Analysis of Congestion Control and Scheduling Algorithms in Multipath TCP

Project Mentor : Prof. Debadatta mishra
By Skand Rajmeet. 170704

1. Introduction

Multipath TCP allows a single data stream to be split across multiple paths. This has benefits of reliability— the connection can persist even when a path fails. It also has benefits for load balancing at multihomed servers and data centers. In order to achieve TCP-friendly Internet deployment of MPTCP, the following three rules of practical multi-path congestion control should be achieved:
1) Rule 1 ("**Improve Throughput**"): A multi-path flow should perform at least as well as a single path flow would on the best of the paths available to it.
2) Rule 2 ("**Do no Harm**"): A multi-path flow should not take up more capacity from any of the resources shared by its different paths than if it were a single flow using only one of these paths. This guarantees it will not unduly harm other flows.
3) Rule 3 ("**Balance Congestion**"): A multi-path flow should move as much traffic as possible off its most congested paths, subject to meeting the first two goals.

In this report on multipath TCP I will provide an in-depth analysis of various congestion control and MPTCP scheduling algorithms available. This report also includes experimental results collected from various sources , detailing which algorithm performs better in various situations.

2. Background

As a high level overview, MPTCP is negotiated via new TCP options (such as MP_CAPABLE) in SYN packets. During this the endpoints exchange connection identifiers, which are used to add new paths/subflows to an existing connection. The subflows share a single send and receive buffer at the endpoints. Each subflow has an individual sequence number mapped to some connection level sequence number. This is helps in loss detection, retransmissions, and reordering the packets at the receiver. Acknowledgements are done at connection-level to implement proper flow control. Addition of new addresses is done by announcement from endpoint to another. For eg, if a connection from A to B has been initially established then A would send the new IP address via the already established connection using ADD_ADDR option. B in turn would decide if it wants to add a new subflow on the announced IP address via MP_JOIN option. In order to prevent malicious subflows in the network Hash-Based Message Authentication Codes (HMAC) are used. The Data Sequence Mapping (DSM)

is used to infer how the sequence space on the subflow maps to the connection level. DSM includes Data Sequence Number (DSN), Subflow Sequence Number (SSN), length of data, and checksum. Since, inferring of cumulative/connection-level data acknowledgement from subflow ACK fields is problematic an explicit cumulative data ACK is sent from the receiver. MPTCP can interact badly with non-MPTCP-aware content-modifying middleboxes. This middleboxes tend to modify the payloads changing their length, resulting in incorrect data mapping from subflow level to connection level. In order to detect such changes, MPTCP involves a second checksum in the DSM itself. The checksum algorithm is the same as TCP checksum. Finally, to close the connection a new DATA_FIN option is used. The meaning of FIN in restricted just to a subflow and does not affects the entire connection. DATA_FIN is an indication that sender has called a close on socket and no longer wants to send more data.

3. Congestion control

3.1 Uncoupled Congestion control MPTCP : Uncoupled congestion control is the simplest form of congestion control for MPTCP. Each subflow is handled like an independent TCP connection, with its own instance of a TCP congestion control. However, this solution is unsatisfactory, as it gives the multi-path flow an unfair (https://ieeexplore.ieee.org/document/6363695) share when the paths taken by its different subflows share a common bottleneck. We can apply TCP congestion control algorithm like Reno , Cubic , BIC, H-TCP, Hybla, Westwood, Vegas etc.on individual subflows of MPTCP. Each of the above algorithms have their pros and cons. Vegas adjusts its congestion window size based on the round trip time (RTT) value. Reno is the most widely used congestion control algorithm, but its bandwidth utilization is not very high, and as the network link bandwidths upgrade, this disadvantage will become more and more obvious. BIC may be too aggressive in low RTT and low-speed networks; Cubic is a modified version of BIC and the current default algorithm for TCP in Linux. During steady state, Cubic increases the congestion window size aggressively when the window is far from the saturation point, and then slowly when it is close to the saturation point. This feature allows Cubic to be very scalable when the bandwidth-delay product of the network is large, and at the same time, be highly stable and also fair to standard TCP flows.

3. 2 Coupled Congestion Control : The basic idea to solve the unfairness issue of uncoupled congestion control on shared bottlenecks is to couple the congestion windows of all subflows of an MPTCP connection with the resource pooling principle (http://ccr.sigcomm.org/online/files/p47-handleyA4.pdf).. The main idea is that by using a coupled congestion control method, the transport protocol can change the congestion

window of each subflow and ensure bottleneck fairness and fairness in the broader, network sense. Several approaches to handle this issue are available, for example LIA (Linked Increases Algorithm), OLIA (Opportunistic LIA), Balia (Balanced LIA) and wVegas (Weighted Vegas).

3.2.1 LINKED INCREASES ALGORITHM (LIA) : LIA has three main objectives: to improve throughput at least as well as an SPTCP connection on the best available path; do no harm to other SPTCP connections when sharing common bottlenecks; and balance congestion by moving traffic from the most congested paths to least congested ones. To this end, when receiving an ACK on subflow r, the subflow's congestion window w_r is increased by the minimum between two terms. First term describes a multipath window increase defined by α/w_{total}, where α is a multipath aggressiveness factor, and $w_{total} = \Sigma_{k \in R} w_k$. The second term is defined by $1/w_r$, which represents the window increase of a regular SPTCP connection would get at the same scenario. The aggressiveness factor α is such that the multipath goodput is equal to the SPTCP on the best path.

Derivation of the parameter alpha of LIA

There are two conditions which a multipath subflow congestion control is supposed to follow ideally:

1) A multipath flow should give a connection at least as much throughput as it would get with single-path TCP on the best of its paths. This ensures there is an incentive for deploying multipath.
2) A multipath flow should take no more capacity on any path or collection of paths than if it was a singlepath TCP flow using the best of those paths. This guarantees it will not unduly harm other flows at a bottleneck link, no matter what combination of paths passes through that link.

The above conditions can be represented in the form of mathematical equations :

$$\sum_{r \in R} \frac{\hat{w}_r}{\mathrm{RTT}_r} \geq \max_{r \in R} \frac{\hat{w}_r^{\mathrm{TCP}}}{\mathrm{RTT}_r}$$

$$\sum_{r \in S} \frac{\hat{w}_r}{\mathrm{RTT}_r} \leq \max_{r \in S} \frac{\hat{w}_r^{\mathrm{TCP}}}{\mathrm{RTT}_r} \quad \text{for all } S \subseteq R.$$

Each window w_r is made to increase on ACKs, and made to decrease on drops, and in equilibrium the increases and decreases must balance out, i.e. rate of ACKs × average increase per ACK must equal rate of drops × average decrease per drop, i.e

$$\left(\frac{w_r}{\mathrm{RTT}}(1-p)\right)\frac{1}{w_{\mathrm{total}}} = \left(\frac{w_r}{\mathrm{RTT}}p\right)\frac{w_{\mathrm{total}}}{2}. \quad ; \; w_{\mathrm{total}} = \sqrt{2(1-p)/p} \approx \sqrt{2/p}$$

We can calculate alpha from the balance equations. At equilibrium, the window increases and decreases balance out on each path, hence

$$(1-p_r)\min\left(\frac{a}{\hat{w}_{\text{total}}},\frac{1}{\hat{w}_r}\right)=p_r\frac{\hat{w}_r}{2}.$$

Making the approximation that pr is small enough that 1 − pr ≈ 1, and writing it in terms of w^{TCP}_r= sqrt(2/pr),

$$\max\left(\hat{w}_r,\frac{\hat{w}_{\text{total}}\hat{w}_r}{a}\right)=\hat{w}_r^{\text{TCP}}.$$

By solving these equations we arrive at the value of alpha:

$$a=\hat{w}_{\text{total}}\frac{\max_r \hat{w}_r/\text{RTT}_r^2}{(\sum_r \hat{w}_r/\text{RTT}_r)^2},$$

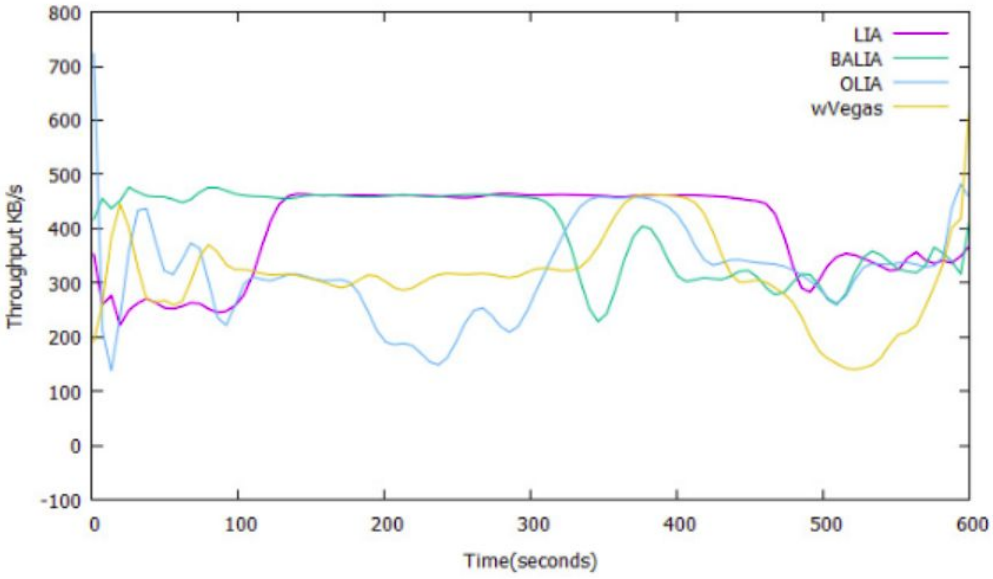
3.2.2 OPPORTUNISTIC LINKED INCREASES ALGORITHM (OLIA) : While LIA forces a tradeoff between optimal congestion balance and responsiveness, OLIA is an alternative that provides simultaneously both these properties. When receiving an ACK on the subflow r, the congestion window wr is increased by adding two terms. The first term is an optimal congestion balancing defined by $(w_r / \tau^2_r)/[\Sigma_{k \in R} (w_k /\tau_k)]^2$, where τ_r is the round-trip time observed on r. The second term is added by α_r /w_r , which guarantees responsiveness to react to changes in the current w_r . Unlike LIA, the parameter α_r is specific to each subflow r and applied to shift traffic among the subflows. To this end, subflows are classified into three different sets: W – set of subflows with the largest w_r ; B – set of best subflows with larger estimates of bytes transmitted on r between the last two losses; and C – set of the best collected subflows with no larger windows, i.e., $r \in B$ and $r < W$. If all $r \in B$ have the largest w_r , then $\alpha_r = 0$ for any r, i.e., the aggregate capacity available is using all the best subflows. When C is not empty, there is at least one best subflow with a small w_r , then α_r is positive for all $r \in C$ and negative for all $r \in W$. This makes w_r to increase faster for $r \in B$ and lower for $r \in W$. As a result, traffic is re-forwarded from subflows fully utilized (W) to paths with free capacity available (C).

3.2.3 BALANCED LINKED ADAPTATION (BALIA) : LIA might be unfriendliness without any benefit to MPTCP when competing with regular SPTCP flows on shared bottlenecks, while OLIA can be unresponsive to changes in network

conditions when the paths experience similar RTTs . From an optimization model designed to guarantee the existence, uniqueness and stability of the network equilibrium, BALIA is an alternative of multipath congestion control that allows w_r oscillation up to an ideal level to provide good balance between friendliness and responsiveness. For the subflow on the best path or for a MPTCP connection of a single path ($|R| = 1$), we have $\alpha_r = 1$. This makes both the increment and decrement of w_r to reduce to the same ones of TCP standard .

3.2.4 WEIGHTED VEGAS (WVEGAS) : While LIA, OLIA and BALIA react to path congestion upon packet loss, wVegas is a delay-based congestion control that estimates the link queueing delay q_r to realize path congestion and, then, proactively adapts w_r . The control of wVegas adjusts αr according to the weight ω_r – a ratio between current subflow's sending rate x_r and the aggregate multipath sending rate. The α_r is adjusted whenever it is smaller than δ_r , which is a difference between expected and actual x_r On the transmission round on r, w_r is adapted in two steps. First, similarly to TCP-Vegas, by incrementing or decrementing w_r according to δ_r and α_r . Second, by readjusting w_r according to q_r in order to drain link queues. This is conducted by monitoring the variation of queue delay q_r through the difference between the average observed RTT and the minimum measured RTT . The value of w_r is decreased by a backoff factor of $\tilde{\tau}_r /(2 * \tau_r)$ when detecting the link queue variation is higher than $2q_r$. Such an approach makes subflow r to occupy fewer link buffers, while reducing packet loss and mitigating bufferbloat. Thus, wVegas provides a congestion avoidance phase more sensitive to network changes than other congestion controls based exclusivity on the event of packet loss.

3.2.5 Experimental results: In this experiment, a comparison of the throughput when downloading a 1 GB file is made between the four different congestion control algorithms discussed in section three. The file is downloaded 4 times, each time different congestion control algorithm is used. These are LIA, OLIA, BALIA and wVegas. WiFi and 3G interfaces are both utilized for MPTCP at normal traffic load. Experiment results show that LIA has the best throughput most of the time (150s – 480s) BALIA also has a good throughput for the time from 0s to 320s at this type of load. OLIA and wVegas did not show good throughput at this load and were not stable most of the time.



4. MPTCP Schedulers

In the research of MPTCP, there are still problems such as out-of-order packet arrival. In order to solve this problem, researchers have done a lot of work and proposed many scheduling algorithms. But, different scheduling algorithms use different design principles. Here is a detailed discussion of widely-deployed scheduling algorithms, namely, LowRTT, DAPS, OTIAS and BLEST.

4.1 LOWEST-RTT-FIRST (LowRTT) The LowRTT algorithm is the default scheduler, which first sends the packet on the subflow with the lowest RTT, until it's congestion window is filled with packet. Then, the packet is sent on the subflow with the next higher RTT. The LowRTT algorithm makes the sub-flow with good path quality bear more data and has a certain load balancing effect. This is better than the RR algorithm, but neither algorithm considers the packet ordering. When the path difference is large, the path utilization will decrease.There are two subflows ($subflow_f$ and $subflow_s$). The congestion window of two subflows are 10, and RTT of two subflows is r_f = 10ms, r_s = 20ms, i.e., the time required to send 2 rounds of data over $subflow_f$ is the same as the time required to send 1 round of packets over subflows . When there are 11 packets to be sent, the $subflow_f$ takes 1, 2, 3, . . . , 10 Packets, subflows take 11 packets. The completion time of the subflowf is r_f , the subflows completion time is r_s , $r_s > r_f$. Thus, it can be seen that the entire completion time is slowed down by the slow flow. When the data is relatively large, the data packets cannot arrive at the receiver in order.

4.2 DELAY-AWARE PACKET SCHEDULER (DAPS) The DAPS algorithm aims to reduce the blocking time of the receive buffer. In the existing network, the link is asymmetric (i.e., each path has different delays, different capacities, different congestion windows), so the packets that can be scheduled for each path is different. The DAPS algorithm considers the difference of the path. It uses the ratio of the forward transmission delay (RTT /2) and the size of the congestion window (CWND) to determine the amount of data that each path should send next time, so that the packets arrive in order, and reducing Head-Of-Line Blocking. In the DAPS algorithm, it is assumed that the data on an MPTCP connection needs to be scheduled to two TCP subflows ($subflow_f$ and $subflow_s$). The congestion window sizes of the two subflows is respectively $cwnd_f$ and $cwnd_s$ the round-trip time is respectively r_f and r_s . If the ratio of the forward transmission delay is equal to the ratio of the Round-Trip Time, the ratio of the forward transmission delay of the two subflows can be expressed as η = j rs rf k . When η less than cwndf , in the next round of data transmission, the data sent by the subflowf is TSN_1, . . . , TSN_1+η, the data sent by the subflows is TSN_1+η+1, . . . , TSN_1+η+$cwnd_s$. When η greater than $cwnd_f$, in the next round of data transmission, the data sent by the $subflow_f$ is TSN_1, . . . , TSN_1 + $cwnd_f$, The data sent by the subflows is TSN_1 + cwndf + 1, . . . , TSN1 + $cwnd_f$ + $cwnd_s$.

4.3 OUT-OF-ORDER TRANSMISSION FOR IN ORDER ARRIVAL SCHEDULER (OTIAS) The OTIAS algorithm mitigates jitter by sending packets out of order on different subflows, enabling packets to arrive in order at the receiver, and hoping to schedule more segments on the subflow than it currently. The OTIAS scheduling algorithm is based on data scheduling for each packet, i.e., for each data packet, the one-way transmission delay of the data packet to the receiver of each path is estimated, and the data packet is sequentially scheduled to the path with a small forward transmission delay. If there is space in the CWND, the segment will be sent immediately. If the CWND is full, the segment must wait in the queue of the subflow. We estimate the one-way transmission delay for the packet i to be transmitted in the subflow j as follows:

$$
\begin{aligned}
&pkt\_can\_be\_sent\_j \\
&\qquad = cwnd_j - unacked_j \\
&RTT\_to\_wait_i^j = \left\lfloor \frac{no\_yet\_sent\_j - pkt\_can\_be\_sent\_j}{cwnd_j} \right\rfloor \\
&\qquad T_i^j = (RTT\_to\_wait_i^j + 0.5) \times srtt_j.
\end{aligned}
$$

In the above formula, pkt_can_be_sent_j is indicated a data packet that can be immediately transmitted in a subflow, no_yet_sent_j is indicated a data packet that has not been sent in the subflow j, and RTT _to_waitj i is a waiting time for the packet i to be transmitted on the subflow j. Compared with the DAPS algorithm, the OTIAS algorithm adjusts the scheduling when the packets are dequeued, while the OTIAS adjusts the scheduling when packets are enqueued. The OTIAS can respond to network changes more dynamically than the DAPS. However, OTIAS also has shortcomings. When the difference of two paths is large and there are packet loss in the link, a large number of data packets will be accumulated on the low-latency link, and the Hol-blocking problem cannot be solved well.

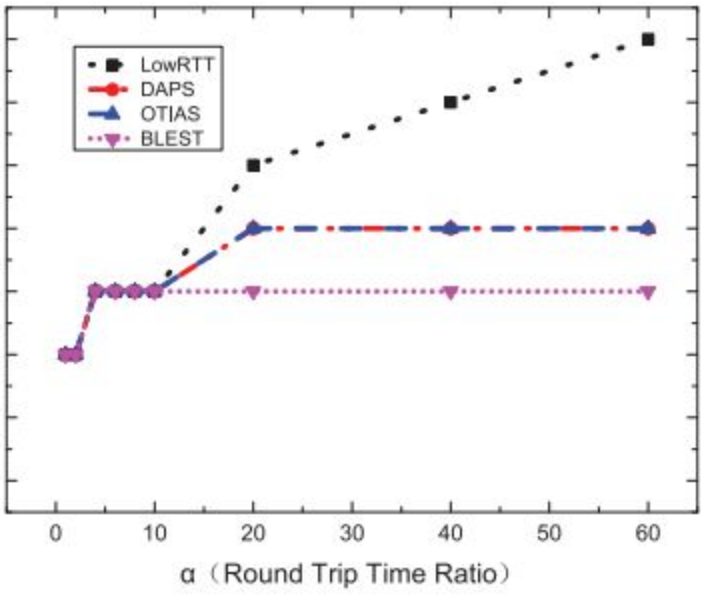
4.4 BLOCKING ESTIMATION-BASED MPTCP SCHEDULER (BLEST) The BLEST algorithm dynamically adapts the schedule by estimating whether a head-of-line blocking will occur, reducing head-of-line blocking, false retransmissions, and improving application performance in heterogeneous scenarios. BLEST estimates packets that can be transmitted in a fast subflow without Hol-blocking. Because the RTT of the slow subflow is relatively large, the data packet arrives at the receiver relatively late. When the amount of data is relatively large, a large number of out-of-order packets are generated, thereby increasing the completion time. Therefore, BLEST tries to use fast subflows to send packets so that they can arrive in order. Suppose there are two subflows ($subflow_f$ and $subflow_s$) that can send data with round trip times of RTT_f and RTT_s . If the data is sent on a slow subflows , BLEST assumes that one segment will occupy space at least RTTs in the MPTCP's send window (MPTCPsw). In order to send data through the fast $subflow_f$ as much as possible, it is estimated that the packet X that can be sent on the fast $subflow_f$ without the Hol-blocking during the period RTTs as follows:

$$rtts = RTT_s / RTT_f$$
$$X = MSS_f \cdot (CWND + (rtts - 1)/2) \cdot rtts$$

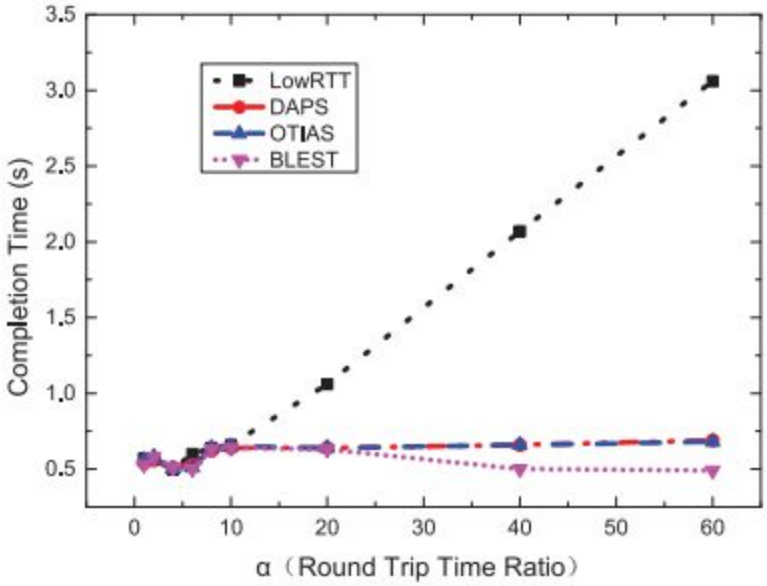
The X estimation may be inaccurate and will be constrained by a λ value. If x × λ >|M| − MSSs· (inflight + 1), i.e. the next segment will not be sent on subflows. Instead, the scheduler waits for the faster subflow to become available. The BLEST algorithm can skip a subflow and wait for a more favorable subflow to send data, which can reduce the risk of Hol-blocking and thus the number of retransmissions triggered. BLEST outperforms the other three algorithms.

4.5 Experimental evaluation of MPTCP Schedulers :

4.5.1 PERFORMANCE EVALUATION IN A NON-SHARED SCENARIO

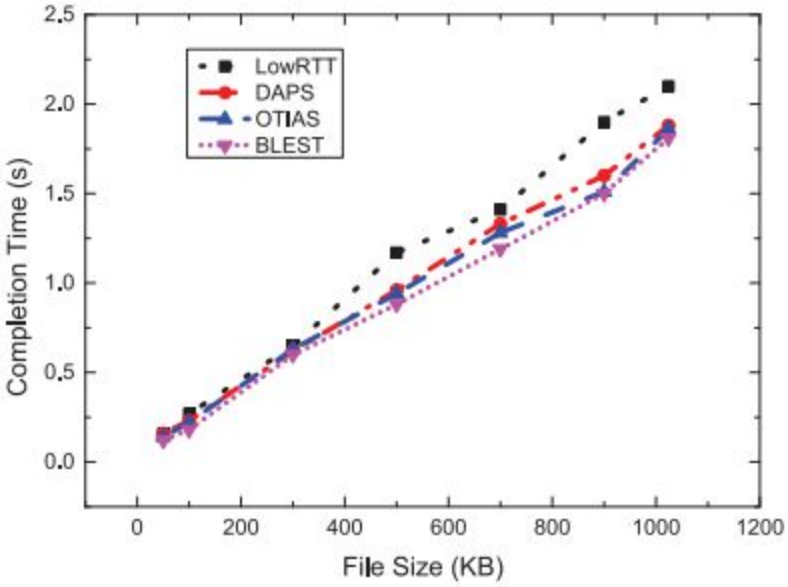


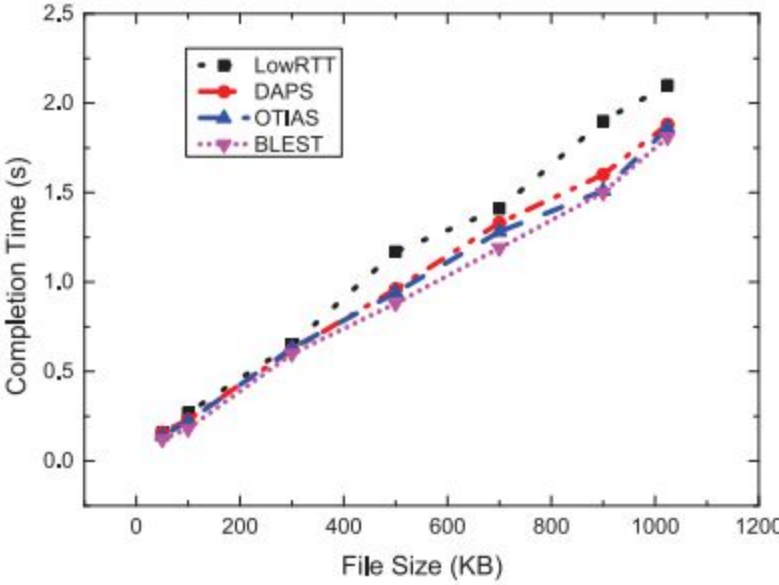
4.5.1.1 PATH HETEROGENEITY TEST In this path heterogeneity test, the bandwidth is 100 Mbps and the number of concurrent flows is 20, RTT1 is 10 ms, and RTT2 ranges from 10ms to 600ms. The number of timeouts and the flow completion time are shown below. As depicted below, the LowRTT has largest number of timeout, and the BLEST has least number of timeout. As a result, BLEST performs best, followed by OTIAS and DAPS, and LowRTT performs worst as described below. The reason lies in that although all the four algorithms consider path heterogeneity, LowRTT does not consider the effects of packets out-oforder. BLEST tries to use fast subflows to send packets so that they can arrive in order, and can reduce head-of-line blocking.



4.5.1.2 DIFFERENT BUFFER SIZE In these experiments, the bandwidth is 5 Mbps, the number of concurrent flows is 20, the RTT of each path is 20 ms, and the file size is 300 KB. BLEST outperforms other algorithms, followed by OTIAS and DAPS. This is because that BLEST can dynamically estimate whether a head-of-line blocking will occur, and reduce the number of out-of-order packets. Although DAPS and OTIAS have the same goal to reduce Hol-blocking, DAPS is insensitive to link changes. When the link characteristics change, a large number of data packets are accumulated on the link, and the data needs to be re-scheduled for one cycle. In addition, OTIAS can respond to network changes

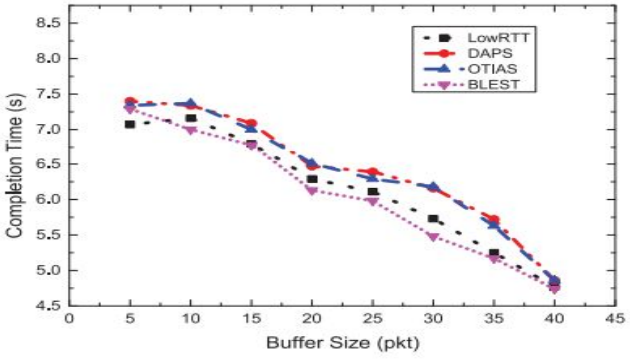
more dynamically than the DAPS. However, when the difference of two paths is large, a large number of data packets will be accumulated on the low-latency link, and the Hol-blocking problem cannot be solved well.



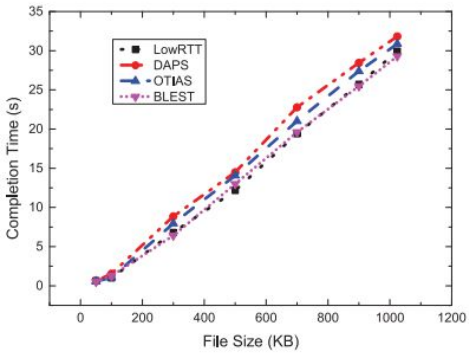


4.5.2 PERFORMANCE EVALUATION UNDER COMPETING TRAFFIC SCENARIOS

4.5.2.1 DIFFERENT BUFFER SIZE In these experiments, the file size is 300 KB. The completion time, becomes smaller with increasing buffer size. The BLEST performs best, followed by LowRTT. The DAPS performs worse compared to OTIAS. This is because that BLEST is able to react more dynamically to network changes, followed by LowRTT and OTIAS. DAPS is insensitive to link changes.



Completion time with different receive buffer sizes.



Completion time with different file sizes.

4.5.2.2 DIFFERENT FILE SIZE: It is shown above the completion time becomes larger with the increasing file size when the buffer size is 20 packets. The results are in accordance with those when the buffer size is different. As analyzed above, the reason lies in that BLEST can respond to network changes more dynamically than the LowRTT and OTIAS, and DAPS is insensitive to link changes.

5. References

1) https://www.usenix.org/legacy/event/nsdi11/tech/full_papers/Wischik.pdf
2) https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8636963
3) https://tools.ietf.org/html/rfc6356
4) https://arxiv.org/abs/1812.03210