Jacobs University Bremen

Guided Research Thesis

TCP Congestion Control Measurements

Author:
Plamen Dimitrov

Computer Science
class of 2011
Jacobs University Bremen
Campus Ring 1
28759 Bremen
Germany

Supervisor:
Prof. Dr. Jürgen Schönwälder

May 8, 2011

Abstract

The Transmission Control Protocol (TCP) is the most widely deployed Internet protocol nowadays. Therefore, its performance undermines the performance of the Internet as a whole, on a global scale. One of the most important elements of TCP with regard to its performance characteristics is congestion control. The aim of this research is to eliminate some of the technical difficulties that occurred at the identification stage of a large-scale experiment which sought to identify the congestion control algorithms used on a geographically distributed set of servers. This was achieved by extending the active probing tool developed and used in the aforementioned study. As a result, a number of host identification pitfalls from the same study was either eliminated or fully avoided.

1 Introduction

The Transmission Control Protocol (TCP) is the most widely deployed Internet protocol nowadays [1]. Therefore, its performance undermines the performance of the Internet as a whole, on a global scale. One of the most important elements of TCP with regard to its performance characteristics is congestion control [2]. The aim of this research is to eliminate some of the technical difficulties that occurred at the identification stage of the large-scale experiment which sought to identify the congestion control algorithms used on a geographically distributed set of servers done by S. Feyzabadi and J. Schönwälder [5]. This was achieved by extending the active probing tool developed and used in the aforementioned study. As a result, a number of host identification pitfalls from the study [5] was either eliminated or fully avoided. The extension of the set of known congestion control algorithms from current two (*Reno* and *Cubic*) to a larger number could decrease the large fraction of "Unknown" hosts. A number of techniques can be employed to circumvent the problems associated with the temporary or permanent relocations of requested resources. In addition, the repetitive requests for different resources on the same host may provide a set of congestion behaviors (instead of a single one), which can be used together in order to associate a host to the specific congestion control with a certain probability. Selection of the hosts, whose congestion control mechanisms will be inferred is a crucial step for ensuring the completeness of the results and will therefore be given special attention.

Figures 1 and 2 show the results of the study by S. Feyzabadi and J. Schönwälder [5] in terms of distributions of web sites with different polynomial degrees of their congestion window (CWND) growth function, measured with a maximum segment size (MMS) of 100 and 200 bytes, respectively. The CWND size limits the amount of data a TCP can send. At any given time, a TCP must not send data with a sequence number higher than the sum of the highest acknowledged sequence number and the minimum of the CWND size and the receiver window (RCWND) size [9]. Using this information, different congestion control algorithms that use different classes of CWND growth functions can be distinguished [5]. As can be seen in Fig. 1 and 2, the CWND growth functions of a significant number of the probed servers could not be determined with the current implementation of the active probing tool used. This leaves the associated congestion control algorithms undetermined.

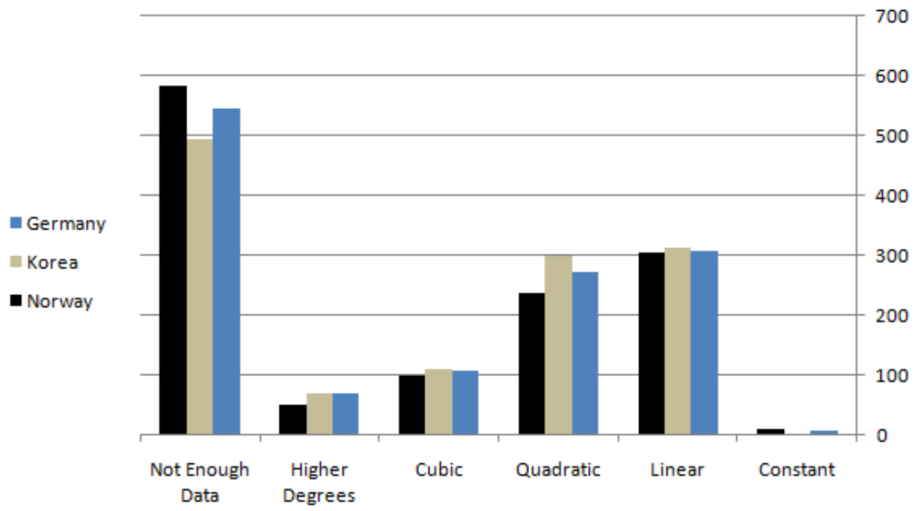

Figure 1: Distribution of web sites with different polynomial degrees of their CWND growth function (MSS = 100) [5]

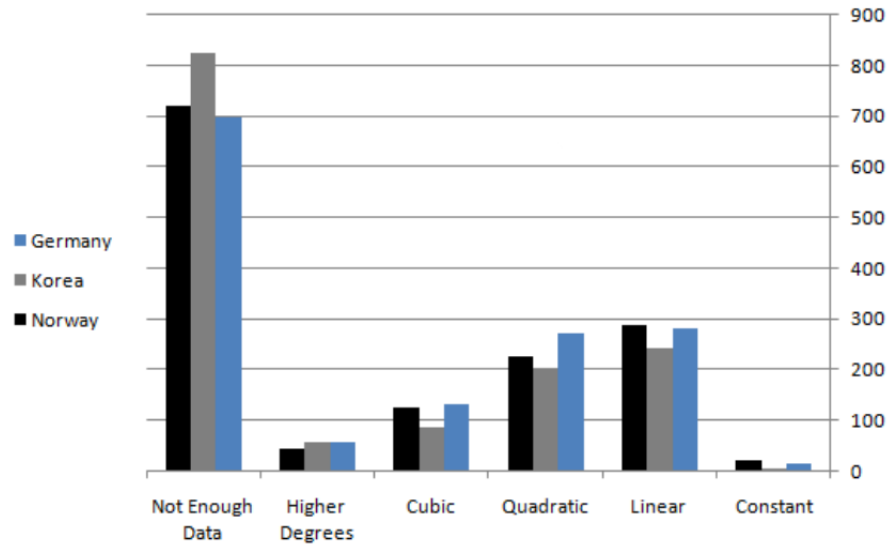

Figure 2: Distribution of web sites with different polynomial degrees of their CWND growth function (MSS = 200) [5]

The remainder of the proposal is structured as follows. In section 2 we look at the the concept of congestion control and present technical aspects of the active probing tool to be used. Section 4 presents an overview of the related work. Some of the proposed and already performed methods aimed at the improvement of the functionality of the tool are presented in section 3.

2 Tools and Concepts

This section presents an overview of several popular congestion control mechanisms, as well as the discussion of the inner-workings of the TCP Behavior Inference Tool (TBIT). Both the original [11] and the modified [5] versions of TBIT are presented towards the end of the section.

2.1 TCP Congestion Control

Most congestion control algorithms have four distinct stages of operation: slow start, congestion avoidance, fast retransmit and fast recovery. Different congestion control mechanisms differ by their congestion window size (CWND) growth functions. In the following sections 2.2.1 and 2.2.2 two popular congestion control algorithms, namely *Cubic* and *Reno* respectively, are briefly described. Other well known congestion control algorithms include *TCP Tahoe, TCP NewReno, TCP SACK, TCP Veno, TCP Bic, TCP Illinois, TCP Vegas, TCP Westwood* among others [2] and will further be studied for potential integration into the mechanism of inferring a host's congestion control scheme.

2.2 Examples of TCP Congestion Control Algorithms

2.2.1 Cubic Congestion Control

Being distributed as the default congestion control mechanism of the Linux kernel since 2006, *Cubic* is presumably one of the widely used algorithms for congestion control [8].

The most characteristic feature of Cubic is that it uses a cubic function for its Congestion Window (CWND) growth function during the congestion avoidance phase. Cubic uses the time elapsed since the last congestion event instead of the RTT as the basis for its growth function [8]. This allows the algorithm to effectively avoid congestion in situations where e.g. two flows competing for the same bottleneck link have different round-trip times (RTTs), and thus the flow with the smaller RTT may acquire all available resources [2].

Cubic uses the following formula to calculate the CWND size w in terms of the elapsed time since the last congestion event Δ:

$$w = C(\Delta - \sqrt[3]{\beta \cdot w_{max}/C})^3 + w_{max} \tag{1}$$

where C is a predefined constant, β is a coefficient of multiplicative decrease in fast recovery and w_{max} is the CWND size before the registered loss detection [2]. The CWND growth function follows a cubic function with a plateau set at w_{max}.

Cubic reduces its CWND size by the factor β (set to 0.2 by default) when a timer expires before an ACK has been received or triggered by explicit congestion notification.

Fig. 3 illustrates the CWND dynamics in Cubic.

2.2.2 Reno Congestion Control

Reno is one of the first TCP congestion control algorithm implemented that supports all the four phases mentioned in section 2.1. The server starts with

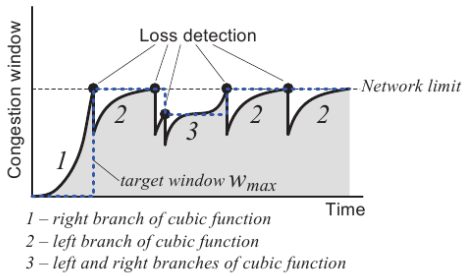

Figure 3: Congestion window dynamics in *Cubic* [2]

an initial CWND size of two packets. During the slow start phase, CWND size doubles every round-trip time (RTT).

After having received the initial threshold of duplicate ACKs, Reno enters fast recovery mode. During fast recovery, the TCP sender is able to make intelligent estimates of the amount of outstanding data by assuming each duplicate ACK received represents a single packet having left the pipe. When congestion avoidance is entered, the CWND increases linearly every round-trip time until congestion is detected by explicit notification or detection of a timer expiration before an ACK receival [4]. With a high probability, a non-duplicate ACK will acknowledge delivery of all data packets previously inferred by the duplicate ACK's received. At this point, congestion window deflation to $CWND/2$ (value of CWND after entering fast recovery) takes place [2]. Fig. 4 illustrates the CWND dynamics in Reno.

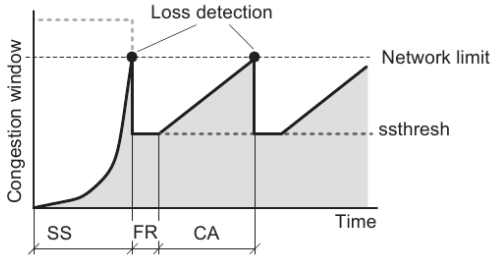

Figure 4: Congestion window dynamics in Reno. SS: slow start, CA: congestion avoidance, FR: fast recovery [2]

2.3 TCP Behavior Inference Tool (TBIT)

The TCP Behavior Inference Tool (TBIT) is an active probing tool which characterizes the behavior of a remote web server by trying to infer the initial size of the congestion window (CWND), the congestion control algorithm used, the response to selective acknowledgments, the response to explicit congestion notification and the time wait duration. TBIT is universal with respect to the kind of server being probed and tries to appear non-hostile in order to achieve maximum effectiveness [11]. TBIT's architecture and process flow are described on Fig. 5.

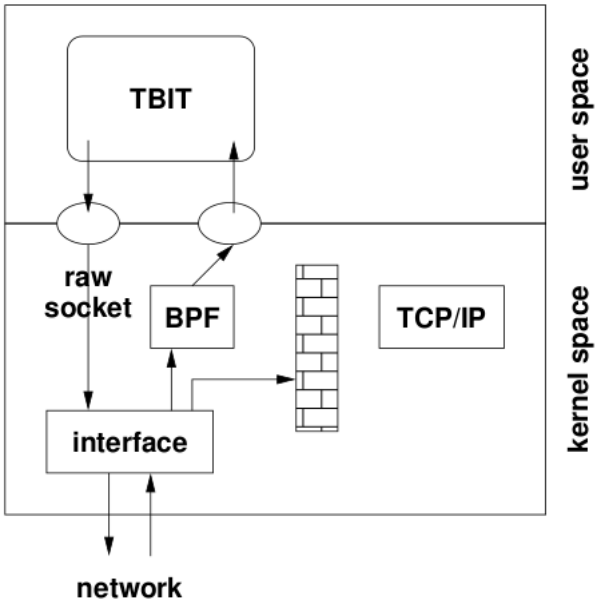

Figure 5: TBIT architecture and process flow [5]

TBIT can identify certain properties of the probed server using two techniques. The first technique works by interpreting the TCP handshake information and extracts static information about the settings of the server (e.g. selective acknowledgments (SACK) or early congestion notification support). The second technique involves initiating and maintaining a data transfer and sending packets in a specific order as to induce certain actions in the probed server and then analyzing the related metrics [5].

Assuming that TBIT is running on host A, and the remote web server is running on host B, the second technique as implemented in the original version of TBIT works in the following way [11]:

- TBIT opens a raw IP socket.
- TBIT opens a BPF device and sets the filter to capture all packets going to and coming from host B.
- TBIT sets up a host firewall on A to prevent any packets coming from host B from reaching the kernel of host A.

- TBIT sends a TCP SYN packet, with the destination address of host B and a destination port of 80. The MSS is set to a small value (e.g. 100 bytes) to force the remote server to send several data packets for the test, even if the requested web page is small in size. TBIT declares a receiver window of 5*MSS.
- TBIT requests the base web page.
- The remote server starts sending the base web page to the TBIT client in 100-byte segments.
- TBIT acknowledges each segment until the 13-th segment is received.
- TBIT drops the acknowledgment for segment 13
- TBIT receives and acknowledges packets 14 and 15. The ACKs sent are duplicate ACKs for packet 12.
- Segment 16 is dropped. All further packets are acknowledged appropriately.
- TBIT closes the connection as soon as 25 data packets are received, including retransmissions.

Based on this stream of 25 packets, TBIT can determine the congestion control behavior of the remote TCP. NewReno TCP is characterized by a Fast Retransmit for packet 13, no additional Fast Retransmits or Retransmit Timeouts, and no unnecessary retransmission of packet 17). Reno TCP is characterized by a Fast Retransmit for packet 13, a Retransmit Timeout for packet 16, and no unnecessary retransmission of packet 17 [11].

Being originally designed to distinguish Reno, NewReno and Tahoe [11], the algorithm described above is already outdated and misclassifies newer congestion control algorithms like Cubic, which perform fast retransmission identically to Reno, as NewReno [5].

A modified version of TBIT used for active probing in the previously mentioned study [5] left the initial setup (initializing the connection) unchanged, while a different identification technique was employed.The evolution of CWND size over time was used instead of the reaction to duplicate ACKs in order to infer certain properties of the congestion control algorithm used at the host's side [5].

The modified version of TBIT mentioned earlier starts the inference process as in the original implementation. After the server starts transmitting the requested web page, the modified TBIT does not reply with ACKs immediately but first stores all received packets instead. Then, having captured packets for a specific time, the client counts the number of the packets in the buffer and sets RCWND as the difference between the last buffer size and the newer buffer size. After acknowledging a packet with a certain sequence number (e.g. 62 in the study [5]), the client stops sending ACKs and ignores all the packets in the buffer. Two more ACKs are then sent for that packet in order to induce the server to go into fast retransmission mode. Then the client resumes sending ACKs for the received packets without dropping any packets, keeps measuring the RCWND size and stores the values in a file. In addition, the modifications

include the intentional use of delays to ensure the receival of all packets of a CWND and reordering of the acknowledgments sent by the client [5]. Fig. 6 illustrates the algorithm's approach to measuring the RCWND size.

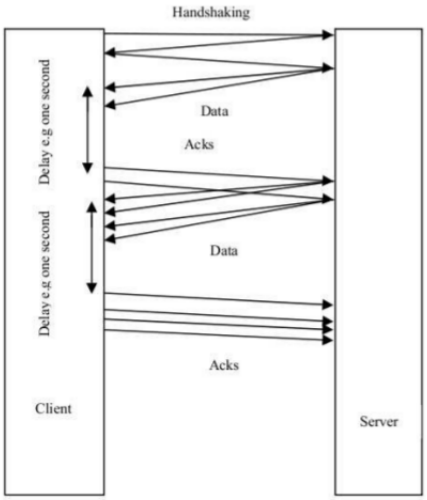

Figure 6: Approach of measuring the RCWND size in the modified version of TBIT [5]

Note that the setup of the BSD Packet Filter (BPF), used to deliver the segments received from the remote host to the TBIT process requires special privileges [10].

3 Improvement Directions

The approach of using the evolution of the congestion window over time to identify congestion schemes with the current (modified) implementation of the active probing tool has two major sets of problems. HTTP errors and MMS-related failures can be seen as probing problems, whereas misclassification and lack of enough recognizable congestion control algorithms can be seen as identification problems.

3.1 Probing problems

3.1.1 HTTP errors

First or all, as can be seen in the results of the previous study [5], a huge percentage of the failures in inferring the congestion control schemes of the remote servers is due to connectivity issues. The HTTP error codes 301 (moved permanently) and 302 (moved temporarily) [7] are quite frequently returned from servers. Especially code 302 accounts to more than 50% of the failures in the scanned German servers for both maximum segment sizes (MMS) 100 and 200 bytes, as well as Korean servers with an MMS of 200 bytes. It is highly unlikely that such a large portion of some of the most visited websites according to Netcraft [5] can no longer be accessed. In those cases, what would greatly diminish the number of failures is just looking at the *Location* field of the response [7].

The modified version of TBIT currently does not handle this in any way, which just results in the response, possibly containing the new location of the requested document, being just too short for the probing algorithm, which results in an identification failure.

The problem can be illustrated by looking at the response to a GET request to the website http://www.google.com/ done through telnet, which is shown below.

```
GET /
HTTP/1.0 302 Found
Location: http://www.google.de/
Cache-Control: private
Content-Type: text/html; charset=UTF-8
Set-Cookie:
PREF=ID=8ab3ae2f06a9a9f4:FF=0:TM=1292941789:LM=1292941789:S=y383ByJfpzVSH--i;
expires=Thu, 20-Dec-2012 14:29:49 GMT; path=/; domain=.google.com
Set-Cookie:
NID=42=aCQxkuZBGH5yIpiNqZ7TcmW6uV3FxfFmoFMrwcBQSXUcPITvjPGRBYE-eykB_T
vocFoZh7ujw IOzw1CI0l7-5VqIPMBPgrDv6ipBiDaq9r2QJuF6s3yDhHt0XPaCksl-;
expires=Wed, 22-Jun-2011 14:29:49 GMT; path=/; domain=.google.com; HttpOnly
Date: Tue, 21 Dec 2010 14:29:49 GMT
Server: gws
Content-Length: 218
X-XSS-Protection: 1; mode=block

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
```

```
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.de/">here</A>.
</BODY></HTML>
```

As expected, since the response is too short, probing http://www.google.com with TBIT (MMS set to 200 bytes) only provides a dataset of 9 points, i.e. TBIT retrieved the page in 9 RTTs.

One possible way of dealing with this problem is to first parse each HTTP response, identify the responses carrying codes 301 and 302, then read the content of the *Location* field and probe the new location. Reading of the *Location* fields for hosts returning codes 301 and 302 can also be done in a step preceding the actual probing, effectively updating the set of hosts to be probed.

Another, though similar approach would be to use a simple python script, which given a hostname returns all the URLs that its main page points to. A subset of these, rooted directly at '/' of the web site can be selected and probed with TBIT afterwards. This was already implemented and tested with very good results.

Both approaches proposed should significantly minimize the number of probing failures.

3.1.2 MMS-related problems

In a considerably large portion of the probed servers, setting the maximum segment size (MMS) as 100 or 200 bytes was problematic. Especially in the 100 bytes case, 45% of the Norwegian, 38% of the Korean and 28% of the German refused to follow the client's request to setting the maximum segment size and thus were unable to be probed by TBIT. The number of failures of the same kind at MMS = 200 bytes was smaller but still significant in each location [5]. One way to deal with this problem is to probe the hosts that failed with a larger MMS value and further allow probing at an even larger MMS value (e.g. 500 bytes) for those hosts that failed at MMS = 200 bytes. This should be done only provided that the web page to be retrieved is large enough so that a sufficient number of packets can be sent to the probing machine.

3.2 Identification problems

Another major shortcoming of the study by S. Feyzabadi and J. Schönwälder [5] that was addressed is the fact that the probing methodology is only able to effectively distinguish between the TCP Reno and Cubic congestion control schemes. Significant efforts need to be made to expand the set of identifiable congestion control algorithms. Improvements need to be done at both the probing and the identification stages.

A significant number of the probed hosts (see Fig. 1 and 2) show a quadratic CWND size growth during the congestion avoidance phase. Currently, it is not clear exactly which congestion control scheme or schemes may account for this behavior. Using an extra additive constant to calculate the evolving CWND size during congestion avoidance is known to be able of producing similar behavior [12]. However it is not too likely that such a significant portion of today's most popular websites use such an approach. According to RFC 2525 - Known TCP

Implementation problems [12], such using an extra additive term to open the CWND more aggressively during congestion avoidance is known to increase loss rate in all connection sharing a bottleneck with the aggressive TCP. Performance can be diminished even on completely uncongested networks [12]. However, this information still does not help identify the algorithms (or classes of such) causing the quadratic growth of CWND during congestion avoidance among the set of known congestion control algorithms. Therefore, this problem needs to be paid special attention in the course or research.

A small experiment aiming at investigating how different congestion control schemes react to probing with TBIT was conducted in a controlled environment. A set of congestion control schemes (Reno,Cubic,Bic, H-TCP, Illinois, Vegas, Veno, Westwood), available as kernel modules, were hotplugged into the kernel of the host (http://tbit.eecs.jacobs-university.de/ running Debian GNU/Linux 5.0 64-bit with a kernel 2.6.26-2-xen-amd64). The host was then probed with TBIT for each congestion scheme at MMS 100, 200 and 500 and the obtained datasets were plotted (Fig. 7, 8 and 9, respectively). These and other results from further experiments with other congestion control algorithms are to be later used with the aim of making more of these algorithms recognizable.

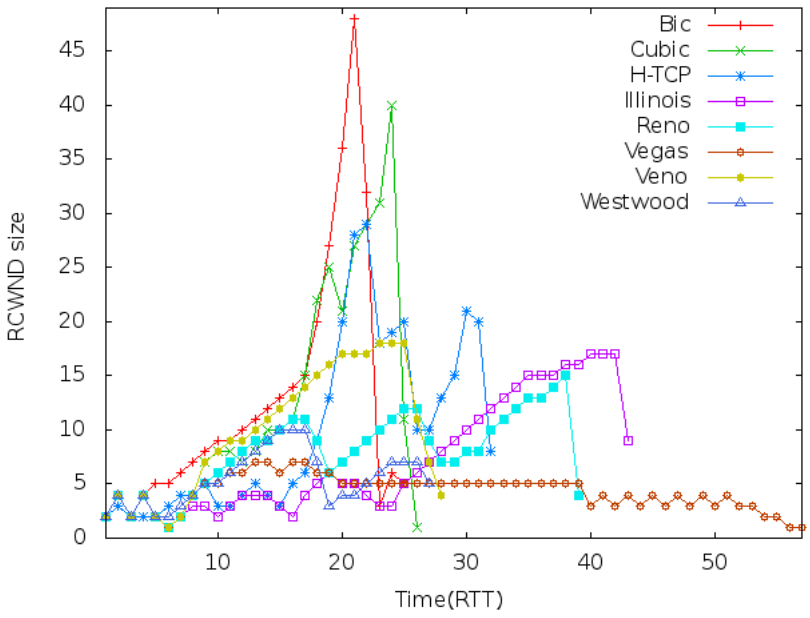

Figure 7: RCWND size over time for different congestion control schemes as measured with TBIT (MMS = 100 bytes)

The study [5] mentions some special cases whereby the probed server uses a fixed value for its CWND size. This was observed when probing the sites www.sbs.com.au, www.orange.es and www.telekon.at among others. In these cases the server first sent two packets as the initial congestion size. Then in the next RTT the server sent 143 packets. The packet drop occurred at packet 6 so 139 packets were left unanswered. This behavior might have been caused by NewReno's fast recovery mode. After a packet loss is detected, retransmis-

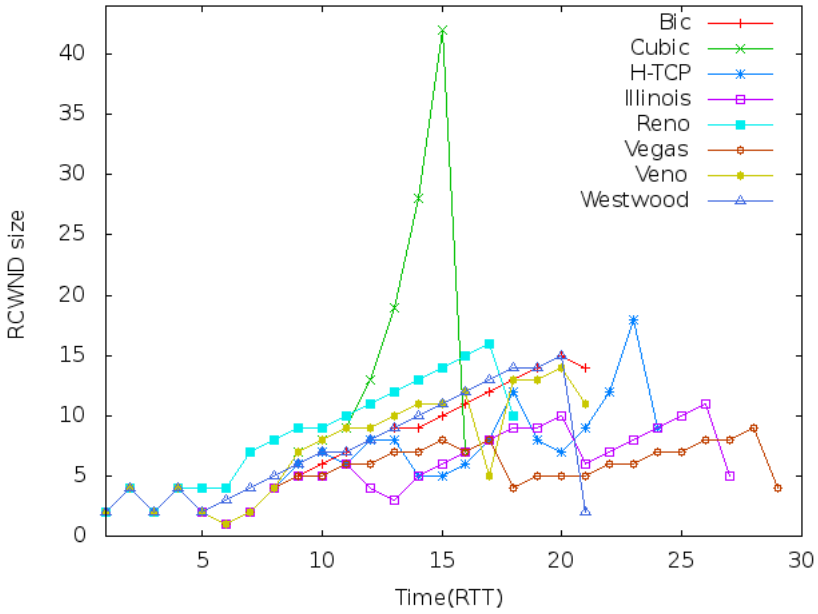

Figure 8: RCWND size over time for different congestion control schemes as measured with TBIT (MMS = 200 bytes)

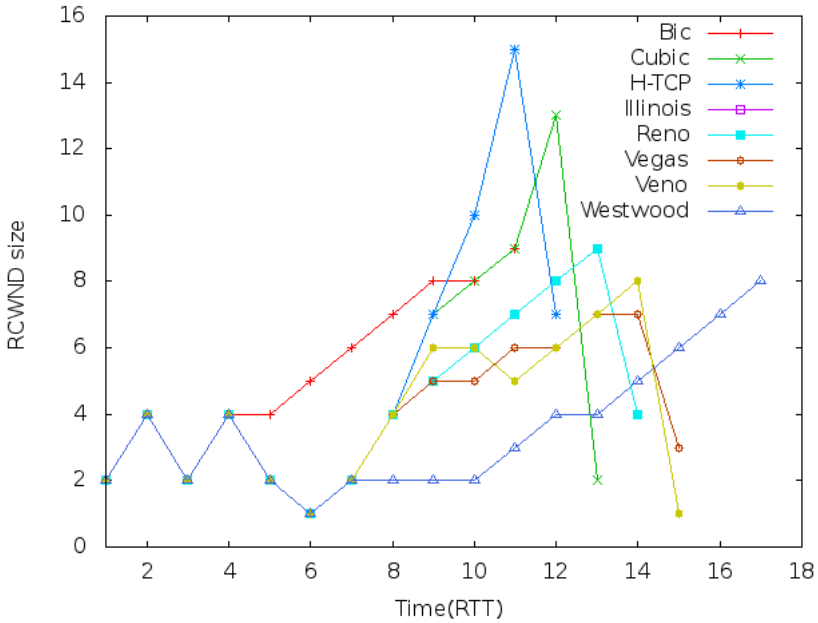

Figure 9: RCWND size over time for different congestion control schemes as measured with TBIT (MMS = 500 bytes)

sion of the lost packet is triggered and each duplicate ACK is "inflating" the CWND. After this stage, each partial ACK is "deflating" the CWND until a non-duplicate ACK is received. Then, retransmission of the lost packet takes place and CWND remains unchanged [2]. This might be a potential explanation for these special cases but the problem has to be further investigated and simulated in a controlled environment.

Ultimately, the proposed research aims at repeating the experiment described in the study by S. Feyzabadi and J. Schönwälder [5] with an improved probing tool and possibly with new approaches to inferring other known congestion control algorithms on a geographically distributed set of servers hosting popular websites. This will allow for a better identification of the popular congestion control algorithms (or classes or such) used globally on the Internet today.

4 Related Work

This research project is based on the TBIT program introduced by J. Padhye and S. Floyd in [11] and on the modifications done to it by S. Feyzabadi and J. Schönwälder described in [5]. The research project directly aims at improving the results of the study by S. Feyzabadi and J Schönwälder [5] by further modifying the TBIT program and possibly devising new techniques to successfully identify the congestion control algorithm used on a server by active probing. The paper Host-to-Host Congestion Control for TCP by A. Afanasyev, et al. [2] was extensively used in the course of this research as a reference to the mechanism of work of some current TCP congestion control algorithms.

5 Implementation stage

5.1 Crawler

As previously discussed in section 1 the RCWND growth functions for a significant portion of the hosts probed with tbit in the previous study [5] were left unidentified due to various errors. Most significantly, probing a very large number of hosts resulted in tbit getting HTTP 301, 302 and 404 errors, respectively indicating that the requested resource was permanently or temporarily unavailable or the resource could no longer be found. In addition, in many of the unsuccessful cases the probed hosts refused to agree with tbit on setting the MSS size. On other occasions the dataset obtained from the probing was just not large enough to ensure a successful curve fitting and thus identification failed.

Of course, given the dynamic nature of the Internet, it will not be realistic to expect that any of these types of errors can be fully eliminated. However, they can be significantly reduced if intelligent action is taken in the selection and preprocessing of the set of resources to be probed. The approach chosen to minimize the occurrences of the above mentioned errors was to develop a Python-based crawler application which traverses the list of hostnames to be probed and does the following:

- checks if the host is accessible at all
- resolves redirects
- performs a breadth-first search on the host main page and constructs a list of paths to resources of sufficient size (minimum allowed size is given as an argument)
- pretends to be a well-known and supported user agent - Mozilla Firefox 4.0

The desired minimum size of a resource, the required number of such resources to be checked per host, the maximum total number of resources to be crawled per host and the maximum time to be spent on crawling a host for resources are all configurable arguments of the crawler. In case a sufficient number of resources of sufficient size is not found for a host within the time and space limits, the host is just not included in the newly constructed list of

(hostname, list-of-paths-to-resources) tuples. The breadth-first-search is implemented with the help of the *SGMLParser* class from the Python library *sgmllib*, which was used as basis for implementing a parser that extracts the contents of all *<a href=“ “>* and *<img src=“ “>* tags from a web page. The crawler does not simply rely on the "Content-Length" [7] field in the HTTP header of the response to determine the size of the resource as the contents of this field might often be missing or incorrectly set. That's why the crawler tries to read the contents of each resource and then estimate its size in bytes. Despite this crawling is generally quite fast - 7000 hosts can be crawled for about an hour in 7 parallel processes, with a minimum resource size set to 6 kb, maximum number of explored resources per host set to 30, required number of resources set to 3 and timeouts set to 3 minutes.

Setting of the user agent to a widely-used and supported one (*Mozilla/5.0 (X11; Linux i686; rv:2.0) Gecko/20100101 Firefox/4.0'*) seemed to have a significant positive impact on the crawling. When using the default python (reference) user (or user web) agent, very often hosts tried to redirect to a more compact ("mobile") version of the main page. For example www.facebook.com/ redirected to the mobile version of the web page (m.facebook.com/) if the default user agent was used. This page was, of course, much smaller in size and therefore much more likely to be ignored by the crawler even if the resource size threshold was considerably low.

The crawler application makes extensive use of the python libraries *urllib* and *urllib2* for connecting to the remote host, resolving the redirects, reading and determining the size of the crawled resources. The python library *urlparse* was used in order to correctly separate the URLs into their components: hostname, path and query string [6].The path and the query string are concatenated and together identify a resource rooted at the crawled host. For each resource, it is checked if its hostname, as parsed by the function *urlparse.urlparse*, matches that of the host being probed. Special care is also taken in order to avoid loops while crawling a host.

5.2 Modifications done to TBIT

Tbit had to be modified in order to allow for GET requests on resources, rather than on the main page of a host only. Previously tbit only did GET / requests on the probed hosts and did not allow for passing the path to a resource as an argument. However, this modification was relatively straightforward since the *TcpSession* structure extensively used in tbit had some support for such functionality.

In order to guarantee better results and further minimize errors, the timeframe between the crawling and the probing phases should be minimized. Unfortunately, combining both phases into one - crawling a host and probing the obtained resources immediately thereafter, suggests considerable performance trade-offs. This is due to the fact that while multiple instances of the crawler can be run in parallel, only a single instance of tbit can be run at a time as it needs to install suitable firewall rules into the kernel for each probing [11]. This logically implies that 301, 302, 404 and other errors are still expected to occur but, of course, at lower rates.

The user agent problem discussed in 5.1 also concerns the probing phase, as hosts might still try to redirect to a more-compact version when requested by

TBIT if its user agent set to "TBIT". That's why it was extremely important for the user agents in both the crawler and tbit to match in order to ensure the resources collected at the crawling step are actually the ones being probed.

A tool was implemented that traverses a list of hosts and associated resources obtained from crawling and runs the newly modified version of tbit on each resource.

5.3 Improving identification

In the previous study done by S. Feyzabadi and J. Schönwälder [5] a significant portion of the probed hosts were classified as quadratic with respect to the CWND growth function of their congestion control algorithms during the congestion avoidance phase. Congestion control algorithms with such properties are rarely, if ever, discussed in scientific publications. One possible explanation for the high occurrences of these cases might be that the approach used during the curve-fitting stage could have been considerably noisy.

Often, during the process of probing with TBIT, the congestion control algorithm of the probed host appears to detect multiple congestion events of reach the network limit multiple times. After the detection of the beginning of the congestion avoidance phase, if curve fitting is performed on the whole section of the dataset after this point, the fitted polynomial may not describe the RCWND evolution during the congestion avoidance phase precisely enough. This can be illustrated by the following example shown below in Fig. 10. The dataset used was obtained in a controlled environment using an implementation of TCP Reno.

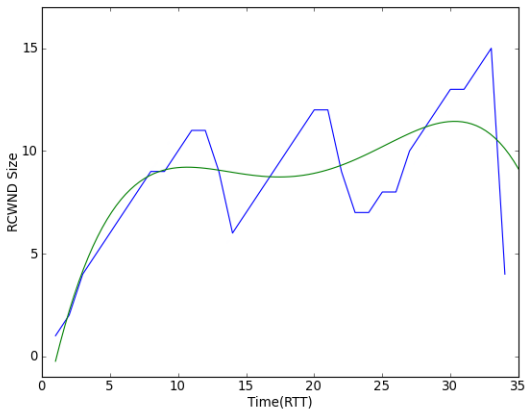

Figure 10: Extracted section of RCWND evolution for Reno after last slow start (blue) and fitted polynomial (green)

Obviously in the observed cases, performing curve fitting on the whole dataset might not be effective in correctly identifying the features of the probed congestion control algorithm. One possible way to alleviate this problem is to locate the longest non-decreasing section of the dataset that appears after the last detected slow start and then use the latter to perform curve fitting. A script

that does that was implemented and integrated into the identification tool. An example showing the obtained section and the polynomial of fourth degree fitted is shown below on Fig. 11.

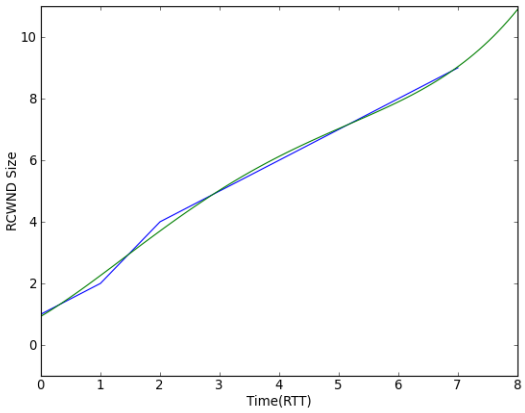

Figure 11: Extracted longest increasing section of RCWND evolution during congestion avoidance for Reno (blue) and fitted polynomial (green)

This simple preprocessing step should significantly diminish the cases of misclassification and expectedly the number of congestion control algorithms classified as quadratic.

The curve fitting was performed using the *polyfit* function from the *numpy* library for numerical computations for Python. A polynomial of fourth degree was fitted to the measured RCWND values of a congestion avoidance phase. The degree of the symbol with the most effective coefficient determines the class of the probed congestion control algorithm - linear, quadratic, cubic or higher. Since the curve fitting is always done with a polynomial of fourth degree, a dataset of at least five points is needed. That's why if the longest non-decreasing segment that appears in a dataset is smaller in length than five, the whole section appearing after the last detected slow start is taken into account for probing.

In case the section of the dataset appearing after the last detected slow start is smaller in size than five, the congestion control algorithm is classified as ïnot enough data:.

The approach towards locating the beginning of the congestion avoidance phase remains unchanged from the one described in the study [5]. Since the CWND size doubles every RTT during slow start [2], as long as the following equation holds, slow start is located.

$$log(2) = log(RCWND_n) - log(RCWND_{n-1}) \tag{2}$$

Since such increase might also occur during congestion avoidance, the logarithms of three consecutive window sizes are taken into account. The identification algorithm tries to locate the last of all consecutive sections of the RCWND evolution dataset that show such increase rate. Then it designates its ending point as the beginning of congestion avoidance.

It is important to note that the approach towards identifying the slow start regions should work only under the assumptions that most of the used congestion control algorithms adhere to the standards specified in RFC 3390 [3]. There is some evidence that some hosts (especially www.google.com/) always send a large number of packets initially in order to speed up the time for reading web pages.

Some of the probed hosts can be characterized by a rather unusual behavior. Their CWND sizes stay constant for a long period throughout the probing. In some cases there are multiple such segments of constant RCWND size obsevered in the datasets obtained in the probing. This suggests that the TCP version used on those hosts just alternates between a few predefined values for their CWND size without showing any polynomial relation of the expected type. An instance of the above-mentioned class of datasets is shown on Fig. 12.

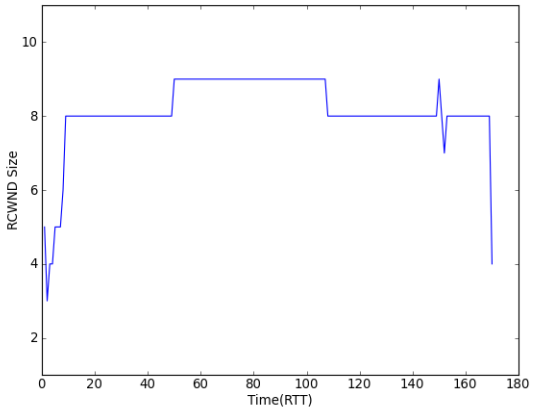

Figure 12: RCWND evolution for the host www.games.com/ showing non-friendly TCP behavior

In order to ensure the successful identification and classification of congestion control algorithms by fragmenting and curve-fitting on the CWND size versus RTT, all such instances need to be filtered out from the rest in a step preceding the fragmentation. This can effectively be done by setting a threshold on the minimum length of sections of constant CWND sizes which appear in these datasets. We refer to instances showing this behavior as a separate class of congestion control algorithms having constant CWND size.

6 Experimental setup

The experiment aiming to identify the aggressiveness of a selected set of high-rank web sites proceeded in three separate stages. The first step consisted of selecting the sample set of hosts and crawling that sample in order to find resources that would be large enough for crawling. The hosts for which such resources could not be found were ignored in the following steps. The second step consisted of probing the obtained list of (hostname, list-of-paths-to-resources)

tuples at three different MSS values (100, 200 and 600). The last stage consisted of processing the results from probing, analyzing the error distributions and identifying the congestion control algorithms associated with the datasets obtained from probing. The results of each stage are discussed in section 7.

6.1 Selection of hosts and crawling

The Alexa list of top 1 million websites was used to select the sample for the experiment. The top 7000 highest ranked hosts were selected for crawling. As the probing was to be done at three different MSS values (100, 200 and 600), two separate crawlings were done with different parameters. The minimum resource size was set to 12 kb for MSS = 100 and MSS = 200 and 40 kb for MSS = 600. This was to ensure that in case the remote host agrees with TBIT on setting the MSS size during the probing stage, a suffiently large dataset could be obtained. The maximum number of resources to explore on a host was in all cases set to 40 and the maximum time for crawling was set to 3 minutes per host.

6.2 Probing with TBIT and processing of the results

Once the three lists of (host, list-of-resources) tuples were obtained from the crawling stage, each was probed with TBIT with the corresponding MSS size and the dataset obtained from each resource rooted at each host was separately stored. The probing was done in parallel on 3 xen virtual machines each having a separate instance of the BSD package scanner.

After the probing, a list of datasets for each host was obtained, each of the datasets corresponding to one probed resource rooted at that host. In the following step, the best (longest) dataset for each host was determined and associated with that host for the identification stage.

6.3 Identification

Since the curve fitting algorithm to be used in this next stage requires at least five points in order to successfully fit a polynomial of fourth degree to the data, all hosts for which the longest dataset was smaller in length than 5 were ignored in the following steps and classified as "not enough data". In the next step, all datasets that contain a contiguous region of constant CWND size were classified as "constant". The minimum length of a constant region to be looked for was set to five for all three dataset lists (MSS = 100, MSS = 200, MSS = 600). The identification algorithm was then run on each list (MSS = 100, MSS = 200, MSS = 600) of datasets. The beginning of congestion avoidance was located for each dataset and in case the number of values after this point was smaller in size than 5, the associated algorithm was classified as "not enough data". If enough points were present after the last slow start, longest non-decreasing region was located and a polynomial of fourth degree was fitted to it. In case a non-decreasing region of sufficient size was not found, the whole section of the dataset after the beginning of congestion avoidance was chosen for fitting.

7 Results

This section presents the results from each stage of the experimental study seeking to identify the congestion control algorithms used on a set of high-rank web hosts.

7.1 Crawling stage

7000 hosts were crawled twice in order to obtain two sets of hostnames with a list of resources rooted at each. Please refer to section 6 for details on the parameters. For the first set, intended to be used for probing at MSS = 100 and MSS = 200, 972 of the initial set of hosts were ignored as no resources of sufficient size were found in the required time and space bounds. The number of hosts to be probed at MSS = 100 and MSS = 200 was 6028, yielding 26306 resources of sufficient size in total. In the second set, intended to be used for probing at MSS = 600, ignored a greater number of hosts - 1130. That was the expected outcome, as the minimum required size for a resource was considerably higher for this set. The number of hosts to be probed at MSS = 600 was 4861, yielding 11891 resources of sufficient size in total. The results from the crawling stage are summarized in Table 1.

	MSS = 100	MSS = 200	MSS = 600
Total number of hosts	7000	7000	7000
Crawled resources of req. size	26306	26306	11891
Hosts selected after crawling	6028	6028	4861

Table 1: Results from the crawling stage

7.2 Probing stage

The resources obtained at the crawling stage were probed with TBIT with the associated MSS values. First set (containing 26306 resources and intended to be probed at MSS = 100 and MSS = 200) produced results for 4811 hosts when probed at MSS = 100 and 5173 hosts when probed at MSS = 200. The results were scanned and only the best (longest) dataset obtained for each host was associated with it. Out of the ones probed at MSS = 100, 4465 hosts had their best dataset longer than five points. For the ones probed at MSS = 200 the number was 4801. Probing the second set (containing 11891 resources and intended to be probed at MSS = 600) produced results for 4573 hosts. Out of these, 4213 had their longest dataset longer than five points. These results are summarized in Table 2. Those having no dataset longer than 5 points counted towards the “not enough data“ class in the identification stage.

The distributions of dataset lengths of the longest dataset for each host 15 for probing at MSS = 100, 200 and 600 respectively. Only sizes in the range (0 - 100) where shown on the figures for readability. The ones not shown follow the same trend in all three cases.

	MSS = 100	MSS = 200	MSS = 600
Hosts for which at least one dataset was obtained	4811	5173	4573
Hosts with longest dataset having at least 5 points	4465	4801	4213
Hosts selected after crawling	6028	6028	4861

Table 2: Results from the probing stage

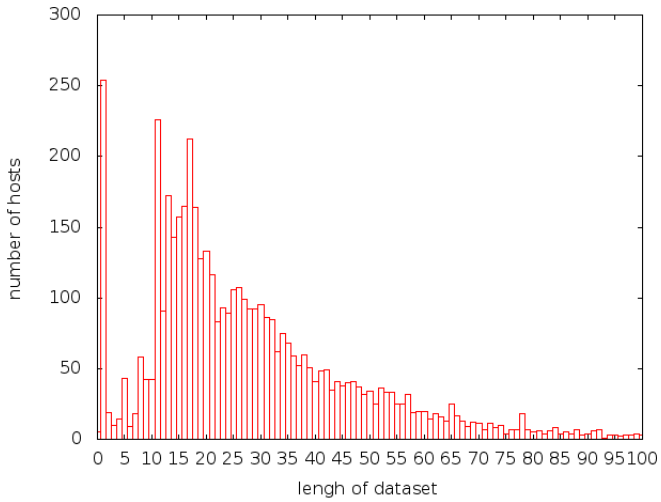

Figure 13: Distribution of hosts with respect to the size of their RCWND evolution dataset. (MSS = 100)

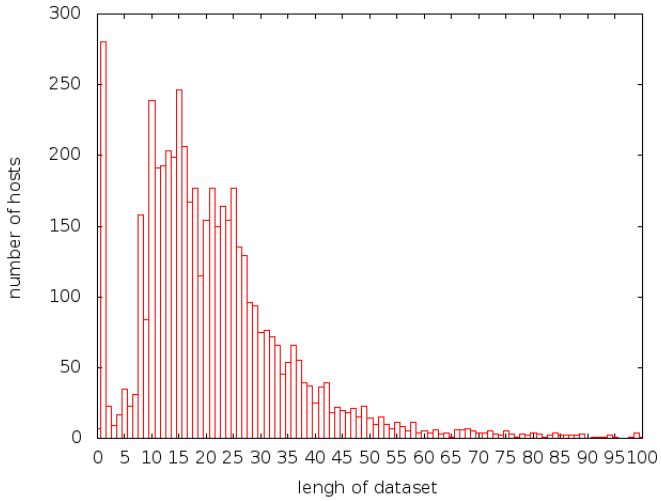

Figure 14: Distribution of hosts with respect to the size of their RCWND evolution dataset. (MSS = 200)

The results from the probing stage differed from what was expected. The results from the set probed at MSS = 600 yielded the smallest number of failures

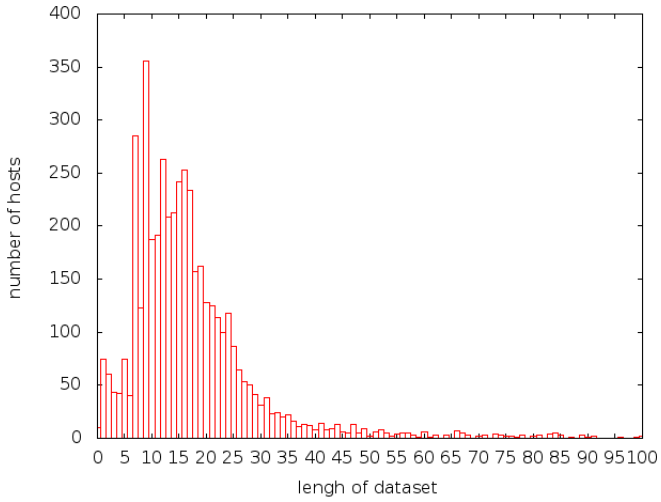

Figure 15: Distribution of hosts with respect to the size of their RCWND evolution dataset. (MSS = 600)

- 324 (number hosts for which no resources could be successfully probed). Those for the ones probed at MSS = 100 and MSS = 200 yielded 1217 and 855 failures, respectively. Although still acceptable, these values were considerably higher than the expected. They directly count towards the "not enough data" class in the identification stage.

In order to investigate the reasons for those failures we looked at the error distributions for each of the three probing sets (MSS = 100, 200 and 600). The values are shown in Table 3.

Reason	MSS = 100	MSS = 200	MSS = 600
Failure to connect (404)	2%	3%	10%
Exceeding MSS	56%	57%	10%
Moved permanently (301)	3%	7%	12%
Moved temporarily (302)	4%	7%	11%
Too Many Packets	27%	14%	45%
Others	8%	12%	12%

Table 3: Distribution of failure types

As expected, the 404, 301 and 302 HTTP errors appeared at much lower rates than in the previous study [5]. This was mostly due to the redirect resolving features of the crawler. The MSS errors were among the most frequent for probing of the MSS = 100 (56 %) and MSS = 200 (57%) sets. This number is significantly decreased when probing was done at MSS = 600. This observation makes sense because it is more likely that that host will refuse to agree with TBIT on setting the MSS when the value is smaller. This ultimately leads to the conclusion that generally it makes sense to probe at MSS values higher than

200. However, looking at the distribution of the "Too Many Packets" errors (highest for MSS = 600 - 45%) suggests that an upper bound to the size of the crawled resources should also be set in order to prevent TBIT from failing due to getting a number of packets exceeding its limit of 10000. This is definitely to be taken into account with regard to improving the crawler application. The "other" errors generally appear at low rates. They feature: HTTP 403, 400, 500, 307 and 503 among others.

7.3 Identification stage

During this stage, the identification script was run on 4465, 4801 and 4213 datasets longer than 5 points, each from the probing at MSS = 100, MSS = 200 an MSS = 600. 154, 218 and 251 hosts were identified as constant in their CWND growth for MSS = 100, 200 and 600, respectively. These hosts were in the range of 3 - 5% in all three cases. Most of the hosts (around 60% in all three cases) were identified as linear. About 15% in all three cases were identified as quadratic. Almost no hosts running cubic congestion control algorithms were identified (in the range of 0 - 0.06%) and no algorithms of higher degrees were identified (0% in all three cases). The number of hosts, for which the section of datasets obtained after the identification of slow start did not contain at least 5 points was added to the number of such hosts identified at the previous stage, to form the "not enough data" class. The number of hosts for which there was not enough data revolves around 20%. This was higher than what was expected but still considerably lower than the results of the previous study [5]. In the latter, the portion of the hosts classified as "Not enough data" reached 44.6 % and 58.7% for MSS = 100 and MSS = 200, respectively.

The summarized results are shown in Table 4. Figures 16, 17 and 18 illustrate the distribution of the identified classes of congestion control algorithms. Since no algorithms exhibiting growth of degree higher than three, the "higher degrees" class was disregarded in those figures.

Polynomial class of TCP algorithm	MSS = 100	MSS = 200	MSS = 600
Constant	154 (3%)	218 (4%)	251 (5%)
Linear	3473 (58%)	3426 (57%)	2883 (59%)
Quadratic	774 (13%)	989 (16%)	769 (16%)
Cubic	2 (0.03%)	4 (0.06%)	0 (0%)
Higher degree	0 (0%)	0 (0%)	0 (0%)
Not enough data	1625 (27%)	1391 (23%)	958 (20%)
Hosts selected after crawling	6028	6028	4861

Table 4: Identification results

The results from the study done by S. Feyzabadi and J. Schönwälder [5] are enclosed below for comparison (Figures 19 and 20).

Although the aim of minimizing the portion of unidentified algorithms from the results of the previous study was fulfilled, the current approach towards identifications only manged to classify very few, in some cases even none, of

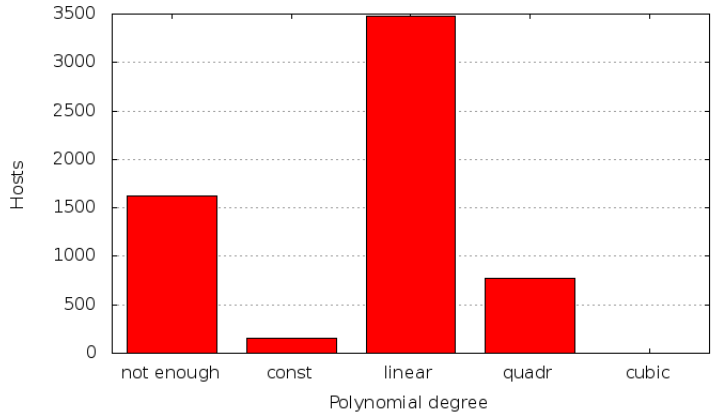

Figure 16: Distribution of web sites with different polynomial degrees of their CWND growth function. (MSS = 100)

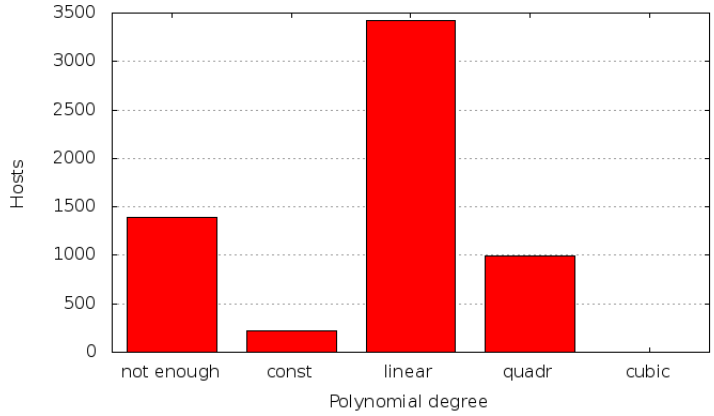

Figure 17: Distribution of web sites with different polynomial degrees of their CWND growth function. (MSS = 200)

the algorithms as “cubic“. This was not expected and suggests that more effort needs to be put into increasing the accuracy of the identification method, although in theory it should be more precise than the original method that was used in the previous study [5]. It is likely that the current identification method still misclassifies a lot of the “cubic“ cases as either “linear“ or “quadratic“.

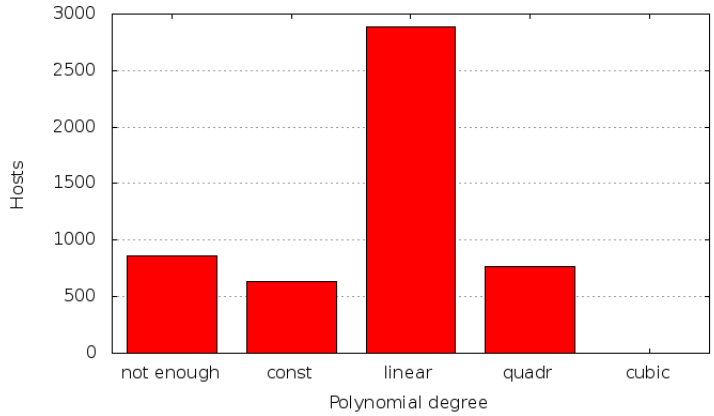

Figure 18: Distribution of web sites with different polynomial degrees of their CWND growth function. (MSS = 600)

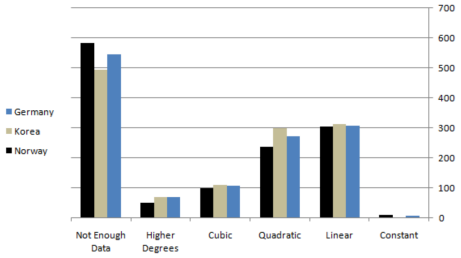

Figure 19: Distribution of web sites with different polynomial degrees of their CWND growth function (MSS = 100) [5]

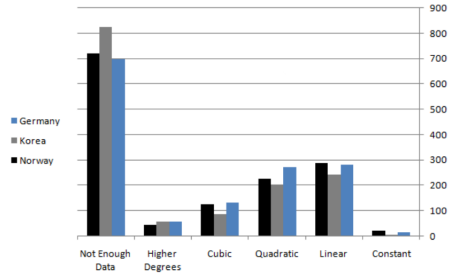

Figure 20: Distribution of web sites with different polynomial degrees of their CWND growth function (MSS = 200) [5]

8 Conclusion

This study aimed at dealing with the shortcomings of the previous study [5] in identifying the congestion control algorithms used on a set of high rank web hosts by active probing. The study [5] failed to identify a large portion of the hosts due to various HTTP errors (301, 302, 404 in particular), MSS-related

failures and possible cases of misclassification during the identification step. The current study tried to address each of these problems separately and managed to minimize the rate of occurrence of some of these, in particular that of HTTP errors. In the course of study, several new approaches to identification and probing were proposed and implemented. The study did not fully resolve the MSS-related issues but managed to examine the problem in detail and propose a possible solution. The cases in which the data obtained from probing was not sufficient to identify an algorithm were effectively diminished by more than 50 %. Although, theoretically more precise, the new approach to identification did not manage to identify almost any cases of congestion control algorithms exhibiting cubic growth. This leaves room for further investigations in seeking new approaches to precisely identify congestion control algorithms with respect to the rate of their CWND size growth.

References

[1] Transmission control protocol. *ISI*, RFC 793, September 1981.

[2] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock. Host-to-host congestion control for tcp. *Communications Surveys Tutorials, IEEE*, 12(3):304 –342, 2010.

[3] M. Allman, S. Floyd, and C. Partridge. Increasing tcp's initial window. RFC 3390, 2002.

[4] Kevin Fall and Sally Floyd. Simulation-based comparisons of tahoe, reno and sack tcp. *SIGCOMM Comput. Commun. Rev.*, 26:5–21, July 1996.

[5] Seyedshams Feyzabadi and Juergen Schoenwaelder. Identifying tcp congestion control algorithms using active probing.

[6] R. Fielding. Relative uniform resource locators. RFC 1808, 1995.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1. RFC 2616, 1999.

[8] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev.*, 42:64–74, July 2008.

[9] V. Paxson M. Allman and E. Blanton. Tcp congestion control. *ISI*, RFC 5681, September 2009.

[10] Steven McCanne and Van Jacobson. The bsd packet filter: a new architecture for user-level packet capture. In *Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings*, pages 2–2, Berkeley, CA, USA, 1993. USENIX Association.

[11] Jitendra Pahdye and Sally Floyd. On inferring tcp behavior. *SIGCOMM Comput. Commun. Rev.*, 31:287–298, August 2001.

[12] V. Paxson, M. Allman, S. Dawson, W. Fenner, J. Griner, I. Heavens, K. Lahey, J. Semke, and B. Volz. Known tcp implementation problems. RFC 2525, 1999.