TCP Congestion Control Comparison

Arne Esterhuizen

20 Janaury 2012

Abstract

The purpose of this report is to investigate the effects that different TCP variants have on each other. These TCP variants differ by the congestion control algorithms they employ. Congestion control algorithms affect how much network traffic is generated by TCP at any one time and aims to prevent a TCP connection from over utilising the network. This report investigates the different congestion control algorithms that are included as loadable modules in the Linux kernel, and discusses experiments performed on how these congestion control algorithms compete for network resources. It will be shown that some TCP variants are able to co-exist, whilst others tend to use excessive bandwidth, potentially smothering other competing TCP connections. Should the results prove interesting, the project may be given to third year computer network students to perform as an assignment.

Contents

1 Introduction **1**

2 Experiments performed **2**

3 Results **3**

3.1 HTCP 3
3.2 Hybla 3
3.3 Illinois 4
3.4 LP 4
3.5 Vegas 4
3.6 Reno 6
3.7 BIC 6
3.8 Westwood 6
3.9 YeAH 6
3.10 Cubic 6
3.11 Highspeed 7
3.12 Scalable 7

4 Technical Details **8**

5 Conclusion **8**

1 Introduction

The *Transmission Control Protocol* (TCP) controls how much data it transmits over a network by utilising a congestion window. TCP cannot send more data than the congestion window allows. The size of TCP's congestion window depends on the condition of the network. In the case of the network experiencing heavy data traffic, the congestion window would be shortened. In the case of the network being only lightly loaded with data traffic, the congestion window would be

expanded. How and when to adjust the congestion window depends on whatever form of congestion control a TCP connection utilises. Congestion control algorithms rely on various indicators to passively ascertain the state of the network. Packet loss is an indication that the network is being over utilised and that routers along the way are dropping IP packets due to limited buffer space. Also, routers can set special flags in an IP packet's header that will inform the receiving host that congestion is about to occur. The receiving host can then inform the the sending host to reduce its sending rate. Other methods include closely observing packet round trip times as well as determining the queueing delay, the time packets spend waiting in a router's buffer. Using one or more of these methods, congestion control algorithms are able to adjust their behaviour according to whether congestion has occurred or could occur. As will be seen, some congestion control mechanisms allow for unfair usage of network bandwidth, while others are able to share bandwidth equally.

The different congestion control mechanisms available for use by the Linux kernel are:

- HTCP
- Hybla
- Illinois
- LP
- Vegas
- Reno
- BIC
- Westwood
- YeAH
- Cubic
- Highspeed
- Scalable

The linux socket interface allows one to change the type of congestion control a TCP connection uses by setting the appropriate socket option.

Experiments were performed to see how these algorithms compete against each other for bandwidth. A description of these experiments is given in Section 2, followed by a discussion of the results in Section 3. Technical difficulties that may affect the practicality of this project as an assignment for students is described in Section 4. Section 5 summarises the presented results.

Throughout this report the words bandwidth and throughput are used interchangeably. The terms TCP variant and congestion control algorithm are also used interchangeably.

2 Experiments performed

Experiments were carried out using client and server programs. A host running a client program generates data which is sent over the network to a host running the server program. A single server host receives data from various client hosts. Each client host is set to use a different congestion control algorithm. The amount of data received from client hosts is measured in megabits per second and is presented as a graph. This shows how much bandwidth the client host is able to utilise. The mean of a client's bandwidth measurements is used to determine it's bandwidth

usage in general. In these experiments, the highest possible throughput on the network is slightly over 100 Mbps.

The experiments were run on the Stellenbosch University network. All hosts involved were located on the same network segment. All experiments were carried out in the same manner, and all TCP variants were tested in pairs. Host A (a client) transmits data to host B (the server), after which host C (another client) also commences sending data to host B. Thus the two client hosts do not commence with data transmission simultaneously, there is always a short delay between one client beginning its transmission and the next client beginning its transmission. Therefore, the first client to commence transmission always utilises the network freely, and is only forced to share bandwidth as soon as an additional client commences transmission. This was found to have an impact on the behaviour of certain TCP variants, as their bandwidth utilisation appeared to change depending on the order in which they were allowed to start (refer to section 3.3).

Every TCP congestion control algorithm was tested against all other algorithms, including itself. This was done in order to establish whether TCP variants could be placed into groups in which all members are well behaved.

3 Results

As stated in section 2, all TCP variants were compared to each other. The results of these experiments will be discussed in this section for each TCP variant individually. For each experiment, the mean of each TCP variant's bandwidth measurements is calculated and tabularised. This allows one an idea of what the average bandwidth usage of a TCP variant is.

3.1 HTCP

From Table 1, HTCP is able to co-exist with itself. HTCP has the lower throughput against Illinois, BIC, Westwood, and Scalable. HTCP is stronger against Hybla, LP, Vegas, Reno, YeAH, Cubic, and Highspeed. Against these it maintains a mean throughput in the range of 64 to 67 Mbps. The exception is vegas, against which HTCP has a mean throughput of about 95 Mbps.

Mean Throughput (Mbps)		
HTPC	Other	
52	52	HTCP
66	37	Hybla
38	65	Illinois
65	38	LP
95	10	Vegas
66	37	Reno
41	62	BIC
19	86	Westwood
64	40	YeAH
67	37	Cubic
64	40	Highspeed
41	64	Scalable

Table 1: Mean throughput of HTCP against other TCP variants.

3.2 Hybla

Referring to Table 2, Hybla seems to be able to co-exist with itself, LP, Reno, Cubic, and YeAH. Hybla seems to do badly against Illinois, BIC, Westwood, and Scalable.

Mean Throughput (Mbps)		
Hybla	Other	
37	66	HTCP
52	51	Hybla
33	70	Illinois
48	54	LP
94	10	Vegas
49	54	Reno
28	75	BIC
14	90	Westwood
46	57	YeAH
53	50	Cubic
40	63	Highspeed
29	74	Scalable

Table 2: Mean throughput of Hybla against other TCP variants.

3.3 Illinois

As can be seen from Table 3, Illinois is very unfriendly, and is unable to fairly share bandwidth with any other congestion control algorithm. The only algorithm to maintain a higher mean throughput than Illinois is Westwood. Illinois has the curious trait of not being able to co-exist with even itself, something all of the other congestion control algorithms are capable of.

In Section 2 it was stated that in some cases the order in which clients are allowed to begin transmission influences the bandwidth utilised. Illinois is one algorithm for which this applies. In Table 3, Illinois is allowed to begin transmission first, and even after a second client begins transmitting data, Illinois still maintains a very high mean throughput. However if the order of transmission is reversed, the competing client is allowed much more bandwidth, even if it is still less than Illinois. This is illustrated by Figures 1 and 2. Illinois shows similar behaviour with all other TCP variants, with the exception of Westwood and Vegas, whose mean throughput remains the same ragardles of the order in which they start transmitting.

Mean Throughput (Mbps)		
Illinois	Other	
65	38	HTCP
70	33	Hybla
13	89	Illinois
95	7	LP
97	6	Vegas
95	8	Reno
93	10	BIC
34	69	Westwood
93	10	YeAH
94	9	Cubic
94	9	Highspeed
88	22	Scalable

Table 3: Mean throughput of Illinois against other TCP variants.

3.4 LP

From Table 4, it is clear that LP is able to co-exist with itself, Hybla, Reno, YeAH, and Cubic. Just as with Hybla, LP fares badly against Illinois, BIC, Westwood, and Scalable.

3.5 Vegas

From Table 5, TCP Vegas is able to co-exist with itself. Vegas performs very poorly against all other TCP variants, achieving a mean throughput of anything between 8 and 13 Mbps, depending on which TCP variant

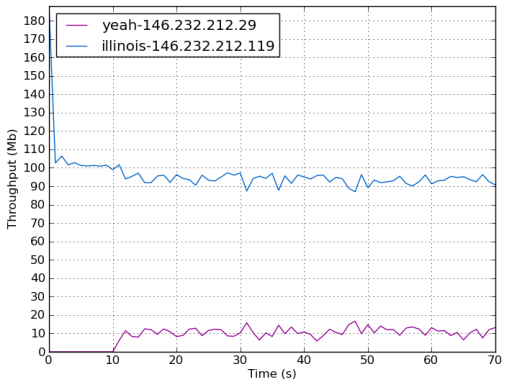

Figure 1: Illinois commences transmission first, and is joind by YeAH after 10 seconds. Illinois maintains a mean throughput of 93 Mbps, while YeAH maintains a very low mean throughput of 10 Mbps.

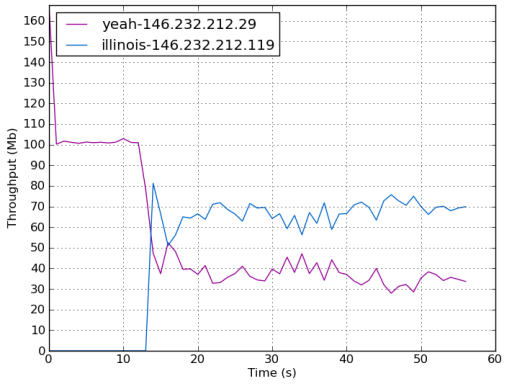

Figure 2: YeAH is now allowed to begin transmission before Illinois. Although still less than Illinois, YeAH now maintains a much higher than before mean throughput of 37 Mbps, whilst Illinois has a lower than before mean throughput of 67 Mbps.

Mean Throughput (Mbps)		
LP	Other	
38	65	HTCP
54	48	Hybla
33	70	Illinois
53	50	LP
95	10	Vegas
50	53	Reno
29	74	BIC
13	90	Westwood
50	53	YeAH
55	47	Cubic
42	61	Highspeed
27	76	Scalable

Table 4: Mean throughput of LP against other TCP variants.

it competes against. This is demonstrated by Figure 3.

Mean Throughput (Mbps)		
Vegas	Other	
10	95	HTCP
10	94	Hybla
6	97	Illinois
10	95	LP
49	50	Vegas
9	94	Reno
13	91	BIC
8	95	Westwood
23	95	YeAH
10	94	Cubic
9	95	Highspeed
8	95	Scalable

Table 5: Mean throughput of Vegas against other TCP variants.

3.6 Reno

From Table 6, it is clear that TCP Reno is able to co-exist with itself, Hybla, LP, YeAH, and Cubic. Again, just as with Hybla and LP, Reno fares badly against Illinois, BIC, Westwood, and Scalable.

Mean Throughput (Mbps)		
Reno	Other	
37	66	HTCP
54	49	Hybla
31	73	Illinois
53	50	LP
94	9	Vegas
51	52	Reno
28	75	BIC
15	89	Westwood
49	54	YeAH
54	49	Cubic
40	62	Highspeed
26	77	Scalable

Table 6: Mean throughput of Reno against other TCP variants.

3.7 BIC

From Table 7 BIC is able to co-exist with itself and Scalable. It's mean throughput is high against Hybla, LP, Vegas, Reno, Cubic and Highspeed. BIC seems to be able to co-exist with Illinois as well, but it was shown in Section 3.3 that Illinois' behaviour against other TCP variants cannot be relied upon to remain consistent.

Mean Throughput (Mbps)		
BIC	Other	
62	41	HTCP
75	28	Hybla
53	51	Illinois
74	29	LP
91	13	Vegas
75	28	Reno
54	49	BIC
19	84	Westwood
73	30	YeAH
77	26	Cubic
73	30	Highspeed
55	51	Scalable

Table 7: Mean throughput of BIC against other TCP variants.

3.8 Westwood

From Table 8, it is seen that Westwood is very aggressive. It's mean throughput always remains above 80 Mbps, allowing very little bandwidth for competing TCP connections. The only exception to this rule is Westwood itself, it shares bandwidth with itself more or less equally.

3.9 YeAH

As seen from Table 9, YeAH is able to co-exist with itself, Hybla, LP, Reno, Cubic, and to a lesser degree, Highspeed. Once more, just as with Hybla, LP, and Reno, YeAH fares badly against Illinois, BIC, Westwood, and Scalable.

3.10 Cubic

Table 10 shows that Cubic is able to co-exist with itself, Hybla, LP, Reno, and YeAH.

Mean Throughput (Mbps)		
Westwood	Other	
86	19	HTCP
90	14	Hybla
69	34	Illinois
90	13	LP
95	8	Vegas
90	13	Reno
84	19	BIC
46	57	Westwood
89	14	YeAH
91	13	Cubic
89	15	Highspeed
80	23	Scalable

Table 8: Mean throughput of Westwood against other TCP variants.

Mean Throughput (Mbps)		
YeAH	Other	
40	64	HTCP
57	46	Hybla
37	67	Illinois
53	50	LP
95	23	Vegas
54	49	Reno
30	73	BIC
19	84	Westwood
52	51	YeAH
54	49	Cubic
45	58	Highspeed
31	72	Scalable

Table 9: Mean throughput of YeAH against other TCP variants.

Again, it also shares these TCP variants' weakness against Illinois, BIC, Westwood, and Scalable.

Mean Throughput (Mbps)		
Cubic	Other	
37	67	HTCP
50	53	Hybla
33	70	Illinois
47	55	LP
94	10	Vegas
49	54	Reno
26	77	BIC
13	91	Westwood
49	54	YeAH
47	55	Cubic
40	64	Highspeed
26	77	Scalable

Table 10: Mean throughput of Cubic against other TCP variants.

3.11 Highspeed

Table 11 shows that Highspeed shares bandwidth well with itself and reasonably well with YeAH. Algorithms like Illinois, Westwood, BIC, and Scalable tend to use most of the bandwidth when competing against Highspeed.

3.12 Scalable

Table 12 shows that Scalable is able to coexist with itself, BIC and Illinois. However, as shown is section 3.3, Illinois' behaviour differs depending on what order the congestion control algorithms are launched in. Scalable is also very aggressive and dominates the amount of available bandwidth when competing against other TCP variants. The only exception to this is Westwood, which maintians

Mean Throughput (Mbps)		
Highspeed	Other	
40	64	HTCP
63	40	Hybla
36	68	Illinois
61	42	LP
95	9	Vegas
62	40	Reno
30	73	BIC
15	89	Westwood
58	45	YeAH
64	40	Cubic
54	50	Highspeed
32	71	Scalable

Table 11: Mean throughput of Highspeed against other TCP variants.

a very high mean throughput against Scalable.

Mean Throughput (Mbps)		
Scalable	Other	
62	42	HTCP
74	29	Hybla
50	53	Illinois
76	27	LP
95	8	Vegas
77	26	Reno
51	55	BIC
23	80	Westwood
72	31	YeAH
77	26	Cubic
71	32	Highspeed
51	52	Scalable

Table 12: Mean throughput of Scalable against other TCP variants.

4 Technical Details

In this section, a short overview of the steps taken to successfully utilise different TCP congestion control algorithms is given. Various TCP congestion control algorithms are available in the Linux kernel as modules. One can force Linux to use a specific congestion control algorithm, but one does not wish to be limited to only one type of congestion control algorithm at any time. Fortunately, Linux makes provision for this. The socket interface allows one to set socket options that allows the use of a different TCP congestion control algorithm for each TCP socket that is created. This was successfully done in the Python programming language. It is important to note that for altering the availability of algorithms, setting which algorithm is in use, as well as setting socket options requires sudo rights.

5 Conclusion

Comparing all the congestion control algorithms to each other has shown that whilst some of the algorithms seem to be able to coexist, others cannot. Vegas consistently had a very low mean throughput of about 10 Mbps against all other TCP variants. It is perhaps the algorithm that is most sensitive to congestion on the Network, as it gives up bandwidth the most easily.

The most aggressive algorithms are BIC, Westwood, Illinois, and Scalable. Of these four, Westwood is the most greedy, taking nearly all available bandwidth for itself. It is perhaps the most incensitive to congestion on the network.

The congestion control algorithms capapble of equally sharing bandwidth are Hybla, LP,

Cubic, YeAH, and Reno. Each of these algorithms share the available bandwidth more or less equally with each of the other four. This is demonstrated by Figures 4, 5, 6, 7, 8, 9, 10, 11, 12, and 13.

Given some guidance, students should be able to figure out how TCP congestion control is implemented in the Linux kernel and successfully conduct experiments of their own. However, as stated in Section 4, this can only be done with super user privileges. The results are certainly interesting and should prove to be a good study into TCP congestion control algorithms.

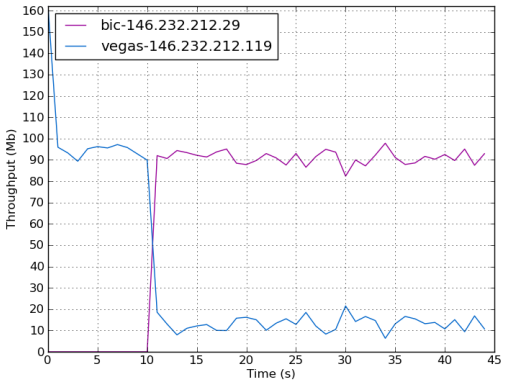

Figure 3: Vegas begins transmission, followed by BIC ten seconds later. BIC immediately achieves a mean throughput of 91 Mbps, while Vegas is reduced to a mean throughput of 13 Mbps. This outcome remains the same regardless of what TCP variant Vegas competes against.

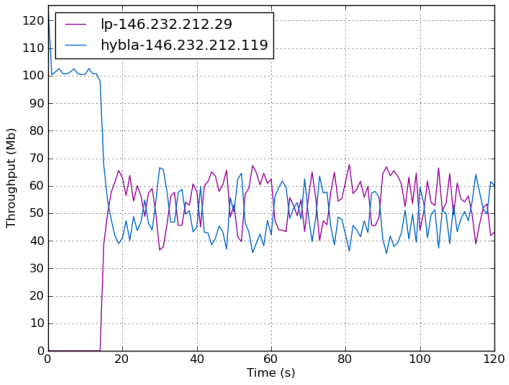

Figure 4: Hybla and LP are able to share bandwidth equally.

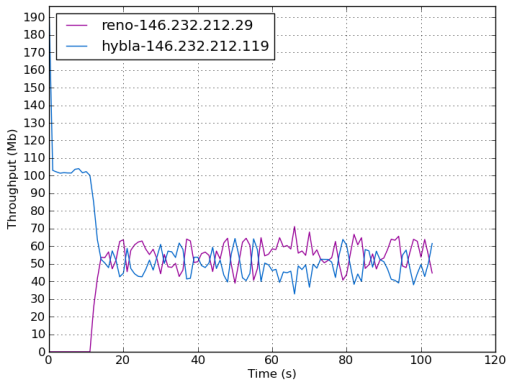

Figure 5: Hybla and Reno are able to share bandwidth equally.

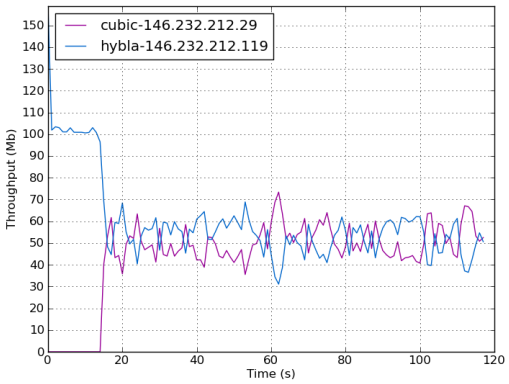

Figure 7: Cubic and Hybla are able to share bandwidth equally.

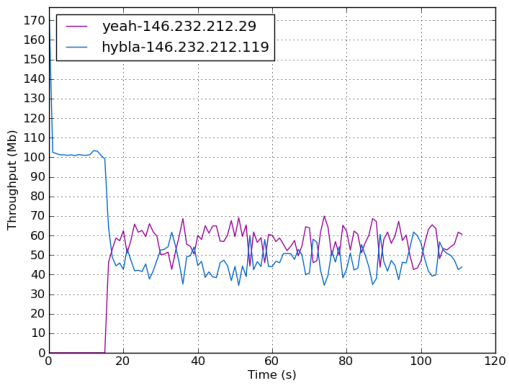

Figure 6: YeAH and Hybla are able to share bandwidth equally.

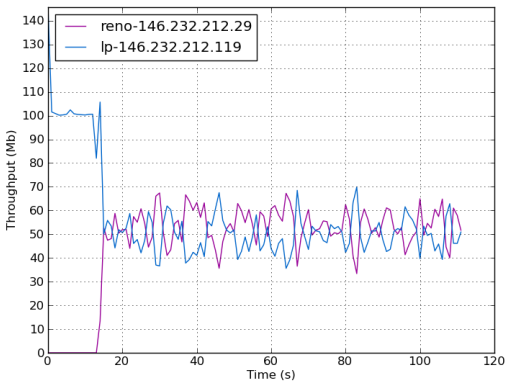

Figure 8: Reno and LP are able to share bandwidth equally.

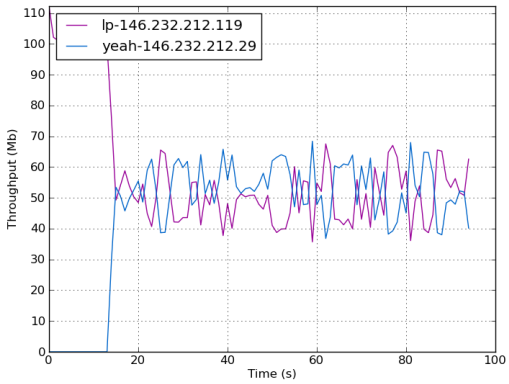

Figure 9: LP and YeAH are able to share bandwidth equally.

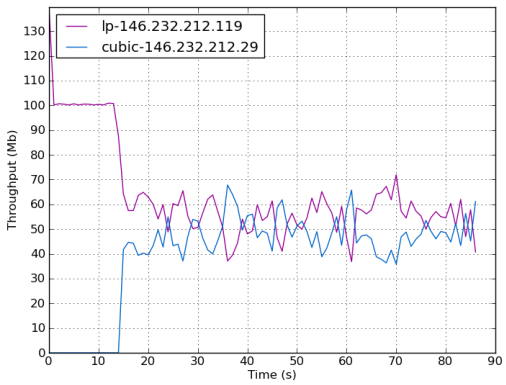

Figure 10: LP and Cubic are able to share bandwidth equally.

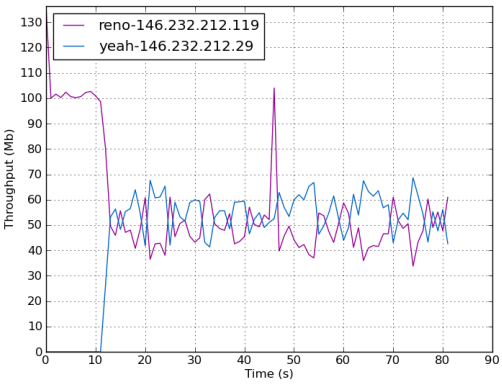

Figure 11: Reno and YeAH are able to share bandwidth equally.

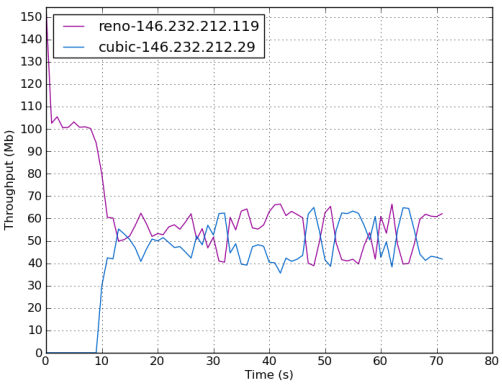

Figure 12: Reno and Cubic are able to share bandwidth equally.

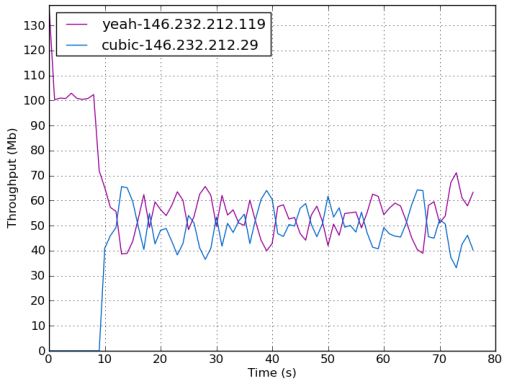

Figure 13: YeAH and Cubic are able to share bandwidth equally.