FLOWER – Fuzzy Lower-than-Best-Effort Transport Protocol

Si Quoc Viet Trang[1], Emmanuel Lochin[1], Cédric Baudoin[2], Emmanuel Dubois[3], Patrick Gélard[3]

[1]Université de Toulouse, ISAE, TéSA, Toulouse, France
[2]Thales Alenia Space, Toulouse, France
[3]CNES Toulouse, France

***Abstract*—We present a new delay-based transport protocol named FLOWER, that aims at providing a Lower-than-Best-Effort (LBE) service. The objective is to propose an alternative to the Low Extra Delay Background Transport (LEDBAT) widely deployed within the official BitTorrent client. Indeed, besides its intra-fairness problem, known as latecomer unfairness, LEDBAT can be too aggressive against TCP, making it ill suited for providing LBE services over certain networks such as constrained wireless networks. By using a fuzzy controller to modulate the sending rate, FLOWER aims to solve LEDBAT issues while fulfilling the role of a LBE protocol. Our simulation results show that FLOWER can carry LBE traffic in network scenarios where LEDBAT cannot while solving the latecomer unfairness problem. Finally, the presented algorithm is simple to implement and does not require complex computation that would prevent its deployment.**

I. Introduction

While standard TCP and its variants endeavor to achieve a fair share of the network bottleneck capacity between flows, the service provided by the network remains best-effort. There exists another service named Lower-than-Best-Effort (LBE) which aims at providing a second priority class inside the network traffic. The rationale is to propose a service for background traffic (e.g. peer-to-peer file transfers, data backup, software updates, ...) or signaling traffic. This kind of traffic might tolerate a high latency and should not disturb the traffic carried out by the best-effort service itself or other services that would propose advanced QoS architecture for time-constrained application such as DiffServ [1]. Today, the LBE service, also called "scavenger" service, is perceived as a potential solution to fetch the unused, sometimes wasted capacity in public network. One of the objective is, for instance, to provide a free Internet access based on this LBE principle, as illustrated by the objectives of GAIA[1] or PAWS[2] project. Last but not least, the LBE service should not exacerbate the bufferbloat issue [2].

Among the different transport protocols providing a LBE service [3], Low Extra Delay Background Transport (LEDBAT) [4] is the most used. LEDBAT is a delay-based congestion control protocol that has been standardized by the Internet Engineering Task Force (IETF). LEDBAT aims to exploit the remaining capacity while limiting the queuing delay around a predefined target τ, which may be set up to $\tau = 100\,\text{ms}$ according to RFC 6817 [4]. Consequently, LEDBAT flows limits the amount of queuing delay introduced in the network and thus lower their impact on best-effort flows such as TCP. As an example of application, the official BitTorrent client is using LEDBAT for data transfer [4].

Despite being a widely deployed protocol, the two main LEDBAT parameters (i.e., target and gain) have been revealed to be complex to determine [5], [6] as their tuning highly depends on the network conditions and not dynamically configurable. Indeed, LEDBAT may become more aggressive than TCP in case of misconfiguration [5], [6]. As an illustration, in a recent study, the authors of [7] conclude that the LEDBAT target parameter should not be higher than $5\,\text{ms}$ in a large bandwidth-delay product (BDP) network. At last, the authors of [8] show that LEDBAT can greatly increase the network latency making its impact on the network not transparent anymore.

Our protocol, FLOWER (**F**uzzy **LOW**er-than-Best-**E**ffo**R**t Transport Protocol), is a promising alternative to LEDBAT. FLOWER overcomes LEDBAT shortcomings and provides an LBE service that is more transparent to the network. The principal difference with LEDBAT is that FLOWER replaces the linear P-type controller (proportional controller) of LEDBAT by a fuzzy controller to modulate the congestion window. Compared to a recent solution named fLEDBAT [9] that proposes to solve the latecomer issue and to the best of our knowledge, there is no universal scheme allowing intra-fair LEDBAT flows to remain LBE compliant, that is, non-aggressive when competing with TCP flows.

We first review in Section II the LEDBAT algorithm and its problems that motivate our work. Section III details the design of FLOWER, while Section IV clearly explains its core component, that is the fuzzy controller. Section V evaluates our new protocol and gives a side-by-side comparison with LEDBAT using the network simulator ns-2.35. We finally conclude our work in Section VI.

II. Contextual background and motivation

While many transport protocols that have been design to carry LBE traffic, such as NICE [10] or TCP-LP [11], only LEDBAT has been reported to be actually deployed [12]. Our

[1]Global Access to the Internet for All (https://sites.google.com/site/irtfgaia).
[2]Public Access WiFi Service (http://publicaccesswifi.org).

work therefore focus on LEDBAT and its design issues that are described in this section.

A. LEDBAT in a nutshell

LEDBAT congestion control is based on queuing delay variations (i.e., the queuing delay is used as a primary congestion notification). LEDBAT is characterized by several parameters: target queuing delay τ, gain γ, minimum one-way delay owd_{min} (also called base delay), and current one-way delay owd_{ack}. The target queuing delay τ embodies the maximum queuing time that a LEDBAT connection is allowed to introduce in the network. The gain γ corresponds to the reactivity of LEDBAT to queuing delay variations. The bigger γ is, the faster LEDBAT congestion control increases or decreases its congestion window. LEDBAT infers the queuing delay by calculating $(owd_{ack} - owd_{min})$ obtained from one-way delays measured by exploiting the ongoing data transfer. To keep the queuing delay around the predefined target, LEDBAT uses a linear P-type controller to modulate the congestion window according to the derived queuing delay. For each ACK received at discrete time k, the new congestion window size $cwnd$ is updated as follows:

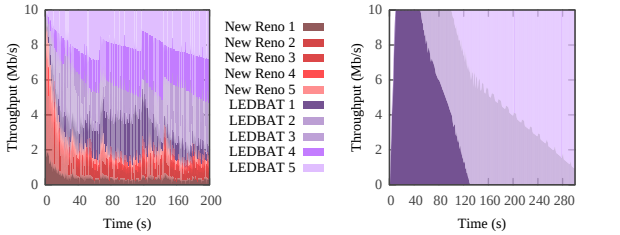
$$cwnd(k) = cwnd(k-1) + \frac{\gamma(\tau - (owd_{ack}(k) - owd_{min}(k)))}{cwnd(k-1)}$$

B. Two main LEDBAT issues

1) Aggressiveness of LEDBAT: RFC 6817 [4] states that, if a compromised target is set to infinity, "the algorithm is fundamentally limited in the worst case to be as aggressive as standard TCP". Actually, it corresponds to the case where the buffer size is too small in comparison to the target. Thus, the queuing delay sensed by LEDBAT never reaches the target. Therefore, LEDBAT always increases its sending rate until a loss event is reported.

However, there are circumstances "worse than the worst case mentioned in RFC 6817" in which hostile LEDBAT makes TCP back off, even in an unfavorable situation for LEDBAT when it starts after TCP. The issue occurs when the buffer size is around the target. In this case, LEDBAT does not have enough time to react to queuing delay before TCP causes a buffer overflow. After that, TCP halves its congestion window, resulting in a reduction of the queuing delay. Since the queuing delay is now below the target, LEDBAT raises again its congestion window conjointly with TCP. Consequently, after several cycles, LEDBAT exploits more capacity than TCP.

To illustrate why the problem is important and the impact of the aggressiveness of LEDBAT on TCP flows, Fig. 1a shows an ns-2 simulation of 5 TCP New Reno and 5 LEDBAT flows sharing the same bottleneck with a capacity of 10 Mb/s. The buffer size is 84 packets (about 100 ms of delay) and the LEDBAT target is set to 100 ms. The result is unequivocal and demonstrates the aggressiveness of LEDBAT flows against TCP flows. Although we present measurements with TCP New Reno, the problem remains the same with Cubic as shown later in the paper.



(a) Aggressiveness (b) Latecomer unfairness

Fig. 1: LEDBAT problems.

2) Latecomer unfairness: when LEDBAT flows start at different times, they may suffer from the latecomer unfairness problem. This problem arises because latecomer flows may sense different minimum one-way delays. In the worst case, when the buffer size is large enough, latecomer flows can starve ongoing flows.

Fig. 1b demonstrates the latecomer unfairness problem. In this case, 3 LEDBAT flows start consecutively every 50 s and share the same bottleneck with a capacity of 10 Mb/s. The buffer size is 167 packets (about 200 ms of delay). The LEDBAT target is set to 100 ms. As can be observed in Fig. 1b, latecomer flows gradually take all bandwidth of ongoing flows.

C. Motivation of FLOWER

Up to this point, we have identified two important problems of LEDBAT. We now present our motivation to develop the new congestion control named FLOWER.

Both LEDBAT key parameters — target and gain — are fixed and do not cope with the diversity of network configurations. Consequently, LEDBAT becomes more aggressive than TCP under some circumstances. One possible solution is to adapt the target/gain to the change of network conditions [7], [13]. However, such adaptive control scheme requires a fine-grained mathematical network model. To prevent the use of such too complex model, we design a new congestion protocol based on the fuzzy logic. Two main advantages of this approach are:

1) a fuzzy control system is a solution that prevents the use of a mathematical model. Such approach is particularly interesting when the model is not trivial, difficult to derive or too complex to be implemented;
2) the fuzzy logic allows to incorporate our heuristic knowledge about how to control the system. In other words, we can use our previous findings [6] as an entry for the fuzzy controller.

An in-depth analysis [6] gives us an insight to overcome the LEDBAT problems, or more specifically, to control the queuing delay. Hence, by means of the fuzzy logic, we integrate our understanding gathered into the fuzzy controller of FLOWER. We also point out that, by using a fuzzy control system, we seek a generic solution that works in several and various network conditions. It means that we are seeking an average use-case and not the "optimal" one.

III. Design and implementation

A. FLOWER overview

FLOWER is a novel delay-based transport protocol which aims at providing an effective LBE service. So, as a potential LEDBAT alternative, FLOWER must tackle its issues while keeping the same goals in terms of LBE service as listed in [4]:

1) to utilize end-to-end available bandwidth and to maintain low queuing delay when no other traffic is present;
2) to add limited queuing delay to that induced by concurrent flows, and;
3) to yield quickly to standard TCP flows that share the same bottleneck link.

To achieve these goals, FLOWER implements a fuzzy controller to manage the target queuing delay algorithm instead of the P-type controller as proposed in [4]. This non-zero target queuing delay ("non-zero": because recommended by the RFC to be fixed by default to $100\,ms$) allows FLOWER to fetch the available capacity, and thus to saturate the bottleneck link, when no other traffic is present. Meanwhile, the queuing delay needs to be kept as low as possible to make FLOWER non-intrusive to standard TCP traffic.

We can represent FLOWER congestion control as a feedback control system depicted in Fig. 2a. The essential components of FLOWER are:

1) *Fuzzy controller*, which is an artificial decision maker that operates based on a set of "If–Then" rules. By using the fuzzy logic, the fuzzy controller determines the congestion window size $cwnd$ such that the future estimated queuing delay eventually matches the target queuing delay τ. The fuzzy controller takes two inputs: queuing delay error e and change of queuing delay error Δe;
2) *Queuing delay estimator*, which exploits measured one-way delays to estimate the current queuing delay q;
3) *Peak-valley detector*, which keeps track of the maximum queuing delay $q_{\max}$ observed in the network. This maximum queuing delay is then used to normalize the queuing delay error.

Basically, FLOWER operates as follows: after each round-trip time (RTT), FLOWER finds out the minimum queuing delay observed during the RTT as the current queuing delay. Queuing delays in an RTT are obtained using the queuing delay estimator. Then, the fuzzy controller compares the target queuing delay with the current queuing delay. The error is positive when the current queuing delay is below the target. In this case, the fuzzy controller increases the congestion window, and thus the sending rate until the queuing delay reaches the target. When the error is negative, meaning that the current queuing delay is beyond the target, the fuzzy controller slows down its sending rate.

B. Comparison of FLOWER and LEDBAT

Fig. 2 shows in blue the differences between FLOWER and LEDBAT. Notably in FLOWER, we replace the P-type controller with the fuzzy controller that, besides the queuing delay error e, also utilizes the error trend Δe. We highlight the fact that while being more robust, the implementation of a fuzzy controller is simple and adds a little complexity to computation compared to the P-type controller of LEDBAT.

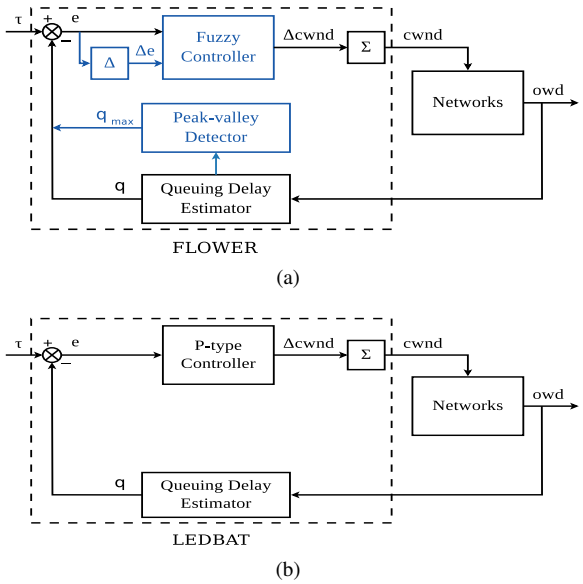


Fig. 2: Block diagram of FLOWER and LEDBAT as feedback control systems.

Another feature added to FLOWER is the peak-valley detector. This detector determines the maximum queuing delay, which is important for the operation of the fuzzy controller. Note that FLOWER uses the same LEDBAT queuing delay estimator, which is fully described in RFC 6817 [4].

C. Peak-valley detection algorithm

To effectively react to congestion events, FLOWER needs to determine the maximum queuing delay $q_{\max}$. For this purpose, we must identify the peaks of queuing delays (local maximum) and find out the maximum queuing delay (global maximum) using a threshold S, which is computed following an exponentially weighted moving average (EWMA) of peaks. To that end, we develop a simple on-line peak-valley detector based on the algorithm described in [14]. The algorithm alternatively identifies the peaks and valleys of queuing delays. Each time a peak is detected, it is then used to calculate a new threshold to find out $q_{\max}$.

D. Slow-start: to do or not to do?

Similarly to LEDBAT, FLOWER might suffer from the latecomer unfairness problem. During our experiments, we notice that the use of the slow-start helps to mitigate (without solving it for LEDBAT) the latecomer issue. This has also been notified by [15]. FLOWER uses slow-start as a synchronization signal and allows to get a preliminary measurement of the queuing delay. The purpose of slow-start is to create a spike in the queuing delay since in the slow-start phase, the congestion window rises exponentially until causing a loss event. If other

FLOWER connections also experience a loss, they restart their congestion window. As a consequence, the queuing delay is reduced, and thus allowing all flows to be able to sense the same base delay. All flows will then rise again at the same time and share the capacity equally. We highlight that slow-start does not necessarily cause loss to other flows. Fortunately, in this situation, the loss detection functionality of the FLOWER fuzzy controller helps ongoing flows to detect the slow-start signal of the latecomer flow, and hence to resynchronize all flows.

IV. FLOWER FUZZY CONTROLLER

At the core of FLOWER congestion control is the fuzzy controller composed by the following modules [16]:

1) *A rule base*, which contains a set of "If–Then" rules that describes how to control the system, in our case, the queuing delay;
2) *An inference mechanism*, which emulates the human expert's decision making about how best to control the system based on the information stored in the rule base;
3) *A fuzzification interface*, which converts controller inputs, e and Δe, into fuzzy values that the inference mechanism can use for its fuzzy reasoning process;
4) *A defuzzification interface*, which converts the conclusions of the inference mechanism into numerical output $\Delta cwnd$.

In the remainder of this section, we briefly introduce these modules and illustrate their operation.

A. The rule base

The rule base models the relationship between the inputs and the output of the system. The FLOWER fuzzy controller has the linguistic rules of the form:

R_i: **if** $e(k)$ is A_i and $\Delta e(k)$ is B_i **then** $\Delta cwnd(k)$ is C_i

where

- R_i is the ith rule ($1 \leq i \leq M$);
- $e(k) = \tau - q(k)$ is the first input variable;
- $\Delta e(k) = e(k) - e(k-1)$ is the second input variable;
- $\Delta cwnd(k)$ is the output variable. Thus, $cwnd(k) = cwnd(k-1) + \Delta cwnd(k)$;
- A_i, B_i, C_i are the fuzzy sets of the input and output variables.

Each fuzzy set A_i, B_i, C_i takes on the following **linguistic values**:

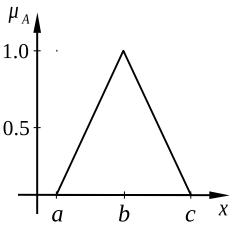
{NVVL, NVL, NL, NM, NM, NS, NVS, Z, PVS, PS, PM, PL, PVL}

where the meaning is: N: negative; P: positive; V: very; Z: zero; S: small; M: medium; L: large. Hence, the linguistic value PVS stands for *positive very small* and so forth.

Each fuzzy set is defined by a triangle membership function with three parameters $\{a, b, c\}$ as follows:

$$\mu_{A_i}(x) : X \mapsto [0,1]$$

$$\mu_{A_i}(x) = \begin{cases} 0 & \text{if } x \leq a, \\ \dfrac{x-a}{b-a} & \text{if } a < x \leq b, \\ \dfrac{c-x}{c-b} & \text{if } b < x < c, \\ 0 & \text{if } x \geq c \end{cases}$$



where $a < b < c$ and b is the center of the triangle membership function (i.e., where it reaches its peak). The membership function quantifies the degree of truth that a numerical value x can be classified linguistically, for example, as PVS.

For a system with two inputs and one output like FLOWER, we can list all rules using a tabular representation as shown in Fig. 3. Note that in the rule table in Fig. 3, we use **linguistic-numeric values** to shorten the description of linguistic values (e.g., 1 represents PVS; 2 represents PS; ...).

An important feature of FLOWER is its capability to react quickly to congestion events caused by TCP. This feature is integrated in the rule base and can be observed at the last column of the rule table, which we call the *loss detection zone* (see Fig. 3). Concretely, when FLOWER detects a very large fall in the queuing delay ($\Delta e(k)$ is 5 or PVL), it must immediately reduce to its minimum congestion window (e.g, set to one packet). This case corresponds to the following output: $\Delta cwnd(k)$ is -6 or NVVL.

Fig. 3 also shows all the membership functions for the inputs and the output of the FLOWER fuzzy controller.

1) Membership functions of $e(k)$: since the queue size varies continuously as a function of the network traffic, we need to make the input error $e(k)$ independent of the network state. For this purpose, before introducing $e(k)$ into the fuzzy controller, we express it as follows:

$$e(k) = \begin{cases} \dfrac{e(k)}{\tau} \times 100 & \text{if } q(k) \leq \tau, \\ \dfrac{e(k)}{\tau - q_{\max}} \times 100 & \text{if } q(k) > \tau \end{cases}$$

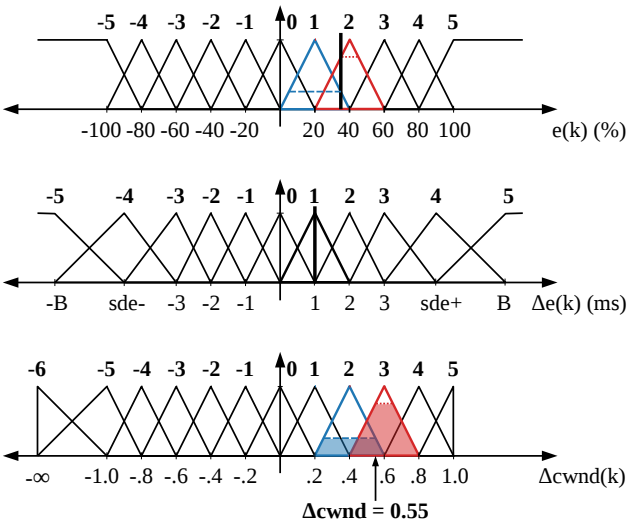
where $q_{\max}$ is the maximum queuing delay observed on the network. Consequently, the membership functions of $e(k)$ is linearly distributed on the universe of discourse $[-100, 100]$ %.

2) Membership functions of $\Delta e(k)$: the queuing delay is ranging from 0 to the maximum value $q_{\max}$. Thus, we have

$$\Delta e(k) = e(k) - e(k-1) = q(k-1) - q(k)$$

where $q(k) \in [0, q_{\max}]$. Then, the universe of discourse for $\Delta e(k)$ is $[-q_{\max}, q_{\max}]$ ms.

The variation of the queuing delay, and thus $\Delta e(k)$, highly depends on the network state. Hence, we need to dynamically adapt the distribution for the membership functions of $\Delta e(k)$. In addition, as seen in the rule table in Fig. 3, the loss detection zone of FLOWER relies only on $\Delta e(k)$. Therefore, we must determine a threshold to define this zone. To this end, we use the exponentially weighted moving average (EWMA) of values of $\Delta e(k)$. As EWMA has higher weights on recent data than on older data, sudden network condition changes



$$e(k)=\tau-q(k)$$
$$\Delta e(k)=e(k)-e(k-1)$$
$$\Delta cwnd(k)=\frac{\sum_i b_i \mu_{R_i}(\Delta cwnd(k))}{\sum_i \mu_{R_i}(\Delta cwnd(k))}$$

Example:

$$\Delta cwnd=\frac{(0.4)(0.25)+(0.6)(0.75)}{0.25+0.75}=0.55$$

Δcwnd		Δe										
		-5	**-4**	**-3**	**-2**	**-1**	**0**	**1**	**2**	**3**	**4**	**5**
e	**-5**	-5	-5	-5	-5	-5	-5	-4	-3	-2	-1	**-6**
	-4	-5	-5	-5	-5	-5	-4	-3	-2	-1	0	**-6**
	-3	-5	-5	-5	-5	-4	-3	-2	-1	0	1	**-6**
	-2	-5	-5	-5	-4	-3	-2	-1	0	1	2	**-6**
	-1	-5	-5	-4	-3	-2	-1	0	1	2	3	**-6**
	0	-5	-4	-3	-2	-1	**0**	1	2	3	4	**-6**
	1	-4	-3	-2	-1	0	1	**2**	3	4	5	**-6**
	2	-3	-2	-1	0	1	2	**3**	4	5	5	**-6**
	3	-2	-1	0	1	2	3	4	5	5	5	**-6**
	4	-1	0	1	2	3	4	5	5	5	5	**-6**
	5	0	1	2	3	4	5	5	5	5	5	**-6**

Legend

-6 = NVVL, **-5** = NVL, **-4** = NL, **-3** = NM, **-2** = NS, **-1** = NVS,
0 = Z,
1 = PVS, **2** = PS, **3** = PM, **4** = PL, **5** = PVL

(**P**: Positive, **N**: Negative, **V**: Very,
Z: Zero, **S**: Small, **M**: Medium, **L**: Large)

B: Buffer, **sde-**: smoothed negative Δe, **sde+**: smoothed positive Δe

Fig. 3: The membership functions and the rule table of the FLOWER fuzzy controller.

are further taken into account in this average. Consequently, the distribution for the membership functions of $\Delta e(k)$ is as follows:

$$-q_{\max}, sde_-, -3, -2, -1, 0, 1, 2, 3, sde_+, q_{\max}$$

where sde_- and sde_+ are the EWMA of the negative and positive values of $\Delta e(k)$, respectively. $\{-q_{\max}, sde_-, sde_+, q_{\max}\}$ are respectively initialized with $\{-5, -4, 4, 5\}$. These values are updated only when the absolute value of a new value is greater than the absolute value of the initial value.

Finally, we underline that, as an effect of the loss detection zone, when $\Delta e(k) > sde_+$, even if the degree of truth $\mu_{PVL}(\Delta e(k))$ is small, FLOWER reduces the congestion window to its initial value.

3) Membership functions of $\Delta cwnd(k)$: outside the loss detection zone, the distribution of $\Delta cwnd(k)$ is linear on the universe of discourse $[-1, 1]$ packet. As a consequence, the maximum ramp-up speed of FLOWER is the same as TCP, i.e., one packet per RTT. When operating in the loss detection zone, $\Delta cwnd(k)$ is set to negative infinity to signal FLOWER to reduce to minimum its sending rate. Otherwise, FLOWER will ramp-down at maximum one packet per RTT.

B. Fuzzification

Whenever the fuzzification module receives a numerical value x, it converts this value into the degree of truth $\mu_{A_i}(x)$ of the corresponding linguistic value.

C. Inference mechanism

The inference mechanism derives the firing strength of each rule from the fuzzy inputs obtained by fuzzification. Using the minimum rule inference, the firing strength of the ith rule is given by

$$\mu_{R_i} = min\{\mu_{A_i}(e), \mu_{B_i}(\Delta e)\}$$

D. Defuzzification

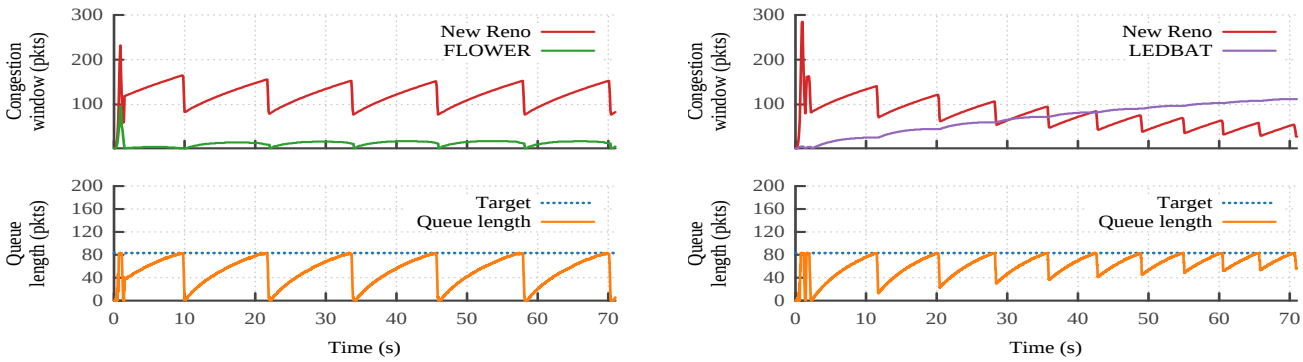
Defuzzification is the process of combining results of the inference mechanism to obtain a numerical output value y. We use the "center-average" defuzzication method which calculates the weighted average of the center values of the output membership function centers:

$$\Delta cwnd = \frac{\sum_{i=1}^{M} b_i \mu_{R_i}}{\sum_{i=1}^{M} \mu_{R_i}}$$

E. Example of fuzzy controller operation

Consider the example in Fig. 3. Suppose that $e(k) = 35$ and $\Delta e(k) = 1$. The fuzzification process gives $\mu_{PVS}(e(k)) = 0.25$ and $\mu_{PS}(e(k)) = 0.75$, whereas $\mu_{PVS}(\Delta e(k)) = 1$. Fig. 3 shows the degrees of truth of the membership functions for the inputs and indicates with black vertical lines the numerical values of $e(k)$ and $\Delta e(k)$. In this case, the corresponding rules are shown as the blue cell and the red cell in the rule table. Thus, the inference mechanism gives $\mu_{R_1} = 0.25$ and $\mu_{R_2} = 0.75$, which correspond to the blue and red regions of the output membership functions, respectively. Lastly, since the output membership function centers of the two rules are $b_1 = 0.4$ and $b_2 = 0.6$, the numerical output given by the defuzzication process is:

$$\Delta cwnd(k) = \frac{0.4 \times 0.25 + 0.6 \times 0.75}{0.25 + 0.75} = 0.55$$



(a) FLOWER vs. TCP New-Reno

(b) LEDBAT vs. TCP New-Reno

Fig. 4: TCP and LBE congestion windows and bottleneck queue length as a function of time.

V. Evaluation of FLOWER

We use the network simulator ns-2.35 to validate our new protocol. For this purpose, we have implemented an ns-2 prototype of FLOWER based on LEDBAT module developed by Valenti et al. [17]. The prototype is implemented as a Linux congestion control module on top of the TCP-Linux framework [18]. Therefore, simulation results are much closer to a real implementation in the Linux kernel and would allow to easy port our implementation inside the Linux kernel (this also been the case for the LEDBAT module [17]).

We specifically focus on the FLOWER performance in terms of respect to a LBE traffic and latecomer unfairness which are the two major drawbacks of LEDBAT.

A. Simulation setup

We use a dumbbell topology where a TCP flow shares a single bottleneck link with a LBE flow (either FLOWER or LEDBAT). Note that to test our protocol, we follow the scenario used in [12] for the sake of comparison. All sources send packets with a size of $P = 1500$ B. The bottleneck link has a capacity set to $C = 10$ Mb/s and a one-way propagation delay $owd \in [10, 50, 100, 150, 200, 250]$ ms. The bottleneck router is a FIFO drop-tail queue with a size of B packets. For convenience, we express the bottleneck buffer B as a ratio to the bandwidth-delay product BDP in terms of packets. Hence, we have $B = \lceil n * BDP \rceil = \lceil n * C * 2 * owd/(8 * P) \rceil$, where the ratio $n \in [0.2, 0.4, 06, 0.8, 1.0]$ and $\lceil x \rceil$ is the ceiling function. Since B must be an integer, we use the ceiling function to get the smallest integer not less than B. We also convert the target τ from milliseconds to packets as follows: $\tau\,(packets) = \tau\,(ms) * C/(8 * P)$. Therefore, a target queuing delay $\tau = 100$ ms corresponds to 83.3 packets and is rounded to 84 packets.

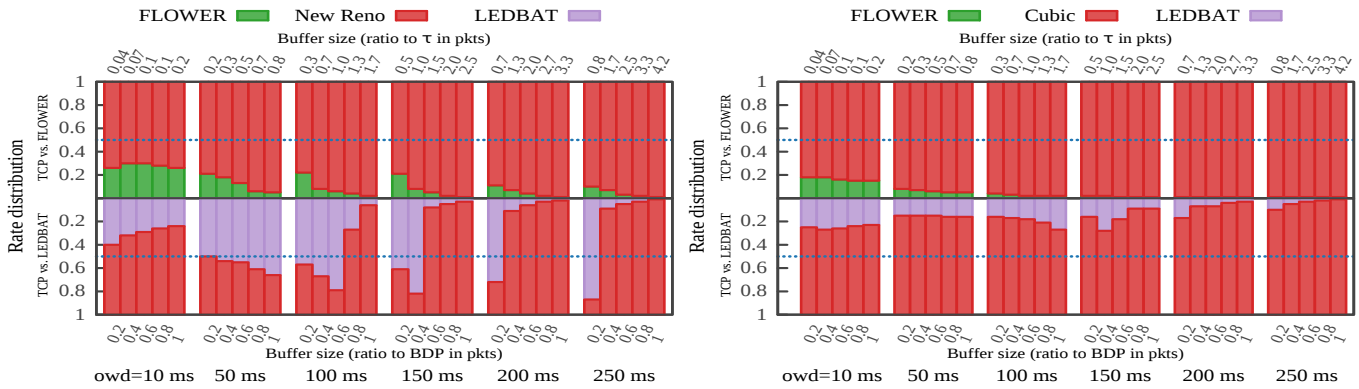
B. Interaction with TCP

In this section, we study the behavior of FLOWER in the presence of TCP and more specifically, the interaction between the FLOWER fuzzy controller and the TCP AIMD (Additive Increase/Multiplicative Decrease) algorithm.

1) Scenario and metrics: two TCP and LBE flows start at $t = 0$ s and stop at $t = 75$ s. In this scenario, $owd = 50$ ms and $B = BDP$. To investigate the behavior of one LBE flow in coexistence with one TCP flow, we consider their congestion windows and the queue length of the bottleneck buffer.

2) Results: Fig. 4 shows both congestion windows (top) as a function of time conjointly with the queue length and the target queuing delay expressed in packets (bottom). The interaction between TCP and FLOWER is shown in Fig. 4a. In the slow-start phase, TCP and FLOWER increase exponentially their congestion window. Thus, the bottleneck queue fills up quickly until loss. Unlike TCP, FLOWER reduces its congestion window to its initial value which equals to one packet in our implementation. After the slow-start phase, approximatively before $t = 3$ s, as the bottleneck queue is half-filled but the resulting queuing delay is small compared to the target, FLOWER and TCP congestion windows conjointly grow. As the queue still increases because TCP keeps sending packets, FLOWER reduces its sending rate (the target is almost reached) and finally stabilizes its congestion window. After exactly $t = 7.5$ s, (we obtained this exact value from the simulation traces) when the queuing delay is close to the target, FLOWER reacts by decreasing its sending rate. Finally, FLOWER reaches the minimum sending rate of one packet per RTT at $t = 9.3$ s. Slightly afterwards, TCP gets losses and enters in its recovery phase. As a consequence, TCP halves its congestion window and the bottleneck queue is drained.

TCP re-enters in the congestion avoidance phase at $t = 10$ s while FLOWER grows at its maximum speed as the queue is not fully filled. FLOWER prevents bottleneck overflow by reducing its sending rate before the knee phase [19] (i.e. when the rate increases gradually but slower than the delay). When TCP halves its congestion window at $t = 21.8$ s, we observe an abrupt fall of the queuing delay. Shortly afterwards, FLOWER detects this fall with the help of the loss detection scheme, hence it drops to the minimum its congestion window. Therefore, the queue is drained and FLOWER enters in a new cycle. Henceforth, both FLOWER and TCP are in steady state.



(a) LBE vs. TCP New-Reno

(b) LBE vs. TCP Cubic

Fig. 5: Rate distribution of TCP and LBE flows.

This first experiment illustrates the good LBE behavior of FLOWER in the presence of TCP. Clearly, the fuzzy controller with the loss detection scheme allows FLOWER to be LBE compliant. In this standard configuration (we recall that $B = BDP$), LEDBAT does not behave as a LBE protocol and is too aggressive as shown in Fig. 4b. This figure also illustrates that the LEDBAT P-type controller does not react correctly to congestion events. We refer the reader to previous studies [6], [15] for further details on the LEDBAT defective behavior.

In the next section, we extend these measurements to several general networking use-cases in order to exhaustively illustrate the good performance of our fuzzy controller scheme.

C. FLOWER versus LEDBAT performance in coexistence with TCP New Reno and TCP Cubic

In this section, we evaluate the impact of FLOWER flows on TCP flows (either New Reno or Cubic) in different network conditions.

1) Scenario and metric: we consider 5 long-lived TCP flows with 5 LBE flows. The simulation lasts $1200\,\text{s}$ where TCP flows start consecutively every $10\,\text{s}$ from $t = 0\,\text{s}$ and keep sending data until the end of simulation. LBE flows start randomly between $t = 350\,\text{s}$ and $t = 450\,\text{s}$ in order for TCP to reach the full capacity.

To assess the impact of LBE on TCP, we define the metric *rate distribution* (X) as the total throughput achieved by all flows F_k where $k \in \{TCP, LBE\}$ over the total throughput of all flows on the link:

$$X_k = \frac{F_k}{F_{TCP} + F_{LBE}} \tag{1}$$

For each combination of network configuration $\{owd, B\}$, we run the simulation 10 times. After each run, we calculate the *rate distribution* over the last 600 seconds. Then, the mean of the 10 metric values is taken as the measured value.

2) Results: in Fig. 5, using histogram, we group the simulation results into different categories of one-way delay (denoted *owd* in Fig. 5), and then into subclasses of buffer size given as a ratio to the *BDP*. For information purpose, note that at the top of the histogram, the equivalent ratio to the *BDP* is converted as the ratio to the target value given in packets as explained in Section V-A. This means we express B as the ratio to the target τ in the same way as with the BDP. For instance, looking at Fig. 5, a buffer sized 0.4 of the BDP at $owd = 100\,\text{ms}$ corresponds to 0.7 of target value in packets. For each buffer size, each stacked column gives the sum of the normalized rates obtained by both TCP and LBE flows. Then, each slice inside a column, represents the part obtained by 5 TCP and 5 LBE flows given by (1).

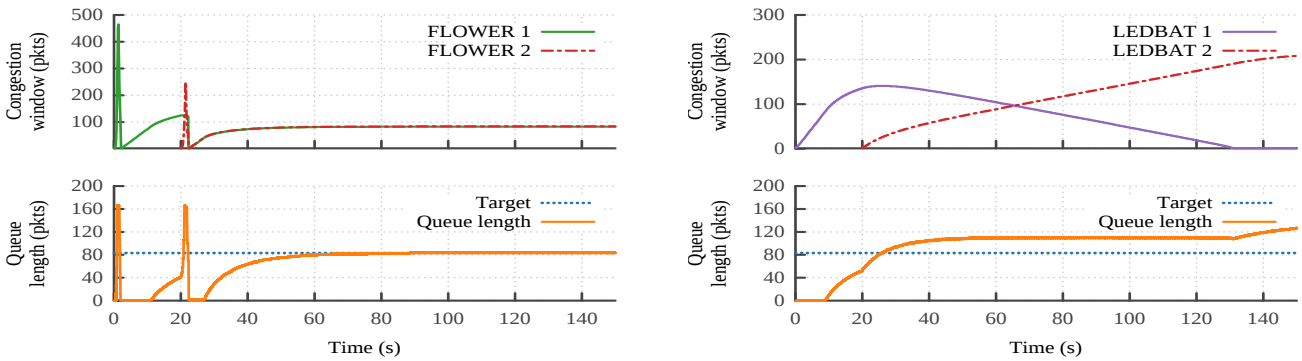
Fig. 5a shows the performance of LEDBAT and FLOWER in the presence of TCP New Reno. We have selected a set of network configurations following our previous study on the LEDBAT performance issues [6]. These network configurations illustrate a large number of use-cases where LEDBAT performs (in Fig. 5a when the ratio of the bottleneck buffer size to the target τ is largely greater than 1) or does not perform correctly (resp. the reverse). As shown in Fig. 5a, LEDBAT obtains sometimes more than TCP New Reno and crosses the fair-share line represented by a dotted line. We then compare the results obtained by FLOWER in these configuration. Fig. 5a allows to easily compare the performance of both protocols in identical situation. The results are unequivocal and illustrate that FLOWER behaves as a LBE protocol where LEDBAT fails in realistic cases.

Using the same network configurations as above, we now study the performance of LEDBAT and FLOWER in coexistence with TCP Cubic in Fig. 5b. TCP Cubic is more aggressive than TCP New Reno but in those cases, the performance of FLOWER is far better than LEDBAT in respect of the LBE principle.

D. Intra-protocol fairness

We finally study the interaction between two FLOWER flows to assess their intra-fairness and determine whether FLOWER is not impacted by the latecomer issue.

1) Scenario and metric: in this scenario, the buffer size B is set to twice the BDP. This configuration is favorable to get the LEDBAT latecomer unfairness phenomenon. The



(a) FLOWER (b) LEDBAT

Fig. 6: LBE congestion windows and bottleneck queue length as a function of time.

bottleneck link has a one-way delay $owd = 50$ ms. The first LBE flow starts at $t = 0$ s and the second starts at $t = 20s$. Both flows last 150 s. As in V-B, we draw their congestion windows and the queue length of the bottleneck buffer.

2) Results: Fig. 6b shows the LEDBAT latecomer issue [17]. The first LEDBAT flow starts when the bottleneck queue is empty, and as a result, senses a base delay. When the second LEDBAT flow starts at $t = 20$ s, the queue is filled with ≈ 50 packets. Consequently, the second flow estimates a higher base delay including the queuing delay of the first one. Since its estimated queuing delays are below the target delay, the second flow raises its sending rate. As a result, the first one senses an increasing queuing delay and begins to decelerate. Finally, it reaches its minimum rate at $t = 131$ s as shown in Fig. 6b. On the contrary, FLOWER does not inherit this latecomer issue thanks to the loss detection scheme described in Section IV as shown in Fig. 6a. This experiment demonstrates that two FLOWER flows can now share fairly the link capacity.

VI. Conclusion and Future Work

We propose FLOWER, a new delay-based congestion control protocol designed to provide a LBE service using results from the fuzzy logic area. The main goal of FLOWER is to overcome both major LEDBAT drawbacks: aggressiveness and latecomer unfairness, while being LBE compliant. To the best of our knowledge, FLOWER **is the first solution that solves both the aggressiveness issue inherent to LEDBAT protocol and the fairness issue**. Our preliminary simulation study over a wide range of network use-cases shows that FLOWER performs better than LEDBAT in case where it usually fails.

We are currently porting and testing this implementation inside the Linux kernel. We expect to present and discuss these results and this fuzzy logic mechanism proposal at the next IETF meeting.

VII. Acknowledgments

The authors wish to thank Pr. Paul Amer and Dr. Nicolas Kuhn for valuable suggestions about this work.

References

[1] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," RFC 2475, Dec. 1998.
[2] V. Cerf, V. Jacobson, N. Weaver, and J. Gettys, "BufferBloat: What's Wrong with the Internet?" *Queue, ACM*, vol. 9, no. 12, pp. 10–20, 2011.
[3] D. Ros and M. Welzl, "Less-than-Best-Effort Service: A Survey of End-to-End Approaches," *Commun. Surveys Tutorials, IEEE*, vol. 15, no. 2, pp. 898–908, 2013.
[4] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)," RFC 6817, Dec. 2012.
[5] G. Carofiglio, L. Muscariello, D. Rossi, and C. Testa, "A hands-on Assessment of Transport Protocols with Lower than Best Effort Priority," in *IEEE LCN*, Oct. 2010.
[6] S. Q. V. Trang, N. Kuhn, E. Lochin, C. Baudoin, E. Dubois, and P. Gelard, "On the existence of optimal LEDBAT parameters," in *IEEE ICC*, June 2014.
[7] N. Kuhn, O. Mehani, A. Sathiaseelan, and E. Lochin, "Less-than-Best-Effort Capacity Sharing over High BDP Networks with LEDBAT," in *IEEE VTC Fall*, Sept. 2013.
[8] D. Ros and M. Welzl, "Assessing LEDBAT's Delay Impact," *Commun. Lett., IEEE*, vol. 17, no. 5, pp. 1044–1047, 2013.
[9] G. Carofiglio, L. Muscariello, D. Rossi, C. Testa, and S. Valenti, "Rethinking the Low Extra Delay Background Transport (LEDBAT) Protocol," *Comput. Netw.*, vol. 57, no. 8, pp. 1838–1852, 2013.
[10] A. Venkataramani, R. Kokku, and M. Dahlin, "TCP Nice: a mechanism for background transfers," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 329–343, 2002.
[11] 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, 2006.
[12] Y. Gong, D. Rossi, C. Testa, S. Valenti, and M. Täht, "Fighting the bufferbloat: on the coexistence of AQM and low priority congestion control," *Comput. Netw.*, vol. 65, pp. 255–267, 2014.
[13] A. Abu and S. Gordon, "A Dynamic Algorithm for Stabilising LEDBAT Congestion Window," in *ICCNT*, Apr. 2010.
[14] G. Palshikar, "Simple algorithms for peak detection in time-series," in *ICADABAI*, 2009.
[15] D. R. Giovanna Carofiglio, Luca Muscariello and S. Valenti, "The Quest for LEDBAT Fairness," in *IEEE GLOBECOM*, 2010.
[16] K. Passino and S. Yurkovich, *Fuzzy Control*. Addison-Wesley, 1998.
[17] D. Rossi, C. Testa, S. Valenti, and L. Muscariello, "LEDBAT: The New BitTorrent Congestion Control Protocol," in *ICCCN*, 2010.
[18] D. X. Wei and P. Cao, "NS-2 TCP-Linux: An NS-2 TCP Implementation with Congestion Control Algorithms from Linux," in *Proc. Workshop Ns-2: The IP Network Simulator*, 2006.
[19] D.-M. Chiu and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Comput. Netw. ISDN Syst.*, vol. 17, no. 1, pp. 1–14, June 1989.