TCP FlexiS: A New Approach to Incipient Congestion Detection and Control

Qian Li

Abstract—Best effort congestion controls strive to achieve an equitable distribution of network resources among competing flows. However, fair resource allocation becomes undesirable when a bandwidth/delay sensitive application shares a bottleneck with a greedy background application. Less than Best Effort (LBE) Congestion Control Algorithms (CCA) are specially designed for background applications, which do not have strict bandwidth/delay requirements. LBE CCAs give foreground applications higher priority in resource allocation by only utilizing spare bandwidth. This can greatly improve network utility at times of congestion. We propose FlexiS – a Flexible Sender side LBE CCA. Unlike most conventional LBE CCAs, which use queue size based congestion detectors and linear rate controllers, FlexiS employs a queue trend based congestion detector and a cubic increase multiplicative decrease rate controller. We have compared FlexiS with LEDBAT and LEDBAT++. Extensive emulation and preliminary Internet tests showed that: 1) FlexiS has comparatively low impact on concurrent best effort TCP flows; 2) it scales to a wide range of available bandwidths; 3) FlexiS flows in aggregation can effectively utilize available bandwidth; 4) contending FlexiS flows can, in most cases, equally share available bandwidth; 5) FlexiS adapts to route changes quickly; and 6) it maintains low priority even when AQM algorithms or shallow buffers are deployed.

Index Terms—Lower than best effort, LBE, low priority, congestion control algorithms.

I. INTRODUCTION

THE Internet is a packet switched network. Packets of different applications share a common set of network resources including transmission medium, switching and routing devices and device buffers. Packets that cannot be transmitted immediately are placed in a buffer and are usually served in first come and first served order. Packets are dropped when the buffer is full. The most widely adopted mechanism for network resource allocation so far has been based on end points through Congestion Control Algorithms (CCA). A CCA determines at what rate a transmitter should be sending, which further determines the bandwidth share of a flow (a sequence of packets that have the same source IP, destination IP, protocol, source port, and destination port). Fairly sharing network resources among competing flows has been one of the design goals of many CCAs. These CCAs are usually referred to as Best Effort (BE) CCAs. The dominant BE CCAs use loss as a signal of congestion.

However, allocating network resources fairly among competing flows is not always desirable. Consider the following scenario. While Alice is enjoying video on demand streaming (app A), a software update (app B) is initiated by a server to transfer 1 GB of software patches to Bob’s computer in the background. Apps A and B share the same bottleneck and both use BE CCAs. The fair competition of app B makes app A lose a portion of bandwidth, which causes video playback to stall from time to time. Consequently, the Quality of Experience (QoE) of Alice is degraded. In comparison, if app B employs a Less than Best Effort (LBE) CCA that can detect app A and never acquires bandwidth more than what is available, the minimum bandwidth requirements of app A can be guaranteed and the QoE of Alice will not be affected. In the meantime, Bob will not notice the increased download time of software patches since the software update runs in the background. Clearly, priority-based resource allocation is more preferable over fair allocation in the above scenario.

In reality, scenarios similar to the one illustrated above are not uncommon. When QoS sensitive foreground applications (such as VoIP, online gaming, video streaming and web browsing) compete fairly with QoS tolerant background applications (e.g., software update, client-to-cloud backup, inter-data-center synchronization and peer-to-peer file sharing), the QoE of Alice is degraded. In comparison, if app B employs an LBE CCA that can detect app A and never acquires bandwidth more than what is available, the minimum bandwidth requirements of app A can be guaranteed and the QoE of Alice will not be affected. Clearly, priority-based resource allocation is more preferable over fair allocation in the above scenario.

In this article, we propose a new LBE CCA named FlexiS – a Flexible Sender side LBE CCA. The objectives of FlexiS are: (1) low intrusion on concurrent foreground applications; (2) effectively utilizing Available Bandwidth (AB); and (3) distributing AB equally among competing FlexiS flows. FlexiS has been implemented as a Linux kernel module. The source code is available as open source software at [7].

Most existing LBE CCAs [1], [2], [3], [4] and delay based CCAs [8], [9], [10], [11] need to estimate base delay in order to detect incipient congestion. Base delay is the time that it takes for a packet to traverse an unloaded network route. In practice, it is very difficult to estimate. First, it can change with the change of route. Second, if a route consists of wireless links, its base delay can change even when route is not changed.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
In particular, it adapts to route changes quickly. It does not suffer from problems related to inaccurate base delay by increased RTT. Extensive evaluation showed that FlexiS does not suffer from problems related to inaccurate base delay estimation. In particular, it adapts to route changes quickly.

Recent LBE CCAs such as LEDBAT [3] and LEDBAT++ [4] use fixed delay TARGETs as congestion thresholds. This technique has two major drawbacks. On the one hand, a fixed amount of extra QD is added to the estimated base delay for most of the time. On the other hand, if the bottleneck has a small buffer or Active Queue Management (AQM) algorithms are deployed, packet loss can occur before the LBE CCAs reach their delay targets. This will make them as aggressive as some BE CCAs in certain scenarios [15]. With the ongoing effort on tackling buffer bloat, recent years witnessed rapid AQM deployment in home gateways [16]. Therefore, retaining low priority in the presence of AQMs has become a must-have feature of LBE CCAs.

FlexiS does not maintain a fixed QD. It reduces rate when RTT samples have an obvious increasing trend. Experiments showed that the extra QD induced by FlexiS is much smaller than that inflicted by LEDBAT and LEDBAT++ in most cases and that FlexiS can preserve low priority even with the presence of AQMs or shallow buffers. Further, backing off earlier can reduce the number of loss events caused by AQMs or buffer overflow, therefore improve bandwidth utilization.

Another problem with many conventional LBE CCAs [1], [2], [3], [4] is the use of linear controls for rate adaptation. The merit of linear controls is simplicity and the potential of converging to fairness. In this article, fairness is defined as the equal distribution of bandwidth among contending flows. It is proven by Chiu and Jain in [17] that Additive Increase Multiplicative Decrease (AIMD) is the most feasible and efficient linear control to realize fairness. Although theoretically sound, AIMD has various issues in practice. The typical application of AIMD is to increase Congestion WinDow (CWND) by one segment per RTT and reduce it by half. However, this application does not guarantee fairness between connections with different RTTs. Floyd proposed in [18] that all connections despite their RTTs should increase their rates by a constant amount of $a$ packets/second during each second. However, it was later on substantiated that this increase function is difficult to successfully deploy in an operational network [19]. In addition to the fairness problem, AIMD used by standard TCP is also known to have low bandwidth utilization problem in large Bandwidth Delay Product (BDP) networks. It is shown in [20] that an unrealistically small bit error rate is required for standard TCP to achieve full bandwidth utilization in such networks.

We evaluated a variety of rate increase/decrease functions. Finally, a cubic increase multiplicative decrease rate controller was chosen. To be specific, FlexiS increases sending rate using a cubic function of elapsed time and decreases rate by a fixed percent. Cubic increase makes FlexiS scalable across a wide range of ABs. And it improves bandwidth utilization in large BDP networks. Cubic increase is used in both initial ramp up and congestion avoidance. Experiments showed that by doing so, the initial long delay and high loss caused by slow start can be avoided. Further, cubic increase multiplicative decrease together contribute to FlexiS’ high intra- and inter-RTT fairness.

The rest of the paper is organized as follows. In section II we review related work. Section III elaborates on the design of FlexiS. Extensive evaluation results are presented in section IV. Finally, section V concludes the paper.

II. RELATED WORK

In this section, we review previous work that either inspired FlexiS or that are similar to FlexiS either in objectives or in design.

A. Work That Inspired FlexiS

The design of FlexiS’ congestion detector is greatly influenced by pathload [21], which is an AB estimator. A pathload sender transmits a fleet of UDP packet streams to a receiver with a predetermined stream rate and inter-stream interval. Upon the receipt of all probe packets and related One Way Delay (OWD) samples of a stream, the receiver analyzes the trend in OWD using two statistics. If the majority of the streams in a fleet cause OWD to have an increasing trend, the rate of the fleet is considered to be higher than the AB. AB can be discovered by sending out multiple fleets with different rates. Inspired by pathload, FlexiS uses an increasing trend in delay as the indicator of incipient congestion. Unlike pathload, FlexiS uses RTT in congestion detection. Further, it uses in-band TCP packets as probe packets and employs linear regression to determine the trend in RTT.

B. Work Similar in Design

Similar to FlexiS, some CCAs also use trend in delay or delay gradients in congestion detection. They differ from FlexiS mainly in how delay gradient is calculated and utilized. ImTCP [22] employs the two statistics technique proposed in pathload to estimate the trend in RTT. Probe Control Protocol (PCP) [23] derives the trend in OWD using least squares. CDG [24] uses the difference of the minimum (or maximum) RTTs measured in consecutive round trips to derive delay gradients. If the average of last $n$ delay gradients is greater than zero, congestion is concluded. Timely [25] uses a positive delay gradient as an indicator of congestion. The delay gradient is a ratio between the average (EWMA) difference between consecutive RTT samples and the minimum RTT. PCC-Vivace [26] and PCC-Proteus [27] use linear regression to generate
RTT gradients, which is used as an independent variable of a utility function. In contrast, FlexiS uses a Theil-Sen estimator to estimate the slope of the linear regression line of RTT samples. Slope exceeding a threshold is used to indicate congestion.

Some other CCAs employ similar increase functions as FlexiS. CUBIC [28] uses a cubic function of elapsed time since last congestion event as its window growth function. It uses both the concave and convex profiles of a cubic function. TCP-LoLa [29] employs two cubic functions for rate increase. One is CUBIC’s window growth function. Another is a cubic function of time elapsed since the start of fair flow balancing. This latter function is used to calculate an Expected amount of Data (ED) in the bottleneck queue, which is used to achieve RTT-independent rate fairness. Similarly, FlexiS also employs a cubic function of time elapsed since the start of the increase epoch for rate increase. But the function is used to update sending rate instead of CWND or ED. Further, it only uses the convex profile of a cubic function and it has a first degree term in its increase function.

C. Work Similar in Objectives

The CCAs presented in this subsection all have LBE as their design objectives.

TCP Nice [1] and TCP-LP [2] are two early LBE CCAs. Both of them use (filtered) QD exceeding a certain percent of maximum QD as congestion indicator. Linear increase functions are used to grow CWND. CWND is halved when congestion is detected. Eclipse [30] is a hybrid of three LBE CCAs. Its target QD calculation is based on TCP Nice and TCP-LP and its rate controller is based on LEDBAT.

ImTCP-bg [31] and TCP Westwood Low Priority [32] are LBE CCAs that are adapted from BE CCAs. The former uses the AB estimated by ImTCP to calculate an upper bound for CWND. The latter adds a backlog based incipient congestion detector to TCP Westwood.

FLOWER [33] adjusts CWND with a fuzzy controller. DA-LBE [34] is a deadline aware LBE framework that is capable of making any non-LBE CCAs to have LBE behavior and the degree of LBEness can be adjusted according to remaining design objectives.

D. Work Used for Comparison

Low Extra Delay BBackground Transport (LEDBAT) [3] and LEDBAT++ [4] are two recent LBE CCAs. Both published specifications with IETF and carry real world traffic in the Internet. In this article, FlexiS is compared with them.

LEDBAT uses QD exceeding a TARGET (100 ms by default) as an indication of congestion. If QD is below TARGET, CWND is Additively Increased, otherwise Additively Decreased (AIAAD). The amount of increase/decrease is in proportion to off_target = (TARGET - QD) / TARGET and a pre-configured increase/decrease GAIN. QD is the difference between the current OWD and base delay.

LEDBAT++ improves LEDBAT using a number of techniques. It replaces OWD with RTT in QD calculation. Slow start quits when QD exceeds 3/4 TARGET to reduce initial intrusion. LEDBAT’s AIAAD controller is replaced by an additive increase multiplicative decrease controller, which is borrowed from fLEDBAT [35]. The authors of fLEDBAT propose to use constant increase to replace proportional increase to speed up convergence and improve efficiency. Additive decrease is attributed to the cause of unfairness and is replaced with multiplicative decrease. In LEDBAT++, GAIN is a dynamic value, which is in proportion to a connection’s RTT. LEDBAT++ periodically reduces CWND to 2 segments in order to discover a true base delay.

III. Design

A. Overview

FlexiS uses an increasing trend in RTT to indicate congestion. On the receipt of the $i_{th}$ ACK, an RTT sample $d_i$ and a timestamp $t_i$ are obtained. $t_i$ is the sending time of the acknowledged data packet. $(t_i, d_i)$ are put into a data structure called RTT sack (denoted as D) as one data point. If enough data points are gathered in D, the slope of the linear regression line of all data points in D is derived and the oldest data point $(t_1, d_1)$ is removed. If the slope is equal to or greater than a threshold $\theta$, RTT is considered to have an increasing trend and FlexiS will reduce its rate by a fixed percent. Otherwise, rate will be increased according to a cubic function of elapsed time. Algorithm 1 shows a pseudo-code of the main logic of FlexiS.

Algorithm 1 The Main Logic of FlexiS

| Variables |
|-----------|
| $L_D$: length of $D$ in ms |
| $N_D$: size of $D$ in number of data points |
| $S_D$: the slope of the regression line of the data points in $D$ |
| $\sigma$: the minimum number of data points in $D$ required to make a trend estimate. default: 3 |
| $\tau$: the minimum length of $D$ required to make a trend estimate. default: 60 ms |
| $\theta$: the minimum slope that makes FlexiS reduce rate. default: 30 |

On the receipt of the $i_{th}$ ACK

put $(t_i, d_i)$ into $D$

$L_D \leftarrow t_i - t_1$

if $L_D < \tau$ then

CWND is unchanged

else

if $N_D < \sigma$ then

increase CWND per Equations 1 & 2 (Section III-C)

else

calculate $S_D$

if $S_D \geq \theta$ then

decrease CWND per Equation 3 (Section III-D)

else

increase CWND per Equations 1 & 2 (Section III-C)

end if

end if

remove the oldest data point $(t_1, d_1)$ from $D$

end if
If the slope is less than $\theta$, the sender enters the increase phase. If sufficient data points are obtained, the sender will then go into the increase phase, which terminates when congestion is detected. The rate increase function is defined in Equation 1. In the following equations, RTT is measured in unit of seconds, time in milliseconds, rate in segments per second and CWND in segments. $d_{\min}(t_1, t_2)$ denotes the minimum RTT observed between times $t_1$ and $t_2$.

$$r = \frac{\Delta t}{\alpha} + \frac{\Delta t}{\beta} + r_0$$

where $\alpha$ and $\beta$ are increase factors, $r_0 = w_0/d_0$ is the estimated sending rate at $t_0$, $w_0$ is the CWND value at $t_0$. $d_0 = d_{\min}(t_{pend}, t_0)$ is the RTT estimate made at time $t_0$. $t_{pend}$ is the start time of the preceding pending phase. Because RTT usually do not increase significantly in the pending phase (otherwise rate would have been reduced), $d_{\min}(t_{pend}, t_0)$ is pretty close to the actual RTT at time $t_0$.

Rate $r$ is then converted to CWND $w$ using Equation 2.

$$w = r \times d_{cur}$$

where $d_{cur} = d_{\min}(t_{pend}, t_{cur})$ is the RTT estimate made at the time of calculation ($t_{cur}$). The actual RTT at $t_{cur}$ can be higher than $d_{cur}$ if the network is congested at the time. This will make the actual sending rate lower than the theoretical value at times of congestion. However, this side effect is desirable because it can make FlexiS less intrusive when congestion is forming but the sender is still not aware of it. This is especially helpful when a connection’s RTT is high.

FlexiS does not distinguish between slow start and congestion avoidance, the above functions are used whenever rate needs to be increased. The rate increase function has a
first degree term. It is used to speed up rate increase at the beginning of an increase epoch. A single cubic term makes rate increase too slowly when elapsed time is small, which has two undesired effects. On the one hand, overly slow rate increments can adversely affect bandwidth utilization. On another hand, in some rare cases, the slow rate increase makes RTT increase so slowly that it cannot be detected by FlexiS. Adding a first degree term to the function can solve both problems.

\( \alpha \) and \( \beta \) determine the curvature of the increase function. The smaller the values are the more aggressive FlexiS is. Very aggressive increase will inflict high intrusion and overly conservative increase can harm utilization. Experiments showed that a good trade-off between intrusion and utilization can be made when \( \alpha \) is set to 100 and \( \beta \) to 10.

We will show next that the increase functions meet all design objectives of FlexiS. First, they make FlexiS low intrusive. During initial ramp up, it makes rate increase in proportion to CWND, which ensures that the QD induced by FlexiS is bounded to a proportion of an RTT. After a congestion event, elapsed time is reset to zero and FlexiS starts from small increments again. This behavior gives the bottleneck sufficient time to drain its queue. It can also delay the occurrence of the next congestion event, therefore reduce intrusion to foreground traffic. Second, the cubic function allows FlexiS to speed up over time, which makes it possible to converge to very large ABs and significantly reduces convergence time compared to linear increase. Finally, the increase functions meet the second necessary condition to fairness. When contending flows enter their respective increase epochs at different times, they will increase at different speeds. However, this asynchronous state will end after a congestion event provided that all flows can detect the congestion. Due to the difference in feedback delays, flows with different RTTs will still enter their respective increase epochs at slightly different times. However, this will not noticeably affect fairness.

D. Rate Decrease

Upon congestion, FlexiS reduces CWND per Equation 3. This function was chosen due to its simplicity and overall good performance.

\[
\text{w'} = \max(w \times \gamma, 2) \tag{3}
\]

where \( w' \) and \( w \) are CWND values after and before reduction, \( 0 < \gamma < 1 \) is the decrease factor. The max operator ensures that CWND is never reduced below 2 segments.

The default value for \( \gamma \) is 0.85. \( \gamma \) should be set in conjunction with \( \alpha \) and \( \beta \). When \( \alpha \) and \( \beta \) are decreased, so should \( \gamma \) so as to counter-act the increased aggressiveness and vice versa. In order to meet the third necessary condition to fairness, all flows should set their decrease factors to the same value.

CWND is halved (but not below 2 segments) on packet loss detected by the fast retransmit and fast recovery algorithm and is reduced to 1 when retransmission timer goes off. Because FlexiS has a very sensitive congestion detector, it does not need to reduce its rate more aggressively than standard TCP at times of loss. FlexiS only reduces CWND once per round trip time. After any type of CWND reduction, \( D \) is emptied.

If a bottleneck employs AQM or has a shallow buffer, packet loss can occur while FlexiS is in rate reduction. CWND will be consequently reduced twice, which hurts bandwidth utilization. As a countermeasure, FlexiS undo's the second CWND reduction upon the exit of fast recovery.

E. Pacing

FlexiS uses pacing, which is a technique used to evenly space packets at specified intervals at times of sending. Pacing has two functions in FlexiS. On the one hand, it can greatly reduce delay variation and packet losses inflicted by FlexiS flows, therefore reduce intrusion to foreground flows. On the other hand, pacing can improve bandwidth utilization of FlexiS. It minimizes the probability that a FlexiS flow backs off before its rate reaches AB due to self-induced queues.

In Linux, per flow pacing can be realized by TCP or by the fair queue packet scheduler. In either way, we need to determine a desired pacing rate. However, in Linux kernel, a congestion control module that only implements congestion avoidance cannot ultimately update pacing rate. As a temporary work-around, we update pacing ratio \( R_p \), which is used to calculate pacing rate by the kernel. \( R_p \) is the ratio between the current rate \( r \) and the rate in one RTT \( r' \). It is updated per equation 4 right after each rate update. If rate \( r \) has just been increased, \( R_p \) is updated with the first sub-equation. Otherwise, the rate in one RTT would be equal to the current rate, so \( R_p \) should be 100%.

\[
R_p = \left\{ \begin{array}{ll}
\frac{r'}{r} = \left( \frac{\Delta t'}{\alpha} \right)^3 + \frac{\Delta t'}{\beta} + r_0 \times 100, & \text{increased} \\
100, & \text{Otherwise}
\end{array} \right. \tag{4}
\]

where \( \Delta t' = t_{cur} + d_{cur} - t_0 \) is the duration between \( t_0 \) and the time in one RTT. \( d_{cur} \) has the same definition as in Equation 2. Again at times of congestion, it can be smaller than actual RTT, which results in reduced pacing rate. However, this estimation error makes the pacing rate more close to the actual sending rate in one RTT.

IV. Evaluation

In this section, we compare and analyze the performance of FlexiS, LEDBAT and LEDBAT++. The LBE CCAs were tested in a variety of scenarios in both emulated networks and in the Internet. In both emulation and Internet tests, background or LBE data packets were only sent in one direction. The direction in which LBE data (or ack) packets travel is referred to as forward (or reverse) direction. Because foreground flows were all BE TCP flows in emulation tests, they are referred to as BE flows in the following text for clarity of expression. Tcpdump was used to capture packets, Tcptrace was used to analyze trace files and bash scripts were used to run the experiments and produce results.

A. Implementation of LEDBAT and LEDBAT++

We have adapted the LEDBAT Linux kernel module implemented by Silvio Valenti [38] for their study [39] to conform to RFC6817. The major change is the CWND update function. In our implementation, the per ack CWND adjustment,
as specified in RFC6817, is in proportion to the number of bytes acknowledged and the (normalized) distance between current delay and the target delay. The adapted code is available at [40]. In our implementation, OWDs are calculated using the timestamp values and timestamp echo replies embedded in returning ACKs at the sender. However, a direct subtraction of these values is not viable, because the two timestamps might be generated by clocks with different resolutions. A special algorithm is used to estimate the receiver’s Timestamp Clock Resolution (TCR). The accuracy of the estimator is seriously affected by congestion in the forward direction. LEDBAT with this estimator enabled has persistently low throughput in various scenarios. Because the receivers’ TCRs were known in all our emulation and Internet tests, they were hardwired in LEDBAT’s code. This ensures that the performance of LEDBAT is not affected by the estimator in the tests. The values for LEDBAT parameters were set in accordance with RFC6817 as follows. TARGET = 100 ms. BASE_HISTORY = 10. The same GAIN was used for both CWND increase and decrease and was set to 1. Noise filter was set to NULL. And MIN_CWND = 2.

We also implemented a Linux kernel module for LEDBAT++, which was adapted from LEDBAT. The implementation conforms to LEDBAT++ draft version 1 [4] and is available at [41]. The draft proposes a per RTT CWND reduction equation. However, it is very difficult to implement in that the amount to be decreased is affected by QD, which changes with each ACK. Therefore, we implemented the per ACK CWND reduction equation proposed in [35], base on which the CWND update functions of LEDBAT++ were developed. In our implementation, the modified slow start is used after periodic slow down. The values for LEDBAT++ parameters were set in conformity with the draft as follows. TARGET = 60 ms. BASE_HISTORY = 10. Min filter was used as current delay filter and its length was set to 4. And MIN_CWND = 2.

B. Performance Metrics

This subsection defines performance metrics. All metrics except retransmission rate and Jain’s fairness index are only used in emulation tests.

Convergence time $CT$ is defined in equation 5 as the time taken for an LBE connection to increase its CWND to available BDP for the first time since connection establishment. Available BDP is defined as the product of AB and base delay.

$$CT = t_c - t_s$$

where $t_s$ is the establishment time of the LBE connection and $t_c$ is the time when CWND of the LBE connection reaches available BDP for the first time.

Bandwidth utilization $U$ measures the throughput of one or more LBE flows as a percentage of AB. It is defined in equation 6.

$$U = \frac{x_i}{B_a} \times 100$$

where $B_a = C - B_u$ is the average AB, $C$ is the bottleneck link capacity and $B_u$ is the average bandwidth consumed by BE flows. $B_a$ is measured by bmon at the bottleneck Network Interface Card (NIC). $x_i$ is the aggregate throughput of all LBE flows that share one bottleneck. A utilization greater than 100% indicates over-use of AB, which implies that the LBE flows are “stealing” bandwidth from BE flows.

Throughput degradation $TD$ measures what percent of throughput BE flows lose due to the contention of LBE flows. It is defined in equation 7.

$$TD = \frac{x_o - x_w}{x_o} \times 100,$$

where $x_o$ is the average throughput of BE flows when they run alone and $x_w$ is the average throughput of the BE flows when LBE flows run in the background.

QD measures the cumulative time that a packet spends waiting in various bottleneck queues along its round trip route. Let $QD_i$ be the QD measured by the $i_{th}$ packet. Then $QD_i = RTT_{i} - RTT_{base}$. Because numerous QD samples can be obtained during an experiment, the 90th percentile ($P_{90}(QD)$) is used to indicate the overall degree of QD.

Extra QD ($\Delta P_{90}(QD)$) is the extra amount of QD added by LBE flows in addition to the original QD inflicted by BE flows. It is calculated as the difference between the 90th percentile QD measured by a BE flow when no LBE flows present ($P_{90}(QD_o)$) and when LBE flows run in the background ($P_{90}(QD_w)$). Equation 8 gives its definition.

$$\Delta P_{90}(QD) = P_{90}(QD_w) - P_{90}(QD_o).$$

Retransmission rate $RR$ is defined in equation 9. It measures the number of retransmitted packets as a percentage of all packets sent. It is used as an estimation of loss rate.

$$RR = \frac{n_r}{n_t} \times 100,$$

where $n_r$ is the total number of packets retransmitted by all LBE (or BE) connections that share one bottleneck and $n_t$ is the total number of packets sent by the same connections.

Extra retransmission rate $\Delta RR$ is the extra amount of $RR$ experienced by BE connections due to the competition of LBE connections. It is defined in Equation 10.

$$\Delta RR = RR_w - RR_o,$$

where $RR_o$ is the RR of the BE connections when they run alone and $RR_w$ is the RR of the BE connections when LBE connections run in the background.

Jain’s Fairness Index [42] $JFI$ is used to measure the degree of equitable distribution of AB among contending flows. It is defined in equation 11.

$$JFI = \frac{\left( \sum_{i=1}^{n} x_i \right)^2}{n \sum_{i=1}^{n} x_i^2},$$

where $x_i$ is the average throughput of the $i_{th}$ LBE flow, $n$ is the number of LBE flows sharing the same bottleneck. $JFI$ is between 0 and 1. The larger the value the higher the fairness.

C. Emulation Tests

In this subsection, we study the performance of FlexiS with a network emulator.
1) Emulation Tools: Emulation tests were conducted on virtual networks emulated by Common Open Research Emulator (CORE) [43], [44]. CORE builds a representation of a real computer network that runs in real time and provides an environment for running real applications and protocols. All virtual nodes on a physical host run the same kernel and share the same set of resources. The limiting factor of performance is the number of times that the operating system needs to handle a packet but not the number of hops and the size of the packets.

Our main emulation platform was a desktop PC with an Intel dual-core 2.9 GHZ processor and a 4 GB memory installed with Ubuntu Desktop 20.04, Linux kernel v5.4, and CORE 7.0. We did not observe obvious performance degradation in most of the tests except the scalability test (section IV-C.6), in which we noticed lowered link utilization of all LBE CCAs when bottleneck capacity is low (caused by hardware limitation) and higher intrusion of FlexiS when bottleneck capacity is high (because CORE 7.0 does not emulate transmission delay). After redoing the scalability test on a PC with a 10-core 2.5 GHZ processor and a 16 GB memory installed with Ubuntu Desktop 22.04, Linux kernel v5.19, and CORE 9.0 (which supports transmission delay), the LBE CCAs have normal performance.

All BE flows except greedy ones were generated by Multi-Generator (MGEN) [45]. It is a packet-level traffic generator capable of generating UDP flows and responsive TCP connections. Standard TCP/IP socket is used to establish a TCP connection between specified source and destination hosts. The source host generates packets based on provided packet size and packet inter-departure time, which can be explicitly specified by setting a statistical distribution or implicitly extracted from a binary trace file (the latter is called trace cloning).

Because MGEN does not support greedy traffic pattern, bulk data transfer applications were written (in the language C) to generate data for LBE and greedy BE connections. The core functions of these applications are the same – the sender pre-fills its send buffer with data, once it is connected to the receiver, it sends the data in its buffer to the receiver without pause. This can to a large extent prevent the sender from being limited by data. The receiver simply discards all the data it receives.

2) Traces Used: Analytical traffic models, albeit easy to obtain and manipulate, cannot accurately model real world traffic [46]. Therefore, in the majority of the emulation tests, we used traces of real traffic to load the virtual networks. The MAWI traffic archive [47] hosts a huge number of Internet traffic trace files captured at various sample points of the WIDE backbone in Japan since 1999. This makes it possible to load our virtual networks without sophisticated processing of the original traces. Because no single trace can represent Internet traffic as a whole, we intentionally selected a large number of traces with diverged bit rates, burstiness and composition. The traces chosen were all captured at sample point F, which is a transit link of WIDE to its upstream ISP in the U.S. TABLE I lists all the trace files used by the emulation tests.

Year specifies in which year a trace was captured. $\bar{R}$ is the average rate of the trace in Mbps. And s is the standard deviation of rate in Mbps.

3) The Dumbbell Topology: The majority of emulation tests were conducted on a dumbbell topology (Fig. 2), which is adapted from the one originally proposed in [48]. The OWD of each link are annotated by the link. They are largely the same as the ones used in [48]. The intention of choosing these OWDs is to make the RTTs of the paths (shown as a column on the right hand side of Fig. 2) fall within the typical Internet RTT range.

In the dumbbell and other emulated networks, links are symmetrical. A link is denoted by the two nodes at both ends of it connected with a line segment as $N1 \rightarrow N2$. When there is only one path between two end hosts $H_i$ and $H_j$, the path is denoted as $H_i \cdot \cdot \cdot H_j$. If there are more than one paths between two hosts, a path is denoted by all nodes on it connected with line segments. A flow is depicted by its source S and destination D connected with an arrow as $S \rightarrow D$.

The central link $R1 \rightarrow R2$ is the bottleneck. Its capacity varies from one test to another. Unless otherwise noted, the capacity of peripheral links are 1 Gbps. The default Queuing DISCIpline (QDISC) is tail-drop First-In First-Out queue in unit of Packets (PFIFO). The bottleneck buffer defaults to 1.5 BDP of a 100 ms connection, which equals a maximum of 150 ms QD (chosen to be larger than LEDBAT’s TARGET). LBE data and ACK flows are depicted in the graph.

Some tests require to load the dumbbell network with MAWI traces. In this case, two traces are needed with one (wideX) used as forward direction load and another (wideY) as reverse direction load. Because there are 9 paths in the network from one side to another, each of wideX and wideY is split into 9 sub-traces. During an experiment, MGEN is used to clone a live TCP connection from a sub-trace. In the end, there are two TCP connections between each pair of hosts with their data traveling in reverse directions. Our experiments showed that the synthesized traffic at the bottleneck inherits the key characteristics of wideX (or wideY).

4) Tests Overview: In all emulation tests, the emulated routers used OSPF as routing protocol. TCP CUBIC was the default CCA for BE flows. LBE flows used the following CCAs: FlexiS, LEDBAT without slow start (LEDBAT-BA), LEDBAT with slow start (LEDBAT-SS) and LEDBAT++. All LBE CCAs used their default parameter settings.

In a test, one network parameter (e.g. bottleneck capacity) was varied within a range. Other parameters were fixed. For
each value in the range, an experiment was conducted. In a typical experiment, the flows using BE CCA were let first run alone and then run simultaneously with each of the LBE CCAs in turn. When BE and LBE flows ran simultaneously, the BE flows were started first. A break of $B$ seconds was inserted between the start of BE and LBE flows. When more than one LBE flows were involved, the flows were started at $I$-second intervals. $B$ and $I$ were respectively 60 and 10 seconds unless otherwise specified. The performance measurement started from the establishment of the first LBE flow and lasted until $300 + I$ seconds after the establishment of the last LBE flow. Each experiment was repeated for 10 times and the average is reported.

In all experiments, TCP send and receive buffers of the emulated nodes were adjusted so that they did not become the limiting factors of sending rate. CPU backlog queue was increased. tcp_no_metrics_save was enabled in order to make the performance of each LBE CCA independent of each other.

5) Default Values of Protocol Parameters:

Before presenting the evaluation results, we first describe how the default values of protocol parameters were selected.

Due to the large solution space, we used a method which progressively discards candidate values that yield bad performance. To be specific, we first determined a range for each parameter based on initial rough calculations. The chosen ranges for $\alpha$, $\beta$ and $\gamma$ were $[20, 200]$, $[2, 20]$ and $[0.5, 0.95]$ respectively. We then set each parameter to a value in its range and studied the performance of FlexiS under a specific scenario. The candidate values that resulted in bad performance were excluded from further consideration. The rest of candidate values were further evaluated in other scenarios (different bottleneck capacities, QDISCs and loads etc.) until one candidate value was left for each parameter. Using the above method, the default values for $\alpha$, $\beta$ and $\gamma$ were set to 100, 10 and 0.85 respectively.

Next, we describe in detail the first round value selection for $\tau$ and $\theta$. Each of the parameters was varied within the range of $[10, 100]$ with a step size of 10. $\alpha$, $\beta$ and $\gamma$ were set to their default values. A hundred experiments were conducted with each evaluating a unique $\tau$ and $\theta$ combination. The experiments were conducted on the dumbbell topology. The bottleneck capacity was set to 100 Mbps. QDISC of the bottleneck router was PIE [49]. The target of PIE was set to 15 ms, which is the default value. The intention of using PIE instead of the default QDISC of PFIFO is to ensure first and foremost that FlexiS does not fail when AQM is deployed. The forward direction load was wide42, and reverse direction load was wide25. Wide42 has a very high variation in rate. The reason we chose it for the first round selection is to guarantee that FlexiS can meet its design objectives under harsh conditions like this. One FlexiS flow $H_4 \rightarrow H_7$ was investigated. After each experiment, the utilization $U$ of the FlexiS flow and the throughput degradation $TD$ of the BE flows were calculated and a utility score (called $uScore$) was derived using the following equation.

$$uScore = U - TD^3 + 93.$$  

Fig. 3 shows the results. The depth of the color is determined by $uScore$. The 100 combinations were ranked based on $uScores$ and the top 15% were chosen for further evaluation. With a $uScore$ of 134, ($\tau = 60$, $\theta = 30$) ranks number 10. It is finally chosen as the default because other combinations that have higher utility scores have lower scores in other scenarios. Due to space limitation, other selection results are not presented here.

6) Scalability Test: The goal of this test is to examine how well FlexiS scales to various ABs. The capacity of the bottleneck link was set to each of the values in turn: 1, 5, 10, 50, 100, 500, and 1000 Mbps. The capacity of the peripheral links were 10 Gbps. The network was not loaded with any BE flows. One LBE flow $H_3 \rightarrow H_6$ was examined and it was let run for 120 seconds. The performance of the LBE CCAs are shown in Fig. 4. Utilization and QD were measured after convergence for 100 seconds.
Convergence time of all LBE CCAs increase with the increase of AB. LEDBAT-SS and LEDBAT++ take less time to acquire large amounts of bandwidth due to the slow start, while FlexiS and LEDBAT-BA are comparatively slower in acquiring large amounts of bandwidth.

At the cost of higher QDs, LEDBAT (henceforth, used to represent both LEDBAT-BA and LEDBAT-SS) has high steady state utilization for all ABs. FlexiS achieves equivalent utilization (around 96%) when bottleneck capacity is high ($\geq 50$ Mbps) but moderately lower values (between 80% and 86%) with small capacity ($\leq 10$ Mbps) bottlenecks. The reason is that as the capacity of the bottleneck decreases, the transmission delay of a given sized packet increases. When sending rate is close to AB, a slight imperfection in packet pacing can cause abrupt increase in RTT, which can make FlexiS back off prematurely. Further, a given amount of rate over-decrease is translated into a higher reduction in utilization for small capacity bottlenecks.

The $P_{90}(QD)$ induced by LEDBAT-SS and LEDBAT++ are around their respective targets. FlexiS induces the least amount of $P_{90}(QD)$ among all LBE CCAs. The maximum $P_{90}(QD)$ induced by FlexiS in all scenarios is 15 ms. LEDBAT-SS suffers from higher RR (between 0.13% and 0.68%) than the rest of LBE CCAs due to the use of the unmodified slow start. Other LBE CCAs have nearly zero RR.

7) Responsiveness Test: The goal of this test is to examine how well FlexiS responds to changes in AB. Bottleneck capacity was set to 100 Mbps. The network was loaded with an on/off BE flow H4→H7, which was generated by MGEN as a responsive TCP connection. It transferred at a constant packet rate of 6250 packets per second (75 Mbps) during the “on” period and 0 Mbps during the “off” period. The “on” and “off” periods had the same duration, which was set to each of the values in turn: 0.001, 0.01, 0.1, 1, 10 and 100 seconds. One LBE flow H5→H8 was studied. The starting interval between BE and LBE flows was 10 seconds. The performance of the LBE CCAs are shown in Fig. 5.

The utilization of all LBE CCAs are affected by on/off duration of the BE flow. LEDBAT and LEDBAT++ have reduced utilization when the duration is equal to or greater than 1 second. FlexiS has lower utilization when the durations are 0.1 and 1 seconds. In such scenarios, the “on” period is long enough to make FlexiS detect the rate increase of the BE flow and reduce its rate. In the meanwhile, the “off” period is too short to allow FlexiS to fully utilize bandwidth released during this period.

Except LEDBAT-SS, all LBE CCAs cause negligible throughput degradation of the BE flow. Generally, the extra $P_{90}(QD)$ induced by LEDBAT and LEDBAT++ have a positive correlation with their utilization. FlexiS induces low extra $P_{90}(QD)$ in most cases. All LBE CCAs add negligible retransmission rate to the BE flow.

LBE flows with different RTTs were also studied. FlexiS has similar performance irrespective of its RTT. Whereas, LEDBAT and LEDBAT++ are more affected by RTT. Their utilization and intrusion increase with the decrease of RTT. Due to space limitation, the results are not shown here.

8) Forward Direction Load Test: This test studies how FlexiS performs with varying levels of traffic in its forward direction. Two groups of experiments were conducted. In both groups, the bottleneck link capacity was set to 100 Mbps. Each of the following traces was used as forward direction load in turn: wide11, wide33, wide52, wide74, wide94 and wide108. The reverse direction load was always wide11. These traces all have low variation in rate. The intention is to ensure load stability while we are introducing burstiness to the network. The first group of experiments examined the performance of one ($n = 1$) LBE flow H4→H7. While the second group investigated nine ($n = 9$) LBE flows, which used three different paths: H3···H6, H4···H7, and H5···H8. There were three flows on each path. Fig. 6 shows the results. Utilization for the 108% loaded scenario is not shown in the figure since there is no AB left in this case and utilization cannot be calculated.

The utilization of a single FlexiS or LEDBAT++ flow declines drastically from around 90% to slightly over 40%. The declination of FlexiS’ utilization can be attributed to two factors. On the one hand, FlexiS operates at a slower frequency than that of AB variation. To be more specific, when the BE flows increase their bandwidth demand, FlexiS reduces its rate immediately. However, when bandwidth demand is reduced, FlexiS increases its rate conservatively. Therefore, the bandwidth released by the BE flows cannot be timely utilized, which results in bandwidth wastage. On another hand, for a given amount of bandwidth wastage, the smaller the AB the lower the utilization. Fig. 6b shows that multiplexing can improve utilization of both FlexiS and LEDBAT++.

A simple FlexiS flow has negligible impact on BE flows. Nine FlexiS flows inflict higher TD when the bottleneck is saturated but do not increase $\Delta P_{90}(QD)$ significantly. The major causes for the higher TD inflicted by FlexiS are: (1) FlexiS does not decrease CWND below 2 segments; and (2) when FlexiS cannot obtain $\sigma$ RTT samples when $L_D \geq \tau$, it will increase CWND.

Increasing the number of LEDBAT connections can also increase its intrusion. When the bottleneck is 108% loaded, nine LEDBAT-BA connections take away almost one fourth of bandwidth (22.6%) from BE flows. LEDBAT and LEDBAT++ inflict higher extra $P_{90}(QD)$ than FlexiS in most
Fig. 6. Impact of forward direction load on the performance of LBE CCAs. NB the Y axes of figures on the left and right hand side columns have different ranges.

Fig. 7. The impact of reverse direction load on the performance of LBE CCAs.

9) Reverse Direction Load Test: This test examines the performance of FlexiS when traffic level in the reverse direction varies. The bottleneck link capacity was set to 100 Mbps. The forward direction load was always wide11. The reverse direction paths were loaded with each of the traces in turn: wide11, wide33, wide52, wide74, wide94 and wide108. One LBE flow H4 → H7 was studied. The results are shown in Fig. 7.

Fig. 8. The impact of RTT on the performance of LBE CCAs.

10) RTT Test: The goal of this test is to investigate the impact of RTT on the performance of FlexiS. We have tested the LBE CCAs with traces having low variation in rate. Next, we will study how they behave with traces that have higher variation in rate. In this test, wide42 (stddev = 20.76 Mbps) was used as the forward direction load and wide25 as the reverse direction load. In order to facilitate cross-test comparison, this pair of traces were also used in most of the tests presented in the rest of this subsection. The capacity of the bottleneck link was set to 100 Mbps. One LBE flow was examined and it took each of the nine paths in turn. Please refer to Fig. 2 for the RTT of each path. The results are shown in Fig. 8.
amplitude of oscillation of AB. As a result, the bandwidth wasted is increased, which is translated into a lower utilization.

The intrusion of FlexiS increases moderately with the increase of RTT. This is caused by the difference in feedback delay for connections with different RTTs, and by the fact that FlexiS keeps increasing its rate before the arrival of congestion feedback. In comparison, the intrusion inflicted by LEDBAT increases with the decrease of RTT.

11) AQM Test: This test studies how Active Queue Management (AQM) algorithms affect the performance of FlexiS. LEDBAT has been shown to have a reprioritization problem by Gong et al. [15] – it becomes as aggressive as TCP NewReno when AQM is deployed at the bottleneck. In this test, we study if FlexiS also suffers from similar problems. In the first group of experiments, the capacity of the bottleneck link was set to 100 Mbps. Wide42 was used as forward direction load and wide25 as reverse direction load. The QDISC of the bottleneck router was set to PIE. Limit of PIE was set to 1250 packets, which corresponds to a 150 ms hard limit on QD. Target of PIE was set to each of the values in turn: 5, 10, 15, 20 and 25 ms. One LBE flow H4→H7 was studied. The starting interval between the BE and LBE flows was 10 seconds. The results are shown in Fig. 9d.

In the second group, the network was loaded with wide11 in both directions. The goal is to examine fairness in an environment without any interference from cross traffic. In the second group, the network was loaded with wide11 in both directions. The goal is to study fairness in a more realistic environment and analyze how cross traffic affects fairness of the LBE CCAs. In both groups, the capacity of the bottleneck link was set to 20 Mbps. The bottleneck buffer was set to 4 BDP of a 100 ms connection. Fairness between three LBE flows was measured. All of them used the path H4→H7. LBE flows were started at 30-second intervals. Fairness was measured for 600 seconds. The results are shown in Fig. 10.

FlexiS and LEDBAT++ achieve high intra-RTT fairness in both unloaded and loaded scenarios with FlexiS’ JFI > 0.99 in both scenarios. While LEDBAT flows cannot fairly share AB when the network is unloaded, which is the result of the so-called late comer advantage problem [3], [13] – the late coming flow takes the QD maintained by early coming flows as part of base delay, therefore become more aggressive than precedent flows. In the loaded scenario, BE flows introduce more delay dynamics, which gives the late coming flows opportunity to discover true base delay. Therefore fairness of LEDBAT is improved.

The third and fourth groups of experiments evaluate inter-RTT fairness. In the third group, the network was unloaded. In the fourth group, Wide42 was used as forward direction load and wide25 as reverse direction load. In both groups of experiments, the capacity of the bottleneck link was set to 100 Mbps. The bottleneck buffer was set to 4 BDP of a 100 ms connection. The fairness between two LBE flows was studied. Starting interval of LBE flows was 10 seconds in the
third group and 30 seconds in the fourth group. Fairness was measured for 300 seconds. The first LBE flow had a fixed path H4 · · · H7 and the second LBE flow took each of the remaining paths plus one more (P10) in turn. P10 (not shown in Fig. 2) has an RTT of 800 ms, which is used to simulate a satellite link. Fig. 11 presents the results.

FlexiS has high inter-RTT fairness (0.974 < JFI ≤ 0.999) when the bottleneck is loaded but slightly lower fairness in an unloaded network when RTT difference of the two flows is large. The reason is that flows may take RTT samples at different times and with different frequencies. When the change in RTT is small (which is typical in an unloaded network), it may be detected by one flow but not by another, which makes the former reduce rate more frequently and consequently obtain less bandwidth share. Smaller bottlenecks can exacerbate this problem, which results in even lower JFIs (graphs are not shown due to space limitation). In a loaded network, packet bursts from cross traffic can make RTT increase more aggressively. As a result, the likelihood for all contending FlexiS flows to detect the increase in RTT becomes higher.

Fairness of LEDBAT-BA is more affected by the difference in RTT. The larger the difference, the lower the JFI. LEDBAT-SS has a very unstable performance in an unloaded network. Its JFI varies greatly between different experiments (i.e. RTT differences) and between runs of the same experiment. Generally, cross traffic has no significant impact on LEDBAT-BA but can improve the fairness of LEDBAT-SS. In an unloaded network, the overall JFI obtained by LEDBAT++ is good, except when the RTT of the second flow is 800 ms. This is because LEDBAT++ employs a dynamic gain to reduce the speed difference in CWND increase. But it is only effective when a flow’s RTT falls within the range of [7.5, 120] ms. Cross traffic has adverse impact on the fairness of LEDBAT++.

13) Route Change Test: An Internet route between two end hosts may change during the lifetime of a TCP connection due to the moving of an end host or failure of a link. This test examines the robustness of FlexiS in the presence of route change. The test was conducted on a diamond topology, which is illustrated in Figure 12.

The bottleneck link is H1−R1. It has a capacity of 10 Mbps. The capacity of the rest of links are 1 Gbps. The OWD of links are annotated in the graph. The bottleneck router buffer is such set that the maximum QD is 150 ms. There are two paths between H1 and H2: P1 = H1−R1−R2−R4−H2 and P2 = H1−R1−R3−R4−H2. The default path is P1. The OWD of P1 is 8 ms. P2’s OWD was set to 24 ms in the first group of experiments and to 124 ms in the second group. The network was not loaded with any BE flows in all experiments.

A simple route change scenario was studied: a link is being announced alternatively as up and down by a router due to the malfunction of a NIC. In our specific case, eth1 of R1 is the malfunctioning NIC and R1−R2 is announced as up and down alternatively. When eth1 is down, route between H1 and H2 is automatically switched from P1 to P2 by the routers. When eth1 is brought up, P1 is used again.

In the experiments, the initial state of eth1 was set to on and then to down and finally to up again. Eth1 of R1 stayed in up or down for the same amount of time. Up/down duration F was set to each of the following values in turn: 10, 40, 160 and 640 seconds. A single LBE flow H1→H2 was studied. It ran for 3 × F seconds. The results are shown in Fig. 13.

Thanks to its queue trend based congestion detector, the utilization of FlexiS (maintained between 84% and 93% for both scenarios) is almost not affected by the OWD of P2. However, it is slightly impacted by up/down duration. In comparison with FlexiS, the utilization of LEDBAT and LEDBAT++ are seriously affected by OWD of P2. When up/down duration is 10 seconds, their utilization drop by approximately 50% when OWD of P2 is increased from 24 ms to 124 ms. This is because when the difference between the OWDs of P1 and P2 is greater than or equal to the targets of LEDBAT and LEDBAT++, the two LBE CCAs will take this difference as QD and keep decreasing their rates until base delay is updated. Because base delay of LEDBAT and LEDBAT++ can be updated to the OWD of P2 when up/down duration is 640 seconds, their utilization is improved greatly. The retransmission rate of all LBE CCAs except LEDBAT-SS are negligible, so it is now shown here.

14) Multiple Bottlenecks Test: In a more realistic scenario, a data flow can traverse multiple congested gateways and wait in various queues for transmission. This test investigates the performance of FlexiS in such scenarios. The topology
Fig. 14. A parking lot topology.

Fig. 15. Performance of LBE CCAs when BE flows run through multiple congested gateways.

Fig. 16. Performance of LBE CCAs when 32 LBE flows ran through multiple bottlenecks.

AB at each bottleneck are different. The intention is to create a virtual network that bears a closer resemblance to the real Internet. Fig. 16 shows the results.

Thirty-two FlexiS flows have a utilization of 92.44%, a max(TD) of 6.89%, a max($\Delta P_{90}(QD)$) of 1 ms and a max($\Delta RR$) of 0.018%. All other LBE CCAs over-utilize AB. As a result, they are more intrusive to the BE flows.

With the same experiment setup, the LBE CCAs have lower utilization and intrusion when only 1 flow runs across multiple bottlenecks. The results are not presented here due to space limitation.

D. Internet Tests

The goal of the Internet test is to study the performance of FlexiS in a realistic environment. The LBE CCAs were tested on two Internet paths – Data center to Data center (D2D) and Data center to Residence (D2R). On each end host, TCP send and receive buffers were expanded and CPU backlog queue limit was increased. Tcp_no_metrics_save was set to 1.

For the D2D path, the sender was a Linode virtual machine locating in Atlanta USA (referred to as atlanta.taht.net). The receiver was another Linode virtual machine in Newark USA (referred to as newark.taht.net). The observed minimum RTT of this path was 17 ms. The throughput was upper bounded by the sender’s subscription plan to 4 Gbps. The two machines were installed with Ubuntu Desktop 22.10 and Linux kernel v5.19.0.

Two tests were conducted on this path. The first examines the throughput, RTT and retransmission rate of a single LBE flow. The second studies fairness between five competing LBE flows of the same kind. The first test was carried out on 2023.03.26, 2023.03.28, 2023.03.30 and 2023.04.01. The dates were so chosen that both work days and weekend were sampled. On each test day, a random time was selected to start the test. During the test, the sender made four successive 60-second bulk data transfers to the receiver with each transfer using a different CCA: FlexiS, LEDBAT-SS, LEDBAT++ or CUBIC. The starting order of the CCAs was randomized. A break of at least 15 seconds was inserted between successive transfers. The test was repeated once per hour for twenty-four times a day.
The average throughput, $99_{th}$ percentile RTT and retransmission rate of each CCA are summarized in box-whisker plots shown in Figs. 17a, 17b, and 17c. The three bars of a box from bottom to top mark the 25th (Q1), 50th (Q2) and 75th (Q3) percentiles respectively. The whiskers extend to 1.5 times of Inter-Quartile Range ($IQR = Q3 - Q1$). The two bars at the ends of the whiskers are minimum and maximum values within the whisker delimited boundary. Values outside the boundary are considered as outliers, which are represented as solid dots in the figure.

As is shown in Fig. 17a, FlexiS is able to achieve comparatively higher throughput in most of the runs. The median throughput of FlexiS, LEDBAT-SS, LEDBAT++ and CUBIC are 2835, 1028, 590, and 2171 Mbps respectively. The IQRs for the CCAs in the same order are 997, 2083, 889, and 2615 Mbps. After analyzing the trace files, we discovered that in most of the runs all CCAs except FlexiS encountered a series of packet losses after slow start, which forced them to reduce their rates well below AB and from then on further losses prevented them from increasing their rates to the AB. Figs. 17b and 17c show that the high throughput of FlexiS is not a result of high intrusion. In 75% of runs, the $P_{90}(RTT)$ of FlexiS is less than or equal to 23 ms. In comparison with FlexiS, the 75th percentile $P_{90}(RTT)$ of LEDBAT-SS, LEDBAT++, and CUBIC are 70, 63, and 61 ms respectively. FlexiS also has consistently low retransmission rate ($\max(RR) = 0.045\%$) in all runs. In comparison with FlexiS, the 75th percentile $P_{90}(RTT)$ of LEDBAT-SS, LEDBAT++, and CUBIC are 70, 63, and 61 ms respectively. FlexiS also has consistently low retransmission rate ($\max(RR) = 0.045\%$) in all runs.

The second test was carried out on 2023.04.22, 2023.04.26, 2023.04.28, and 2023.05.01. On each chosen day, a random time was selected to start the test. During the test, the sender made four successive 60-second bulk data transfers to the receiver with each transfer using a different CCA: FlexiS, LEDBAT-SS, LEDBAT++, or CUBIC. The starting order of the CCAs was randomized. The test was repeated once per hour for twenty-four times a day. Based on our trace analysis, each of FlexiS and LEDBAT++ has one run in which the five concurrent connections have remarkably distinctive RTT statistics, which implies that they did not share the same bottleneck. Data of these runs have been excluded from the final results (see Fig. 17d).

In most of the runs, FlexiS has high JFIs. The median JFI of FlexiS is 0.97 and that of LEDBAT-SS and LEDBAT++ are 0.87 and 0.56 respectively. In some runs, FlexiS obtained very low JFIs (shown as outliers). Due to the lack of control over the Internet paths under investigation, we cannot give a confident explanation why FlexiS obtained such low JFIs. It might be the result of flows not sharing the same bottleneck or other random factors. LEDBAT-SS has comparatively higher JFI than LEDBAT++. Considering that it also has higher RR, we believe that frequent packet losses and faster rate increase after backing off are two major contributors to its higher fairness.

In the D2R test, newark.taht.net was the sender and a home PC located in mainland China was the receiver. The observed minimum RTT was 218 ms and the throughput was limited by the receiver’s subscription plan to 100 Mbps. The home PC was installed with Ubuntu Desktop 22.04 and Linux kernel v5.19.0. The test was carried out on 2023.04.10, 2023.04.12, 2023.04.14, and 2023.04.16. On each test day, a random time was selected to start the test. During the test, the sender made four successive 60-second bulk data transfers to the receiver using each of the LBE CCAs in turn: FlexiS, LEDBAT-SS, LEDBAT++, and CUBIC. The starting order of the CCAs was randomized. A break of at least 15 seconds was inserted between successive transfers. The test was repeated once per hour for twenty-four times a day. In eleven runs of the test, the sender could not get any ACK from the receiver, so they were excluded from the final results. Fig. 18 shows the performance of the LBE CCAs.
of all valid runs is around 300 ms. And they have abnormally high retransmission rates in at least 25% of the runs. To be specific, the 75th percentiles of RR are 11.6%, 8.6%, 10.1%, and 15.5% for FlexiS, LEDBAT-SS, LEDBAT++, and CUBIC respectively. Because no abnormality was observed in the traces, we believe that the high RR is the result of severe congestion.

Second, it scales to ABs that differ by several orders of magnitude. In the meantime, it also has a slower convergence time than slow start based LBE CCAs. Bandwidth utilization of FlexiS is independent of RTT, AQM and the amount of AB but is affected by bottleneck capacity and the oscillation of AB. Usually, the utilization is low when bottleneck capacity is small. And as both frequency and amplitude of oscillation of AB increase the utilization of FlexiS decreases. The severity of this issue has a negative correlation with the number of FlexiS flows that share one bottleneck. Therefore, in a realistic setting such as the Internet, effective utilization of AB should not be a serious problem. We will leave improving a single flow’s bandwidth utilization as a future work.

Third, due to the use of RTT in congestion detection, reverse direction congestion can reduce FlexiS’ bandwidth utilization in the forward direction. One option is to replace RTT with OWD. However, an accurate estimate of OWD is difficult to obtain. On the other hand, TCP’s timestamps cannot be directly used in OWD estimation because the sender and receiver may have different TCRs. On the other hand, the clocks of the sender and receiver may have different skews and drifts, which may make estimated OWDs increase over time when the network is not congested. We will leave tackling this problem as a future work.

Fourth, FlexiS has high intra-RTT fairness in both loaded and unloaded networks regardless of bottleneck capacity. It also has high Inter-RTT fairness in most situations. However, in an unloaded network with small bottleneck capacity, FlexiS flows with large RTT difference will have reduced fairness.

Fifth, FlexiS preserves low priority when AQM algorithms are deployed at the bottlenecks.

Sixth, FlexiS adapts to route changes quickly. Its performance is almost not affected by the RTT of the alternative route but slightly impacted by route change intervals.

Seventh, because FlexiS employs a Theil-Sen estimator to estimate the trend in RTT, it has comparatively higher storage and computation demands.

Finally, it is a sender side only CCA, which makes it easy to deploy.

V. CONCLUSION

We have proposed a novel LBE CCA named FlexiS. It monitors RTT through a sliding observation window. If RTT samples in the window have an upward trend, FlexiS assumes that the network is congested and it will reduce rate by a fixed percent. It will otherwise increase sending rate according to a cubic function of time.

Extensive emulation tests and preliminary Internet experiments showed that FlexiS has the following properties. First, in most cases, it inflicts very low intrusion to BE flows. In other CCAs, it experiences high RTTs in some cases, which may cause a decrease in throughput. Second, FlexiS shows that the RTT in period II is always higher than that in period I, which suggests that the network was congested in period II. Therefore, the low throughput of FlexiS in period II should be the result of yielding bandwidth to cross traffic at times of congestion.

ACKNOWLEDGMENT

The author would like to thank the anonymous reviewers for their time and effort in providing valuable feedback to this article. She truly appreciate the help of Dave Täht on making their Linode virtual machines atlanta.taht.net and newark.taht.net available to them for the Internet tests. She would also like to thank Michael Welzl for the introduction of LBE services and the CORE emulator and for the suggestions and advice given to earlier design proposals of the LBE CCA. This work is a continuation of a previous effort in designing an LBE CCA, which was funded by the University of Oslo.

REFERENCES

[1] A. Venkataramani, R. Kokku, and M. Dahlin, “TCP nice: A mechanism for background transfers,” ACM SIGOPS Oper. Syst. Rev., vol. 36, no. SI, pp. 329–343, Winter 2002.
[2] A. Kuzmanovic and E. W. Knightly, “TCP-LP: Low-priority service via end-point congestion control,” IEEE/ACM Trans. Netw., vol. 14, no. 4, pp. 739–752, Aug. 2006.
[3] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind. Low Extra Delay Background Transport, document RFC 6817, 2012. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6817
[4] (2020). LEDBAT++: Congestion Control for Background Traffic. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-irtf-icrg-ledbat-plus-plus-01
[5] M. Bagnulo, A. Garcia-Martinez, G. Montenegro, and P. Balasubramanian. (2022). rLEDBAT: Receiver-Driven Low Extra Delay Background Transport for TCP. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-irtf-icrg-rledbat-05
[6] Z. Wang and J. Crowcroft, “A new congestion control scheme: Slow start and search (Tri-S),” ACM SIGCOMM Comput. Commun. Rev., vol. 21, no. 2, pp. 9–16, Apr. 1991.
[7] Q. Li. Source Code of Flexis. Accessed: Sep. 30, 2023. [Online]. Available: https://github.com/finale77/Flexis
[8] R. Jain, “A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks,” ACM SIGCOMM Comput. Commun. Rev., vol. 19, no. 5, pp. 56–71, Oct. 1989.
[9] Z. Wang and J. Crowcroft, “A new congestion control scheme: Slow start and search (Tri-S),” ACM SIGCOMM Comput. Commun. Rev., vol. 21, no. 1, pp. 32–43, Jan. 1991.
[10] Z. Wang and J. Crowcroft, “Eliminating periodic packet losses in the 4.3-Tahoe BSD TCP congestion control algorithm,” ACM SIGCOMM Comput. Commun. Rev., vol. 22, no. 2, pp. 9–16, Apr. 1992.
[11] L. Brakmo, S. O’Malley, and L. Peterson, “TCP Vegas: New techniques for congestion detection and avoidance,” in Proc. ACM SIGCOMM, 1994, pp. 24–35.
