

Network Congestion Control

managing internet traffic

MICHAEL WELZL
University of Innsbruck, AUSTRIA

As the Internet becomes increasingly heterogeneous, the issue of congestion control becomes ever more important.

In order to maintain good network performance, mechanisms must be provided to prevent the network from being congested for any significant period of time. Michael Welzl describes the background and concepts of Internet congestion control in an easily comprehensible format. Throughout the book, not just the how, but also the why of complex technologies including the Transmission Control Protocol (TCP) and Active Queue Management are explained. The text also gives an overview of the state-of-the-art in congestion control research and an insight into the future.

Network Congestion Control:

- > Presents comprehensive, easy-to-read documentation on the advanced topic of congestion control without heavy maths.
- > Aims to give a thorough understanding of the evolution of Internet congestion control: how TCP works, why it works the way it does, and why some congestion control concepts failed for the Internet.
- > Explains the Chiu/Jain vector diagrams and introduces a new method of using these diagrams for analysis, teaching & design.
- > Elaborates on how the theory of congestion control impacts on the practicalities of service delivery.
- > Includes an appendix with examples/problems to assist learning.
- > Provides an accompanying website with Java tools for teaching congestion control, as well as examples, links to code and projects/bibliography.

This invaluable text will provide academics and researchers in computer science, electrical engineering and communications networking, as well as students on advanced networking and Internet courses, with a thorough understanding of the current state and future evolution of Internet congestion control. Network administrators and Internet service and applications providers will also find *Network Congestion Control* a comprehensive, accessible self-teach tool.

Companion Website

www.welzl.at/congestion



LIBRARY OF CONGRESS



0 016 072 258 7

ISBN 0-470-02528-X



9 780470 025284

WELZL

Network Congestion Control
managing internet traffic



TK 5105
.875
.157
W454
2005
Copy 1

WILEY

Network Congestion Control

managing internet traffic

MICHAEL WELZL



Wiley Series in Communications Networking & Distributed Systems



Network Congestion Control

Managing Internet Traffic

Michael Welzl

*Leopold Franzens
University of Innsbruck*



John Wiley & Sons, Ltd

Copyright © 2005

John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester,
West Sussex PO19 8SQ, England

Telephone (+44) 1243 779777

Email (for orders and customer service enquiries): cs-books@wiley.co.uk

Visit our Home Page on www.wiley.com

All Rights Reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except under the terms of the Copyright, Designs and Patents Act 1988 or under the terms of a licence issued by the Copyright Licensing Agency Ltd, 90 Tottenham Court Road, London W1T 4LP, UK, without the permission in writing of the Publisher. Requests to the Publisher should be addressed to the Permissions Department, John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England, or emailed to permreq@wiley.co.uk, or faxed to (+44) 1243 770620.

This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold on the understanding that the Publisher is not engaged in rendering professional services. If professional advice or other expert assistance is required, the services of a competent professional should be sought.

Other Wiley Editorial Offices

John Wiley & Sons Inc., 111 River Street, Hoboken, NJ 07030, USA

Jossey-Bass, 989 Market Street, San Francisco, CA 94103-1741, USA

Wiley-VCH Verlag GmbH, Boschstr. 12, D-69469 Weinheim, Germany

John Wiley & Sons Australia Ltd, 42 McDougall Street, Milton, Queensland 4064, Australia

John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop #02-01, Jin Xing Distripark, Singapore 129809

John Wiley & Sons Canada Ltd, 22 Worcester Road, Etobicoke, Ontario, Canada M9W 1L1

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Library of Congress Cataloging-in-Publication Data

Welzl, Michael, 1973–

Network congestion control : managing Internet traffic / Michael Welzl.

p. cm.

Includes bibliographical references and index.

ISBN-13: 978-0-470-02528-4 (cloth : alk. paper)

ISBN-10: 0-470-02528-X (cloth : alk. paper)

1. Internet. 2. Telecommunication–Traffic–Management. I. Title.

TK5105.875.I57W454 2005

004.67'8 – dc22

2005015429

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

ISBN-13 978-0-470-02528-4

ISBN-10 0-470-02528-X

Typeset in 10/12pt Times by Laserwords Private Limited, Chennai, India

Printed and bound in Great Britain by Antony Rowe Ltd, Chippenham, Wiltshire

This book is printed on acid-free paper responsibly manufactured from sustainable forestry in which at least two trees are planted for each one used for paper production.

whether control and/or measurement actions from within the network should be included. Since it seems obvious that adding these functions will complicate things significantly, we postpone such considerations and start with the simpler case of *implicit* feedback, that is, measurements that are taken at the receiver and can be used to deduce what happens within the network.

In order to determine what such feedback can look like, we must ask the question, What can happen to a packet as it travels from source to destination? From an end-node perspective, there are basically three possibilities:

1. It can be delayed.
2. It can be dropped.
3. It can be changed.

Delay can have several reasons: distance (sending a signal to a satellite and back again takes longer than sending it across an undersea cable), queuing, processing in the involved nodes, or retransmissions at the link layer. Similarly, packets can be dropped because a queue length is exceeded, a user is not admitted, equipment malfunctions, or link noise causes a checksum of relevance to intermediate systems to fail. Changing a packet could mean altering its header or its content (payload). If the content changed but the service provided by the end-to-end protocol includes assurance of data integrity, the data carried by the packet become useless, and the conclusion to be made is that some link technology in between introduced errors (and no intermediate node dropped the packet due to a checksum failure). Such errors usually stem from link noise, but they may also be caused by malicious users or broken equipment. If the header changed, we have some form of explicit communication between end nodes and inner network nodes – but at this point, we just decided to ignore such behaviour for the sake of simplicity. We do not regard the inevitable function of placing packets in a queue and dropping them if it overflows as such active participation in a congestion control scheme.

The good news is that the word ‘queue’ was mentioned twice at the beginning of the last paragraph – at least the factors ‘delay’ and ‘packet dropped’ can indicate congestion. The bad news is that each of the three things that can happen to a packet can have quite a variety of reasons, depending on the specific usage scenario. Relying on these factors therefore means that implicit assumptions are made about the network (e.g. assuming that increased delay always indicates queue growth could mean that it is assumed that a series of packets will be routed along the same path). They should be used with care.

Note that we do not have to restrict our observations to a single packet only: there are quite a number of possibilities to deduce network properties from end-to-end performance measurements of series of packets. The so-called *packet pair* approach is a prominent example (Keshav 1991a). With this method, two packets are sent back-to-back: a large packet immediately followed by a small packet. Since it is reasonable to assume that there is a high chance for these packets to be serviced one after another at the bottleneck, the interspacing of these packets can be used to derive the capacity of the bottleneck link. While this method clearly makes several assumptions about the behaviour of routers along the path, it yields a metric that could be valuable for a congestion control mechanism (Keshav 1991b). For the sake of simplicity, we do not discuss such schemes further at this point and reserve additional observations for later (Section 4.6.3).

solutions for congestion control. While it is well known that Internet access now spans a broad range of devices and environments (from the cell phone in my pocket to the satellite way above me – in fact, there is now even research on the ‘Interplanetary Internet’), we have totally neglected this aspect of network communications up to now. However, it does play a significant role for congestion control.

Briefly put, the influence of heterogeneous environments can falsify the interpretation of network measurements. As already mentioned in Section 2.4, only three things can happen to a single packet as it traverses the network: (i) it can be delayed, (ii) it can be dropped, and (iii) it can be changed. At least two of these effects lend themselves to interpretations by an end-to-end receiver for the sake of providing the sender with feedback about the state of congestion in the network: delay, which can be a result of growing queues, and packet drops, which can be a result of an overflowing queue. We have also seen that, whether increasing delay is interpreted as a sign of congestion or not, measuring the RTT is useful for several things. Here is a brief (non comprehensive) list of different environments with their properties that have an influence on end-to-end delay and loss:

Wireless links: In wireless usage scenarios, bit errors are frequent; they can, for instance, occur when a user passes by a wall or enters an elevator. At the wireless receiver, receiving noisy data means that a checksum will fail. Depending on the specific technology that is used, this can mean two things:

1. The packet is dropped (because its contents are now useless).
2. The packet is delayed, because the link layer retransmits it if an error occurs. Even then, if multiple retransmissions fail, the packet can eventually be dropped.

Typically, bit errors are not statistically independent, that is, they are clustered in time (Karn et al. 2004).

Satellite links: Satellites combine a number of potentially negative factors:

- Connections are noisy, which means that they share the negative properties of wireless links described above. This problem may be compensated for with *Forward Error Correction* (FEC), but this can add delay.
- They show a long delay, which is always a problem because of its impact on the precision of RTT measurements: the larger the delay, the larger its fluctuations. In the case of *GEOs* – satellites in geostationary orbit – the delay is between 239.6 and 279.0 ms (Allman et al. 1999a). While ‘low earth orbit’ satellites (*LEOs*) seem to alleviate this problem via their significantly reduced one-way propagation delay of about 20 to 25 ms, it may be necessary to route data *between* them (‘LEO handoff’), which can cause packet loss and propagation delay fluctuations (Hassan and Jain 2004).
- Satellites are typical examples of so-called long fat pipes – links with a large bandwidth $\times$ delay product. As we have discussed earlier, this value dictates the amount of data a protocol should have ‘in flight’ (transmitted, but not yet acknowledged) in order to fully utilize a link (Allman et al. 1999a). Such links cause problems for mechanisms like AIMD: additively increasing the rate with a constant factor until the link is saturated can take a very long time (perhaps

hop after the sender; this can be attained by choosing a suitable packet (actually *frame*) type from the underlying link layer technology. For instance, 802.3x defines a PAUSE frame that can be used as a gap packet when its 'pause time' is set to zero.

4.3.3 Tuning parameters on the fly

Another way to control the behaviour of TCP without changing its implementation is to adaptively tune its parameters. One particularly interesting parameter is the buffer size of the sender, which automatically imposes an upper limit on its window. Tuning the maximum window size itself may seem more attractive because its impact is perhaps more immediate, but this parameter is not always available – remember that the goal is to influence TCP without changing its implementation. Since the maximum sender window should ideally equal the bandwidth $\times$ RTT product of the connection, the required buffer size varies as widely as the environments that TCP/IP is used in; an example range given in (Hassan and Jain 2004) and (Feng et al. 2003) is a short modem connection with a capacity of 56 kbps and a delay of 5 ms – corresponding with a window size of 36 bytes – versus a long-distance ATM connection with a capacity of 622 Mbps and a delay of 100 ms, which corresponds with a window size of 7.8 MB. Choosing the latter window size for the former scenario wastes over 99% of its allocated memory whereas choosing the former window size for the latter scenario means that up to 99% of the network capacity is wasted.

One simple solution to the problem is to manually tune the buffer size depending on the environment. The problem with this approach is that both the network capacity that is available to an end system and the RTT fluctuate, which is caused by effects such as routing changes or congestion in the network. It is therefore desirable to automatically adapt the buffer size to the given network conditions. While they are not the only methods available (e.g. the memory management technique in Linux kernel version 2.4 also does this), two well-known approaches can be seen as representatives:

1. *Auto-tuning* utilizes TCP header information and the Timestamps option to estimate the bandwidth $\times$ RTT product and adapt the buffer size on the fly; this is a sender-side kernel modification where several concurrent TCP connections can share a single buffer (Semke et al. 1998).
2. *Dynamic Right-Sizing (DRS)* is a receiver-side modification that makes the sender change its maximum window by tuning the advertised window (Feng et al. 2003); thus, the *flow control* functionality of TCP is used by this approach. The receiver estimates the current *cwnd* of the sender by monitoring the throughput; the RTT is estimated by measuring the time between sending an ACK and the reception of data that are at least one window beyond the ACKed data. Since this assumes that the sender will transmit new data right away when it receives an ACK, but this may not always be the case (i.e. the sender may not have new data to send), this RTT is interpreted as an upper bound. If the receiver is itself sending back data (remember, TCP connections are bidirectional), the RTT will automatically be estimated by the TCP implementation.

While these techniques do not require changing the TCP code, both auto-tuning and DRS operate at the kernel level. For instance, the advertised receiver window is usually not

function that retains the strengths of BIC yet simplifies its window control and enhances its TCP-friendliness. As the name suggests, the function that was found is cubic; its input parameters are the maximum window size (from normal BIC), a constant scaling factor, a constant multiplicative decrease factor and the time since the last drop event. The fact that this 'real' time dominates the control is what makes it independent of the RTT: for two flows that experience loss at the same time, the result of this function depends on the time since the loss occurred even if they have different RTTs.

The cubic response function enhances the TCP-friendliness of the protocol; at the same time, the fact that the window growth function does not depend on the RTT allows TCP to be more aggressive than CUBIC when the RTT is small. Therefore, CUBIC additionally calculates how fast a TCP sender would open its window under similar circumstances and uses the maximum of this result and the result given by the standard CUBIC function.

TCP Westwood+

TCP Westwood+ (a refined version of 'TCP Westwood', first described in (Grieco and Mascolo 2002)) resembles HSTCP and STCP in that it changes the response function to react not in a fixed manner but in proportion to the current state of the system. It is actually quite a simple change, but there is also a fundamental difference: in addition to the normal inputs of a TCP response function, TCP Westwood+ utilizes the actual rate at which packets arrive at the receiver as an input – this is determined by monitoring the rate of incoming ACKs. Strictly speaking, this protocol therefore uses slightly more implicit feedback than the ones discussed so far. A result of this is that it works well over wireless networks: it will reduce its rate severely only when a significant number of packets are lost, which may either indicate severe congestion or a long series of corruption-based loss events. A single loss event that is due to link noise does not cause much harm – this is different from standard TCP. Also, while this protocol is TCP-friendly according to (Grieco and Mascolo 2004), it does not distinguish between 'low loss' and 'high loss' scenarios.

The response function of TCP Westwood+ is easily explained: its only divergence from standard TCP Reno or NewReno behaviour is the update of *ssthresh* (the starting point for fast recovery) in response to a congestion event. Instead of applying the fixed rate decrease by half, it sets this variable to the product of the estimated bandwidth (as determined from the rate of ACKs) and the minimum RTT – this leads to a more-drastic reduction in a case of severe congestion than in a case of light congestion. The key to the efficiency of TCP Westwood+ is the function that counts and filters the stream of incoming ACKs; this was designed using control theory. While the original version of TCP Westwood was prone to problems from 'ACK compression' (see Section 4.7), this was solved in TCP Westwood+ by means of a slightly modified bandwidth estimation algorithm.

4.6.2 Delay as a congestion measure

When congestion occurs in the network, a queue grows at the bottleneck and eventually, if the queue length cannot compensate, packets will be dropped. It therefore seems reasonable to react upon increasing delay. Note that only changes can be used, as a fixed delay measure does not explain much: two mildly congested queues may yield the same delay as a single severely congested one. Only delay *changes* can be used, that is, measurements must always be (implicitly or explicitly) combined with a previously recorded value. Delay measurements

already control the behaviour of TCP via its RTT estimate, and increasing delay (which might be caused by congestion) will even make it react slower; still, the measurements are only interpreted for the sake of proper self-clocking and not as a real congestion indication.

Using delay in this manner has the potential advantage that a congestion control mechanism can make a more reasonable decision because it has more feedback. At the same time, it is non-intrusive, that is, it does not require any additional packets to be sent, and routers do not have to carry out extra processing. However, this feedback must be used with care, as it is prone to misinterpretations (see Section 2.4) – delay can also be caused by routing or link layer ARQ.

The idea of delay-based congestion control is not a new one; in fact, a mechanism that does this was described as early as 1989 (Jain 1989). In this scheme, the previous and current window and delay values are used to calculate a so-called normalized delay gradient, upon which the decision whether to increase or decrease the rate is based. Since then, several refinements were proposed; we will look at two popular ones.

TCP Vegas

An implicit form of delay-based congestion control can be found in a well-known TCP variant called *TCP Vegas* (Brakmo et al. 1994). This is a sender-side TCP modification that conforms to the normal TCP Reno standard: the specification does not forbid sending *less* than a congestion window, and this is exactly what TCP Vegas does. Like TCP Westwood+, it determines the minimum RTT of all measurements. The expected throughput is then calculated as the current size of *cwnd* divided by the minimum RTT. Additionally, the actual throughput is calculated by sending a packet, recording how many bytes are transferred until reception of the corresponding ACK and dividing this number by the sample RTT; if the sender fully utilizes its window, this should be equal to *cwnd* divided by SRTT. The calculation is carried out every RTT, and the difference between the actual rate and the expected rate governs the subsequent behaviour of the control.

The expected rate minus the actual rate is assumed to be non-negative, and it is compared against two thresholds. If it is underneath the lower threshold, Vegas increases *cwnd* linearly during the next RTT because it assumes that there are not enough data in the network. If it is above the upper threshold (which means that the expected throughput is much lower than the actual sending rate), Vegas decreases *cwnd* linearly. The congestion window remains unchanged when the difference between the expected and the actual rate is between the two thresholds. In this way, it can converge to a stable point of operation – this is a major difference between TCP Vegas and TCP Reno, which always needs to exceed the available capacity in order to detect congestion. Vegas can detect incipient congestion and react early, and this is achieved by monitoring delay: if the sender fully utilizes its congestion window, the only difference between the actual and expected rate calculated by Vegas is given by the minimum RTT versus the most-recent RTT sample.

This mechanism is the core element of TCP Vegas; there are two more features in the protocol: (i) it uses the receipt of certain ACKs as a trigger to check whether the RTO timer expired (this is an enhancement over older implementations with coarse timers), and (ii) it only increases the rate every other RTT in slow start so as to detect congestion during the intermediate RTTs. Despite its many theoretical advantages, Vegas is hardly used in practice nowadays because it is less aggressive than TCP Reno; if it was used in the Internet, it would therefore be ‘pushed aside’ (Grieco and Mascolo 2004). The value of TCP Vegas lies