objectivo de aliviar a fila de espera, e é colocado todo o tráfego do *host* num *buffer* separado. Quando um *host* é separado de todos os outros *host* e é colocado num *buffer* secundário, este é colocado num estado isolado, num estado de quarentena. Com isso o *router* tem menos tráfego a acumular-se na fila de espera principal.

A separação é feita assim que o controlador identificar que o fluxo em causa está a transmitir demasiado, sendo, nesse momento, reduzido o tamanho do *buffer* principal. Essa redução de tamanho é feita dependendo do número de conexões/fluxos existentes com o *router*. Quando um fluxo é colocado em quarentena, ou seja, quando é retirado do *buffer* principal, é decrementado do valor total do *buffer* principal o valor da divisão desse mesmo tamanho pelo número de conexões existentes, incluindo as ligações em quarentena.

Tomando como exemplo um aparelho da rede que possuí 4 fluxos ligados, a transmitir pacotes, sendo que um deles é colocado em quarentena, o tamanho do *buffer* de quarentena vai ter um valor igual ao tamanho da fila de espera original a dividir pelo número de fluxos em transmissão. Desta forma, a fila de espera é constituída por pequenas porções, todas iguais e de quantidade igual ao número de fluxos, que, ao reduzir o seu tamanho devido a uma quarentena, é como se isolasse uma dessas porções do *buffer* num local à parte do *buffer* principal.

3.2 Aplicações

Como referido anteriormente, quando são introduzidas perdas numa rede em congestão, o protocolo TCP é o primeiro a sofrer as consequências, pelo facto do seu algoritmo reiniciar a janela de congestão mais frequentemente. Isto deve-se ao facto do seu protocolo ser desenhado para *buffers* de grandes dimensões e para redes com poucos descartes que são criadas para "servir" as necessidades do TCP. Estes factores não contribuem para que haja harmonia entre rede e aplicação. Por esse motivo, a aplicação deve ser desenhada conforme os requerimentos da rede e não o oposto.

No caso do UDP não existe nenhum controlo de congestão como o do TCP. Os pacotes são enviados e não existe nenhum mecanismo para saber se ocorreu uma falha na rede. Em caso de congestão, não é possível informar a origem que tem de haver uma redução no ritmo de envio.

Pelos motivos dos parágrafos anteriores, este estudo procura uma harmonização entre protocolos. O objectivo deste trabalho é abrir possibilidades para novos protocolos de transporte, que tanto mantêm um controlo de congestão para com a rede, como mantêm um ritmo de envio aceitável entre aplicações.

Foram criados três tipos de aplicações que se diferenciam umas das outras pelo modo como se adaptam às necessidades da rede. Sendo que, o controlo de congestão foi desenvolvido tanto com *Rate Base* como com *Window Base*. O objectivo dos testes é ver se as aplicações têm ritmos superiores, em redes congestionadas, aos das aplicações UDP. A partir desse ponto é necessário efectuar outro tipo de testes, que não serão abordados durante este trabalho, para garantir que essas aplicações são mesmo mais eficazes que os protocolos já existentes.

3.3 Desenvolvimento

Pretende-se que a rede possua um baixo atraso na entrega dos dados aos *hosts* de destino. Isso é conseguido através da redução do tamanho dos *buffers*. Estes passam a ter uma capacidade de armazenamento de 20 Kbits, numa linha com ritmo máximo de transferência de 10 Mbps. Isto significa que o atraso máximo que um pacote vai sofrer num *buffer* destes é de 2 ms.

A rede que se pretende construir possui um trajecto máximo de 8 dispositivos ligados em linha. Este é o máximo que um pacote pode percorrer até chegar ao *host* de destino. Deste modo, é possível calcular o atraso máximo que um pacote pode sofrer quando a rede está saturada. Se para um *buffer* o atraso máximo é 2 ms, para 8 *routers*, o atraso é de 16 ms.

Para redes tão rápidas é necessário abdicar da capacidade de armazenamento de pacotes, aumentando assim as suas perdas. Tem que haver uma resposta instantânea, por parte dos algoritmos da camada de transporte, aos sinais de uma possível congestão, mas sem que haja uma perda muito acentuada na taxa de transferência. Por esse motivo foram pensadas em formas para que um algoritmo se adaptasse a essas perdas de forma mais eficiente do que o TCP.

Foram desenvolvidas aplicações que tencionavam agregar as funcionalidades do TCP, em termos de controlo de congestão através de uma janela de congestão, com a ideia de controlo de ritmo e usando o UDP. O objectivo é tornar o algoritmo de controlo de congestão o mais estável possível no ponto óptimo de transmissão. Com base em protocolos de controlo de congestão como o TCP Vegas [2] e o QUIC [3], tenciona-se obter resultados idênticos ao do TCP Vegas, mas com ritmos de transmissão mais elevados.

Pelo facto de a rede ser um estudo de uma possível rede AQM, é necessário experimentar vários tipos de algoritmos de controlo de congestão. Por esse motivo, foram feitos 3 tipos de aplicações: duas que possuem tanto janela de congestão, como possuem controlo de ritmo; e uma outra aplicação com uma janela de congestão fixa e controlo de ritmo variável.

4 Descrição e testes da Rede

A rede proposta tem como objectivo impedir tráfego, que está a saturá-la com o envio de grandes quantidades de pacotes, de intervir na sua congestão. Este mecanismo de impedimento funciona através da separação dos fluxos prejudiciais à rede, por parte de *buffers* secundários que apoiam o *buffer* principal. Se um *host* estiver a prejudicar a rede com o seu tráfego, o sistema de controlo de congestão da rede entra em acção ao colocar esse mesmo *host* em quarentena num *buffer* separado.

A Figura 4.1 serve para ilustrar um equipamento de rede, neste caso, um *router*. Como foi referido anteriormente, é em equipamentos como este que ocorre o processo de separação de tráfego, ou seja, o tráfego prejudicial é colocado em *buffers* secundários.

Os *buffers* secundários são do tamanho da fila principal a dividir pelo número de entradas do *router* em causa. Isto é, considerando que no exemplo da Figura 4.1, um *router* com uma fila de espera de 20kbits (como o caso do *buffer* usado para os testes), que possuí 4 entradas (λ_1, λ_2, λ_3, λ_4) e 1 saída (λ), sofria de uma saturação na linha, a fila de espera principal é reduzida num valor igual a cada *buffer* secundário criado. Para o

caso de ser somente um *host* a prejudicar a ligação, este seria realocado para um *buffer* secundário de 5.0 kbits e o *buffer* principal ficaria com 15 kbits de tamanho.

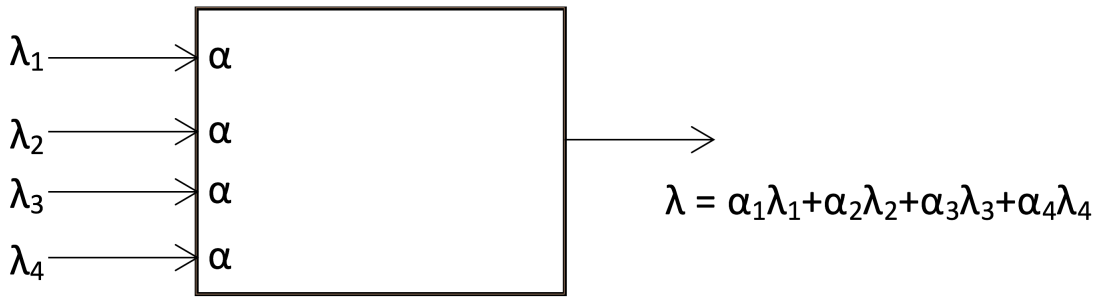


Figura 4.1: Esquema representativo do router.

Foram elaborados 3 testes na rede, com 5 tipos de aplicações cada: uma aplicação do tipo *P2P* que utiliza TCP para transmitir os seus pacotes, uma aplicação do tipo *Longo Web* com UDP como método de transporte, outra aplicação *Short Web* que também utiliza UDP, ainda outra aplicação UDP denominada *Continuous Stream* e por fim uma aplicação *On-Off Stream* também utilizando UDP. São aplicações que diferem dos 3 tipos falados anteriormente. Os 5 tipos de aplicações testados na rede servem somente como padrão, enquanto que as 3 aplicações desenvolvidas no capítulo 5 foram criadas para serem comparadas com os valores padrão, sendo estas, 3 tipos de aplicações completamente novas.

É necessário referir que o tamanho dos pacotes usados para as aplicações TCP é de 64 bits enquanto que o tamanho dos pacotes UDP é de 55 bits. Esta escolha teve em conta o tamanho mínimo que cada pacote poderia ter tendo em conta o tamanho dos cabeçalhos de cada protocolo.

Na Tabela 4.1 são indicados os valores de transmissão, usados nas aplicações de teste da rede, em relação às larguras de banda/ritmos, tempo médio por aplicação e o número médio de aplicações, dos *hosts* que transmitem a uma taxa não prejudicial, ou seja, transmitem com um ritmo que não causa congestão.

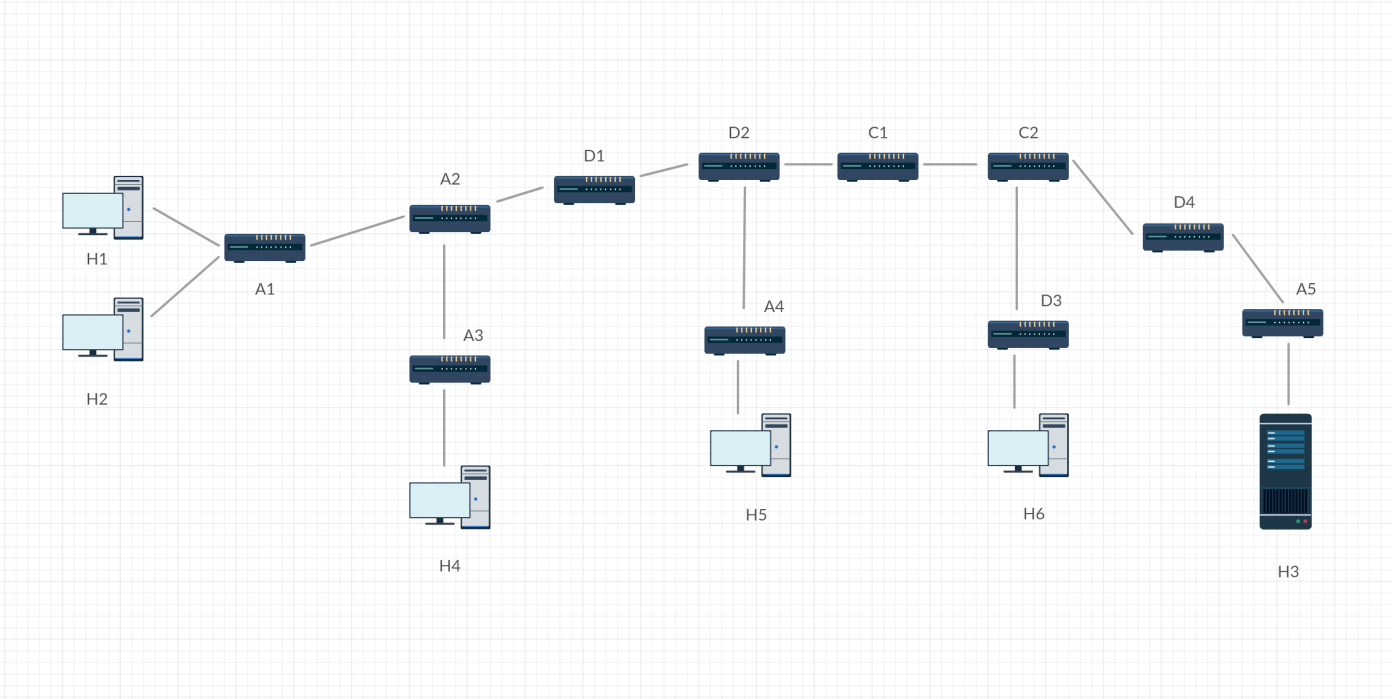
No programa de simulação, usado para testar a rede e definir o comportamento de cada tipo de tráfego, é possível criar-se automaticamente os géneros de tráfego que se pretende implementar (P2P, Long Web, Short Web, Continous Stream e On-Off Stream). Desta forma não é possível explicar em detalhe o porquê dos valores escolhidos, devido ao facto de ter sido feita uma análise por tentativa e erro até alcançar os valores pretendidos:

Aplicações	Largura de banda/ritmo de transferência	Tempo médio por aplicação	Número médio de aplicações
P2P	5 kbits	40 ms	160
Long Web	5 kbits	15 ms	250
Short Web	180 ms	3 ms	230
Continous Stream	420 ms	30 ms	155
On-Off Stream	80 ms	30 ms	178

Tabela 4.1: Valores de transmissão de cada aplicação em regime não prejudicial.

Seguidamente, a Figura 4.2 representa o esquema de rede que foi implementada para se realizar os testes. Esta rede é composta por 6 *hosts*, dos quais 5 (H1, H2, H4, H5, H6) são utilizadores comuns, que se encontram a navegar na rede com aplicações de *web browsing*, aplicações de *email*, aplicações de vídeo conferência ou videojogos. O último *host* (H6) é o servidor da rede. É este o *host* que recebe todo o tráfego dos restantes utilizadores da rede, ou seja, é o destino de todos os pacotes enviados pelas aplicações.

Os restantes dispositivos da rede são *routers* a que foram atribuídos o controlo de congestão desenvolvido ao longo deste trabalho. São estes dispositivos os responsáveis



por entregar todos os pacotes ao servidor, fora os pacotes que são descartados por motivos de controlo de congestão.

Figura 4.2: Esquema representativo da rede.

4.1 Teste 1: 2 *hosts* a transmitir abaixo do limite

Este teste é elaborado com dois *hosts* ligados ao *router* A1 que transmitem com um ritmo que não tem qualquer efeito de saturação na rede. Isto significa, nenhum dos *hosts* é colocado em quarentena por transmitir mais rápido do que devia. Como foi falado anteriormente, os *buffers* têm um tamanho máximo de 20480 bits. Neste caso, se um *host* for colocado em quarentena e estando 2 *hosts* (H1 e H2) ligados ao *buffer* principal, o *buffer* de quarentena (*buffer* secundário) terá o tamanho de 10240 bits. Após a separação dos *buffers*, a fila principal será reduzida para o tamanho de 20480 bits, menos o tamanho de cada *buffer* de quarentena, ou seja, neste caso terá um valor de 10240 bits, valor igual ao *buffer* de quarentena.

Nestas condições é de esperar que somente o *router* A1 tenha um tráfego mais acentuado, mas não em congestão, pelo facto de ser o único *router* ligado a dois dispositivos activos na rede. Sendo assim, este dispositivo A1 funciona como um filtro de congestão na rede. Os restantes dispositivos que estão localizados após o *router* A1, não tendo mais nenhum *host* a transmitir, só possuem uma linha de transmissão possível. Essa linha unitária é constituída pelo dispositivo D1, seguindo do D2, C1, C2, D4, A5 e por fim o servidor H3.

Por estes motivos, é de esperar que a fila de espera do *router* A1 vá aumentando ao longo do tempo até que atinja um valor estável, sendo este inferior ao valor máximo de armazenamento. É de esperar também que não seja colocado nenhum *host* em quarentena.

O ritmo de transferência é variável conforme o gráfico da Figura 4.3. A sua amplitude está entre os 10Mbps, que é o máximo possível na rede, até os 5Mbps, metade da velocidade máxima. Através deste gráfico é possível observar uma zona mais escurecida da imagem, sendo possível localizá-la entre os 5.5Mbps e os 7Mbps. Essa zona representa a média de ritmo de transferência que se situa por volta dos 6.5Mbps, com algumas sobrecargas aos 40 segundos e aos 100 segundos.

É importante saber o ritmo de transferência do *router* A1 para perceber se este está muito tempo a transmitir ao máximo. Se o *router* estiver a transmitir a um ritmo de

10Mbps ou próximo, por longos períodos de tempo, é provável que o *router* seguinte tenha mais probabilidade de congestionamento se introduzido uma outra fonte de tráfego. Desta forma é necessário transmitir a um ritmo médio o mais baixo possível.

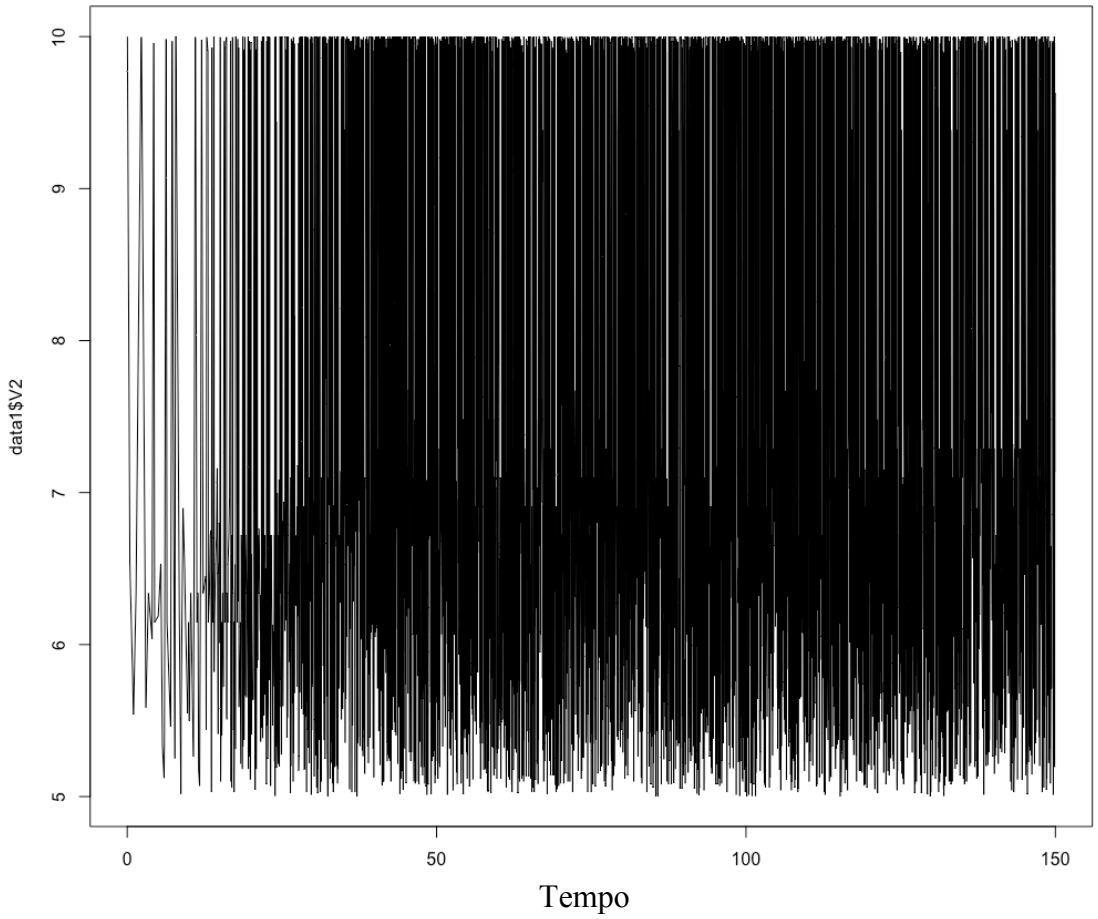


Figura 4.3: Ritmo de transferência do router A1.

Na Figura 4.4 é apresentada um gráfico de linhas que representa a ocupação da fila de espera principal no *router* A1. É possível constatar, através das zonas listadas de branco e preto, que ocorre uma grande flutuação de valores (aproximadamente entre os 12Kbits e os 6Kbits), sendo que a ocupação não atinge o seu máximo, o que significa que está a metade do tamanho máximo do *buffer*. É possível notar que a zona escurecida da figura atinge valores de 0bits, o que significa que existem períodos em que o *buffer* se encontra vazio.

Desta forma, é possível confirmar os resultados esperados. Os *hosts* não foram colocados em quarentena, o que significa que a rede se encontra no seu funcionamento normal sem ocorrer congestão, não tendo sido ultrapassado metade do valor máximo de capacidade do *buffer*, como previsto.

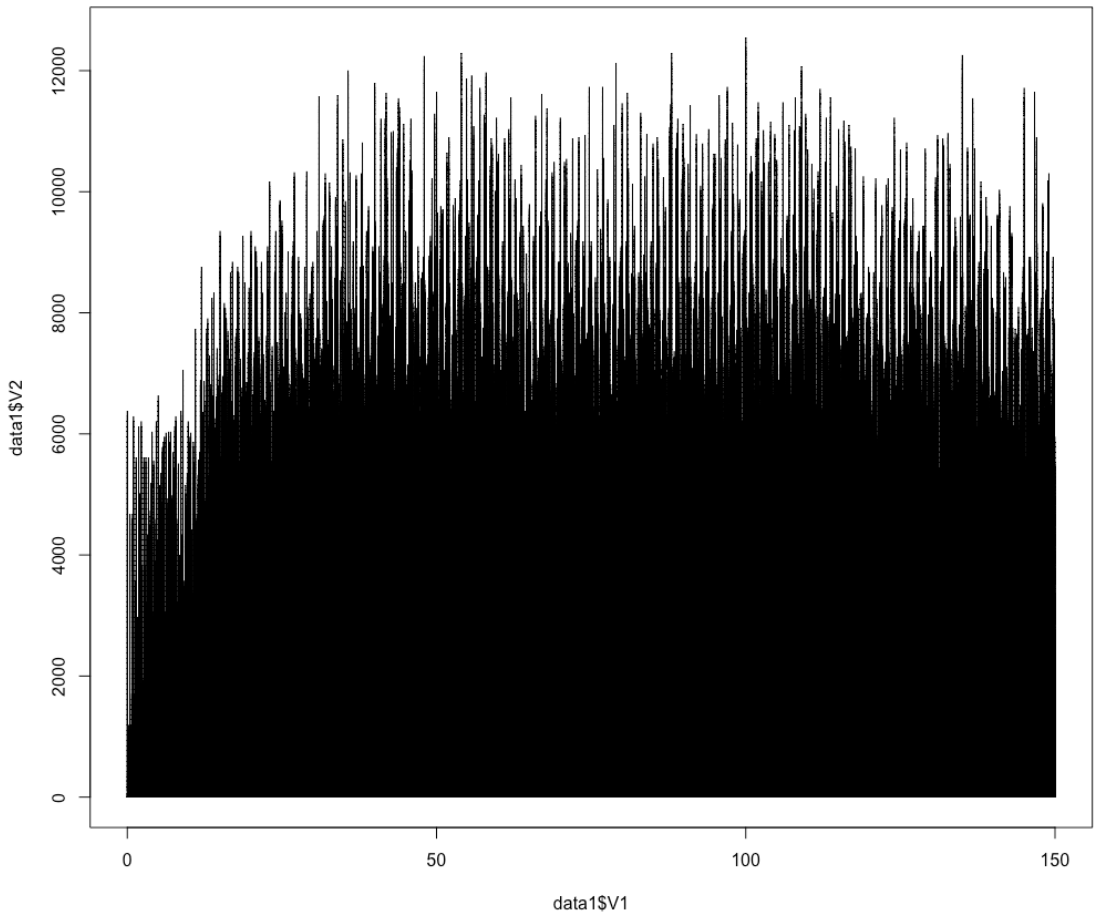


Figura 4.4: Tamanho do buffer do router A1.

A figura 4.5 mostra a fila de espera do *router* A2 que é sempre constante mantendo sempre o seu tamanho nos 120bits. Como já foi dito anteriormente, o *router* A1 é utilizado como filtro de grandes variações de tráfego, por isso mesmo, o tráfego do *router* A1 não influencia na congestão do *router* A2, mas se houvesse mais uma entrada conectada ao *router* os valores seriam diferentes. Haveria dois utilizadores a transmitir para o mesmo dispositivo, criando-se assim um estreitamento em funíl: várias entradas para uma saída. Isto deve-se ao facto de passar a haver mais do que um *host* a utilizar a memória e o ritmo

de entrada passa a ser maior do que o ritmo de saída, por esse motivo houve uma acumulação de pacotes.

A Figura 4.6 é o resultado do último *buffer* (dispositivo) antes dos pacotes serem entregues ao *host* de destino (o servidor). Nesta figura o tamanho do *buffer* deveria variar entre os 55 bytes e 64 bytes, o que não ocorre. Este efeito deveria ocorrer pelo facto dos pacotes UDP terem o tamanho de 55 bits e os pacotes TCP terem o tamanho de 64 bits. Uma possível razão para esta pequena alteração de valores esperados, deverá ser a redução de imagem, ou seja, por falta de precisão gráfica.

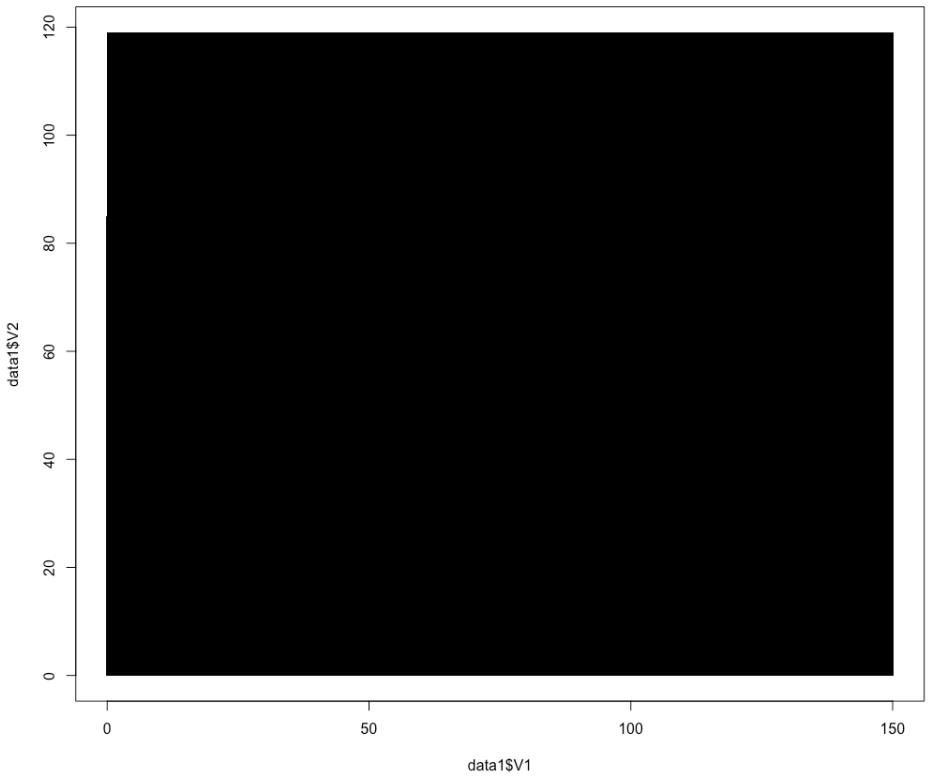


Figura 4.5: Tamanho do buffer do router A2.

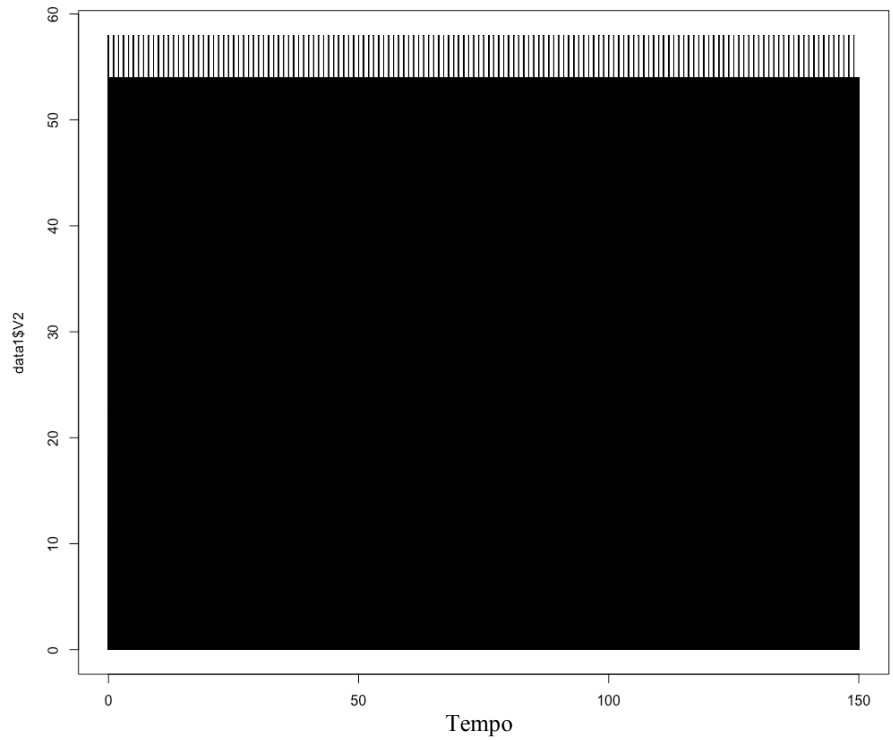


Figura 4.6: Tamanho do buffer do router A5.

4.2 Teste 2: 5 *hosts* a transmitir abaixo do limite

O teste 2 usou as mesmas definições do teste 1, em relação às taxas de transferência, seguindo os valores da Tabela 4.1, e o mesmo tamanho de pacotes (64 bits para TCP e 55 bits para o UDP). Foram adicionados mais 3 *hosts* à rede, todos a enviar pacotes ao servidor H3. Este teste tem como objectivo concluir se as condições mudam no caso de serem adicionados mais *hosts* à rede e como é que a rede reage a essa adição de *hosts*.

Neste teste todos os *hosts* estão a transmitir a um ritmo menor ao do limite de transferência na rede, mas mesmo assim a partir do *router* A2 existe algum tráfego que obriga o *buffer* a entrar em modo de congestão, para prevenir que haja uma maior saturação no *router*. Isto significa que existe a necessidade de reduzir um pouco mais o ritmo teórico óptimo de transferência, pelo facto de se aproximar demasiado do limite da rede. Sendo algo que tem de ser alterado em trabalhos futuros.

A Figura 4.7 mostra o ritmo de transferência do *router* A1. É possível reparar que o ritmo de transferência é quase sempre na ordem dos 10Mbps, com algumas descidas a

5Mbps. Assumindo que a média de valores de transferência ronda os 8.5Mbps, pelo facto de se localizar grande parte da zona escurecida do gráfico. Sendo este um valor muito superior ao valor do teste anterior.

Uma possível razão para este aumento de ritmo ter ocorrido deve-se ao facto de haver uma maior congestão na rede. Como os recursos são mais escassos, em termos de espaço nos *buffers*, não ocorre a existência de momentos mortos de transferência. Por esse motivo, os dispositivos da rede são obrigados a transmitir mais rápido para diminuir o tamanho da sua saturação.

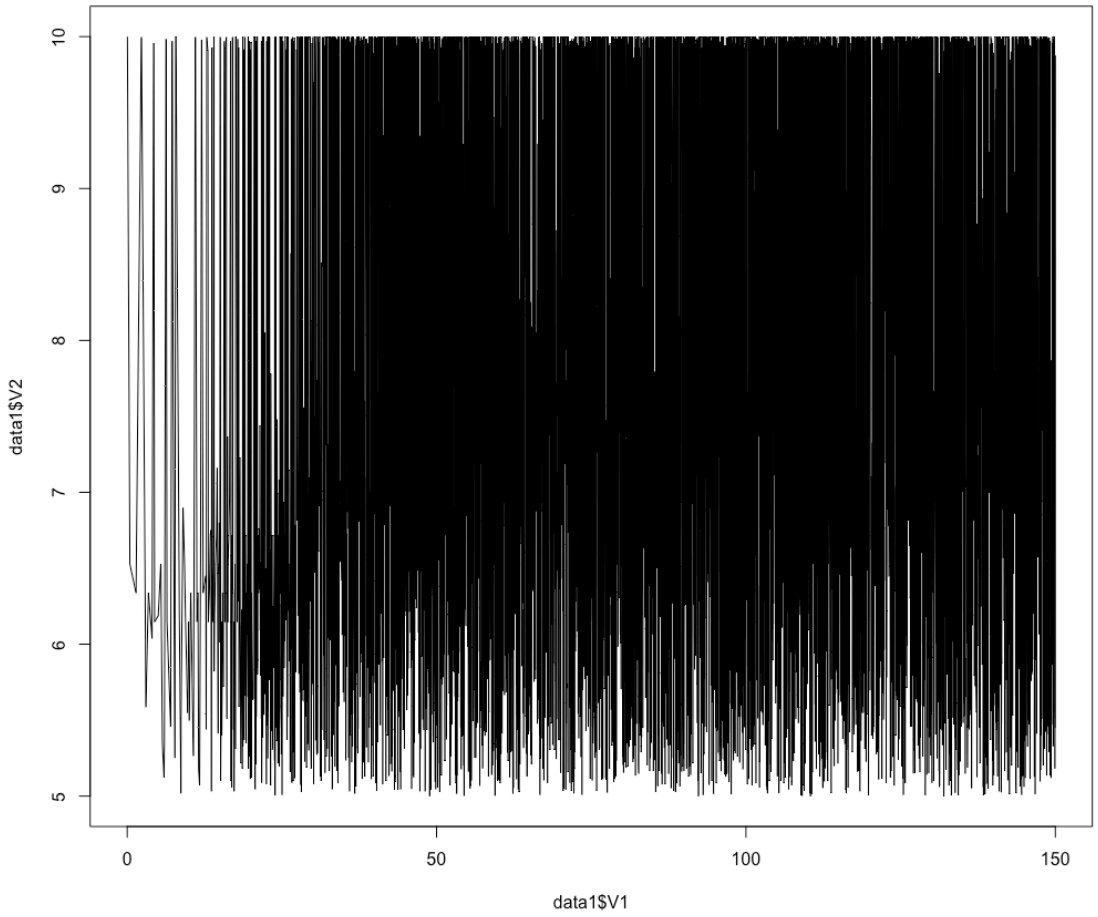


Figura 4.7: Ritmo de transferência do router A1.

A Figura 4.8 mostra a ocupação da fila de espera no *router* A1, que está conectado a dois *hosts*: o *host* H1 e o *host* H2, como no teste anterior. Aqui ainda não foi activado o mecanismo de controlo de congestão, pelo facto de não haver suficiente tráfego para justificar a separação dos fluxos. Por esse motivo, o resultado obtido é idêntico ao 1º teste efectuado.

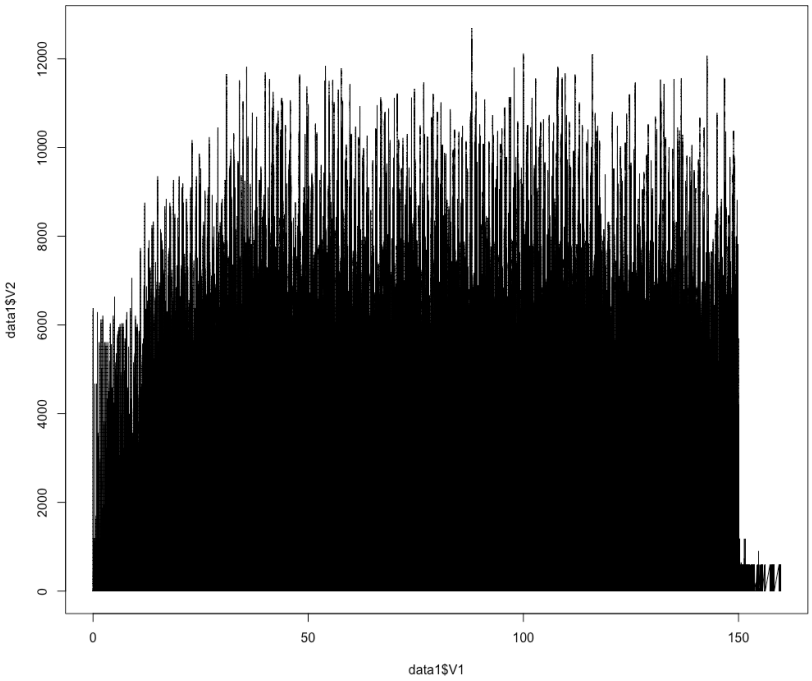


Figura 4.8: Tamanho do buffer do router A1.

A Figura 4.9 mostra a situação em que já se encontra activo o sistema de controlo de congestão do dispositivo, com o mecanismo de separação de fluxos, com início perto dos 30/40 segundos. Isto ocorre pelo facto de que haver mais um *host* a transmitir o *router* A2. Como foi adicionada uma linha de tráfego, que consiste no utilizador H4 a transmitir para o servidor H3, o *router* em causa não consegue processar o tráfego originário do *router* A1, que é constituído por dois *hosts*, mais o tráfego do *host* H4. Isto faz com que os pacotes comecem a acumular-se no *buffer* do *router* A2, fazendo com que seja isolando o tráfego do *router* A1, pelo facto de ser o dispositivo com uma maior taxa de transferência.

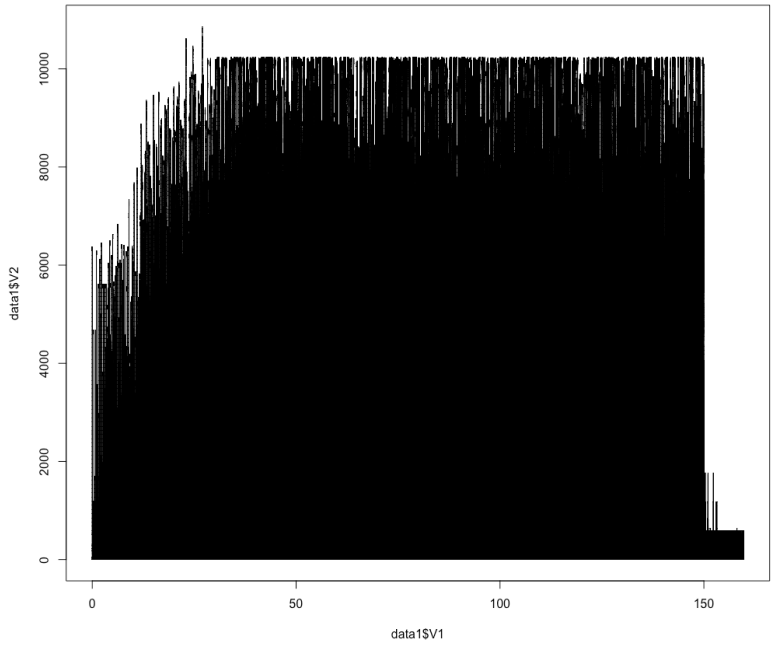


Figura 4.9: Tamanho do buffer do router A2

A Figura 4.10 representa o tamanho do *buffer* do *router* que suporta todo o tráfego da rede (*router* C2). É possível observar que existe uma grande congestão na fila de espera. Tendo sido iniciado o mecanismo de controlo de congestão, através da separação de tráfego, aos 20/30 segundos.

É possível notar que a partir dos 40 segundos, o tamanho do *buffer* do *router* C2 tem menos quantidade de tráfego na zona inferior ao gráfico do que exemplos anteriores. É possível observar pela zona branca que ocorre na zona inferior do gráfico. Isto deve-se ao facto de haver uma maior congestão no *buffer*, impedindo que o tamanho do *buffer* decremente tantas vezes até valores de 0 bits.

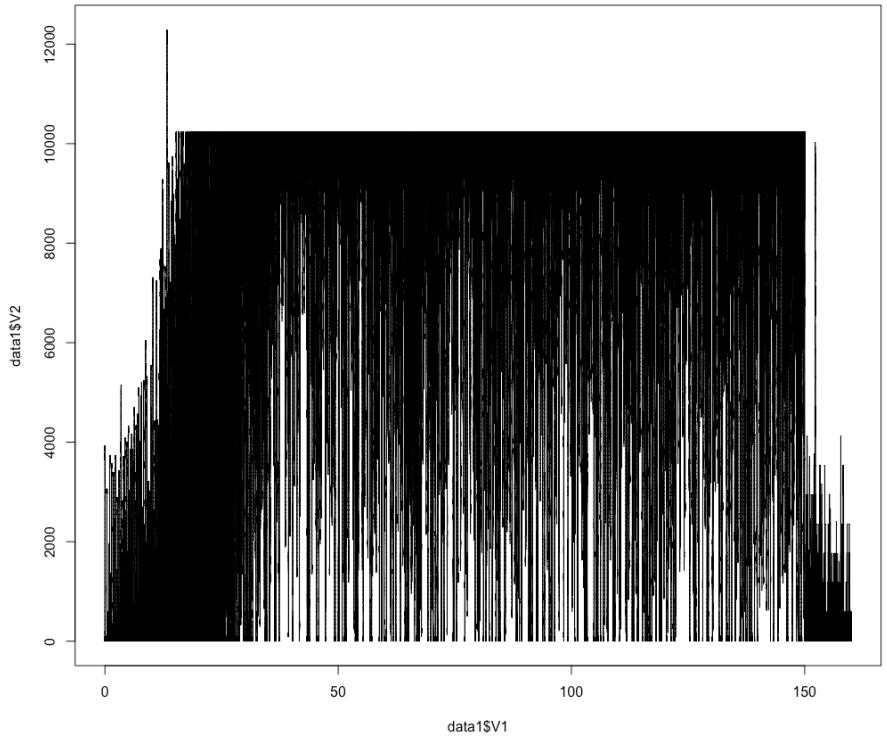


Figura 4.10: Tamanho do *buffer* do *router* C2.

Em termos de resultados, a partir do *router* C2, os valores são idênticos aos dispositivos A2 e A5 do teste 1, isto porque, neste caso, o dispositivo que serviu como filtro de tráfego foi o *router* C2. Nesse caso, o *router* A5 vai ter a mesma variação de pacotes, com uma variação entre 55 e 64 bits, como no teste 1, e o *router* A2 possuí um valor estável de tamanho do *buffer*.

Pode-se concluir que este teste obteve resultados mais severos para a rede, não estando em completa congestão, mas estando num estado de limite de transmissão, com alguns dispositivos em controlo de congestão. Sendo assim, é reforçada a mesma opinião obtida no teste anterior: que os valores poderiam ter sido reduzidos para não afectar tanto a rede quando nenhum dos *hosts* está a prejudicar a mesma.

4.3 Teste 3: 5 *hosts* a transmitir com o H4 a um ritmo elevado

O teste 3 vai ser o exemplo usado para testar aplicações que se adaptem a este tipo de rede, pelo facto de ser mais idêntico com a realidade. Este teste consiste em ter a rede anterior, sendo o *host* H4 o responsável pelo tráfego prejudicial à rede. Com este teste pretende-se perceber se existe grande diferença entre o tráfego do teste anterior e este teste.

Na Figura 4.11 é possível observar no tamanho do *buffer* do *router* A1 e reparar que não ocorrem grandes diferenças para o teste anterior. Sendo assim, é possível assumir que o tráfego que ocorre nos *routers* seguintes da rede não afecta os dispositivos anteriores, neste caso, o *router* A1.

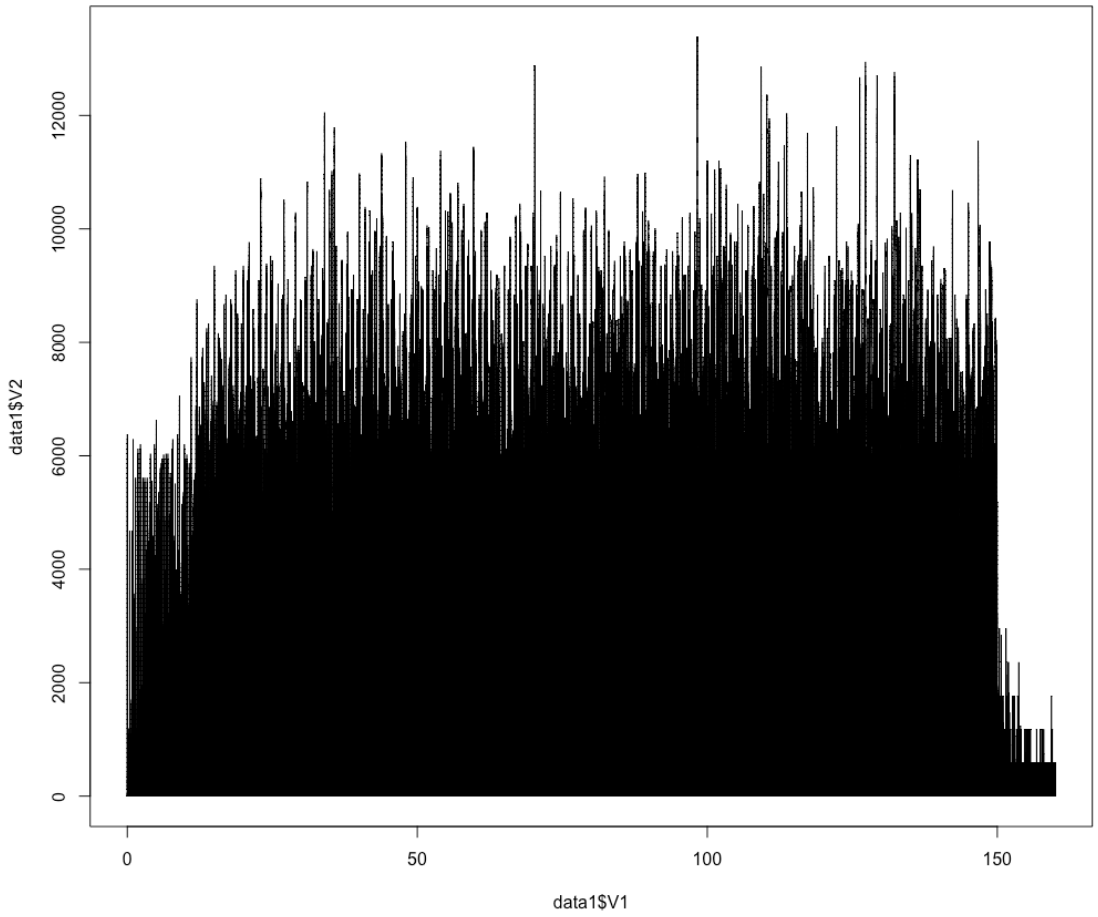


Figura 4.11: Tamanho do buffer do router A1.

Na figura 4.12 observa-se que existe uma grande congestão no *router* A2, pelo facto do tamanho dos *buffers* estarem limitados a metade da sua capacidade máxima (separação de fluxos). Esta saturação tem proporções muito superiores ao exemplo da Figura 4.9 do 2º teste. A fila, após os 20/25 segundos, torna-se constante, com um tamanho de 10120 bits, algo que no teste 2 não era tão acentuado. Os valores do tamanho do *buffer* da Figura 4.9 tinham zonas listadas onde o tamanho não atingia o seu máximo, o que significava que o *buffer* não se encontrava totalmente saturado. No caso deste teste, não existe nenhuma zona listada com valores inferiores ao tamanho, máximo. O *buffer* encontra-se completamente saturado.

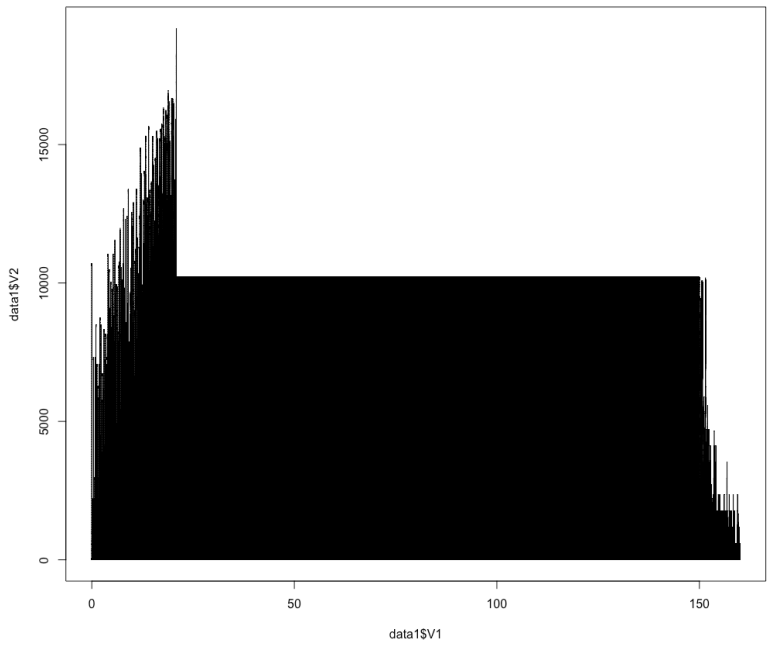


Figura 4.12: Tamanho do buffer do router A2.

Para terminar, a Figura 4.13 representa o tamanho da fila de espera no *router* C2, que se encontra em controlo de congestão, pelo facto de o tráfico ser limitado a 10120 bits, metade do valor máximo. Este resultado é idêntico ao exemplo da figura anterior, mas com a particularidade de ocorrer, neste *router*, uma maior variância durante o período de congestão, que tem início, aproximadamente, aos 20 segundos. Sendo outra das diferenças, o facto de não ter havido um grande pico de tráfego antes da isolação do fluxo prejudicial, também localizado aos 20 segundos, no momento pré-congestão.

Muito do tráfego da rede fica no A2, pelo facto do *host* prejudicial se encontrar ligado a uma das suas linhas, propagando-se pelos restantes *routers* com valores diluídos de congestão. Isto é, o *router* A2 funciona como um filtro de tráfego para os restantes *hosts* da rede, eliminando, o melhor possível, os fluxos que se encontravam a prejudicar.

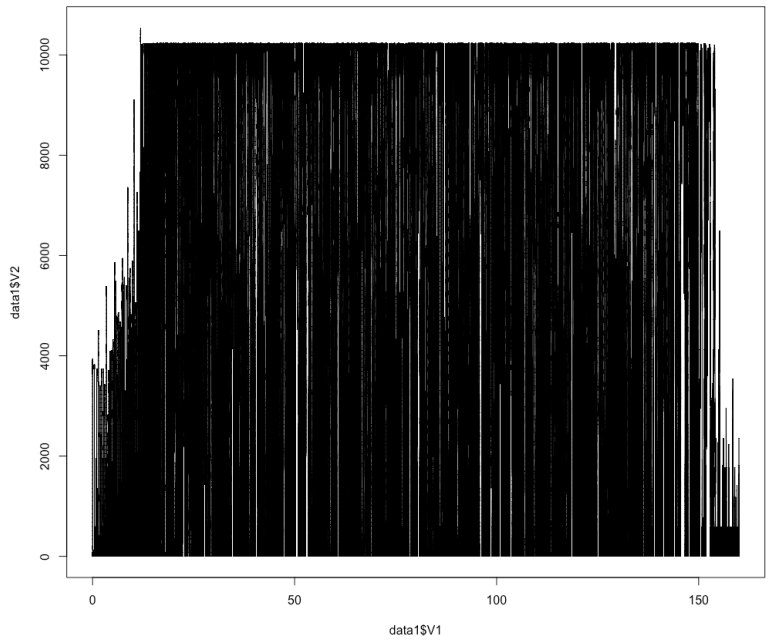


Figura 4.13: Tamanho do buffer do router C2.

4.4 Análise de resultados

Teste da rede

É de concluir que a rede, quando entra em modo de recuperação de congestão, é protegida pelo mecanismo de separação de *buffers*. Este mecanismo obtém resultados um pouco abaixo do esperado, por esse motivo, é aconselhável não alterar o tamanho do *buffer* principal quando a rede entra em modo de congestão. Assim, o *host* responsável pela congestão é isolado e mantém-se as condições normais do *router*, mas com menos um *host*. Desta forma, aumenta-se o tamanho médio do *buffer* para cada *host*, o que em teoria reduz a congestão da rede.

Existe a possibilidade ser feito um mecanismo em que o *host* prejudicial não será colocado em quarentena, mas sim num estado intermédio. Esse mecanismo poderá ser utilizado para suavizar o tráfego prejudicial. Se a rede não obtiver uma melhoria significativo em termos do seu tráfego, o responsável será colocado em quarentena num *buffer* separado.

5 Descrição e teste de aplicações

Sabendo que o TCP sofre um grande prejuízo se o tráfego for muito acentuado numa dada rede, combinou-se as vantagens da transferência do UDP com a fiabilidade do TCP. Dessa forma obtém-se um sistema que beneficia de um *throughput* mais elevado, como é o caso das aplicações UDP, e de uma maior eficácia de envio, como o exemplo do TCP. O objectivo destes testes é provar que mais pacotes são recebidos pelo *host* que possui a aplicação mista, que combina o protocolo TCP com o protocolo UDP, do que somente pelo protocolo UDP, mantendo uma certa fiabilidade de chegada de pacotes ao *host* de destino. Essa fiabilidade é definida pela chegada dos pacotes na ordem em que foi enviada. Se houver falhas de pacotes, a situação é informada à origem.

São apresentados três tipos de protocolos desenvolvidos, com o intuito de criar diferentes estratégias de atacar o mesmo problema. São criadas 3 aplicações com protocolos de controlo de congestão diferentes. Todos estes possuem os mesmos cabeçalhos de envio e todos são constituídos por controlos de congestão baseados em ritmo de envio, alguns combinando com envio em janela.

O cabeçalho é constituído por três campos, como é indicado na Tabela 5.1, juntamente com o respectivo tamanho, em bytes: o tamanho total da janela que foi enviada; o número do pacote em relação à janela enviada e o ID do pacote em relação ao total de pacotes que já foram enviados e recebidos com sucesso. Em todos os testes o protocolo foi posto à prova por um *flow* UDP com um ritmo de 15 ms e pacotes de 55 bits, de forma a assemelhar-se ao ritmo de uma aplicação com um uso excessivo na rede.

Tabela 5.1: Estrutura do cabeçalho dos pacotes enviados.

Tamanho da janela	Número do pacote	ID do pacote
4 bytes	4 bytes	4 bytes

Estas aplicações foram testadas com a rede em modo de congestão com o *host* H4 (*Figura 4.2*) a prejudicar o tráfego e a congestionar a rede. O tráfego TCP (P2P) possui um ritmo de 5Kbps, tal como o tráfego *Long Web*. Para o *Short Web*, o *Stream* contínuo e o *Stream On*-Off, os ritmos de transferência foram, respectivamente, 180ms, 420ms, 80ms. Estes valores são os valores usados nos testes do capítulo 4, *Tabela 4.1*, em rede não congestionada.

5.1 Conceito de Janela

Este conceito é implementado em todos os protocolos desenvolvidos, como já foi referido anteriormente. Este conceito caracteriza-se por possuir um tamanho fixo, ou variável, de envio, à medida que vai sendo confirmado na sua totalidade pela chegada de um *Acknowledge*, o seu tamanho pode aumentar ou manter-se com o valor da janela anteriormente enviada. Consequentemente, com a chegada de pacotes *Not Acknowledge* o tamanho diminui, ou pode manter-se igual. As condições da janela são definidas dependendo do protocolo em questão, por esse motivo, serão detalhadas as condições quando forem explicadas as aplicações mais à frente.

O que torna este conceito diferente do TCP e do UDP é que, enquanto que o TCP necessita de confirmar todos os pacotes da janela, que chegam ao destino, no caso das aplicações desenvolvidas neste projecto, os pacotes são todos confirmados quando a janela inteira é recebida com sucesso pelo destino. Pode-se comparar à situação de ter um pacote grande, dividido em partes muito mais pequenas e enviar essas partes com um certo ritmo, também ajustável (mais detalhe na próxima secção), que, ao chegar todos esses pedaços, o destino envia a confirmação à origem de que o pacote maior foi todo enviado.

O objectivo deste mecanismo é fazer com que, o tráfego que estamos a gerar, adapte-se ao tráfego da rede em causa e assim controle como vai ser a sua resposta ao tráfego

actual da rede. Para tornar o tráfego o mais responsivo possível é necessário alterar também o ritmo de envio de pacotes.

5.2 Controlo de Dados

Como já foi referido anteriormente, o cabeçalho dos pacotes é constituído por três campos. Desses campos, o tamanho da janela e o número do pacote são utilizados para verificar se houve algum pacote perdido dentro da janela enviada. Se o número do pacote recebido for igual ao número do pacote esperado tudo correu como devia e não houve detecção de falhas. Por outro lado, se o pacote esperado for diferente do número do pacote recebido, então houve uma falha na transmissão. Quando o número esperado é igual ao tamanho da janela enviada, o valor do pacote esperado volta a ser 1 e é enviada uma confirmação à origem de que a janela chegou sem falhas. Por exemplo, se é transmitida uma janela de tamanho 10, o destino já sabe que se receber o pacote número 10 é necessário confirmar a chegada de toda a janela ao enviar um *Acknowledge*. Se todos os pacotes esperados estiverem a chegar correctamente ao destino, então não é enviada nenhuma confirmação à origem de que não ocorreram falhas durante o envio dos pacotes. Por outro lado, se o destino estiver à espera do pacote número 3 e recebe o pacote número 4, é enviado um *Not Acknowledge* para informar que houve a falha de um pacote de dados dentro daquela janela (o pacote número 3).

Sempre que é enviado um pacote, seja *Acknowledge*, *Not Acknowledge* ou de dados, são acionados *timers* que permitem a retransmissão sempre que não houver resposta ao envio. Estes *timers* são configurados com o tempo médio de resposta, sendo que, somente o primeiro pacote a ser enviado tem um *timer* com um valor fixo. Depois de ser recebido o primeiro pacote, é estabelecido o tempo ideal para o seguinte *timer*.

No caso de falha de pacote, é memorizado o ID do último pacote recebido correctamente dentro da ordem esperada. Isto tem o objectivo de, quando é enviado um *NACK* à origem, o ID do pacote anterior ser enviado conjuntamente com o *NACK*, assim, a origem é informada que tem que enviar o pacote com ID uma unidade a cima do que está no cabeçalho do *NACK*.

Desta forma, e com poucos dados no cabeçalho, é possível detectar falhas na transmissão de pacotes e assegurar a fiabilidade do envio de pacotes como no TCP, mantendo o *throughput* de envio existente no protocolo UDP. Esta combinação de controlos de congestão permite que não haja uma queda de ritmo de envio tão acentuada, sempre que

ocorre uma falha, aproximando-se assim, do protocolo UDP. A semelhança com o protocolo TCP dá-se ao facto de haver uma confirmação, positiva ou negativa, quanto ao estado e à ordem de envio. Isto tem o objectivo de tornar a confirmação de pacotes mais flexível e mais leve, em vez de ser tão estática como no TCP (um pacote de cada vez). No caso destas aplicações criadas é confirmada a chegada de um conjunto de pacotes esperados pelo destino.

No próximo ponto será explicado o conceito de ajuste de ritmo, implementado nas aplicações desenvolvidas.

5.3 Conceito de Ritmo

O maior objectivo deste estudo é tornar o tráfego o mais flexível possível a prováveis congestionamentos na rede, de maneira a que não haja tantas perdas por parte da rede. Sendo esta uma rede que compromete os *hosts* que se encontram a enviar a uma taxa muito superior à cedida. Por este motivo, os *ACKs* e os *NACKs* não servem somente para assegurar o envio de pacotes pela ordem correcta, nem somente para alterar o tamanho da janela de envio, mas também para ser calculado o tempo de transmissão de cada pacote ou o *Round Trip Time* (RTT). É o conjunto das partes (fazer o controlo de chegada e o calculo do RTT) de cada aplicação que torna o tráfego flexível às congestões na rede.

O ajuste de ritmo funciona com base no primeiro pacote da janela, até à recepção do *ACK* da janela. Este tempo é cronometrado e depois é dividido pelo número de pacotes que constituem a janela. Assim é obtido o ritmo de transferência de cada pacote existente na janela seguinte. Em teoria, cada janela tem um ritmo de transferência específico para a mesma, alterando-se no final de cada transmissão. Esta propriedade é variável conforme o estado da rede e o estado da janela. Quando os dispositivos da rede estão saturados, mais tempo os pacotes vão permanecer nos *buffers*, aumentado assim o tempo transmissão da sua janela e consequentemente o ritmo de envio da janela seguinte.

A situação de cada aplicação será estudada mais detalhadamente quando for abordada na secção seguinte.

5.4 Caractrização dos protocolos

Todos os protocolos possuem ritmos de envio variáveis, sendo que alguns possuem janela variável e outros somente o ritmo. Este ritmo varia consoante o tempo de transmissão ao longo da rede.

Com a fixação da janela num dado valor temos como objectivo perceber o que ocorre se somente for alterado o ritmo. Ou seja, consoante o estado da rede, a mesma janela, com o mesmo exacto tamanho, vai demorar mais ou menos tempo a ser transmitida pela rede.

O primeiro protocolo teve como objectivo experimentar o que aconteceria com a rede se possuíssemos um tráfego prejudicial à rede. Tendo sido esperado um valor menos bom do que os restantes testes, pelo facto de a janela, por norma, estabilizar num valor elevado. Sendo que essas espectativas não se realizaram. Posteriormente a janela foi fixada num valor de 5 pacotes por janela, em que, com este teste, foram obtidos resultados mais interessantes para prosseguir com pesquisas mais detalhadas em trabalhos futuros. Sendo esperados valores piores do que os obtidos. Por fim, foi feito um teste em que seria tudo livre, tanto a janela, de inicio ao fim do teste, como o ritmo de envio. Desta forma seria interessante perceber se o protocolo se adaptava bem à rede. Tendo sido esperado que este obtive-se os melhores valores em termos de transferência, não correspondendo à realidade. Ao invés, foi o teste com os piores valores de transferência.

Para terminar, todas as aplicações têm o nome de FCCP (*Fast Congestion Control Protocol*). Isto deve-se ao facto de o estudo ter como objectivo desenvolver um controlo de congestão, integrado num protocolo de Transporte/Aplicação, que seja o mais rápido possível em redes que são muito rigorosas com tráfego que se encontra a prejudicar os restantes utilizadores (é o caso da rede do capítulo anterior).

5.4.1 Controlo de congestão por Janela variável com estabilização

Este protocolo tem a particularidade de começar como um sistema adaptável e estabilizar num valor que será o melhor para a rede. É possível observar, na Figura 5.1, um exemplo genérico de como é que a janela de transferência aumenta, reage e estabiliza. Para iniciar, este protocolo começa com a janela a uma unidade. Ao receber o primeiro *Acknowledge*, a janela duplica o seu valor. Este mecanismo vai duplicando o valor da janela sempre que recebe um *Acknowledge* até receber o primeiro *Not Acknowledge*. Nesse momento, o valor da janela é decrementado, a uma unidade e sempre que é recebido um *NACK*, até o valor da janela ser igual ao valor do cabeçalho do primeiro *NACK*

recebido. Esse valor é a subtracção do tamanho da janela no momento da falha, pelo número de pacotes que faltavam ser entregues ao destino.

Desde o momento que a janela chega ao valor definido pelo primeiro *NACK*, o tamanho vai ser sempre o valor que foi recebido. Desta forma a rede vai estabilizar o valor da janela no mais justo para ela e assim a única mutação que vai ocorrer, no protocolo, será somente o ritmo de envio da janela de pacotes.

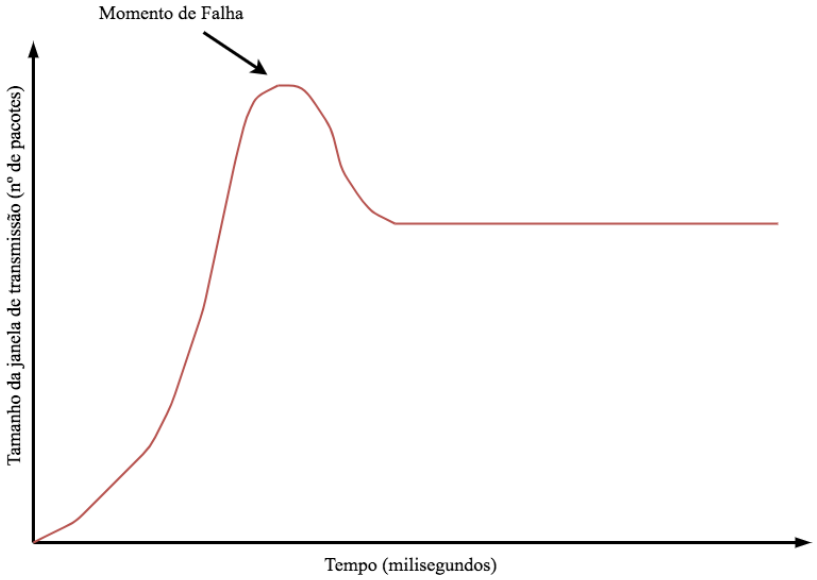


Figura 5.1: Exemplo genérico do tamanho da janela durante o 1º teste.

A única preocupação que foi colocada para este protocolo é a grande dimensão que a janela pode atingir e, sendo só o ritmo o elemento de adaptação, qual o efeito desse elemento para tornar a aplicação mais flexível aos congestionamentos na rede.

Foi elaborado um teste, com a rede em congestão, devido ao *host* H4 e ao mesmo tempo colocado uma aplicação UDP a transmitir a 15 ms.

Tabela 5.1: Demonstração de valores da primeira aplicação em teste.

	H1	H3	%
FCCP	14498	13702	0,945095875
UDP	10001	8471	0,847015298

Como é possível observar, existe um grande envio de pacotes por parte do *host* H1, cerca de 14498 pacotes foram enviados pelo protocolo FCCP, face aos 10001 pacotes enviados pela *stream* UDP. Ou seja, neste caso o FCCP obteve um ritmo de transmissão superior ao ritmo do UDP. Dos 14498 pacotes enviados pelo FCCP, 13702 pacotes foram recebidos com sucesso, o que dá uma taxa de sucesso cerca de 94.5% de envio, enquanto o UDP teve uma taxa de sucesso cerca de 84.7%.

Para além de comparar o número de pacotes enviados e recebidos pelo protocolo, é necessário referir que, dos 13702, foram recebidos 9702 dentro de ordem, ou seja, dos 14498 pacotes enviados, 9702 pacotes foram aproveitados pelo controlo de dados do protocolo FCCP. São estes 9702 pacotes que podem ser ordenados de modo a que não ocorra falhas em ficheiros ou documentos. Deste modo, o FCCP enviou mais dados que o *stream* UDP e ainda manteve um controlo de fiabilidade dos dados recebidos. Este valor foi obtido através de um ficheiro gerado durante o decorrer dos testes em que indicava o número dos pacotes enviados e o número dos pacotes recebidos, sendo que, o ultimo pacote tenha o número de referência 9702º. Sendo que, estes números não podem ser saltados. Sempre que ocorre uma falha na recepção, o número do pacote volta ao primeiro pacote que falhou e repete o processo com uma janela de envio diferente.

Sabendo que os fluxos TCP têm um ritmo de transferência de 5,00 kbps, ao calcular o ritmo de transferência do FCCP (pacotes enviados por tempo de envio) chegamos ao valor de 5,30 kbps, ou seja, somente por adaptar-se à rede que se encontrava, o protocolo desenvolvido foi capaz de aproximar-se do ritmo estabelecido como o "correcto" para a rede. É de notar que nenhum valor foi estabelecido de antemão, somente o tamanho dos pacotes é indicado. Através da adaptação do FCCP foi possível chegar ao ritmo desejado para a rede com taxas de sucesso superiores.

5.4.2 Controlo de congestão por Janela estável

Neste teste, o protocolo teve uma janela estável de 5 unidades, sem aumentos nem decrementos. A única alteração que ocorreu durante todo o processo de adaptação na rede à congestão elevada foi através do ritmo de envio.

Este valor de 5 pacotes por janela foi estimado através da média, em caso de congestão na rede, em que o 3º teste (com ritmo e janela variável) estabilizava. Para tal, foram efectuados, primeiramente, *dummy tests* com janela e ritmo variável, de forma a obter essa mesma média de pacotes por janela.

Como a janela não tem tendência a estabilizar em valores com grandes dimensões, como no caso do teste 5.4.1, existe uma menor quantidade de pacotes para confirmar por

Acknowledge. Desta forma, menos pacotes são enviados, mas mais pacotes chegam na ordem correcta, daqueles que foram enviados. É de notar que a origem tem de esperar menos tempo pela resposta da confirmação, comparando com o TCP que tem de esperar por cada pacote enviado.

Os testes elaborados a este protocolo estão apresentados na tabela em baixo. Esta tabela é idêntica à do teste anterior.

	H1	H3	%
FCCP	9990	9481	0,949049049
UDP	10001	8504	0,850314969

Tabela 5.2: Demonstração de valores da segunda aplicação em teste.

Neste teste é possível notar-se que a velocidade de transmissão não é tão elevada como no teste anterior, pelo facto de terem sido enviados somente 9990 pacotes, comparando com os 14498 pacotes enviados no 1º teste. Sendo que existe uma superioridade em relação à eficácia do teste anterior. A diferença é decimal, mas é algo que pode justificar futuros testes para perceber qual é o protocolo que melhor se encaixa numa rede tão reactiva como a que foi criada ao longo deste projecto. Uma das possíveis razões para esta mesma maior eficácia, pode dever-se ao facto de transmitir menos pacotes que o caso 5.4.1, sendo necessário efectuar alguns testes futuros para concluir o contrário.

No caso deste teste, 7945 pacotes chegaram dentro da ordem, face aos 9702 pacotes do teste anterior. É possível concluir que o primeiro teste enviou mais pacotes dentro de ordem do que o segundo, mas vendo por outro lado, 7945 pacotes em 9990 são 79 pacotes em 100, comparando com os 9702 pacotes em 14498 do teste 5.4.1 que dá, aproximadamente 67 pacotes em 100. Sendo assim, o segundo teste tem uma maior probabilidade de entregar pacotes em ordem do que o primeiro caso, mesmo tendo um ritmo de transferência inferior. Desta forma, obtém-se dois protocolos com dois fins diferentes: enquanto um tem um ritmo mais elevado, mas uma taxa de fiabilidade menor, o outro possuí um ritmo inferior de transferência, mas uma maior taxa de fiabilidade.

A largura de banda desta aplicação é de 3,66 kbps, um pouco inferior à largura de banda do FCCP anterior. Isto deve-se ao facto de haver uma adaptação no ritmo de transferência. Por haver uma congestão na rede, o protocolo adapta-se melhor ao facto da janela ser, na maior parte das vezes, menor que a janela da aplicação anterior.

5.4.3 Controlo de congestão por Janela variável sem estabilização

Para haver uma aplicação com liberdade total de movimento na rede, foi feito um teste em que a janela é variável, como na primeira situação, mas sem a estabilização final. Desta forma, a janela está sempre a alterar o seu valor, tal como o ritmo de transferência.

O objectivo deste teste é ter um modelo de comparação entre aplicações, para ser mais fácil escolher qual a melhor opção a ser implementada numa aplicação real, para este tipo de rede.

O teste começa por possuir uma janela de 1 pacote, como no primeiro teste, aumentando o seu valor para o dobro do valor anterior sempre que recebe um *Acknowledge*. No momento em que recebe o primeiro *Not Acknowledge*, a janela reduz o seu valor para o indicado no cabeçalho do *NACK*. A partir desse momento os ajustes à janela vão ser unitários, ou seja, sempre que a aplicação receber um *ACK* a janela aumenta em uma unidade o seu valor, e sempre que receber um *NACK*, o valor da janela é decrementado em uma unidade. Havia a possibilidade de ter incrementos e decrementos dependendo do valor que se recebe no *ACK*, mas havia problemas técnicos e de estabilidade que tornava o protocolo mais complicado. Por esse motivo foi decidido tornar mais simples, por enquanto, abrindo a possibilidade futura de fazer uma aplicação com um protocolo desse género, para perceber qual o beneficio à rede e aos utilizadores.

Na tabela seguinte estão indicados os valores obtidos no teste efectuado ao protocolo em questão.

Tabela 5.3: Demonstração de valores da terceira aplicação em teste.

	H1	H3	%
FCCP	13072	11235	0,859470624
UDP	10001	8507	0,850614939

É possível de observar que este teste é, de todos, o que obteve menos eficácia de envio. Sendo que, foram enviados mais pacotes do que a aplicação UDP que competia pela rede juntamente com o FCCP. Podemos concluir que, em termos de eficácia, não é uma grande mais valia que o UDP, mas em termos de velocidade, é tão capaz como a 1ª aplicação de teste.

Dos 13072 pacotes enviados, somente 5178 pacotes foram recebidos dentro de ordem e aproveitados pelo controlo de dados do FCCP. Isto significa que obteve o pior desempenho de todas as aplicações testadas. Com 39 pacotes recebidos correctamente em 100, face aos 67 pacotes do 1º teste e 79 pacotes do 2º teste. Isto significa que este teste obteve os piores resultados em termos de fiabilidade de pacotes.

De todos os testes, este foi o que menor eficácia de envio obteve. Sendo que, a velocidade de transmissão tenha sido idêntica ao 1º teste, com uma perda de fiabilidade de 30%.

A largura de banda foi de 4,80 kbps, o que significa que possui o valor mais próximo definido para o TCP, de todas as aplicações, mas com a eficácia mais baixa em todos os aspectos possíveis. Pode-se dizer que, sendo esta aplicação a mais idêntica ao TCP, as outras, que obtiveram melhores valores nos testes, são possíveis substitutas aos protocolos correntemente usados nas redes.

5.5 Análise de Resultados

Através destes três testes podemos perceber o que favorece a aplicação numa rede em que, quem está a cumprir os limites de transmissão, não sofre tantas perdas e quem não cumpre, é punido com uma maior taxa de perdas em pacotes enviados. O controlo de congestão que mais se adapta à rede é o controlo de ritmo, pelo facto dos primeiros dois testes terem a janela estabilizada num dado valor e terem sido os testes com melhores prestações em termos de ritmos de transferência e de eficácia.

Através destes testes é possível distinguir dois tipos de aplicações, de entre as 3 aplicações desenvolvidas: uma que possuí menos eficácia, mas que transmite a um ritmo superior e outra que possuí uma maior eficácia, mas transmite a um ritmo inferior. A escolha da aplicação recaí para o que se pretende da rede. Por exemplo: para aplicações que necessitem de uma maior eficácia de transmissão, independentemente do seu ritmo (E-mail, Chats, *Homebanking*, *Web browsing*, etc.) o protocolo do 2º teste será o mais indicado; enquanto que, para aplicações de *Stream*, Voice Over IP e Video conferências, que necessita de um ritmo de envio superior, mas não de fiabilidade, será melhor a 1ª aplicação desenvolvida.

O protocolo que mais se destacou foi o segundo protocolo, pelo facto de ter sido mais eficaz que o primeiro protocolo, tendo enviado quase o mesmo número de bits na ordem esperada do que o primeiro, mas mantendo um ritmo de transferência menor.

Teste da 1ª aplicação

Na primeira aplicação criada é possível observar um melhoramento de quase 10% na eficácia de recepção de pacotes. Pode-se concluir que de todas as aplicações foi a melhor em termos de ritmo de transferência. Isto deve-se ao facto de a janela ser escolhida quando a rede ainda não está sobrelotada. Como a janela de transferência é escolhida de acordo com o que a rede é capaz de suportar, no inicio dos testes, o valor é o óptimo para aquelas condições. O que varia a seguir é o ritmo de transferência, que vai depender da congestão da rede.

Ainda é necessário realizar-se testes mais intensivos para se perceber quais os resultados quando são iniciados os testes em plena congestão da rede. Uma das possibilidades que se deve testar é se a janela mantém-se pequena durante todo o teste e se os ajustes do ritmo poderão compensar o valor da janela, aumentando o seu tamanho.

Teste da 2ª aplicação

Na segunda aplicação é possível reparar que houve um decréscimo na velocidade de transmissão, mas que houve uma maior eficácia na recepção dos pacotes. Para perceber se a maior eficiência de transferência deve-se ao facto de haver menos pacotes enviados, ou deve-se simplesmente ao facto de o algoritmo ser melhor em termos de eficácia é necessário efectuar-se mais testes.

Conclui-se então que esta aplicação possui uma velocidade de transferência menor que a 1ª aplicação, mas uma eficácia superior, tanto em pacotes recebidos com sucesso como em pacotes recebidos dentro de ordem.

Teste da 3ª aplicação

O 3º protocolo testado foi o protocolo com os piores valores em termos de velocidade de transferência e em termos de eficácia. Por este motivo pode-se concluir que o melhor para este tipo de redes é um mecanismo de ajuste de ritmo. Isto deve-se pelo facto de que os protocolos, que possuem somente ajuste no tempo, são os protocolos que se saíram melhor nos testes realizados.

É necessário realizar mais testes para se poder concluir que o melhor caminho para uma rede destas é a adaptação do ritmo e não outro tipo de adaptações.

Conclui-se que protocolos de janela idênticos ao TCP não são a melhor solução para redes com muitas perdas, mas é melhor optar por controlos de congestão com base no ritmo ou no tempo.

Para termos práticos, foi desenhado um cenário possível para a utilização completa dos 3 protocolos numa rede com controlo de congestão AIMD. Para facilitar a explicação, serão denominados o 1º protocolo, o 2º protocolo e o 3º protocolo, respectivamente, por protocolo A, B e C.

Para uma utilização mais fiável da aplicação na rede, é necessário que o maior número de pacotes chegue o mais ordenado possível ao destino, podendo ser abdicadas taxas de transferências elevadas por uma maior fiabilidade. Tendo sido, esse objectivo comprido por parte do protocolo B. Assim sendo, numa rede deste género, o protocolo B será o responsável por entregar ficheiros e documentos sem falhas, um pouco ao estilo do que o protocolo TCP tem feito ao longo dos anos.

Por outro lado, em relação a ligações *Stream* (por exemplo), em que a fiabilidade não é tão requerida, o protocolo A poderá ser a alternativa ao UDP. Pelo facto de possuir um grande ritmo de transferência juntamente com uma taxa de sucesso elevada. Estes são factores que tornam este protocolo muito útil para aplicações de vídeo *online*, seja *live*, ou não, fazendo deste protocolo um mecanismo muito útil para quando a rede se encontra em condições aceitáveis de transmissão.

Em caso de congestionamento na rede, poderá ser acionado o protocolo C para fazer face às novas exigências. Sendo reduzida a velocidade de transmissão do protocolo A e a sua eficácia. Desta forma, a aplicação continuaria a correr, mas em condições satisfatórias para o uso da aplicação.

6 Conclusões

Este trabalho teve o objectivo de elaborar um algoritmo de congestão de rede de maneira a tornar mais justo a utilização da rede entre os *hosts* que a utilizam. Após o desenvolvimento da rede foram criadas três aplicações que se adaptassem ao tráfego em congestionamento para ver qual a melhor caminho a seguir.

É de concluir que a rede, ao isolar os tráfegos correctamente, está a tornar mais rígida a sua resposta aos congestionamentos e a utilizadores abusivos. É o dever das aplicações adaptarem-se ao estado do congestionamento da rede. Assim, existem regras que cada *host* deve cumprir para não ser isolado do *buffer* principal.

Para demostrar essa adaptação foram criadas 3 aplicações de exemplo com diferentes tipos de controlo de congestão, sendo que a que obteve melhores ressoltados foi a aplicação com estabilização de janela variável. Foi possível concluir que será necessário, para futuras aplicações, controlos de congestão com janela variável com estabilização e variação de ritmo para uma melhor performance. ***Problemas ocorridos***

Durante o processo deste trabalho ocorreu alguns imprevistos que não permitiram um maior desenvolvimento das aplicações e dos seus testes por de falta de tempo.

Um desses problemas foi a necessidade de elaborar uma rede para que as aplicações corressem. Não havendo muito tempo para elaborar muitos testes, nem à rede, nem às diferentes aplicações elaboradas no decorrer do projecto. Desta forma, este documento torna-se um documento com um âmbito mais introdutório às diferentes soluções existentes.

Outro problema que ocorreu foi o ritmo com que os elementos da rede transmitiam. No inicio foi estabelecido que o ritmo que de transmissão seria 250 Mbps, mas devido à alta exigência de processamento e por ser necessário utilizar um servidor por

Secure Shell, foi decidido baixar para valores a rondar os 10 Mbps para realizar os testes localmente. Esta redução deve-se também ao facto de cada teste (com o ritmo a 250 Mbps) rondar as 7/8 horas de processamento (com falhas na rede em vários testes), em vez de 1 hora de teste a correr numa máquina local, sem qualquer tipo de falhas. Isto afastou um pouco do objectivo principal deste documento, que seria aproximar os valores o melhor possível da realidade.

Trabalhos Futuros

Para trabalhos futuros é essencial continuar com melhoramentos a nível de protocolo de congestionamento de rede, de modo a que a rede se torne mais adaptativa a tráficos prejudiciais. Estes ajustes devem ser tomados em conta, de maneira a que haja mais independência da rede em relação à aplicação. Desta forma, *hosts* que congestionam a rede devem sofrer as consequências e terem mais perdas do que os *hosts* que estão a transmitir dentro do seu limite.

É necessário criar uma protecção, tanto na rede como na aplicação, que venha a prevenir congestões causadas pelo uso intensivo da rede. Com um mecanismo deste género, o tráfego em rajada passa a ser um tráfego mais regulado, mais espalhado ao longo do tempo, sem grandes variações de ritmo de transferência.

Para que esta protecção seja criada de forma eficaz é necessário fazer mais testes para perceber qual a solução mais adequada à futura rede, quando estiver completamente implementada. Por esse motivo, haverá alterações nas aplicações criadas no âmbito deste documento, pelo facto de estas não serem as versões finais, mas sim exemplos para possíveis versões finais.

Um outro ponto que deverá ser abordado em futuros estudos é a possibilidade de implementar toda esta arquitectura numa arquitectura SDN (*Software define Networking*). Esta possibilidade torna mais prática a elaboração do controlo de congestão já que em arquitecturas SDN, o controlo de dados está separado da rede.

No caso actual o controlo é efectuado individualmente por cada equipamento da rede. No caso do SDN, os *routers* enviariam somente os dados entre eles e o controlador seria responsável por transmitir as regras a cada um dos equipamentos na rede.

Para concluir, os trabalhos futuros serão destinados a tornar a rede mais justa, mais segura a congestionamentos e mais independente das aplicações, mas para isso será necessário criar aplicações que se adaptam a essas alterações a nível da rede, pelo facto do UDP e o TCP não possuírem mecanismos de congestão capazes para redes idênticas à criada neste documento.

Referências

[1] C. Jin, D.Wei, S. Low, S. Hegde, "Fast TCP", in *Engineering & Applied Science*, Caltech, (n.d.).

[2] L. Brakmo, *Student member*, *IEEE*, L. Peterson, "TCP Vegas: End to End Congestion", in *IEEE journal on selected areas of communications*, vol.13, no.8, Oct. 1995

[3] F. Gratzer, "QUIC – Quick UDP Internet Connections", in *Seminar Innovative Internet – Technolgien und Mobilkommunikation*, doi: 10.2313/NET-2016-09-1_06, Sep. 2016.

[4] Y. Gu, R. Grossman, "UDT: UDP-based Date Transfer for High-Speed Wide Area Networks", in *National Center for Data Mining*, (n.d.).

[5] D. Stenberg, *HTTP2 Explained ed.2*.

[6] Peng Yang, Juan Shao, Wen Luo, "TCP Congestion Avoidance Algorithm Identification", *IEEE/ACM Transactions on Networking,* vol. 22, no. 4, Aug. 2014, pp. 1311-1324.

[7] I. Rhee and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant", in *Proc. PFLDNet,* Feb. 2005, pp. 1–6.

[8] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound TCP ap- proach for high-speed and long distance networks", in *Proc. IEEE IN- FOCOM*, Barcelona, Spain, Apr. 2006, pp. 1–12.

[9] E. Kohler, M. Handley, and S. Floyd, "Datagram congestion control protocol (DCCP)", RFC 4340, Mar. 2006.

[10] R. Stewart, "Stream control transmission protocol", RFC 4960, Sep. 2007.

[11] M. Allman, V. Paxson, "TCP Congestion Control", RFC 5681, Sep. 2009.

[12] V. Jacobson, “Congestion avoidance and control”, in *Proc. ACM SIG- COMM*, Stanford, CA, USA, Aug. 1988, pp. 314–326.

[13] T. Henderson, Boeing, S. Floyd, “The NewReno Modification to TCP’s Fast Recovery Algorithm”, RFC 6582, Apr. 2012.

[14] Ratna Pavani K., N. Sreenath, “Evaluating the Performance of TCP-Reno, TCP-NewReno and TCP-Vegas on na OBS Network”, in *International Journal of Advances in Engineering & Technology*, vol. 6, issue 2, May 2013, pp. 724-729.

[15] F. Baker, Cisco Systems, G. Fairhurst, “IETF Recommendations Regarding Active Queue Management”, RFC 7567, Jul. 2015.

[16] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, and D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang, “Recommendations on Queue Management and Congestion Avoidance in the Internet”, RFC 2309, Apr. 1998.

[17] Floyd, S., “Connections with Multiple Congested Gateways in *Packet-Switched*, Networks Part 1: One-way Traffic.”, in *Computer Communications Review*, Oct. 1991.

[18] Postel J., “User Datagram Protocol”, RFC 768, Aug. 1980.

[19] Dah-Ming Chiu, R. Jain, “Analysis of increase and decrease algorithms for congestion avoidance in computer networks”, in *Computer Networks and ISDN systems* 17, pp. 1–14.

[20] Y. R. Yang, S. S. Lam, “General AIMD Congestion Control”, in *Proceedings 2000 International Conference on Network Protocols*, pp. 187-198.

[21] D. Katabi, M. Handley, C. Rohrs, “Congestio Control for High Bandwidth-Delay Product Networks”, in *Sigcomm’02*, Aug. 2002.

[22] R. Rejaie, M. Handley, D. Estrin, “RAP: Na End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet”, Jul. 1998.

[23] C. V Hollot, V. Misra, D. Towsley, and W.-B. Gong, “On Designing Improved Controllers for AQM Routers Supporting TCP Flows,” in *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, 2001, vol. 3, pp. 1726–1734.

[24] F. Yanfie, R. Fengyuan, and L. Chuang, “Design a PID controller for active queue management,” in *Proceedings of the Eighth IEEE Symposium on Computers and Communications. ISCC 2003*, 2003, vol. 2, pp. 3–8.

Anexos

Anexo A: Teste 1 da rede.

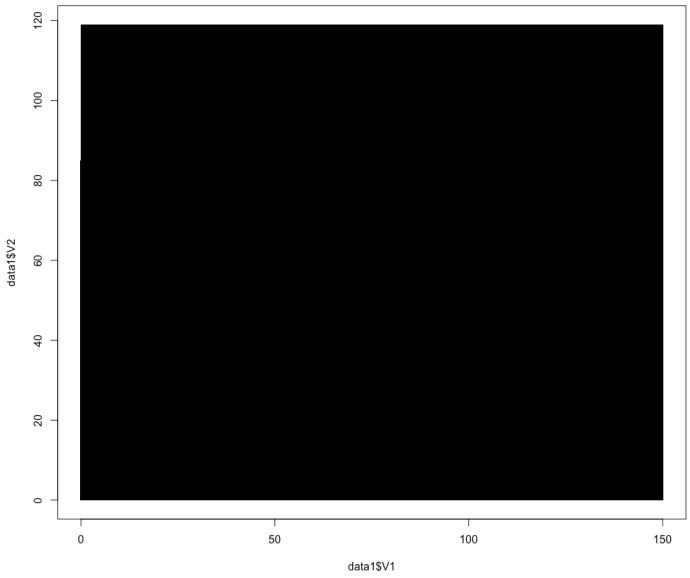


Figura A.1: Tamanho da fila de espera do router D2.

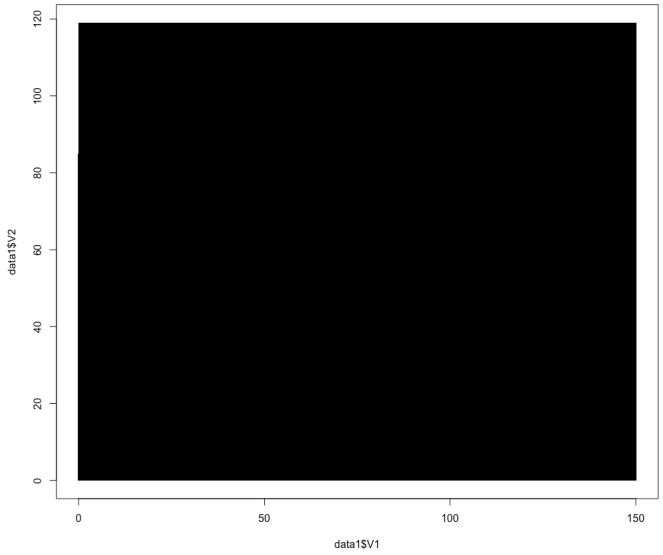


Figura A.2: Tamanho da fila de espera do router C2.

Anexo B: Teste 2 da rede.

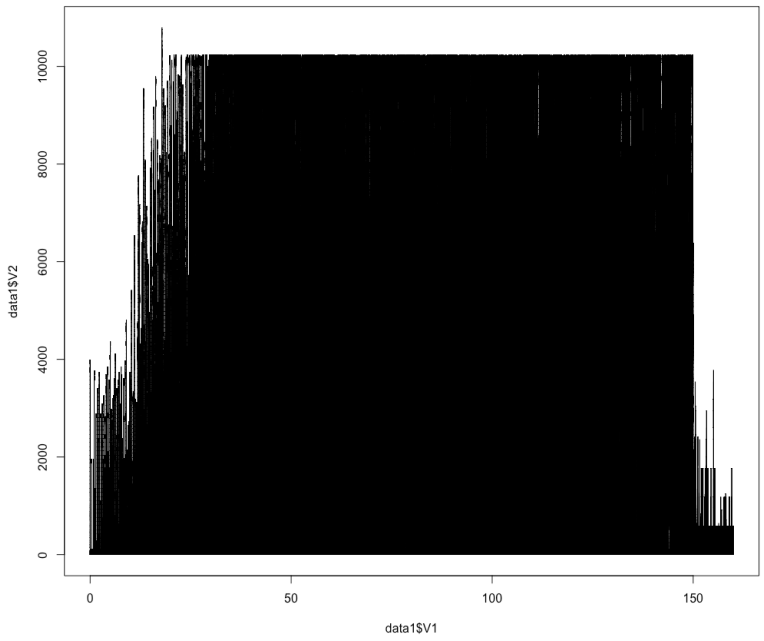


Figura B.1: Tamanho da fila de espera do router D2.

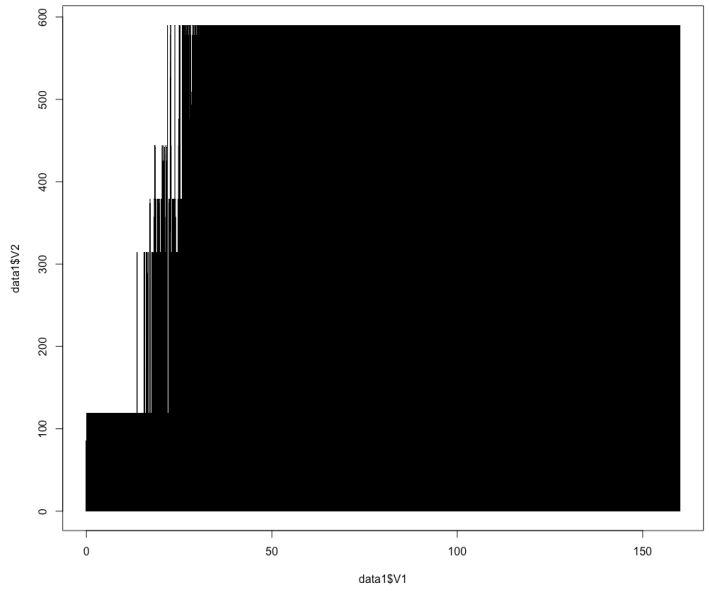


Figura B.2: Tamanho da fila de espera do router C1.

Anexo C: Teste 3 da rede.

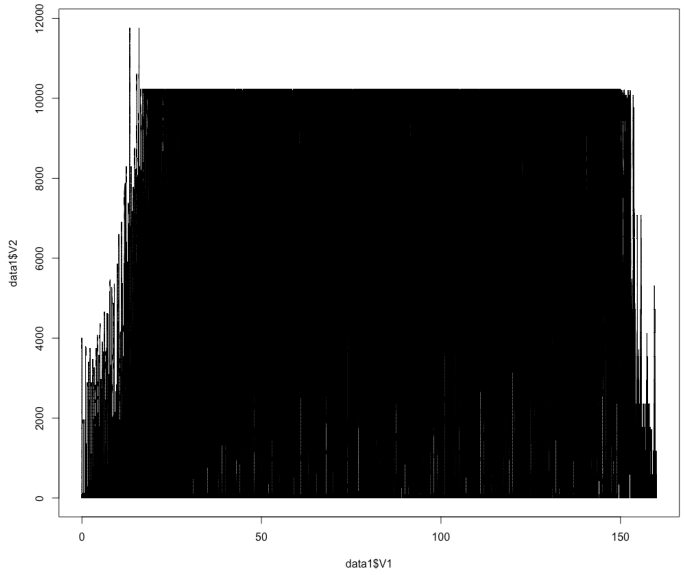


Figura C.1: Tamanho da fila de espera do router D2.

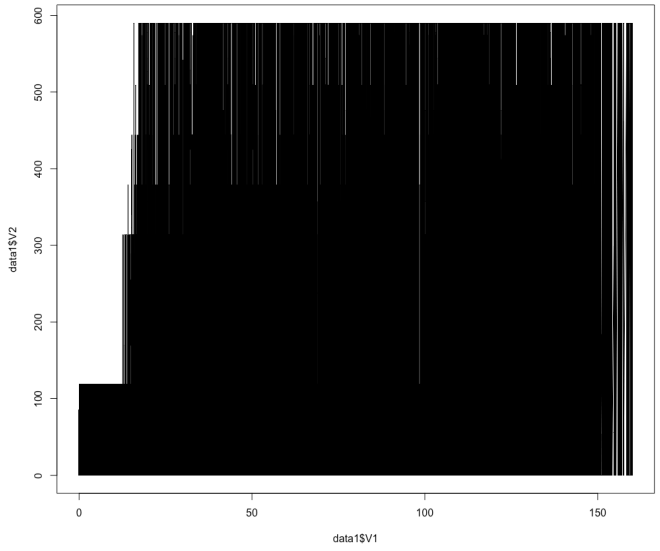


Figura C.2: Tamanho da fila de espera do router D4.

.