An experimental evaluation of TCP startup algorithms

How do flow startup mechanisms impact the performance of TCP?

Matei Grigore

Supervisor(s): Fernando Kuipers, Adrian Zapletal

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 13, 2025

Name of the student: Matei Grigore
Final project course: CSE3000 Research Project
Thesis committee: Fernando Kuipers, Adrian Zapletal, Asterios Katsifodimos

Abstract

Most TCP data transfers in the Internet are short. This makes the startup algorithms an important factor that impacts TCP performance. Several startup algorithms have been developed. However, not a lot of research has been conducted into how these behave and interact when used for short flows. This paper aims to provide a thorough evaluation of these algorithms and their interactions under different network conditions, focusing on short flows and using *ns-3*. We have observed that JumpStart seems to outperform the other algorithms used when it comes to flow completion time for short flows. That is likely because it starts to send data with an aggressive initial congestion window and the flow is finished after the first few RTTs. However, JumpStart performs more poorly when the flows are longer. We have shown that JumpStart has a great potential to make communication more efficient in the Internet but further research has to be conducted into its behavior in more adversarial conditions that represent real life situations better.

1 Introduction

The Transmission Control Protocol or TCP [6] is the primary transport protocol used in the Internet. It ensures reliable end-to-end communication between a sender and a receiver. Since it is used in the Internet, TCP is faced with a wide range of different network conditions that impact its performance. It is essential to understand how its performance is affected by different conditions such that improved versions can be developed in the future.

Many congestion control algorithms (CCA) have been developed over the years to try to improve TCP's performance such as CUBIC [9], Vegas [2] or the more recent Google algorithm, BBR [3]. These were later compared to each other, in order to understand their behavior under different conditions and to be able to improve them in the future [19, 22, 11].

Another important part of TCP literature is focused on developing new startup algorithms and analyzing them. Startup algorithms are used by TCP at the beginning of the connection to increase the congestion window (*cwnd*) until it reaches the optimal value and it can properly use the available bandwidth. Several startup algorithms have been developed over the years, such as HyStart [8], JumpStart [13] or the BBR startup [3].

In the Internet, the majority of flows are short, as it was shown in several studies [12, 17]. That is because many user activities correspond to short data transfers such as GET requests, loading small web pages or loading short form content from different social media apps. The large number of small flows in the Internet makes the impact of startup algorithms on TCP performance an increasingly important issue. Since the flows are so small, many would not even get to exit the startup phase before they are done sending. However, there is not enough research conducted into analyzing the performance of different startup algorithms on short flows.

Our research aims to answer the question of "How do flow startup mechanisms impact the performance of TCP?". We achieved this by providing a thorough evaluation of a few of the important startup algorithms, to understand their behavior and in which conditions each algorithm performs better. To this extent, we focused on three algorithms for our experiments: HyStart because it is currently the default startup algorithm used in CUBIC, which is the most common TCP version used in the Internet [14, 21, 15]; BBRv3 because it is the latest algorithm developed by Google and they have already started to use it for their public Internet traffic [4]; and our custom implementation of JumpStart in *ns-3* because its aggressive initial sending rate shows potential when dealing with small flows. To compare them thoroughly, we developed multiple experiments that simulated different real world network conditions. The main performance metric used was flow completion time (FCT) because it is known to best represent user experience [5]. Other metrics were throughput over time, *cwnd* growth and Jain's Fairness Index (JFI) [10] for competing flows.

We discovered that JumpStart outperforms the other two algorithms in FCT, when the flows are short. This makes it a very valuable algorithm to be studied further and potentially improved. This evaluation was important to understand the potential of JumpStart in the Internet and hopefully convince researchers that it is a topic that deserves more attention. It is important that future work will also explore a wider variety of conditions in which JumpStart should be tested, that mimic real world traffic more closely.

2 Background

In academic literature, there has been a lot of research conducted on TCP. Many studies focus on evaluating the performance of TCP under different conditions. This is essential to understand its potential problems and how it can be improved in future versions. There are also several academic papers that introduce new TCP versions or startup algorithms.

Many improvements of the protocol have been proposed over the years, specifically targeting the congestion control algorithms (CCA) used, such as CUBIC [9], Vegas [2], or the latest Google algorithm, BBR (Bottleneck Bandwidth and Round-trip time) [4, 3]. CUBIC is now the most common algorithm used in the Internet and BBR is already used by Google on all their public Internet traffic [4]. This makes the performance of these algorithms and their interactions a very important topic which has been studied extensively in literature [19, 22, 11].

One of the main factors that impact the performance of TCP is the flow startup mechanism used. The default TCP startup algorithm, Slow Start, doubles the *cwnd* every RTT until it exceeds a certain threshold or packets are lost. Several improvements to standard TCP Slow Start were proposed in literature. HyStart [8], is one of these solutions, and uses ACK trains

and RTT delay samples to combat Slow Start's overshoot by finding an early exit point. It has also been implemented as the default slow start for CUBIC in the Linux kernel. Google's CCA, BBR also slightly modifies slow start such that it exits the startup phase when the bottleneck bandwidth estimation plateaus or, the loss or ECN thresholds are exceeded [22]. Another different approach observed in literature is that of JumpStart [13], which completely skips over the startup phase and paces as many packets as the advertised window of the receiver over the first RTT. This also has several drawbacks, but its performance hasn't been studied in depth. One of the latest proposed algorithms, SUSS [1] shows a lot of potential for improving slow start especially for short flows where the small improvements are more relevant. It uses HyStart to predict when the *cwnd* will be increased and it quadruples it instead of doubling and paces extra packets to reduce congestion, speeding up slow start and utilizing more of the available bandwidth [1]. QuickStart [18] has been classified as a network assisted approach to slow start [1] as it uses the network to set an optimal initial *cwnd* size. During the handshake, the sender sends its desired *cwnd* and this is then modified by routers along the path to fit each one of them. Another network approach, P4air [20], uses P4 programmable routers in the network to calculate the fair share of the bandwidth for each flow and forces them to finish slow start with a good *cwnd* by actively dropping packets. A different approach on startup algorithms encountered in literature is a stateful approach such as Stateful-TCP [7]. It makes use of previous information about flows with the same destination to set the initial sending rate, which is useful when several similar requests are made one after another to the same website. A similar approach can be observed in TCP-RL [16] that uses reinforcement learning to set the initial window based on prior transmissions.

All of these algorithms were thoroughly evaluated when introduced. However, not all of them were compared to each other under different conditions. Since HyStart and BBR are currently heavily used in the Internet [14, 21, 15], their interactions with other algorithms is an interesting topic of research. JumpStart is also a promising approach for short flows, which are very common in the Internet [12, 17]. Therefore, we explore and compare the performance of each of these algorithms as well as their interactions, because, to the best of our knowledge, this was not previously done.

3 Methodology

This section will provide a detailed description of the testbed we used for experiments as well as the implementation of this setup and the specific algorithms used. Each experiment will also be described in detail.

3.1 Experiment setup

The experiments evaluated in this paper were run using the *ns-3* network simulator. The network topology used for all of these experiments is visually depicted in Figure 1. It is a dumbbell topology with 2 senders (S1 and S2 in Figure 1) connected to the same bottleneck router (BR1 in Figure 1) using high speed point to point links with a bandwidth of 10 Gbps. The router is connected to another router (BR2 in Figure 1) through a slower point to point link creating a bottleneck whose bandwidth was set to 10 Mbps, 100 Mbps or 1 Gbps depending on the experiment. The second router is then also connected to 2 receivers (R1 and R2 in Figure 1) using high speed links (10 Gbps). The one way delay of the bottleneck link was set to 5 ms, 25 ms or 125 ms depending on the experiment. This setup allows to simulate a realistic Internet environment and observe the different behaviors of each of the startup algorithms used.

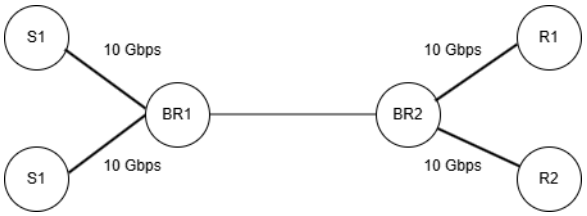


Figure 1: Dumbbell topology used for experiments

The experiments consisted of 1 or 2 TCP flows with small sizes, either 50, 100. 200, 400 KB, as these represent the majority of flows in the Internet [12, 17] and therefore accurately simulated real world scenarios. The flows used either CUBIC with HyStart, BBRv3 or CUBIC with JumpStart as their TCP algorithms.

The experiments with one flow had most parameters fixed except for one, which was either bottleneck bandwidth or RTT. This was set to several different values as mentioned above. One experiment focused on different bottleneck bandwidth values and another on different RTT values. The aim of this approach was to test for each startup algorithm, in which conditions it thrives and compare their performance with all the other algorithms. Their performance was measured using different metrics such as flow completion time (FCT), throughput and *cwnd* increase over time.

The other type of experiments was the one with 2 flows competing for resources. Three experiments of this kind were performed. For these the main network parameters remained the same, but the startup algorithm used differed. The parameters used were 100 Mbps bottleneck bandwidth and 50 ms RTT. One experiment featured a long lived flow (100 MB) and a short lived flow (100 KB). The long flow was started before the other one and continued after that was done sending data. The long flow used CUBIC while the algorithm of the short flow was varied from run to run. The aim of this experiment was to observe the interaction of these short flows with the already existing traffic, in a situation that could be frequently encountered in the Internet. The performance of the short lived flow was measured with the metrics used before, but additionally also using the Jain's fairness index (JFI). The other two experiments of this kind featured 2 medium sized flows (both 10 MB) started at the same time. One experiment compared the interactions of flows having different startup algorithms while the other focused on flows with the same one. This aimed to analyze the fairness of these algorithms when flows compete for

resources.

3.2 Implementation

As mentioned above, the experiments were run using the *ns-3* network simulator. To accomplish this, code had to be written both for the experiment as well as for extending the *framework* with implementations of algorithms that were not present yet. This code can be found in our repository[1], a fork of the official *ns-3* repository[2]. In particular the code used for these project can be found on the *startup-algorithms* branch.

For the framework extension part, we added two algorithms to its *internet* model. One is our custom implementation of the JumpStart algorithm on top of the CUBIC implementation already present. This consisted in copying the code of the *TcpCubic* class to other files, *tcp-cubic-jumpstart.h* and *tcp-cubic-jumpstart.cc*, and adding the algorithm specific logic. The implementation simply sets the initial *cwnd* to the advertised window of the receiver and paces these packets over the first RTT before reverting back to the CUBIC algorithm. The other implemented algorithm was BBRv3. This implementation was taken from the open source community and can be found on this repository[3]. The code was added, in our repository, to *tcp-bbrv3.h* and *tcp-bbrv3.cc*, respectively.

The simulation script was built on top of a common setup within our project group and we added project specific features. The script sets up the dumbbell topology using *ns-3*'s *PointToPointDumbbellHelper*. All of the parameters of the network can be easily changed using this setup. Furthermore, we implemented logging for several important metrics mentioned above such as FCT, throughput, *cwnd* over time. We aggregated the logs into plots using python scripts.

4 Evaluation

This section will provide a thorough analysis of the results obtained from the experiments described above and their corresponding visual representations.

4.1 Varying bandwidth and size

This experiment tested each of the startup algorithms mentioned above in the dumbbell topology presented with an RTT of 50 ms with different bandwidth values (10 Mbps, 100 Mbps, 1 Gbps) and different flow sizes (50 KB, 100 KB, 200 KB, 400 KB).

An observation that can be made based on the observed FCTs in Table 1 is that the JumpStart algorithm always performs significantly better under the given conditions than its counterparts. That is probably due to its aggressive initial *cwnd* that allows it to occupy more bandwidth earlier. CUBIC outperforms BBRv3 consistently because its startup algorithm is more aggressive than BBR's, however, the difference in FCTs is considerably smaller than the difference between JumpStart and CUBIC. The differences between FCTs of JumpStart and CUBIC are often around 100 or 150 ms, with the highest observed being 154 ms, when the bandwidth is set to 10 Mbps and the flow has 400 KB. In the case of CUBIC and BBRv3 the differences are usually of 10 or 15 ms with some even below 10 ms, for example, in the case where the bandwidth is 10 Mbps and the flow has 400 KB.

In terms of throughput, it can be observed in Figure 3 that JumpStart again achieves much higher throughput than its counterparts when flows are shorter (50 KB and 100 KB). That is not always the case when flows are bigger. For example, when the bandwidth is 10 Mbps and the flow has 200 KB, JumpStart reaches the same maximum throughput as the other algorithms. When the flow has 400 KB, it achieves lower throughput than the other algorithms. However, it can be observed that JumpStart has a very different behavior than the other 2 algorithms. This can be attributed to its aggressive initial congestion window, as seen in Figure 2. Thanks to this specific characteristic, the algorithm manages in most cases with short flows to achieve a higher throughput earlier, and therefore utilizing more of the available bandwidth, compared to the careful approach of CUBIC and BBRv3. In Figure 2 for flows of 50 KB, the JumpStart *cwnd* is not present in the plots because the initial window is larger than the size of the flow, so it is never increased.

4.2 Varying RTTs

The second experiment performed is similar to the previous one, but this tested the behavior of algorithms in a setting with fixed bandwidth (100 Mbps) and flow size (100 KB) but with varying RTTs (10 ms, 50 ms, 200 ms).

A similar trend in FCTs as in the previous experiment can be observed here as well. Looking in Table 2, it can be seen that JumpStart outperforms CUBIC and BBRv3 under these conditions too. The same ranking of performance as observed previously is apparent in this experiment, as well with JumpStart being fastest, followed by CUBIC and lastly BBrv3. The differences in FCTs between JumpStart and CUBIC are approximately 40 ms, 140 ms, and 520 ms corresponding to RTTs of 10 ms, 50 ms and 200 ms respectively. In the case of CUBIC and BBRv3 smaller differences can be observed, respectively, 5 ms, 25 ms, and 100 ms.

The throughput measurements in Figure 5 and of *cwnd* increase in Figure 4 are very similar to the patterns observed and explained above in experiment 1. JumpStart achieves a higher throughput than its counterparts and earlier, making it more efficient in the situations observed above, as also supported by the measured FCTs in Table 2.

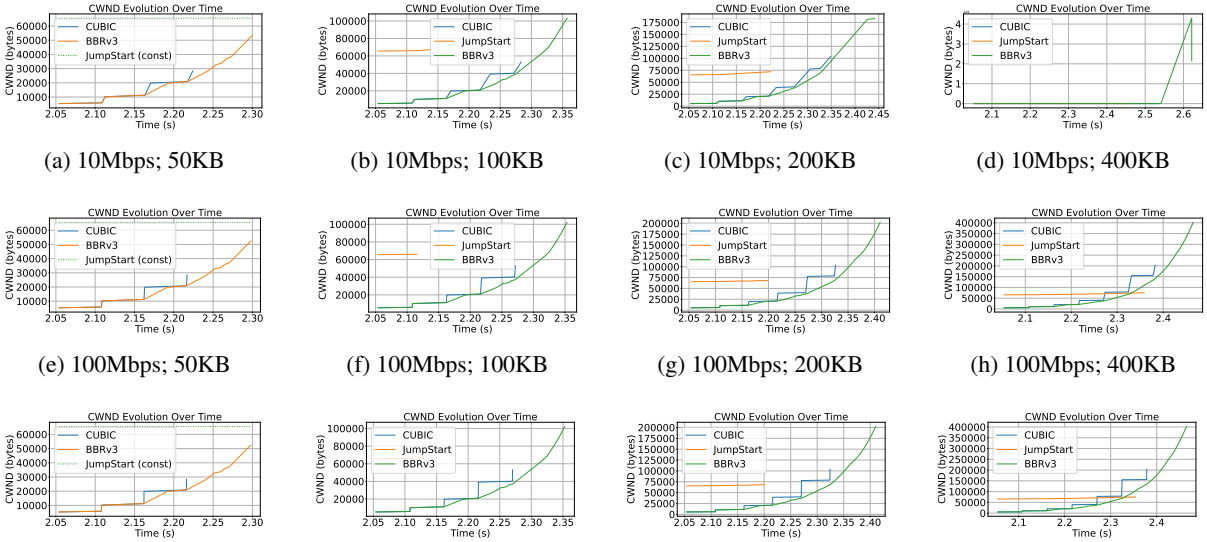
Table 2: Flow Completion Time (FCT) in seconds for different TCP algorithms and RTTs for 100 Mbps bandwidth and 100 KB flow size. The smallest FCT is shown in green, the second smallest in orange and the last in red

Algorithm	10ms	50ms	200ms
JumpStart	0.055315	0.213019	0.804443
CUBIC	0.093626	0.353626	1.328107
BBRv3	0.099205	0.380374	1.437718

[1] https://github.com/AlexandruTabacaru/ns-3-dev-git-rp
[2] https://github.com/nsnam/ns-3-dev-git
[3] https://github.com/Aruuni/ns3-bbrv3

Table 1: Flow Completion Time (FCT) in seconds for different TCP algorithms, bandwidths, and flow sizes. The smallest FCT is shown in green, the second smallest in orange and the last in red

Algorithm	10 Mbps				100 Mbps				1 Gbps			
	50KB	100KB	200KB	400KB	50KB	100KB	200KB	400KB	50KB	100KB	200KB	400KB
JumpStart	0.179327	0.223345	0.311423	0.487537	0.169267	0.213019	0.297924	0.442608	0.169232	0.212904	0.297751	0.441953
CUBIC	0.312668	0.377218	0.465296	0.641367	0.298570	0.353626	0.409689	0.467760	0.297160	0.351267	0.405474	0.459882
BBRv3	0.327032	0.384409	0.472487	0.648558	0.325015	0.380374	0.437622	0.492146	0.324813	0.380118	0.437340	0.491811

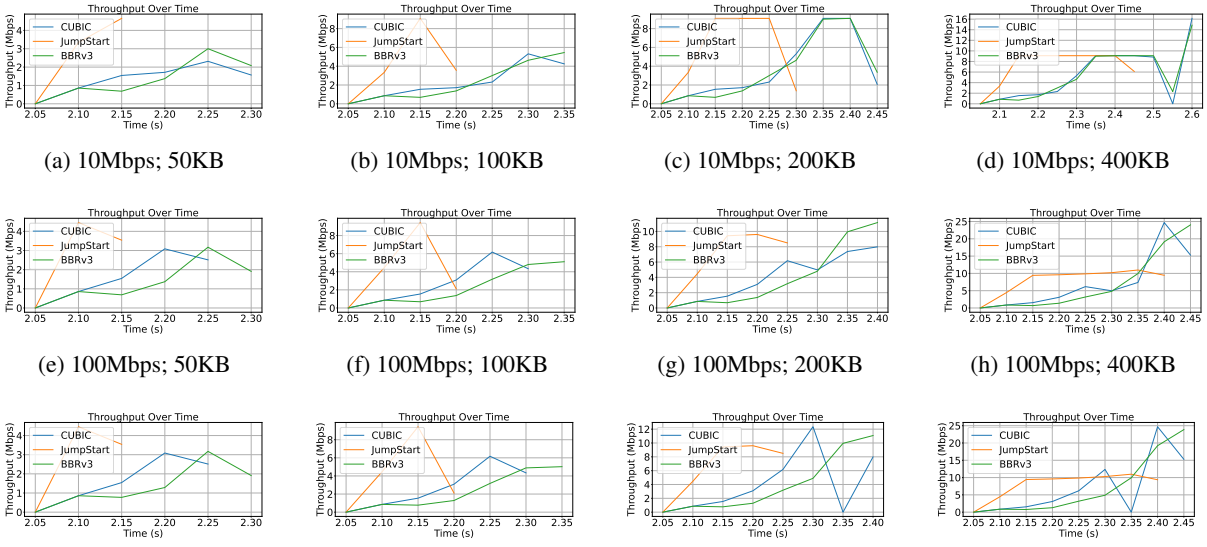


(a) 10Mbps; 50KB (b) 10Mbps; 100KB (c) 10Mbps; 200KB (d) 10Mbps; 400KB

(e) 100Mbps; 50KB (f) 100Mbps; 100KB (g) 100Mbps; 200KB (h) 100Mbps; 400KB

(i) 1Gbps; 50KB (j) 1Gbps; 100KB (k) 1Gbps; 200KB (l) 1Gbps; 400KB

Figure 2: Congestion window increase over time for different bandwidths and flow sizes

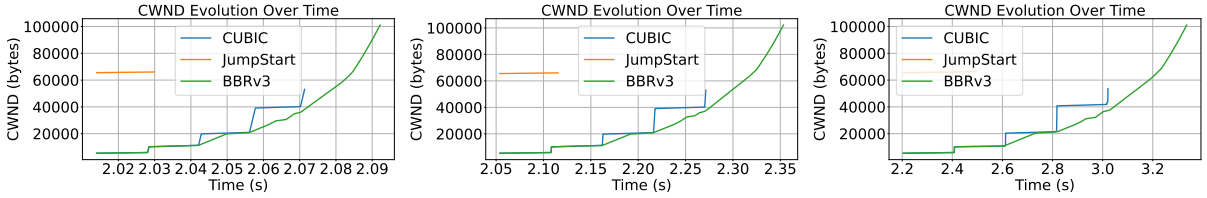


(a) 10Mbps; 50KB (b) 10Mbps; 100KB (c) 10Mbps; 200KB (d) 10Mbps; 400KB

(e) 100Mbps; 50KB (f) 100Mbps; 100KB (g) 100Mbps; 200KB (h) 100Mbps; 400KB

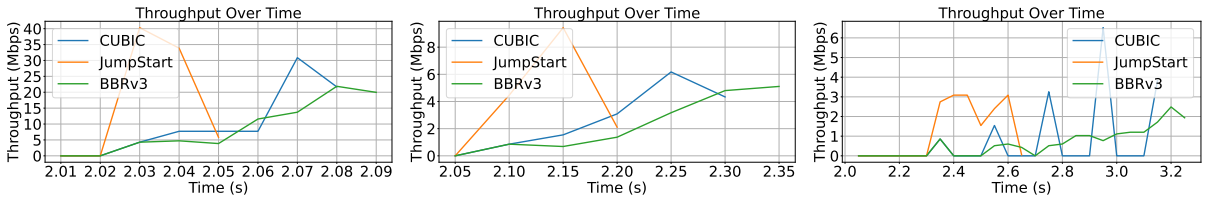
(i) 1Gbps; 50KB (j) 1Gbps; 100KB (k) 1Gbps; 200KB (l) 1Gbps; 400KB

Figure 3: Throughput over time for different bandwidths and flow sizes



(a) 10ms (b) 50ms (c) 200ms

Figure 4: Congestion window increase for different RTTs



(a) 10ms (b) 50ms (c) 200ms

Figure 5: Throughput over time for different RTTs

4.3 Interactions with existing flows

This experiment featured a simulation with 2 flows, one long (100 MB) and one short (100 KB). The aim of the experiment was to test the behavior of the algorithms when the flow competes, during startup, with an already existing flow that has reached its optimal sending rate.

In terms of FCTs, the same pattern can be observed again in Table 3, with JumpStart being the fastest, followed by CUBIC and BBRv3. The FCT of the long flow doesn't seem to be impacted much by the different algorithms, it is the same for both CUBIC and BBRv3 and only slightly faster when competing with JumpStart. However, comparing the FCTs of the short flows with those corresponding to the same conditions but without competing flows in Table 1 and Table 2, a significant increase can be observed. This is expected because of sharing the bandwidth. For JumpStart this difference is of approximately 100 ms while for CUBIC and BBRv3 200 ms.

Table 3: Flow Completion Time (FCT) in seconds for different TCP algorithms and both flows for 100 Mbps bandwidth and 50 ms RTT. The smallest FCT is shown in green, the second smallest in orange and the last in red

Algorithm	Long Flow	Short Flow
JumpStart	9.375789	0.322366
CUBIC	9.375866	0.528677
BBRv3	9.375866	0.570119

For this experiment, the JFI was also measured in Table 4. The fairness indices measured are very low for all three algorithms. In the ideal case, which can rarely be achieved in the real world, the JFI should be 1. In this experiment JFIs are 0.51, 0.53 and 0.51, for CUBIC, JumpStart and BBRv3 respectively, which are generally low values. However, a high JFI wouldn't be realistic in this case. The flows take time to ramp up and for most of their lifetime the difference in throughput is inevitably large, leading to a bad JFI value.

By analyzing the throughput measurements in Figure 6 it can also be observed that all the algorithms have a similar impact on the long flow. They slightly disturb its throughput when they start and cause a short period of throughput variation while the long flow tries again to converge to its optimal bandwidth utilization at around 90 Mbps.

Table 4: Jain's Fairness Index (FCT) for different TCP algorithms for 100 Mbps bandwidth and 50 ms RTT. The highest JFI is shown in green, the second highest in orange and the last in red

Algorithm	JFI
JumpStart	0.529104
CUBIC	0.517756
BBRv3	0.516466

4.4 Inter-Algorithm Fairness

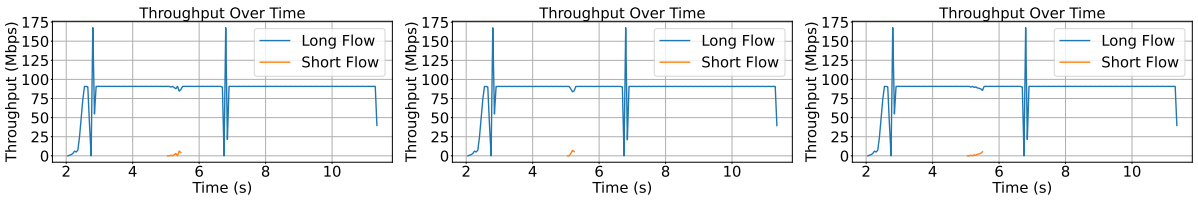
This experiment aimed to test the interactions between two flows started at the same time on the dumbbell topology, but using different algorithms. The network parameters were, as before, a bottleneck bandwidth of 100 Mbps and an RTT of 50 ms. However, for this experiment we chose to use slightly bigger flows (10 MB) such that we can observe how they compete for resources. We have ran this using smaller flows of 100 KB, but they end before reaching the available bandwidth so they never really compete for resources.

Analyzing the FCTs in Table 5 it can be observed that it is no longer the case that JumpStart is the fastest. It appears that, when using larger flows, it is slower than its counterparts by almost 2 seconds. That is because after the aggressive initial *cwnd*, JumpStart switches to normal CUBIC behavior, which is a slow additive increase in *cwnd*. Therefore, its throughput increases slowly after startup. On the other hand, CUBIC and BBRv3 start with smaller *cwnds* but they increase it more and after startup, they end up using more bandwidth compared to JumpStart. This behavior can also be seen in Figure 7

Table 5: Flow Completion Time (FCT) in seconds for different TCP algorithm combinations for 100 Mbps bandwidth, 50 ms RTT and 10 MB flow size. The smallest FCT is shown in green, the second smallest in orange and the lst in red

Algorithms	Flow 1	Flow 2
CUBIC & JumpStart	1.468073	3.397860
CUBIC & BBRv3	2.215472	2.194623
JumpStart & BBRv3	2.972063	1.546323

When analyzing the JFI measurements in Table 6, it can be observed that when CUBIC or BBRv3 compete with JumpStart the JFI is lower than when the first two compete with each other, 0.86 and 0.90 compared



(a) CUBIC (b) JumpStart (c) BBRv3

Figure 6: Throughput over time for different algorithms

to 0.99 respectively. This is a consequence of the observation from above. Since JumpStart takes longer to ramp up to the available bandwidth, it has a lower throughput for a long time and therefore the JFI is lower.

Table 6: Jain's Fairness Index (JFI) for different TCP algorithm combinations. The highest JFI is shown in green, the second highest in orange and the last in red

Algorithms	JFI
CUBIC & JumpStart	0.863770
CUBIC & BBRv3	0.999980
JumpStart & BBRv3	0.909156

Looking at the throughput measurements in Figure 7, we have observed that as mentioned before, JumpStart's throughput increases slower than the other algorithms' throughput. Because of the additive increase employed after the startup phase, the *cwnd* increases slowly. That is why at the beginning of the connection the other algorithm, CUBIC or BBRv3, has more bandwidth available and finishes their transmission quicker. However, when CUBIC competes with BBRv3, their throughput is very similar. They ramp up their sending rate similarly and therefore follow the same trend, with peaks and lows, eventually converging towards their fair share of 50 Mbps. Therefore we can say that for longer flows, JumpStart is not aggressive enough to achieve a fair share of the bandwidth or a small FCT.

4.5 Intra-Algorithm Fairness

This experiment featured the same setup as the previous one, only this time the two flows will use the same algorithm.

In Table 7 it can be seen that the FCTs are similar to the one in the previous experiment. It can also be observed that the difference between FCTs of competing flows with the same algorithm is very small, much smaller than when the algorithms differ. When CUBIC flows compete, the FCT of both flows is higher than that of CUBIC when it competed with JumpStart, about 800 ms higher. However, they are smaller than when CUBIC competed with BBRv3. The two competing JumpStart flows have lower FCTs than in the previous experiments. BBRv3 flows have larger FCTs. The difference between these flows and when BBRv3 competed with JumpStart is much bigger than when it competed with CUBIC. These results show that every algorithm is more fair when it competes with the same algorithm. This is probably because they have a similar way of using the bandwidth. This has been observed before in literature [19].

Table 7: Flow Completion Time (FCT) in seconds for different TCP algorithm combinations. The smallest FCT is shown in green, the second smallest in orange and the last in red

Algorithms	Flow 1	Flow 2
CUBIC & CUBIC	2.207829	2.203184
JumpStart & JumpStart	2.878466	2.896766
BBRv3 & BBRv3	2.213762	2.213762

In the case of JFIs it can be observed in Table 8 that all the pairs of flows share bandwidth in a fair way. This can also be seen when looking at small differences in FCTs Table 7 and at the throughput measurements in Figure 8. It can also be observed that each of the two flows has a similar throughput at any moment of the simulation.

Table 8: Jain's Fairness Index (JFI) for different TCP algorithm combinations. The highest JFI is shown in green, the second highest in orange and the last in red

Algorithms	Flow 1
CUBIC & CUBIC	0.999999
JumpStart & JumpStart	0.999990
BBRv3 & BBRv3	0.999997

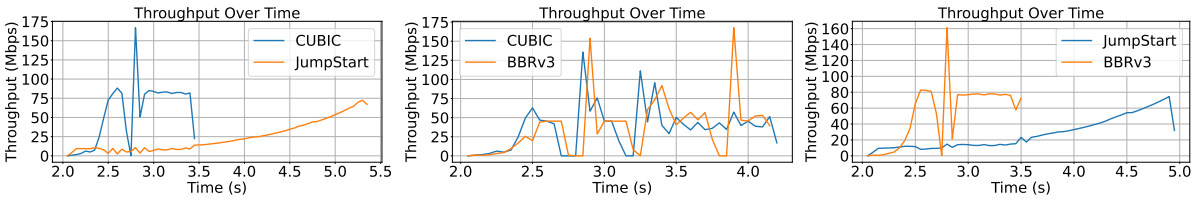
5 Responsible Research

This section aims to present the responsible research aspects of this paper.

All of the experiments presented previously, have been run using a publicly available, open source network simulator called *ns-3*[4]. The network setup for each of the experiments has been thoroughly described in the paper such that they can be easily reproduced. Since the experiments are a simulation, anyone can run them and get the same results without the need of any physical hardware. Moreover, the code used for the simulation and additions to the existing framework has been made open source on GitHub[5] as previously stated in the methodology section. Using all of this

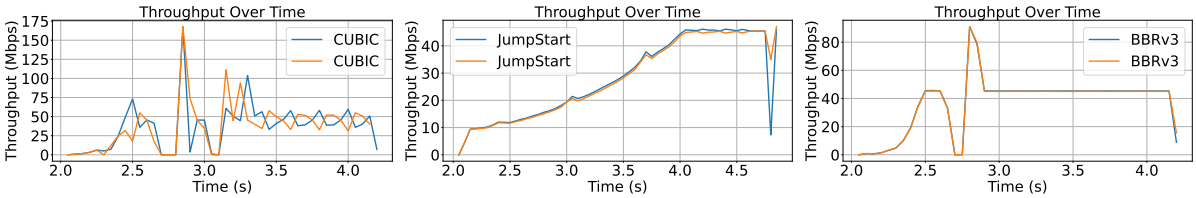
[4]https://www.nsnam.org/

[5]https://github.com/AlexandruTabacaru/ns-3-dev-git-rp/tree/startup-algorithms



(a) CUBIC & JumpStart (b) CUBIC & BBRv3 (c) JumpStart & BBRv3

Figure 7: Throughput over time for different algorithm combinations



(a) CUBIC & CUBIC (b) JumpStart & JumpStart (c) BBRv3 & BBRv3

Figure 8: Throughput over time for different algorithm combinations

information, anyone reading this paper can easily reproduce the experiments and verify the validity of the observed results.

LLMs were used for debugging purposes and understanding strange code behavior. The few lines of code written by LLMs were carefully analyzed to make sure they serve the purpose they were intended to.

Since these experiments do not rely on existing datasets or any kind of data collection, the issues regarding data privacy are not relevant to this research.

6 Discussion and Future Work

This section will discuss the observed results and propose what future work can be done to build on top of these discoveries.

Looking at the results from all of the performed experiments, we observed that in all conditions with short flows, the JumpStart algorithm has outperformed the other two algorithms we have compared, namely CUBIC, which uses HyStart and BBRv3 with its own startup phase implementation. This does not come as a surprise, given the conditions under which the algorithms were tested. Short-lived flows are favorable for JumpStart since it aggressively sends packets at the beginning of the connection. Therefore, the algorithm manages to utilize more bandwidth earlier in the connection, being more efficient than its counterparts. Also, due to the small size of the flows, the initial aggressive sending rate does not get to cause congestion on the network that would heavily impact its performance and other present flows. It has been shown that JumpStart may perform worse when dealing with a lot of traffic on the network [13].

In the case of the experiments with 2 mid-length flows competing, we have observed that JumpStart performs more poorly than the other algorithms. This is because after the initial *cwnd* is set to a relatively high value, the standard CUBIC additive increase takes over, which increases the *cwnd* slowly. While the other algorithms start with a smaller *cwnd*, they increase it more aggressively in the beginning and exit startup with a bigger value than JumpStart does. This makes CUBIC and BBRv3 take up more bandwidth at the beginning and therefore finish the transmission faster in the case of longer flows. However, as we have seen, the majority of flows in the Internet are short, and this makes a strong argument why the use of JumpStart in the real world could still be beneficial.

Future work should be conducted into analyzing the performance of JumpStart in more diverse and real world environments, while still focusing on short flows. While this research has shown that JumpStart can have a lot of potential compared to the algorithms already in use today in real Internet traffic, it is not enough to be sure that it would perform as well in real world conditions. Given the disadvantages previously observed in literature, it is necessary that improved versions of JumpStart will be developed in the future that can be deployed in the Internet. However, we believe that this research has proven the importance of further studying and improving this algorithm, to achieve more efficient communication on the Internet.

Another possible approach that we recommend for future research would be a thorough comparison of the other startup algorithms mentioned in the Background section under similar conditions as in this paper. This will lead to a better understanding of their performance especially in short flow situations. Such research may prove essential for further improving them or developing new startup algorithms that would make Internet communication more efficient.

7 Conclusion

The experiments we have performed have shown that JumpStart often outperforms the other two startup algorithms, CUBIC's HyStart and BBRv3. It achieved significantly better FCT than its counterparts in experiments with short flows. This is likely because short flows don't get to occupy the whole bandwidth during startup and they finish sending data before exiting startup. However, when we tested with longer flows, JumpStart did not perform as well. Looking at JFI, when JumpStart competes with other algorithms for bandwidth, it achieves lower fairness than when

the other two, CUBIC and BBRv3 compete. Therefore, based on the behavior we observed, we believe that JumpStart has potential to increase performance of TCP in the Internet. Nevertheless, further research should be conducted both into the behavior of JumpStart in different conditions, as well as into other startup algorithms to find a good way of improving them.

References

[1] Mahdi Arghavani et al. "SUSS: Improving TCP Performance by Speeding Up Slow-Start". en. In: *Proceedings of the ACM SIGCOMM 2024 Conference*. Sydney NSW Australia: ACM, Aug. 2024, pp. 151–165. ISBN: 979-8-4007-0614-1. DOI: 10.1145/3651890.3672234. URL: https://dl.acm.org/doi/10.1145/3651890.3672234 (visited on 04/24/2025).

[2] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. "TCP Vegas: new techniques for congestion detection and avoidance". en. In: *Proceedings of the conference on Communications architectures, protocols and applications - SIGCOMM '94*. London, United Kingdom: ACM Press, 1994, pp. 24–35. ISBN: 978-0-89791-682-0. DOI: 10.1145/190314.190317. URL: http://portal.acm.org/citation.cfm?doid=190314.190317 (visited on 04/25/2025).

[3] Neal Cardwell et al. "BBR: congestion-based congestion control". In: *Commun. ACM* 60.2 (Jan. 2017), pp. 58–66. ISSN: 0001-0782. DOI: 10.1145/3009824. URL: https://dl.acm.org/doi/10.1145/3009824 (visited on 05/02/2025).

[4] Neal Cardwell et al. "BBRv3: Algorithm Bug Fizes and Public Internet Deployment". In: July 2023. URL: https://datatracker.ietf.org/doc/slides-117-ccwg-bbrv3-algorithm-bug-fixes-and-public-internet-deployment/.

[5] Nandita Dukkipati and Nick McKeown. "Why flow-completion time is the right metric for congestion control". In: *SIGCOMM Comput. Commun. Rev.* 36.1 (Jan. 2006), pp. 59–62. ISSN: 0146-4833. DOI: 10.1145/1111322.1111336. URL: https://dl.acm.org/doi/10.1145/1111322.1111336 (visited on 05/02/2025).

[6] W. Eddy. *RFC 9293: Transmission Control Protocol (TCP)*. USA: RFC Editor, July 2022.

[7] Lingfeng Guo and Jack Y. B. Lee. "Stateful-TCP—A New Approach to Accelerate TCP Slow-Start". In: *IEEE Access* 8 (2020), pp. 195955–195970. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3034129. URL: https://ieeexplore.ieee.org/abstract/document/9240937 (visited on 05/02/2025).

[8] Sangtae Ha and Injong Rhee. "Taming the elephants: New TCP slow start". In: *Computer Networks* 55.9 (2011). Publisher: Elsevier, pp. 2092–2110. URL: https://www.sciencedirect.com/science/article/pii/S1389128611000363 (visited on 04/22/2025).

[9] Sangtae Ha, Injong Rhee, and Lisong Xu. "CUBIC: a new TCP-friendly high-speed TCP variant". en. In: *ACM SIGOPS Operating Systems Review* 42.5 (July 2008), pp. 64–74. ISSN: 0163-5980. DOI: 10.1145/1400097.1400105. URL: https://dl.acm.org/doi/10.1145/1400097.1400105 (visited on 04/25/2025).

[10] Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Hawe. "A quantitative measure of fairness and discrimination". In: *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA* 21.1 (1984), pp. 2022–2023. URL: https://www.academia.edu/download/49798765/fairness.pdf (visited on 06/04/2025).

[11] Douglas J. Leith, Robert N. Shorten, and Gavin McCullagh. "Experimental evaluation of cubic-TCP". In: (2008). URL: https://mural.maynoothuniversity.ie/1716/1/Hamiltonpfldnet2007_cubic_final.pdf (visited on 04/22/2025).

[12] Qingxi Li, Mo Dong, and P. Brighten Godfrey. "Halfback: running short flows quickly and safely". In: *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. CoNEXT '15. New York, NY, USA: Association for Computing Machinery, Dec. 2015, pp. 1–13. ISBN: 978-1-4503-3412-9. DOI: 10.1145/2716281.2836107. URL: https://dl.acm.org/doi/10.1145/2716281.2836107 (visited on 05/22/2025).

[13] Dan Liu et al. "Congestion control without a startup phase". In: *Proc. PFLDnet*. 2007, pp. 61–66. URL: https://www.icir.org/mallman/pubs/LAJW07/LAJW07.pdf (visited on 04/24/2025).

[14] Ayush Mishra et al. "Keeping an Eye on Congestion Control in the Wild with Nebby". In: *Proceedings of the ACM SIGCOMM 2024 Conference*. ACM SIGCOMM '24. New York, NY, USA: Association for Computing Machinery, Aug. 2024, pp. 136–150. ISBN: 979-8-4007-0614-1. DOI: 10.1145/3651890.3672223. URL: https://dl.acm.org/doi/10.1145/3651890.3672223 (visited on 06/05/2025).

[15] Ayush Mishra et al. "The Great Internet TCP Congestion Control Census". In: *Proc. ACM Meas. Anal. Comput. Syst.* 3.3 (Dec. 2019), 45:1–45:24. DOI: 10.1145/3366693. URL: https://dl.acm.org/doi/10.1145/3366693 (visited on 06/05/2025).

[16] Xiaohui Nie et al. "Dynamic TCP Initial Windows and Congestion Control Schemes Through Reinforcement Learning". In: *IEEE Journal on Selected Areas in Communications* 37.6 (June 2019), pp. 1231–1247. ISSN: 1558-0008. DOI: 10.1109/JSAC.2019.2904350. URL: https://ieeexplore.ieee.org/abstract/document/8668690 (visited on 06/04/2025).

[17] Feng Qian et al. "TCP revisited: a fresh look at TCP in the wild". In: *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*. IMC '09. New York, NY, USA: As-

sociation for Computing Machinery, Nov. 2009, pp. 76–89. ISBN: 978-1-60558-771-4. DOI: 10.1145/1644893.1644903. URL: https://dl.acm.org/doi/10.1145/1644893.1644903 (visited on 05/22/2025).

[18] Pasi Sarolahti et al. *Quick-Start for TCP and IP*. Request for Comments RFC 4782. Num Pages: 82. Internet Engineering Task Force, Jan. 2007. DOI: 10.17487/RFC4782. URL: https://datatracker.ietf.org/doc/rfc4782 (visited on 05/02/2025).

[19] B. Turkovic, F.A. Kuipers, and S. Uhlig. "Interactions between congestion control algorithms". English. In: *TMA - Proc. Netw. Traffic Meas. Anal. Conf.* Ed. by Secci S. et al. Journal Abbreviation: TMA - Proc. Netw. Traffic Meas. Anal. Conf. Institute of Electrical and Electronics Engineers Inc., 2019, pp. 161–168. ISBN: 978-390317617-1 (ISBN). DOI: 10.23919/TMA.2019.8784674. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85071160753&doi=10.23919%2fTMA.2019.8784674&partnerID=40&md5=6ada6dba9deab372339baab945e38ff2.

[20] Belma Turkovic and Fernando Kuipers. "P4air: Increasing Fairness among Competing Congestion Control Algorithms". In: *2020 IEEE 28th International Conference on Network Protocols (ICNP)*. ISSN: 2643-3303. Oct. 2020, pp. 1–12. DOI: 10.1109/ICNP49622.2020.9259405. URL: https://ieeexplore.ieee.org/abstract/document/9259405 (visited on 05/02/2025).

[21] Ranysha Ware et al. "CCAnalyzer: An Efficient and Nearly-Passive Congestion Control Classifier". en. In: *Proceedings of the ACM SIGCOMM 2024 Conference*. Sydney NSW Australia: ACM, Aug. 2024, pp. 181–196. ISBN: 979-8-4007-0614-1. DOI: 10.1145/3651890.3672255. URL: https://dl.acm.org/doi/10.1145/3651890.3672255 (visited on 06/05/2025).

[22] D. Zeynali et al. "Promises and Potential of BBRv3". English. In: *Lect. Notes Comput. Sci.* Ed. by Richter P., Bajpai V., and Carisimo E. Vol. 14538 LNCS. Journal Abbreviation: Lect. Notes Comput. Sci. Springer Science and Business Media Deutschland GmbH, 2024, pp. 249–272. ISBN: 03029743 (ISSN); 978-303156251-8 (ISBN). DOI: 10.1007/978-3-031-56252-5_12. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85189368557&doi=10.1007%2f978-3-031-56252-5_12&partnerID=40&md5=14c884b6ef75f8d4563a5956b024ddfb.