Université Fédérale

Toulouse Midi-Pyrénées

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut Supérieur de l'Aéronautique et de l'Espace

Présentée et soutenue par :
Si Quoc Viet TRANG

le jeudi 3 décembre 2015

Titre :

FLOWER, an innovative Fuzzy LOWer-than-best-EffoRt transport protocol

FLOWER, un protocole de transport innovant, Lower-than-Best-Effort, basé sur la logique floue

---

École doctorale et discipline ou spécialité :
ED MITT : Réseaux, télécom, système et architecture

Unité de recherche :
Équipe d'accueil ISAE-ONERA MOIS

Directeur(s) de Thèse :

M. Emmanuel LOCHIN (directeur de thèse)

Jury :

M. Congduc PHAM, Université de Pau et des Pays de l'Adour, France - Président, Rapporteur
M. Toufik AHMED, Université Bordeaux, France- Rapporteur
M. Paul AMER, University of Delaware, États-Unis
M. Emmanuel LOCHIN, Professeur ISAE-SUPAERO, France - Directeur de thèse

To my wife Lan Hương
and my daughter Tuệ Anh
without whom this thesis would have
been completed one year earlier.

Acknowledgment

I am greatly indebted to my supervisor, Prof. Emmanuel Lochin, for giving me a chance to work with him. Many thanks to him for all the constructive comments, freedom and encouragement he has given me over the last three years at ISAE. I will never forget his optimism and support beyond thesis supervision.

I am grateful to Centre National d'Etudes Spatiales (CNES) and Thales Alenia Space (TAS) for funding my research. I owe my gratitude to my co-supervisors: Patrick Gélard (CNES), Emmanuel Dubois (CNES) and Cédric Baudoin (TAS), who have given valuable comments on my research direction.

I would like to thank my thesis committee members, Paul Amer, Toufik Ahmed and Congduc Pham for their advice and comments.

I would also like to thank all my friends, both at ISAE and outside, for their enthusiastic support.

A very special thanks to my wonderful wife Lan Hương and my little angel Tuệ Anh for their immense support and endless love. They are my source of motivation and the force that keeps me going.

Finally, thanks to my family for their continuous support and encouragement: my parents, parents-in-law, brother, brother-in-law and sisters-in-law.

Contents

List of Figures

List of Tables

Abstract

In this thesis, we look at the possibility to deploy a Lower-than-Best-Effort (LBE) service over long delay links such as satellite links. The objective is to provide a second priority class dedicated to background or signaling traffic. In the context of long delay links, an LBE service might also help to optimize the use of the link capacity. In addition, an LBE service can enable a low-cost or even free Internet access to remote communities via satellite communication. Two possible deployment levels of an LBE approach exists: either at the MAC layer or at the transport layer. In this thesis, we are interested in an end-to-end approach and thus specifically focus on transport layer solutions. We first propose to study LEDBAT (Low Extra Delay Background Transport) because of its potential. Indeed, LEDBAT has been standardized by the IETF and is widely deployed within the official BitTorrent client. Unfortunately, the tuning of LEDBAT parameters is revealed to highly depend on network conditions. In the worst case scenario, LEDBAT flows can starve other traffic such as commercial traffic performing over a satellite link. LEDBAT also suffers from an intra-unfairness issue, called the latecomer advantage. These reasons often prevent operators from using LBE protocols over wireless and long-delay links as a misconfiguration can overload link capacity. Therefore, we design FLOWER, a new delay-based transport protocol, as an alternative to LEDBAT. By using a fuzzy controller to modulate the sending rate, FLOWER aims to solve LEDBAT issues while fulfilling the role of an LBE protocol. Our simulation results show that FLOWER can carry LBE traffic not only in the long delay context, but in a wide range of network conditions where LEDBAT usually fails.

Keywords Congestion control; Lower-than-Best-Effort; LEDBAT; Fuzzy logic;

Résumé

Dans cette thèse, nous examinons la possibilité de déployer un service Lower-than-Best-Effort (LBE) sur des liens à long délai tels que des liens satellites. L'objectif est de fournir une deuxième classe de priorité dédiée à un trafic en tâche de fond ou un trafic de signalisation. Dans le contexte des liens à long délai, un service LBE peut aider à optimiser l'utilisation de la capacité du lien. En outre, un service de LBE peut permettre un accès à Internet à faible coût ou même gratuit dans les collectivités éloignées via la communication par satellite. Il existe deux niveaux de déploiement possible d'une approche de LBE: soit à la couche MAC ou soit à la couche de transport. Dans cette thèse, nous nous intéressons à une approche de bout-en-bout et donc nous nous concentrons spécifiquement sur les solutions de la couche transport. Nous proposons tout d'abord d'étudier LEDBAT (Low Extra Delay Background Transport) en raison de son potentiel. En effet, LEDBAT a été normalisé par l'IETF et est largement déployé dans le client BitTorrent officiel. Malheureusement, le réglage des paramètres de LEDBAT dépend fortement des conditions du réseau. Dans le pire des cas, les flux LEDBAT peuvent prendre toute la bande passante d'autre trafic tels que le trafic commercial sur le lien satellite. LEDBAT souffre également d'un problème intra-inéquité, appelé latecomer advantage. Toutes ces raisons empêchent souvent les opérateurs de permettre l'utilisation de ce protocole sur le lien sans fil et à long délai puisqu'une mauvaise configuration peut surcharger la capacité du lien. Pour répondre à l'ensemble de ces problèmes, nous proposons FLOWER, un nouveau protocole de transport, qui se positionne comme alternative à LEDBAT. En utilisant un contrôleur de logique floue pour réguler le débit des données, FLOWER vise à résoudre les problèmes de LEDBAT tout en remplissant le rôle d'un protocole de LBE. Dans cette thèse, nous montrons que FLOWER peut transporter le trafic de LBE non seulement dans le contexte à long délai, mais dans plusieurs conditions du réseau où LEDBAT se trouve en échec.

Synthèse en Français

I. Introduction

L'utilisation du contrôle de congestion applicatif de faible priorité LEDBAT (*Low Extra Delay Background Transport*) au sein du client BitTorrent a suscité l'intérêt de la communauté de recherche Internet au service *Lower-than-Best-Effort* (LBE). Dédié à transporter du trafic non-critique, le service LBE cherche à utiliser, d'une manière non intrusive, la capacité du réseau inutilisée par les flux *best-effort*. Dans cette thèse, nous examinons la possibilité de déployer un service LBE sur des liens *large bandwidth-delay product* (LBDP) tels que les liens satellites. Dans ce contexte, un service LBE peut optimiser l'utilisation de la capacité du lien et pourrait permettre d'offrir un accès Internet aux régions concernées par la fracture numérique [1]. En effet, d'une part le satellite dispose d'un large éventail de couverture et peut donc atteindre des zones lointaines à un coût relativement faible. D'autre part, un protocole LBE comme LEDBAT pourrait être utilisé pour exploiter la capacité restante ou inutilisée du lien satellite pour fournir cet accès Internet, tout en restant transparent pour le trafic commercial.

Pour offrir un service LBE, il est nécessaire de laisser rapidement la place au trafic *best-effort* en cas de congestion, afin de ne pas perturber ce dernier, d'où sa qualification de *less-than-best-effort*. Alors qu'un service LBE pourrait être réalisé par d'autres couches[1] [2], dans cette thèse, nous nous focalisons sur son déploiement au niveau de la couche de transport afin de bénéficier de l'approche de bout-en-bout. En outre, nous nous concentrons principalement sur LEDBAT car il est le protocole LBE le plus déployé à ce jour. Ce dernier est notamment utilisé pour la sauvegarde des données, le *prefetching*, la distribution de contenu Internet, le transfert de fichiers peer-to-peer [3].

[1]Par exemple, un ordonnanceur à priorité type *priority-queuing* peut offrir un service LBE localisé entre deux nœuds d'un même lien.

Malheureusement, en dépit d'être un protocole LBE prometteur, LEDBAT possède un comportement plus agressif que TCP dans certaines configurations de réseaux [4, 5]. Néanmoins, bien que les auteurs de [5] suggèrent une solution possible, ces études ([4,5]) ne couvrent pas exhaustivement un large éventail de conditions réseaux.

Outre son agressivité, LEDBAT souffre également du problème d'inéquité intra-protocolaire connu sous le nom de *latecomer advantage* ou *latecomer unfairness* — le symptôme avec lequel un nouveau flux LEDBAT en concurrence avec d'autres flots du même type peut prendre jusqu'à toute la bande passante disponible [5–9]. Contrairement au problème d'agressivité, le problème du *latecomer* a reçu beaucoup d'attention et de nombreuses solutions ont été proposées afin de le résoudre [5, 8, 9]. Pourtant, ce problème de *latecomer* est considéré comme moins important que celui de l'agressivité. La justification est que puisque LEDBAT a été initialement conçu pour être une classe de trafic à très faible priorité [5], son intra-équité (i.e. entre flots LEDBAT) n'a pas été considérée comme un objectif en soi. A noter que ce problème disparaît en présence des flux TCP [6]. Etant donné le potentiel de LEDBAT, nous croyons que le problème du *latecomer* doit être abordé en même temps que le problème de l'agressivité. Bref, nous ne considérons pas l'intra-équité de moindre importance que l'inter-équité.

Suivant cette discussion, le premier objectif de cette thèse est de réaliser une étude approfondie sur le caractère agressif de LEDBAT. Le second est de fournir une solution efficace afin de résoudre ces deux principaux problèmes que sont l'agressivité et le *latecomer advantage*. Bien que notre objectif initial soit l'étude de la faisabilité d'un déploiement de LEDBAT sur les réseaux LBDP, tout au long de cette thèse, nous chercherons à obtenir une solution qui fonctionne sur un large éventail de configuration de réseaux.

II. Analyse de LEDBAT

Dans cette section, nous étudions les facteurs qui détournent LEDBAT de son objectif initial (i.e., se comporter comme un protocole LBE) et qui le rend plus agressif que TCP. Ensuite, nous tenterons de déterminer un ensemble de paramètres optimaux permettant au trafic LEDBAT de se comporter comme tel.

II.a. Le protocole LEDBAT

LEDBAT est conçu pour limiter le délai de la file d'attente autour d'une valeur fixe, appelée *target queuing delay* τ (le délai de la file d'attente cible). Dans ce but, LEDBAT

utilise un contrôleur PID de type P pour contrôler sa fenêtre de congestion en utilisant le délai de la file d'attente de bout-en-bout comme l'indicateur de congestion. A chaque l'instant k, le fonctionnement du contrôleur de congestion peut se résumer comme suit :

$$cwnd(k+1) = \begin{cases} cwnd(k) + \dfrac{\gamma e(k)}{cwnd(k-1)} & \text{si pas de perte,} \\ \dfrac{1}{2} cwnd(k) & \text{si perte} \end{cases}$$

où $e(k) = \frac{\tau - q(k)}{\tau}$ est la différence entre le *target queuing delay* et le délai de la file d'attente actuel $q(k)$ estimé par LEDBAT. La réactivité de LEDBAT aux variations du délai de la file d'attente est en plus ajustée par le gain γ. Ce gain se découpe en *increase gain* et *decrease gain* correspondant respectivement à la phase croissante et à la phase décroissante de la fenêtre de congestion.

Fondamentalement, LEDBAT fonctionne comme suit : la différence $e(k)$ est positive lorsque le délai courant de la file d'attente est au dessous du *target queuing delay*. Dans ce cas, le contrôleur doit augmenter la fenêtre de congestion, et donc le taux d'émission jusqu'à ce que le délai de la file d'attente atteigne le *target queuing delay*. Lorsque la différence est négative, ce qui signifie que le délai courant de la file d'attente est au-dessus du *target queuing delay*, le contrôleur doit ralentir son taux d'émission. De plus, la taille de la fenêtre de congestion est modifiée proportionnellement à la différence $e(k)$ pour éviter d'osciller. En conséquence, la fenêtre de congestion reste inchangée lorsque la différence est égale à zéro.

II.b. Agressivité de LEDBAT

Selon la RFC de LEDBAT [10], si le *target queuing delay* est volontairement[2] ou involontairement fixé à l'infini, le comportement de LEDBAT se limite à être aussi agressif que TCP dans le pire des cas. En fait, ceci correspond au cas où la taille du buffer est trop petite par rapport au *target queuing delay*, comme montré par la figure 1. Sur cette figure, la taille du buffer est de 9 paquets, ce qui signifie que le ratio entre la taille du buffer et le *target queuing delay* est de 0,1. Ainsi, le délai de la file d'attente estimé par LEDBAT n'atteint jamais le *target queuing delay*. Par conséquent, LEDBAT augmente toujours son taux d'émission jusqu'à ce qu'une perte soit détectée.

Cependant, il existe des circonstances dans lesquelles LEDBAT devient hostile et dégrade le service offert par TCP. La figure 2 illustre ce comportement agressif de LEDBAT envers TCP. La taille du buffer dans cet exemple est de 92 paquets. Dans sa

[2] Cas d'une configuration d'un utilisateur malintentionné.

phase de *slow start*, TCP augmente de façon exponentielle sa fenêtre de congestion. En conséquence, le buffer se remplit immédiatement et la fenêtre de congestion de LEDBAT se retrouve bloquée à un paquet. Après la phase de *slow start*, de $t = 3\,\text{s}$ à $t = 5\,\text{s}$, puisque le délai de la file d'attente est petit par rapport au *target queuing delay*, les fenêtres de congestion de LEDBAT et TCP augmentent en même temps et à la même vitesse. Comme la file d'attente ne cesse d'augmenter, LEDBAT réduit la vitesse d'augmentation de sa fenêtre de congestion alors qu'entre-temps, TCP continue d'augmenter linéairement sa fenêtre de congestion. Après $t = 11\,\text{s}$, quand le délai de la file d'attente est supérieur au *target queuing delay*, LEDBAT diminue lentement sa fenêtre de congestion. Bien que la taille du buffer est plus grande dans cet exemple, elle reste relativement petite comparée au *target queuing delay*. En effet, le ratio entre la taille du buffer et le *target queuing delay* est de 1,1. Par conséquent, LEDBAT n'a pas assez de temps pour réagir face à l'augmentation du délai de la file d'attente avant que TCP ne déborde du buffer à $t = 15\,\text{s}$. Après cela, TCP divise par deux sa fenêtre de congestion, ce qui entraîne une réduction du délai de la file d'attente. Puisque le délai de la file d'attente est maintenant en dessous du *target queuing delay*, LEDBAT augmente à nouveau sa fenêtre de congestion conjointement avec TCP. En conséquence, après plusieurs cycles, LEDBAT exploite plus de capacité que TCP.

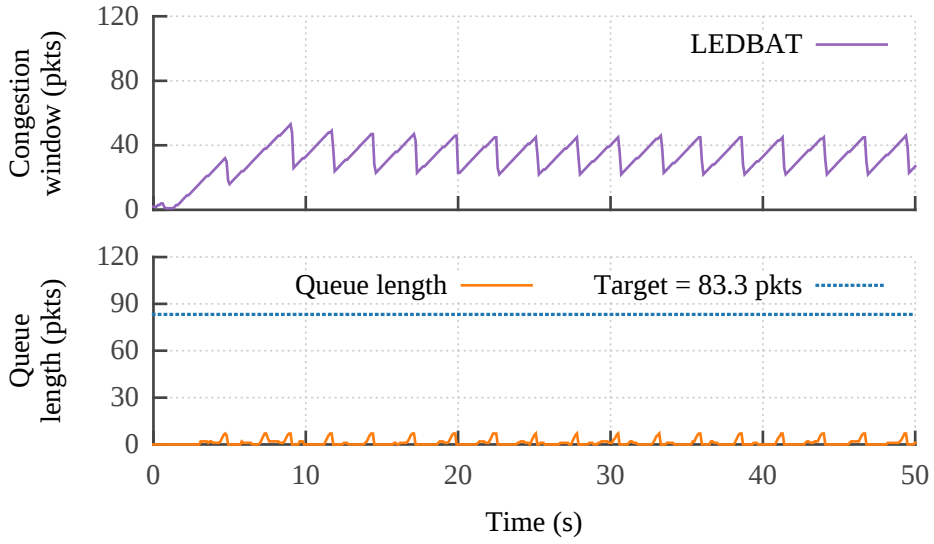


Figure 1: LEDBAT se comporte comme TCP standard dans le cas d'un *target queuing delay* à l'infini.

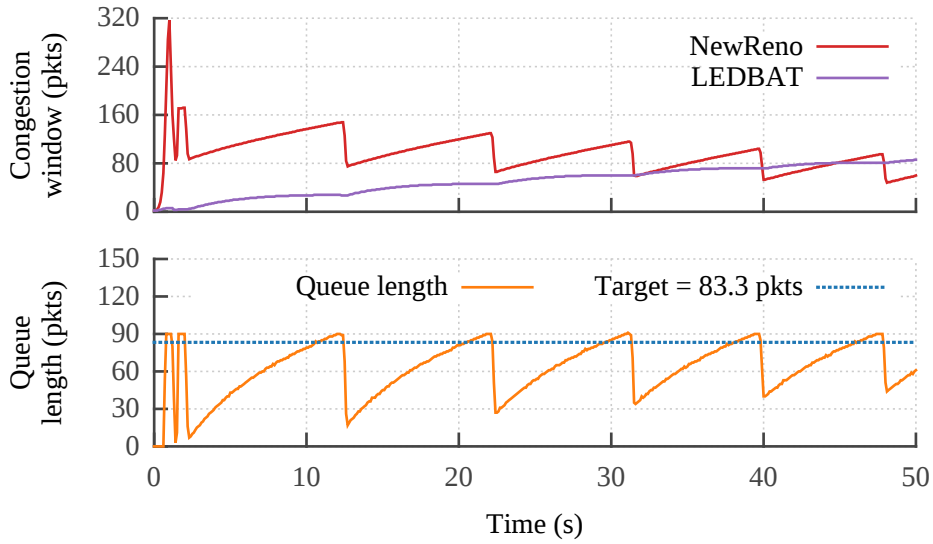


Figure 2: Agressivité de LEDBAT envers TCP.

II.c. Combinaison optimale de *target queuing delay* et *decrease gain*

Dans un premier temps, nous avons cherché l'existence d'une combinaison de *target queuing delay* et *decrease gain* qui permettrait à LEDBAT de se comporter comme un protocole de LBE quelle que soit la configuration réseau. Dans ce but, nous considérons les différentes capacités $C \in [1, 5, 10, 20, 50]$ Mb/s et les différents délais de propagation unidirectionnels (*one-way propagation delays*) $d \in [10, 50, 100, 150, 200, 250]$ ms du lien de goulot d'étranglement. Pour chaque réglage C, D, et B du réseau, nous explorons l'ensemble des *target queuing delays* $\tau \in [5, 25, 50, 75, 100]$ ms et l'ensemble des *decrease gains* $\gamma_{dcr} \in [1, 10]$. Nous considérons un flot LEDBAT commençant à $t = 0$ s et un flot TCP commençant plus tard à $t = 200$ s. Puis, nous calculons le taux d'utilisation du lien η de chaque flot pour chaque réglage du réseau.

Nous utilisons une méthode de partitionnement de données (*clustering method*) pour classer nos résultats. Pour chaque combinaison de *target queuing delay* et *decrease gain*, lorsque $B = BDP$, si le taux d'utilisation du lien de TCP est $\eta_{TCP} \geq 0,8$, alors nous choisissons le taux d'utilisation du lien de LEDBAT comme référence $\eta_{ref} = \eta_{LEDBAT}$. Ensuite, pour chaque autre valeur de B, nous calculons $\Delta = |\eta_{LEDBAT} - \eta_{ref}|$. Si $\Delta \leq \varepsilon$ et $\eta_{\text{TCP}} \geq 0,8$, alors nous classons B dans le cluster "Droite" (noté R), sinon, dans le cluster "Wrong" (noté W). Dans notre étude, nous utilisons $\varepsilon = 0,15$. Donc, pour les tailles de buffer différentes, si η_{TCP} est toujours $\geq 0,8$ et la différence entre η_{LEDBAT}

Figure 3: Nombre de cas R et W pour chaque réglage du *target queuing delay*.

et η_{ref} est dans une limite de 15%, nous considérons que la combinaison courante du *target queuing delay* et *decrease gain* fonctionne bien pour ces tailles de buffer. Enfin, nous effectuons une analyse statistique pour déterminer la combinaison de *target queuing delay* et *decrease gain* fonctionnant dans la plupart des configurations du réseau.

Choix du *target queuing delay*

Nous commençons par la recherche de la valeur optimale du *target queuing delay*. Dans la figure 3, utilisant un histogramme, nous regroupons les résultats de simulation dans les différentes catégories de capacités du réseau, puis, dans les sous-classes de *target queuing delay*. Pour chaque valeur du *target queuing delay*, la colonne empilée représente le nombre de cas R et W. La hauteur de la colonne correspond à l'ensemble des simulations de toutes les valeurs possibles du *decrease gain* en combinaison avec une valeur du *target queuing delay* donnée en abscisse.

La figure 3 montre qu'une valeur de *target queuing delay* de 5 ms fonctionne dans la plupart des cas. Ces résultats nous permettent donc de conclure que le réglage du *target queuing delay* à 5 ms est optimal.

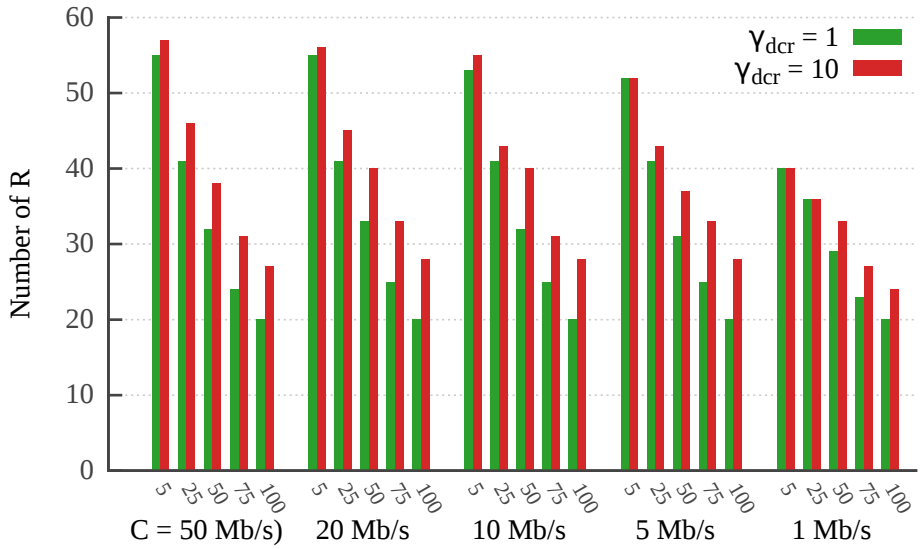


Figure 4: Nombre de cas R quand $\gamma_{dcr} = 1$ et $\gamma_{dcr} = 10$ pour chaque réglage du *target queuing delay*.

Choix du *decrease gain* et de la combinaison des paramètres optimaux de LEDBAT

Après avoir obtenu cette valeur de *target queuing delay* optimale, nous utilisons encore l'histogramme pour trouver le *decrease gain* optimal. Pour chaque paramètre de *target queuing delay*, nous présentons le nombre de cas R pour chaque valeur de *decrease gain* correspondante. Comme le montre la figure 4, régler le *decrease gain* à 10 est préférable car cela augmente le nombre de cas R.

II.d. Discussion

La combinaison de *target queuing delay* et de *decrease gain* (5 ms; 10) est globalement optimale, c'est-à-dire, optimale sur un large éventail de configurations de réseau. Cependant, ce n'est pas nécessairement une combinaison localement optimale pour une configuration de réseau spécifique. Par exemple, avec $C = 10\,\text{Mb/s}$, $d = 50\,\text{ms}$ et $B = 84\,\text{paquets}$, passer de la combinaison globalement optimale (5 ms; 10) à la combinaison (65 ms; 1) diminue le taux d'utilisation du lien de TCP de 6.84% tandis que le taux d'utilisation du lien de LEDBAT augmente de 778.37%. En outre, le problème du *latecomer advantage* persiste même avec la combinaison de ces paramètres globalement optimaux. En conclusion, bien que nous approchons d'une solution générique, il nous

faut toujours trouver une solution qui résout les deux principaux problèmes de LEDBAT.

III. FLOWER, un protocole de transport Lower-than-Best-Effort basé sur la logique floue

Dans cette section, nous présentons FLOWER, un nouveau protocole de transport visant à fournir un service LBE tout en résolvant les deux principaux problèmes de LEDBAT — l'agressivité envers TCP et le *latecomer advantage*.

III.a. Motivation et objectifs de FLOWER

Motivation

Les deux principaux paramètres de LEDBAT — *target queuing delay* et *gain* — sont fixes et ne peuvent pas faire face à la diversité des configurations de réseau. Par conséquent, LEDBAT devient plus agressif que TCP dans certaines circonstances. Une solution possible serait d'adapter le *target queuing delay* et/ou le *gain* au changement des conditions réseaux [11, 12]. À cette fin, nous devons d'abord dériver un modèle mathématique du réseau, puis construire un modèle mathématique du contrôleur. Cependant, le réseau est un système complexe de part son hétérogénéité. Ainsi, il n'est pas trivial de dériver un modèle de réseau suffisamment précis et générique considérant les réseaux mobiles, cellulaires, filaires, pour être utilisé par un système de contrôle adaptatif. D'où notre choix d'utiliser un contrôle à logique floue pour relever les difficultés dans le développement et l'analyse de ce système complexe. Les avantages principaux du contrôle à logique floue sont :

- un modèle mathématique n'est pas nécessaire pour développer un système de contrôle. En effet, le contrôle à logique floue nous permet d'intégrer un heuristique représentant le contrôleur. Cette approche est particulièrement intéressante lorsque le modèle n'est pas trivial, difficile à dériver ou trop complexe à implémenter;

- comme ils sont non linéaires en général, les contrôleurs à logique floue peuvent opérer dans un éventail de conditions d'opérations plus larges que les contrôleurs PID;

- les contrôleurs à logique floue sont faciles à personnaliser car ils sont constitués de règles linguistiques qui sont faciles à comprendre et à modifier;

- les opérations du contrôleur à logique floue sont simples à manipuler et à mettre en œuvre.

Compte tenu des inconvénients du système de contrôle adaptatif et des avantages du contrôle à logique floue, nous proposons un nouveau protocole de congestion LBE basé sur ce principe. Les travaux présentés dans la section II. nous informent sur la façon dont il faudrait contrôler le délai de la file d'attente. C'est ensuite sous forme d'une heuristique que ces mêmes travaux sont intégrés au sein du mécanisme de contrôle flou de FLOWER. Nous soulignons aussi que, en utilisant un système de contrôle à logique floue, nous cherchons une solution générique qui fonctionne dans diverses conditions réseaux. Cela signifie que nous sommes maintenant à la recherche d'un cas d'utilisation moyen.

Objectifs

Proposé comme une alternative à LEDBAT, FLOWER se doit bien évidemment de résoudre les problèmes de LEDBAT tout en gardant les mêmes objectifs en termes de service LBE comme indiqués dans [10] :

1. utiliser la bande passante de bout-en-bout disponible et maintenir un faible délai de traversée de la file d'attente lorsque aucun trafic n'est présent dans cette dernière;
2. ne pas ajouter de délai de traversée de la file d'attente supplémentaire à celui induit par les flux concurrents;
3. laisser rapidement la place aux flux TCP qui partagent le même goulot d'étranglement.

Pour atteindre ces objectifs, FLOWER implémente un contrôleur à logique floue pour gérer le délai de la file d'attente à la place du contrôleur de type P proposé dans [10]. Le *target queuing delay* non nul permet à FLOWER d'aller chercher la capacité disponible, et donc de saturer le lien de goulot d'étranglement lorsqu'aucun autre trafic est présent. En même temps, le délai de la file d'attente doit être maintenu aussi faible que possible pour que FLOWER soit non intrusif au trafic TCP standard.

III.b. Contour du design de FLOWER

Nous pouvons représenter FLOWER comme un système de contrôle en boucle fermée comme illustré par la figure 5. Les composantes essentielles de FLOWER sont :

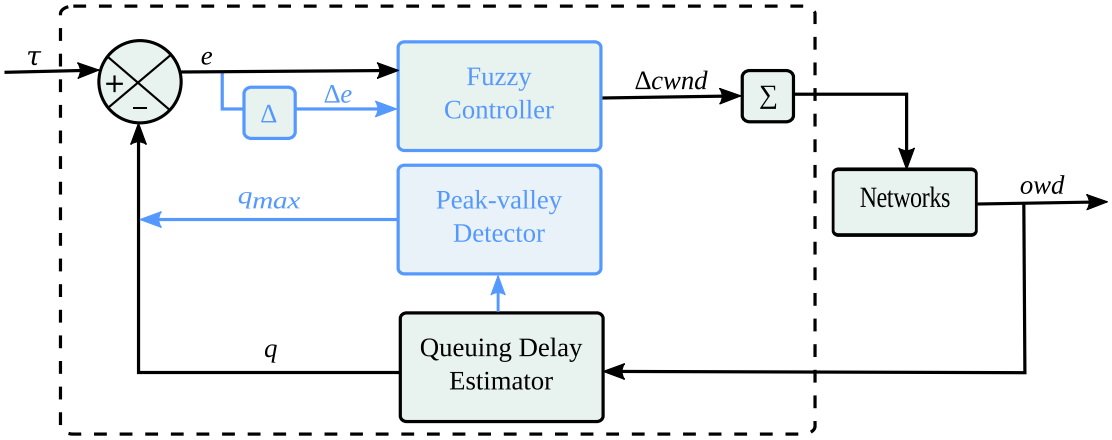


Figure 5: Schéma de FLOWER

1. *en entrée* :

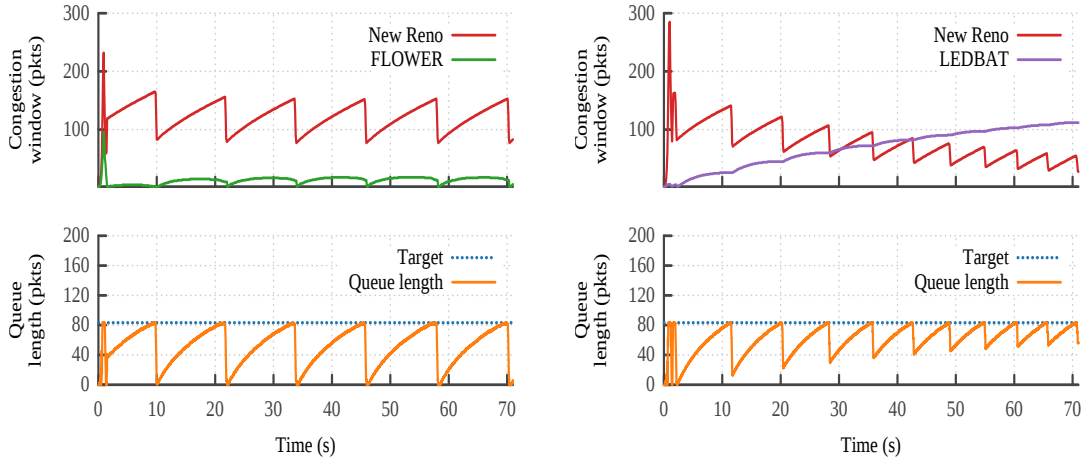
   - τ : *target queuing delay*, qui est le délai de la file d'attente maximum qu'un flux de FLOWER est autorisé à introduire dans le réseau;
   - e : *queuing delay error*, qui est la différence entre le *target queuing delay* et le délai de la file d'attente estimé;
   - Δe : *change on queuing delay error*, qui est la tendance de l'erreur.

2. le *queuing delay estimator*, qui exploite les délais unidirectionnels mesurés pour estimer le délai de la file d'attente courant q;

3. le *peak-valley detector*, qui garde la trace du délai de la file d'attente maximum $q_{\max}$ observé dans le réseau. Ce délai de la file d'attente maximum est ensuite utilisé pour normaliser la *queuing delay error* (l'erreur de délai de la file d'attente);

4. le *fuzzy controller*, qui est un décideur artificiel fonctionnant sur la base d'un ensemble de règles "If–Then". En utilisant la logique floue, le contrôleur détermine la taille de la fenêtre de congestion `cwnd` telle que le futur délai de la file d'attente estimé corresponde au *target queuing delay*. Le contrôleur à logique floue intègre également un mécanisme de détection de perte dans sa base de règles pour améliorer la détection de la congestion.

FLOWER utilise le *slow-start* pour obtenir une mesure préliminaire du délai de la file d'attente. D'ailleurs, FLOWER pourrait souffrir du problème *latecomer advantage*



(a) FLOWER vs. TCP New-Reno (b) LEDBAT vs. TCP New-Reno

Figure 6: Fenêtres de congestion des flux TCP et LBE et longueur de la file d'attente du goulot d'étranglement en fonction du temps.

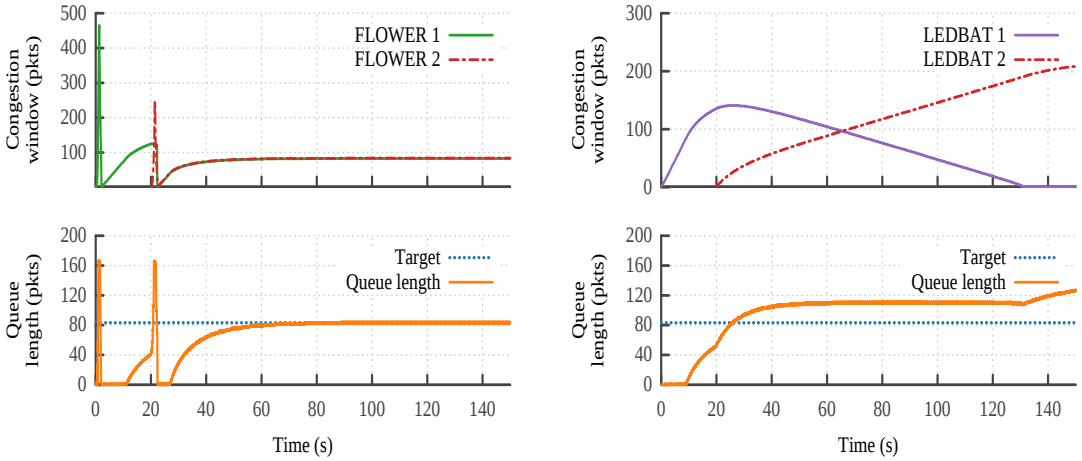
comme LEDBAT. Heureusement, dans cette situation, le mécanisme de détection de perte du contrôleur à logique floue de FLOWER et le *slow-start* aide à resynchroniser tous les flux en cours. Finalement, en cas de perte, FLOWER réinitialise sa fenêtre de congestion au minimum.

IV. Evaluation de FLOWER

Dans cette section, nous étudions la performance de notre nouveau protocole par le biais de simulations sous ns-2.

IV.a. Interaction avec TCP

Nous considérons deux flux TCP et LBE (LEDBAT, FLOWER) qui commencent à l'instant $t = 0$. La taille du buffer dans ce cas est égale à la BDP. La figure 6 montre les fenêtres de congestion (en haut) en fonction du temps, la longueur de la file d'attente et le *target queuing delay* exprimés en paquets (en bas). L'interaction entre TCP et FLOWER est représentée à la figure 6a et celle entre TCP et LEDBAT à la figure 6b. Cette simulation illustre le bon comportement de FLOWER en tant que protocole LBE en présence de TCP. De toute évidence, le contrôleur utilisant la logique floue avec le mécanisme de détection de perte permet à FLOWER de se conformer au principe LBE.



(a) FLOWER (b) LEDBAT

Figure 7: Fenêtres de congestion des flux LBE et longueur de la file d'attente du goulot d'étranglement en fonction du temps.

Dans cette configuration standard ($B = BDP$), LEDBAT ne se comporte pas comme un protocole LBE et est trop agressif envers TCP. Le comportement défectueux de LEDBAT s'explique par la réaction inexacte de son contrôleur de type P face à la congestion.

IV.b. Equité intra-protocole

Dans cette expérience, la taille du buffer B est fixée à deux fois la BDP. Cette configuration est favorable pour reproduire le phénomène *latecomer advantage* de LEDBAT. Le premier flux LBE (LEDBAT, FLOWER) commence à $t = 0\,\text{s}$ et le second commence à $t = 20\,\text{s}$. Comme dans la section IV.a., nous traçons les fenêtres de congestion et la longueur de la file d'attente du buffer. La figure 7b montre bien le problème de latecomer de LEDBAT. Au contraire, FLOWER n'hérite pas de ce problème grâce au mécanisme de détection de perte. La figure 7a montre que deux flux FLOWER partagent effectivement équitablement la capacité du lien.

IV.c. Coexistence de FLOWER et AQM

LEDBAT a été conçu pour fonctionner principalement avec la politique de file d'attente DropTail. En présence de mécanismes AQM plus évolués, LEDBAT perd sa caractéristique LBE et se comporte alors comme un TCP standard [7, 13, 14]. La RFC de LEDBAT le constate également [10]: *"If Active Queue Management (AQM) is configured to drop or*

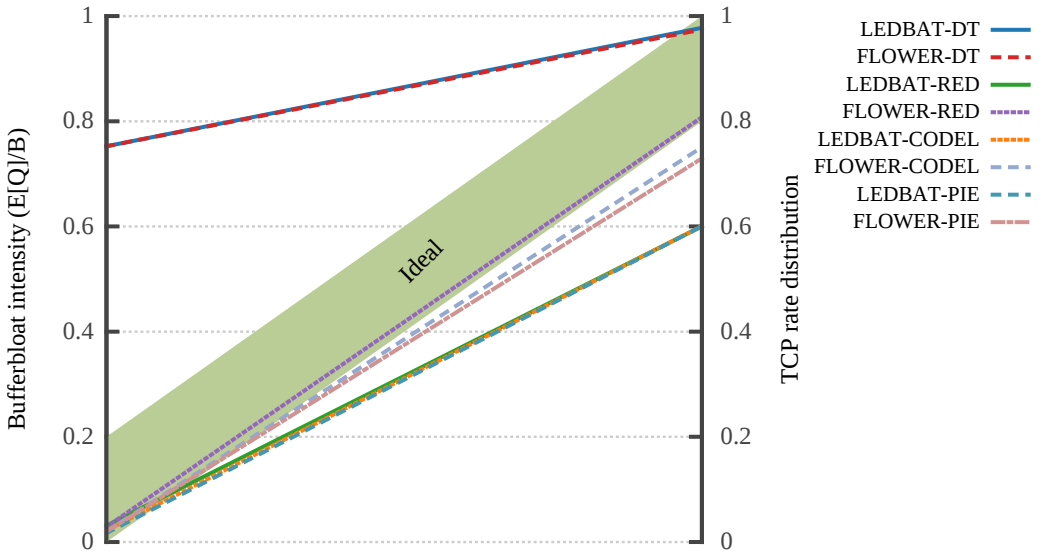


Figure 8: Impact des mécanismes d'AQM sur les protocoles de LBE.

ECN-mark packets before the LEDBAT flow starts reacting to persistent queue buildup, LEDBAT reverts to standard TCP behavior rather than yielding to other TCP flows". Par conséquent, lors de la conception et de la proposition d'un nouveau protocole LBE (voire d'un protocole de transport de façon générale), il est essentiel d'étudier sa coexistence avec les mécanismes AQM existants.

Dans cette section, nous évaluons l'impact des AQMs comme RED [15], CoDel [16] et PIE [17] sur la conformité au service LBE proposé par FLOWER en présence de TCP. À cette fin et pour avoir une base de comparaison équitable, nous employons directement les scripts utilisés par les auteurs de [14], disponibles sur [18]. Nous considérons 5 flux TCP en concurrence avec 5 flux LBE. Tous les flux commencent à l'instant $t = 0$. La taille du buffer B est égale à 3 fois la BDP afin de reproduire le *bufferbloat* — le phénomène de faible débit et de délai élevé causé par un buffer de taille excessive. En général, un mécanisme AQM est considéré comme la meilleure solution pour résoudre ce phénomène [19]. Nous mesurons la répartition du débit des flux TCP (X_{TCP}), la longueur moyenne de la file d'attente en termes de paquets ($E[Q]$) et l'intensité du *bufferbloat* définie comme $E[Q]/B$.

Nous présentons les résultats de la simulation dans la figure 8. L'interaction de chaque combinaison des mécanismes AQM et des protocoles LBE est représentée par une ligne reliant les deux métriques $E[Q]/B$ (l'axe y à gauche) et X_{TCP} (l'axe y à droite). L'interaction idéale est illustrée par la région oblique verte dans la figure 8, où

le délai de la file d'attente est faible tandis que le trafic LBE reste en faible priorité.

Avec la politique de file d'attente DropTail, FLOWER et LEDBAT se conforment tous au principe LBE. Dans ce cas, TCP maximise également l'intensité du *bufferbloat*. Au contraire, la figure 8 montre clairement que l'utilisation d'un mécanisme AQM résout le problème de *bufferbloat*. Cependant, un tel mécanisme AQM compromet également la caractéristique de faible priorité des protocoles LBE et augmente leur agressivité envers TCP. Dans tous les cas, FLOWER se conforme toujours plus au principe LBE que LEDBAT et tend vers la région idéale. Il y a deux raisons qui expliquent ce résultat : premièrement, FLOWER a une zone de détection de perte dans sa base de règles qui lui permet de réagir mieux que LEDBAT face à la congestion; deuxièmement, FLOWER réinitialise sa fenêtre de congestion au minimum en cas de perte afin d'atténuer son impact sur les flux de priorité plus haute.

V. Conclusion

Dans cette thèse, nous examinons la possibilité de déployer un service LBE sur des liens LBDP tels que des liens satellites. Nous nous intéressons à une approche de bout-en-bout et donc nous nous concentrons spécifiquement sur les solutions de la couche transport. Nous proposons tout d'abord d'étudier LEDBAT en raison de son potentiel. Cependant, LEDBAT est connu pour avoir deux problèmes principaux : son agressivité envers TCP et le *latecomer advantage*. En explorant l'ensemble des paramètres de LEDBAT, nous constatons que (5 ms; 10) est la combinaison optimale de *target queuing delay* et *decrease gain* permettant à LEDBAT d'être conforme au principe de LBE sur un large éventail de conditions réseaux. Malheureusement, le problème du *latecomer advantage* de LEDBAT persiste même avec cette combinaison optimale de paramètres. Par conséquent, nous proposons FLOWER, un nouveau protocole de transport visant à fournir un service LBE tout en résolvant deux principaux problèmes de LEDBAT. A notre meilleure connaissance, FLOWER est la première solution qui résout à la fois l'agressivité et le *latecomer advantage* de LEDBAT tout en conservant ses propriétés LBE en présence de mécanismes AQM.

1 Introduction

The switch of the official BitTorrent client to Low Extra Delay Background Transport (LEDBAT) [10], a new low-priority congestion-control algorithm, has once again raised the interest of the Internet research community in the Lower-than-Best-Effort (LBE) service. Dedicated to carrying non-critical traffic, the LBE service seeks to use, in a non-intrusive manner, the remaining network capacity unused by best-effort flows. Hence, LBE service leads to many applications such as background or signaling traffic. This thesis looks to deploy the LBE service over large bandwidth-delay product (LBDP) networks, in particular satellite networks, since this service helps to optimize the full use of the link capacity. Furthermore, the LBE service over satellite communications can enable a low-cost or even free Internet access to remote communities, as discussed below.

While an LBE service can be achieved at other layers, in this thesis, we solely study its deployment at the transport layer [2] because of the scalability and simplicity of such end-to-end approach. Moreover, we mainly focus on LEDBAT since it is the most widely deployed LBE protocol. Developed by BitTorrent and later standardized by IETF, LEDBAT rapidly gains notoriety and plays an important role in Internet traffic. Therefore, the evaluation of its performance is an obvious concern for the Internet research community.

Intuitively, one may question whether users are willing to use a service worse than best effort. Therefore, in Section 1.1, we first illustrate the benefits of the LBE service through an introductory example. Then we present the motivation of this thesis in Section 1.2 and our contributions in Section 1.3. Finally, we outline the thesis in Section 1.4.

1.1 LBE + Satellites = Internet Access for All

In 2012, the Internet Society conducted an online survey of more than 10000 Internet users across 20 countries [20]. This study highlighted that 83 percent of the respondents believe that access to the Internet should be considered as a basic human right. Indeed, in today's digital world, the Internet is a critical infrastructure enabling many services, including communication, digital economy, education, employment, e-governance, remote health care, social networks, and more. For many of us, sending and reading emails or surfing the Internet is an integral part of our daily activities. In addition, the rapid evolution of technology makes the Internet faster and cheaper. Thus, for millions of people, Internet access is a trivial matter, so trivial that we may unconsciously ignore that many others do not even have access to basic services, let alone the Internet. Giving its crucial role, Internet access must be available for anyone from everywhere: a vision that is shared among both major stakeholders and global governments [1]. In reality, many projects have already launched to provide an affordable Internet access, for example, Loon [21] by Google and Internet.org [22] by Facebook. The Internet Engineering Task Force (IETF) also formed the Global Access to the Internet for All Research Group (GAIA) [23] to address the problem.

While the objective sounds straightforward, the realization encounters many challenges. The most important challenge is that the population in remote communities are usually scattered over wide areas. Therefore, the investment cost for Internet service providers to deploy infrastructures is simply too high. However, in this case, the satellite communication coupled with an LBE protocol such as LEDBAT offers a promising alternative solution. On one hand, the satellite has a wide range of coverage and thus can reach remote areas at relatively low cost. On the other hand, LEDBAT, also known as µTP (Micro Transport Protocol) [24], is a popular LBE protocol that can be employed to exploit the unused capacity of satellite links for free Internet access, while remaining transparent to commercial traffic.

Sharing the same conviction as the authors of [25], we believe that Internet access for all is not just an insane idealism, and the LBE service is one key solution to make it happen.

1.2 Motivation

Delivering an affordable Internet access, as presented in the previous section, is just one example of many potential applications offered by an LBE protocol such as LEDBAT.

Other examples are data backup, prefetching, Internet content distribution, peer-to-peer file transfer, and more [3]. Given its focus on carrying heavy low-priority traffic, a misbehavior of an LBE protocol can seriously disturb best-effort traffic. Hence, before deploying an LBE protocol on a large scale, we need to carefully study its impact, if any, on best-effort traffic. Unfortunately, in this context, despite being a promising LBE protocol, LEDBAT has been pointed out to be more aggressive than TCP in some configurations [4, 5]. Nevertheless, these works do not cover a wide range of network configurations. Moreover, although the authors of [5] hint a possible solution, there is no in-depth study concerning the potential solutions to this problem.

Besides its aggressiveness, LEDBAT also suffers the latecomer advantage — the symptom in which a newly arriving LEDBAT flow can starve ongoing LEDBAT flows [5–9]. Unlike the aggressiveness problem, the latecomer problem has received a lot of attention, and many solutions have been proposed [5, 8, 9]. Yet, the latecomer problem is argued to be less important than the aggressiveness problem since LEDBAT is designed for a low-priority and delay-insensitive traffic class [5]. In addition, the latecomer advantage is remedied in the presence of TCP flows [6]. However, given LEDBAT's potential to enable Internet access for all, we believe that the latecomer problem also needs to be addressed.

From what has been discussed above, the first objective of this thesis is to realize a thorough study to gain insight into the aggressiveness of LEDBAT. The second objective is to provide an effective solution that solves both aggressiveness and latecomer problems of LEDBAT. Furthermore, while our initial goal is to study the feasibility of a deployment of LEDBAT over LBDP networks, throughout this thesis, we take a further step towards a solution that works over a wide range of network configurations.

1.3 Contributions

The two main contributions of this thesis are:

Analysis of LEDBAT aggressiveness behavior: Using simulation, we deeply study the impact of LEDBAT internal parameters on its performance in the presence of TCP over a wide range of network configurations. The study shows that a misconfiguration of LEDBAT parameters leads to a total starvation of TCP flows. We further explain the nature of this problem and demonstrate that the setting of LEDBAT parameters is highly depend on the network configurations. Therefore, we propose an optimal combination of parameters that allows LEDBAT to behave

as an LBE protocol in different network configurations. However, we observe that the aggregated use of optimized LEDBAT sources in a highly loaded network still disturbs the overall traffic performance. As a result, we underline the necessity for additional strategies to limit the number of LEDBAT flows. Our study has been published in [26].

Design of FLOWER — a new LBE protocol: While the optimal combination of LEDBAT parameters mitigates the aggressiveness issue in some cases, it also limits the LEDBAT performance in other cases. Moreover, the optimal combination does not solve the latecomer problem. Therefore, we design the Fuzzy Lower-than-Best-Effort (FLOWER) transport protocol to solve both aggressiveness and latecomer problems of LEDBAT. Our simulation study over a wide range of network use-cases shows that FLOWER performs better than LEDBAT in cases where LEDBAT usually fails. Furthermore, FLOWER exhibits a good interplay with AQM schemes. To the best of our knowledge, FLOWER is the first solution that solves all three LEDBAT issues: the aggressiveness, the intra-unfairness, and the bad interaction with AQM schemes. The design of FLOWER and its performance evaluation, except the results with AQMs, have been published in [27].

1.4 Organization

In Chapter 2, we briefly describe the background assumed in the rest of this thesis and review previous work that is relevant to the problem being addressed. Especially, an extensive review on the state of art of LEBDAT points out the lack of a solution for the aggressiveness problem, one of two main LEDBAT problems. The other problem is the latecomer advantage.

In Chapter 3, we explore the set of optimal parameters allowing LEDBAT to effectively perform as an LBE protocol. We found that a target of 5 ms and a decrease gain of 10 are globally optimal, that is, optimal over a wide range of network configurations. However, the latecomer unfairness issue of LEDBAT still persists even with the global optimal combination of parameters. Therefore, it is necessary to find a solution that solves both the aggressiveness and latecomer unfairness of LEDBAT.

Based on the insight gained from the LEDBAT analysis in Chapter 3, we design in Chapter 4 a new LBE transport protocol, named FLOWER, by employing the fuzzy control concept.

Finally, in Chapter 5, we evaluate the performance of the new LBE protocol FLOWER

by means of simulation. The results show that FLOWER behaves as an LBE protocol in network scenarios where LEDBAT fails while solving the latecomer unfairness problem. We also demonstrated that FLOWER is more LBE-compliant than LEDBAT when coexists with AQM schemes.

2 Background and Related Work

In this chapter, we briefly describe the background assumed in the rest of this thesis and review previous work that is relevant to the problem being addressed. First, we present the Internet access via satellite in Section 2.1. Then, we review TCP and its variants in Section 2.2. Next, Section 2.3 describes the principle of LBE and Section 2.4 introduces the feedback control and the PID control. Afterwards, we describe LEDBAT in Section 2.5. Finally, an extensive review on the state of art of LEDBAT is given in Section 2.6.

2.1 Internet Access via Satellite

As discussed in Chapter 1, an attractive application of LBE is to enable free Internet access for rural communities via satellite communication. In the meantime, paying satellite Internet access is already available for people in remote areas. In this section, we provide an overview of satellite communication and describe how satellite Internet works.

2.1.1 Characteristics and Applications of Satellite Communication

The **satellite communication** [28] is one variety of wireless communication systems. In this system, two or more points earth stations communicate via a communication satellite. A **communication satellite** is an artificial satellite orbiting around Earth. It functions as a radio relay station that receives, amplifies and redirects analog and digital signals from and to one or many places around the world.

Characteristics

A main characteristic of the satellite communication is its expensive investment cost. The first factors that contribute to the high cost of satellites are the equipment and materials employed to construct them. Another factor is the excessive expense for launching satellites into orbit. Once a satellite is put into orbit, we need human operators to monitor it from a ground facility. Therefore, we must also take into account the maintenance cost of satellites. Satellite communication is further characterized by a relatively long one-way delay. For instance, the one-way delay for a communication with Geostationary Earth Orbit (GEO) is typically on the order of 250 ms. Furthermore, the throughput of a satellite link is constrained mainly by both power and bandwidth resources available.

Nevertheless, despite these aforementioned limitations, satellite communication offers many attractive advantages. Indeed, a satellite provides a wide geographical coverage which largely surpasses that of a terrestrial system. Moreover, satellite networks can be deployed faster than terrestrial infrastructures. Satellites are further not vulnerable to natural disasters (earthquake, tsunami...) or man-made disasters (nuclear explosion, blackout...). The availability and deployability make the role of satellite communication effective in disaster management primordial. Finally, satellite communication is economical over long distances.

Applications

One popular application of satellite communication is satellite phones. Satellite phones are seldom used when cellular phones are available because the satellite phones are cumbersome and their service is more costly. However, satellite phones are still needed in situations where wired or other means of connections are unavailable such as on planes, ships or in remote areas. Today, satellite communication is well known for its application in television and radio broadcasting. Moreover, it is used for providing Internet access which is helpful for users in remote areas without a broadband connection. Satellite communication also finds its applications in military, in search and rescue and much more.

2.1.2 Satellite Internet

We now have a basic concept of the satellite communication. In what follows, we outline how to access the Internet using satellite communication.

Figure 2.1 illustrates a typical satellite network providing Internet access. The two-way (forward and return) communication satellite connects all user terminals, via a

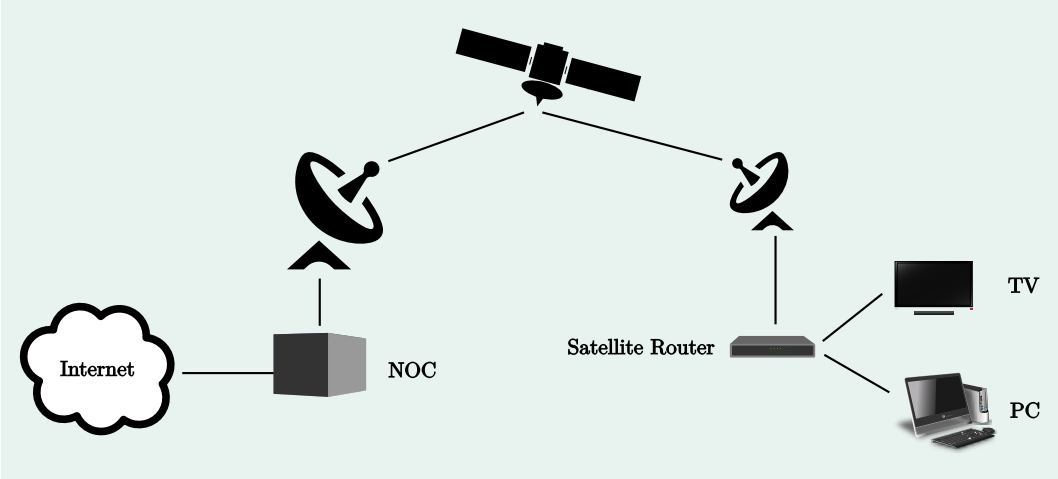


Figure 2.1: A satellite network

satellite router, to a central gateway or hub at an earth station called the Network Operations Center (NOC). The NOC gateway, in its turn, connects to the wide Internet.

To offer Internet access, satellite links must support TCP/IP (Transmission Control Protocol/Internet Protocol), the de facto standard for connecting hosts on the Internet. The responsibility of IP is to deliver packets from a source host to a destination host, using an address scheme to identify hosts. IP provides a **best-effort delivery service**. This means that IP makes every effort to deliver packets to their destination, but does not offer any guarantee. Packets may be lost, duplicated or corrupted.

In its turn, TCP provides an end-to-end, reliable, in-order data delivery service to applications on top of the unreliable best-effort service of IP. TCP also employs a **flow-control mechanism** to prevent the sender from overwhelming the receiver. Finally, TCP implements a **congestion-control mechanism**, which will be described in the next section, to keep all of the senders together from overloading the network.

2.2 Transmission Control Protocol (TCP)

In this thesis, we only focus on the congestion-control aspect of TCP. Therefore, in this section, we describe the standard congestion-control mechanism named TCP NewReno (hereafter denoted as standard TCP or TCP) and some of its most well-known variants. We also discuss the difficulties of providing an efficient end-to-end communication between pairs of hosts over LBDP networks such as satellite networks.

2.2.1 Standard TCP Congestion Control

A TCP sender maintains a state variable called congestion window (`cwnd`), which is its estimation of the number of packets that it can send over the network without them being discarded by the intermediate routers. When a TCP receiver receives a packet, the receiver replies with an acknowledgment (ACK) to indicate to the sender that the transmission is successful. A TCP sender detects packet losses either through retransmission timeout or three duplicate acknowledgments. The evolution of the congestion window consists of two phases: slow start and congestion avoidance. TCP enters the slow-start phase at the beginning of the connection or after a packet retransmission timeout. In the slow-start phase, the sender increases `cwnd` by one packet whenever it receives an ACK. As a consequence, `cwnd` is doubled every round-trip time (RTT). TCP exits the slow-start phase when it notices a packet loss or when `cwnd` is greater than a slow-start threshold (`ssthresh`). This threshold is set to half of the current `cwnd` when a packet loss is detected by a retransmission timeout.

After leaving the slow-start phase, TCP enters the congestion-avoidance phase. In this phase, TCP uses an Additive Increase/Multiplicative Decrease (AIMD) algorithm to probe for available network bandwidth. For each RTT, a TCP sender increases `cwnd` by one packet, that is, increases `cwnd` by $\frac{1}{cwnd}$ on each successful ACK, until it experiences a packet loss. If this packet loss is detected by three duplicate ACKs, TCP reduces its `cwnd` by half. If the packet loss is instead detected by a timeout, TCP sets its `cwnd` to one packet and re-enters the slow-start phase.

2.2.2 High-Speed TCP Congestion Control Algorithms

Despite being one of the decisive factors behind the success of Internet, the AIMD behavior of TCP does not scale well and thus results in the poor performance of TCP in LBDP networks [29]. In addition, TCP cannot distinguish whether a packet is dropped due to congestion or corruption. Therefore, TCP reduces its congestion window size in the presence of any packet-drop signals. As a consequence, for wireless LBDP networks such as satellite networks, packet corruptions due to transmission errors unnecessarily degrade the TCP throughput [28].

In what follows, we will present CUBIC and Compound TCP, two currently used TCP variants designed particularly to solve the problem of standard TCP over LBDP networks.

CUBIC TCP

CUBIC [30] uses a cubic function of time to modulate its congestion window. Consequently, the growth rate of the CUBIC congestion window is independent of the RTT. Indeed, it depends on the time since the last congestion event.

By employing the cubic function [30], the congestion window growth of CUBIC is divided into two phases. In the first concave phase, CUBIC rapidly increases the congestion window to the size achieved before the last congestion event. Then, in the second convex phase, CUBIC probes for more bandwidth. Since it is the default congestion control algorithm in the Linux kernel since the version 2.6.19, CUBIC is widely deployed in Internet. However, CUBIC also suffers a number of issues, noticeably the slow convergence time, as pointed out in [31].

Compound TCP

Compound TCP [32] is designed to improve link efficiency and RTT fairness. It uses the delay as the congestion indicator. Compound TCP maintains in parallel two congestion windows. The first one is a regular loss-based congestion window employing the AIMD mechanism of standard TCP The second one is a delay-based congestion window inspired from TCP Vegas [33]. The resulting congestion window is the sum of these two windows.

If the estimated delay is small, the delay-based window quickly increases to exploit the available bandwidth. When the queue starts to fill up, to keep the summed congestion window constant, the delay-based window decreases to counterbalance the increase of the loss-based window. Compound TCP falls back to standard TCP under heavy network congestion. Despite being designed especially for LBDP networks, Compound TCP can suffer poor scaling behavior similar to standard TCP as the BDP increases [34].

2.3 Principle of Lower-than-Best-Effort (LBE)

As the main subject of this thesis, we present in this section the **Lower-than-Best-Effort (LBE) service** that aims at providing a second priority class inside the network traffic.

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. However, not all traffic have a same degree of priority. In fact, background traffic such as data backup, software updates, or peer-to-peer file transfers are less critical than

other traffic and may be willing to accept a lower quality of service than best-effort service. Therefore, LBE service, also called *scavenger service*, is proposed to carry low-priority traffic. This kind of traffic tolerates a higher latency and, most importantly, they should not disturb the traffic needing best-effort service itself or other services that would propose advanced QoS architecture for time-constrained applications such as DiffServ [35]. Moreover, the LBE service should also not exacerbate the bufferbloat issue [36], which is the phenomenon of low throughput and high latency caused by excessive buffering.

To offer an LBE service, the main idea is to exploit any unused bandwidth not being used by best-effort service while being transparent to the later, that is, LBE traffic must yield to best-effort traffic under congestion. As a result, LBE traffic may undergo a high packet loss rate or even starvation. An LBE service can be deployed at different layers [2,3,37–39]. Yet, in this thesis, we only focus on the LBE service at the transport layer [2] because of the scalability and the ease of deployment of such an end-to-end approach.

However, introducing LBE traffic always disturbs TCP traffic, as shown by the authors in [12] and by our study in Chapter 3. Therefore, a difficulty when dealing with LBE service is to determine its degree of disturbance: how much disturbance would be acceptable and considered as non-intrusive to TCP traffic? There is no common answer for this question as each operator or user would subjectively define their own degree of disturbance. In this thesis, since we mainly focus on LEDBAT, a widely deployed LBE protocol developed by BitTorrent, we take its performance as reference for discussion. More specifically, we consider as LBE-compliant if and only if an LBE transport protocol satisfies both following criteria:

- first, the LBE protocol must limit the maximum "disturbing" queuing delay that it is allowed to introduce into the network around a fixed target;
- second, at the latest, the LBE protocol must yield to TCP, that is, resets its congestion window to minimum, immediately after the buffer overflow caused by TCP.

2.4 Feedback Control and PID Control

In this thesis, we deeply study LEDBAT, an LBE protocol developed by employing feedback control theory, more specifically, PID control theory. Therefore, we devote this

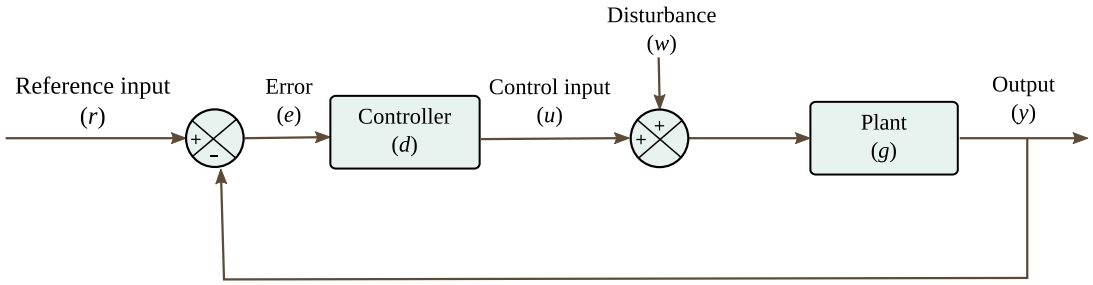


Figure 2.2: Block diagram of a feedback control system.

section to outline general feedback control systems, before delving into the LEDBAT design in the next section.

2.4.1 Overview of Feedback Control

Feedback control systems are everywhere in our daily life, from household applications such as the thermostat in a house to aeronautical applications such as the autopilot in an aircraft. The goal of feedback control is to determine the **input** of the **controlled system** so that its **output** matches a **reference input** in spite of **disturbances**. Figure 2.2 illustrates fundamental elements of a feedback control system [40]:

Controlled system The system to be controlled, that is, to be regulated its properties.

Control input u The controllable variable of the controlled system.

Control output y The measurable property of the controlled system.

Reference input r The desired output of the controlled system.

Disturbance input w The uncontrollable and undesirable input that affects the measured output of the controlled system in conjunction with the control input.

Control error e The difference between the reference input and the measured output.

Controller The controller calculates the control input required to reach the reference input, that is, reduce the control error to zero.

Congestion control mechanisms are indeed feedback control systems. For standard TCP, the controller is the AIMD mechanism at the sender. The controlled system is the network including the receiver. Outputs of the controlled system are ACK packets from the receiver. The inter-arrival time of ACK packets is then converted into a binary signal that indicates whether or not a packet-loss event happens. The output of the controller is the congestion window. Notice that there is no reference input for the controller in this case. As as result, the maximum throughput of a flow is proportional to the maximum congestion window achieved before a loss is incurred due to congestion. Lastly, from the standpoint of a flow, other flows are disturbance inputs.

2.4.2 PID Control

Having considered the feedback control, we now briefly describe the **PID (Proportional-Integral-Derivative) control** [41], which is the most popular feedback control mechanism. Simple, reliable, and easy to understand are the reasons why it is widely used in the industry. The equation of the output $u(t)$ of a PID controller, which is also the control input of the controlled system, is as follows:

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{d}{dt} e(t) \tag{2.1}$$

where

K_p: Proportional gain,

K_i: Integral gain,

K_d: Derivative gain,

$e(t)$: Control error.

Roughly speaking, the P-term $K_p e(t)$ represents the present error, the I-term $K_i \int e(t)dt$ represents the accumulation of past errors, and the D-term $K_d \frac{d}{dt} e(t)$ represents the prediction for future errors. The performance of the PID controller strongly depends on the tuning of three gain parameters K_p, K_d, and K_i. Finally, we do not always need to use all the three terms of PID controller. There exist other forms of PID controllers such as the P-type controller and PI-type controller. As explained in the next section, LEDBAT uses the P-type controller to control its congestion window.

2.5 Low Extra Delay Background Transport (LEDBAT)

In this section, we present Low Extra Delay Background Transport (LEDBAT), a widely deployed LBE protocol that has been standardized by the Internet Engineering Task Force (IETF) [10]. As we will see, while being a pervasive LBE protocol, LEDBAT suffers from a number of issues, making it ill suited for certain networks such as constrained wireless networks. Consequently, we cannot carelessly use LEDBAT to deploy LBE services over satellite networks, which is the aim of this thesis.

2.5.1 LEDBAT in a Nutshell

LEDBAT is a delay-based transport protocol that aims at offering an LBE service. As stated in RFC 6817 [10], the design goals of LEDBAT are:

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 its goals, LEDBAT implements two mechanisms. The first mechanism is a **P-type controller** that controls the congestion window using the end-to-end queuing delay as the congestion indicator. The objective of the controller is to limit the queuing delay around a fixed target. This non-zero target queuing delay allows LEDBAT 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 LEDBAT non-intrusive to standard TCP traffic. The LEDBAT controller relies on the second mechanism, the **queuing delay estimator**, to measure the delay that will be used as its input. More specifically, the queuing delay estimator estimates only the end-to-end queuing delay on the forward path.

In the following subsections, we will describe in detail the two main components of LEDBAT. For a better comprehension, we represent the LEDBAT congestion control algorithm as a feedback control system depicted in Figure 2.3.

2.5.2 P-type Controller

The heart of LEDBAT is the P-type congestion controller (see Section 2.4.2). On receipt of every ACK, the controller adjusts the congestion window to match the current queuing

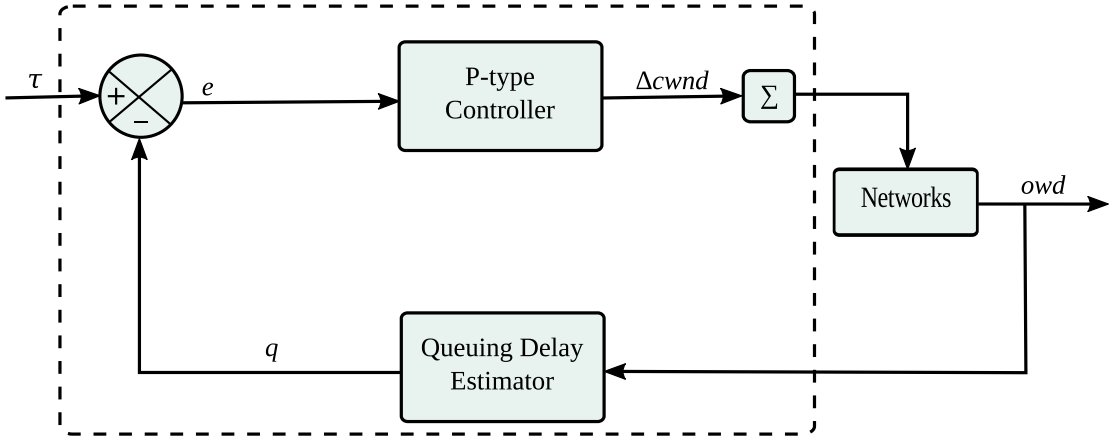


Figure 2.3: Block diagram of LEDBAT as a feedback control system.

delay to the target queuing delay τ. This target embodies the maximum queuing delay that a LEDBAT connection is allowed to introduce in the network.

For each ACK received at discrete time k, the controller uses as input the queuing delay error:

$$e(k) = \frac{\tau - q(k)}{\tau} \tag{2.2}$$

where $q(k)$ is the current queuing delay, estimated by the queuing delay estimator. The queuing delay error is positive when the current queuing delay is below the target. In this case, the controller needs to increase 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 controller must slow down its sending rate. Moreover, the size of the congestion window is modified proportionally to the queuing delay error to avoid oscillation. Consequently, the congestion window remains unchanged when the queuing delay error equals to zero. Following what we have discussed so far, the controller calculates as output the change of congestion window $\Delta cwnd(k)$

$$\Delta cwnd(k) = \frac{\gamma e(k)}{cwnd(k-1)} \tag{2.3}$$

where $\Delta cwnd(k)$ represents the pace at which the controller must increase or decrease the congestion window to match the queuing delay to the target queuing delay. The reactivity of LEDBAT to queuing delay variations is further adjusted by the gain γ. The bigger γ is, the faster LEDBAT's congestion controller increases or decreases its congestion window. Notice the resemblance of (2.3) to the P-term of (2.1). The size of

the congestion window $cwnd(k)$ is finally calculated by

$$cwnd(k) = cwnd(k-1) + \Delta cwnd(k) \tag{2.4}$$

Finally, to be TCP friendly, LEDBAT behaves like TCP in case of loss, that is, a LEDBAT sender halves its congestion window. The operation of the congestion controller can be summed up as follows:

$$cwnd(k+1) = \begin{cases} cwnd(k) + \dfrac{\gamma e(k)}{cwnd(k-1)} & \text{if no loss,} \\ \dfrac{1}{2} cwnd(k) & \text{if loss} \end{cases}$$

Choosing Target and Gain

RFC 6817 [10] recommends the target queuing delay to be fixed at 100 ms to make LEDBAT non-disturbing to delay-sensitive traffic such as voice. This recommendation is controversial because a non-compliant implementation could introduce the unfairness issue between LEDBAT flows, as studied in [4, 7].

To prevent LEDBAT from being too aggressive, the RFC states that γ must be lower than 1. Furthermore, we can distinguish two types of gain: increase gain γ_{icr} and decrease gain γ_{dcr} which are involved in the increase and decrease phases of the congestion window. In the increase phase of the congestion window (that is, $q(k) < \tau$), γ_{icr} must be set to 1 so that LEDBAT does not increase its congestion window faster than TCP. On the contrary, in the decrease phase of the congestion window (that is, $q(k) > \tau$), γ_{dcr} may not be limited to let LEDBAT raise the pace at which it decreases its congestion window when the queuing delay increases. Increasing γ_{dcr} allows LEDBAT to be more sensitive to the increase of queuing delay.

Finally, setting LEDBAT parameters is a tradeoff as a target too small could make LEDBAT sensitive to network conditions and a decrease gain too high could reduce significantly its performance. We will analyze this tradeoff in more detail in Chapter 3.

2.5.3 Queuing Delay Estimation

The **one-way delay (OWD)** or **end-to-end delay** is the time taken for a data packet to travel from one end system to another. This delay is composed by (see Figure 2.4):

Processing delay The amount of time it takes for a node to inspect the packet header and determine where to send the packet;

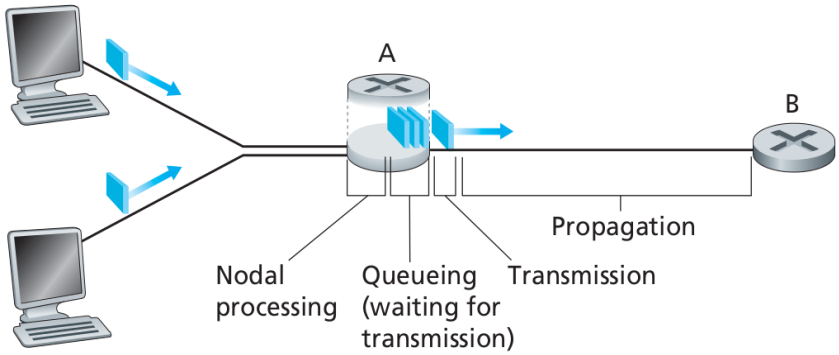


Figure 2.4: Components of a link delay.

Transmission delay The time required to transmit an entire packet onto the link;

Propagation delay The time needed for a bit of the packet to propagate from one node to the subsequent node;

Queuing delay The time in the queue for a packet waiting to be forwarded. If a packet arrives at an idle link, it is transmitted immediately onto the link and therefore suffers no queuing delay. On the contrary, if the link is busy transmitting data, the packet is placed into the queue. As long as packets arrive faster than the rate at which they are transmitted, the queue keeps filling up. Consequently, the queuing delay experienced by a newly arriving packet increases. The maximum queuing delay is proportional to the buffer size at a node.

While the other delays, except for some noise, are considered as constant, the queuing delay can vary from packet to packet. In consequence, we can assume that the base delay — the sum of the constant delay components — is the minimum observed on the end-to-end path. The queuing delay is then the difference between the current OWD and the estimated base delay.

Having discussed the underlying logic, we now detail the LEDBAT process of estimating the queuing delay. The sender places a timestamp from its local clock in each sending data packet. When a receiver receives the data packet, the receiver calculates the OWD as the difference between the timestamp from the receiver's local clock and the timestamp in the data packet. The receiver then feedbacks the computed OWD to the sender through an ACK. In addition, the sender maintains a sliding window of history of base delays. The objective of the sliding window is to discard old base delay values in order to cope with route changes.

An experienced reader will probably question about the problem of clock synchronization. However, LEDBAT bypasses this problem by using only the difference between the estimated OWD and the observed base delay. Since the estimated OWD and the observed base delay both suffer from the same clock bias, the queuing delay estimation process cancels any clock offset between the sender and the receiver as illustrated by Figure 2.5. Suppose that the sender and receiver clocks in Figure 2.5 are desynchronized by a constant offset Δ. At the beginning, the base delay D_{min} of the sender is set to infinity. After receiving the first data packet from the sender, the receiver feedbacks to the sender the OWD D_1 that equals to the difference between its local timestamp y_1 and the sender timestamp x_0 (see Figure 2.5). Upon receiving the ACK from the receiver, the sender updates its base delay D_{min} and calculates the current queuing delay q as follows:

$$\begin{aligned} D_1 &= y_1 - x_0 = x_1 - x_0 + \Delta \\ D_{min} &= min(D_{min}, D_1) = x_1 - x_0 + \Delta \\ q &= D_1 - D_{min} = 0 \end{aligned}$$

The same process is repeated for the second data packet from the sender. Suppose now that the queue is build up and thus, $D_2 > D_1$. Then, we have:

$$\begin{aligned} D_2 &= y_3 - x_2 = x_3 - x_2 + \Delta \\ D_{min} &= min(D_{min}, D_2) = x_1 - x_0 + \Delta \\ q &= D_2 - D_{min} = (x_3 - x_2 + \Delta) - (x_1 - x_0 + \Delta) \\ &= (x_3 - x_2) - (x_1 - x_0) \end{aligned}$$

We can observe that the clock bias Δ is automatically removed when computing the current queuing delay.

Another clock issue that can affect the OWD estimation is the clock skew, which is the difference between the rates of two clocks. However, unlike the synchronization problem, the clock skew is not eliminated in the computation of the queuing delay. To solve this issue, the RFC 6817 suggests some clock skew correction mechanisms.

2.6 Related Work on LEDBAT

Since it has been developed by BitTorrent, Inc. and later standardized by IETF, LEDBAT has attracted significant attention from the scientific research community. The

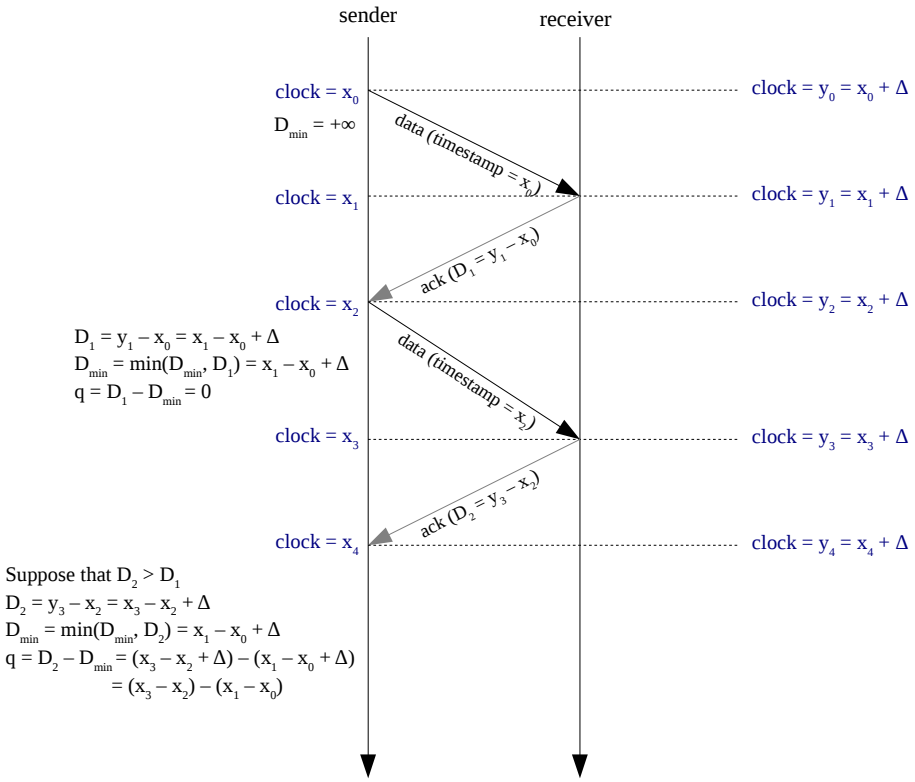


Figure 2.5: Queuing delay estimation process.

related work on LEDBAT can be classified into many categories as shown in Table 2.1. Moreover, each work can belong to various categories. However, to ease the discussion, in what follows, we only review each work once in a subsection which is most relevant to the work.

2.6.1 LBE Compliance and Issues of LEDBAT

Many studies on LEDBAT performance demonstrate that LEDBAT successfully achieves its design goals to be an LBE-compliant protocol. However, at the same time, these studies also reveal LEDBAT limitations.

Using their own LEDBAT implementation in ns-2, Rossi et al. [6] study the LEDBAT performance in the presence of TCP. They conclude that LEDBAT ensures compliance with the LBE principle. The authors indicate that in case of misconfiguration, LEDBAT behaves as standard TCP. Their results also point out the unfairness issue between LEDBAT flows. This intra-unfairness, called **latecomer advantage**, arises because

Subsection	Reference	Categories										
2.6.1	Rossi et al. [6]	✓		✓		✓						
	Rossi et al. [42]		✓	✓			✓					
	Andreica et al. [43]		✓	✓								
	Schneider et al. [7]		✓	✓		✓	✓					
	Ros and Welzl [44]	✓					✓					
	Carofiglio et al. [8]	✓				✓						
	Kühlewind and Fisches [5]	✓			✓	✓						
	Carofiglio et al. [9]	✓				✓						
	Komnios et al. [45]	✓		✓	✓							
2.6.2	Carofiglio et al. [4]	✓		✓	✓		✓	✓				
	Abu and Gordon [11]	✓						✓				
	Kuhn et al. [12]	✓		✓				✓				
2.6.3	Testa et Rossi. [46]	✓							✓			
	Testa et al. [47]		✓						✓			
	Testa et al. [48]		✓						✓			
	Sanhaji et al. [49]	✓								✓		
	Chirichella et al. [50]		✓								✓	
	Chirichella and Rossi [51]		✓								✓	
2.6.4	TCP Nice [52]	✓	✓									✓
	TCP-LP [53]	✓	✓									✓
	NF-TCP [54]	✓										✓
Approaches	**Simulation**											
	Experimental											
LBE Compliance												
Issues	**Aggressiveness**											
	Latecomer											
	Others											
Parameter Tuning												
Applications	**BitTorrent**											
	YouTube											
	Bufferbloat Measurement											
Other LBE protocols												

Table 2.1: Related work on LEDBAT

latecomer LEDBAT flows may sense different minimum one-way delays and thus can starve ongoing flows. However, this issue does not persist in the presence of TCP. Finally, the authors suggest a potential solution for the latecomer-advantage problem using the well-known slow-start algorithm.

In another work, Rossi et al. [42] investigate the evolution of the BitTorrent client, from TCP version to LEDBAT-over-UDP version. Experiments are conducted in a controlled testbed and over the Internet. Once again, they verify the promising LBE aspect of LEDBAT. The authors further point out that TCP traffic on the reverse (backward) path makes the queuing delay estimation of LEDBAT inexact. In this case, LEDBAT greatly underutilizes the link capacity of the forward path.

Along the same lines, Andreica et al. [43] confirm the non-intrusive property of LEDBAT in a real testbed using a user-level Python implementation. This study also shows that there exists an underutilization of the available bandwidth. However, this problem seems due to the computational overhead of their Python implementation.

Schneider et al. [7] implement their own version of the LEDBAT algorithm for testing with a variety of home gateways. Their results show that while LEDBAT fulfills its role as an LBE protocol, modern home gateways using sophisticated queuing schemes provide a much more effective LBE service than LEDBAT. In their work, the authors also analyze the latecomer-advantage issue of LEDBAT. They question and later condemn the use of slow-start as a solution for this issue. Additionally, their study exposes two other problems of LEDBAT: **intra-unfairness** of LEDBAT flows with **different target values** or **different RTT (RTT-unfairness)**. In fact, a LEDBAT flow having a higher target queuing delay or a lower RTT can starve another LEDBAT flow with a lower target or higher RTT, respectively. The authors finally point out that a home gateway with a queuing discipline enabled degrades the LEDBAT performance. As a consequence, LEDBAT is not LBE-compliant anymore.

Moreover, LEDBAT can greatly increase the network latency making its impact on the network no longer transparent, as revealed by Ros and Welzl [44] via simulation. Indeed, LEDBAT can suffer from the **base delay growth** problem. To cope with route changes, LEDBAT employs a sliding window of history of base delay estimations. According to the LEDBAT RFC, the size of this sliding window is 10 so that LEDBAT discards old base delay values every 10 min. At the end of a 10-min period, if the buffer never empties, the LEDBAT sender will take into account its own induced queuing delay as part of the new base delay. As a consequence, after every 10 min, the base delay raises by an amount that equals to the target queuing delay, which is 100 ms in the LEDBAT RFC. Furthermore, the authors show that LEDBAT flows increase the transfer time of

short-lived TCP flows.

Addressing the latecomer advantage

The LEDBAT latecomer unfairness is further studied by Carofiglio et al. in [8] by means of simulation. They propose different schemes to fix this latecomer problem including using slow start, random drops, probabilistic decrease, and in particular multiplicative decrease. In conclusion, the authors lean towards the multiplicative decrease scheme and argue that this scheme is the most appropriate solution.

Subsequently, Kühlewind and Fisches [5] review the multiplicative decrease scheme proposed in [8] and its variants employing a simulative approach. The results point out that multiplicative decrease schemes deteriorate the link utilization and may raise the completion time. Consequently, the authors argue that the default linear decrease scheme, which achieves a higher link utilization, is more suitable for LEDBAT as an LBE protocol. They also exhibit the **aggressiveness** of LEDBAT towards TCP, yet another severe LEDBAT issue. Finally, they advocate a higher decrease gain to solve this issue.

Carofiglio et al. [9] design **fLEDBAT**, a LEDBAT variant that aims to solve the latecomer unfairness issue. The algorithm of fLEDBAT is as follows:

$$cwnd(k+1) = \begin{cases} cwnd(k) + \dfrac{\gamma}{cwnd(k)} & \textbf{if no loss and } e(k) \leq 0, \\ cwnd(k) + \dfrac{\gamma}{cwnd(k)} + \dfrac{\zeta}{\tau} e(k) & \textbf{if no loss and } e(k) > 0, \\ \dfrac{1}{2} cwnd(k) & \textbf{if loss} \end{cases}$$

where γ still is the gain. The main difference with LEDBAT is that fLEDBAT uses a multiplicative decrease scheme with the decrease factor ζ when the estimated queuing delay is above the target queuing delay τ. The authors validate the efficiency and fairness of the new LBE protocol using a fluid model. They further evaluate the performance of fLEDBAT through ns-2 simulation. Although this protocol solves the latecomer advantage problem, fLEDBAT also adds another degree of freedom — the decrease factor ζ — besides the target τ and the gain γ which complicates the parameter tuning as shown by the sensitive analysis in [9].

Komnios et al. [45] evaluate the performance of LEDBAT and fLEDBAT in sub-packet regimes, where the per-flow throughput is less than one packet per RTT. More specifically, by means of simulation, the authors assess the feasibility of employing an

LBE protocol, such as LEDBAT, to help users of wireless community networks in developing regions efficiently share low-bandwidth backhaul Internet links. The results show that in the case where all flows use the same access method in sub-packet regimes, LEDBAT and fLEDBAT attain a higher link efficiency and fairness than TCP. However, fLEDBAT flows become aggressive towards TCP flows when they compete at same bottleneck link in sub-packet regimes. Although only the interaction between fLEBDAT and TCP is shown, the authors expect the same aggressive behavior in case of LEDBAT.

2.6.2 LEDBAT Parameter Tuning

The sensitive analysis of Carofiglio et al. [4] reveals the difficulty in tuning two key LEDBAT parameters — the target and gain. By means of simulation, the authors evaluate the impact of various settings of target and gain on the LEDBAT performance in two scenarios: inter-protocol (against TCP Reno) and intra-protocol (against LEDBAT itself). In the inter-protocol scenario, while LEBDAT is likely to provide an LBE service, LEDBAT also shows its aggressiveness towards TCP Reno in case of misconfiguration of the target. In the worst case, LEDBAT can even starve TCP Reno. On the other hand, the intra-protocol scenario shows that LEDBAT flows with a higher target will have advantage over other LEDBAT flows and can easily starve the latter. In both scenarios, tuning the gain has little impact on LEDBAT performance. In their work, Carofiglio et al. also give a comparison of LEDBAT with TCP-NICE [52] and TCP-LP [53], other LBE congestion control mechanisms. The result indicates that while generally being more LBE-compliant than the other two, LEDBAT suffers from RTT-unfairness, that is, the symptom in which the queuing delay of a small-RTT LEDBAT flow reaches its target first and thus starves another LEBDAT flow with higher RTT.

Along the same lines, Abu and Gordon [11] study the effect of different settings of gain on the LEDBAT performance. Their result shows that increasing the gain decreases the time taken for LEDBAT to reach the steady state. But a too high value of the gain causes oscillations of the LEDBAT congestion window in the steady state. The impact of the gain is obvious since this is an intrinsic property of the P-type controller, which is already well studied in the control theory. They propose and implement in ns-2 an algorithm for dynamically tuning the parameter gain of the LEDBAT controller.

In their work, Kuhn et al. [12] investigate the impact of different values of the target queuing delay on LEDBAT's performance over high bandwidth delay product (BDP) networks, in particular, over satellite networks. Their experiments are driven using an ns-2 extension named Cross-Layer InFormation Tool (CLIFT) [55]. This extension allows

ns-2 to simulate with real physical layer traces for more realistic results. In their study, the authors use satellite link traces provided by CNES[1]. They conclude that, in the context of high BDP networks, the target value of 5 ms is an optimal trade-off between the impact of the LBE traffic on the primary traffic and effectively exploiting the link capacity. They also suggest dynamically adapting the target queuing delay to different network conditions.

2.6.3 LEDBAT Applications

We now review in this section well-known applications of the LEDBAT protocol.

BitTorrent

BitTorrent is one of the most widespread peer-to-peer (P2P) file sharing protocol developed by BitTorrent, Inc. To send or receive files, a user requires a client program that implements the BitTorrent protocol. Since December 2008, LEDBAT has been implemented in µTorrent, a popular freeware and closed source BitTorrent client proprietary to BitTorrent, Inc. Although LEDBAT has been widely deployed within µTorrent, few studies have focused on the LEDBAT performance in P2P settings.

Testa et Rossi. [46] are the first that evaluate the impact of LEDBAT based on **download completion time**, the main BitTorrent user-centric metric, by means of ns-2 simulation. More specifically, they integrate the open-source BitTorrent module for ns-2 [56] with their own ns-2 implementation of LEDBAT to investigate two types of swarm population: **homogeneous and heterogeneous swarms**. In homogeneous swarms, all peers use either TCP or LEDBAT as their transport protocol. Conversely, in heterogeneous swarms, half of the population are TCP peers and the rest are LEDBAT peers. The simulation results reveal that LEDBAT peers experience a smaller download completion time in case of heterogeneous swarms. The authors explain that this time gain is a benefit of faster signaling traffic owing to lower uplink queuing delays introduced by LEDBAT peers. To the contrary, no performance difference between TCP and LEDBAT peers is observed in homogeneous swarms.

In their later work [47, 48], Testa et al. confirm the results of [46] using an experimental approach. The experiments in both studies are carried out over the Grid'5000 platform [57], a testbed for designing and evaluating large-scale distributed systems. In particular, the study in [48] focuses on the trade-off between the data plane throughput and the control plane delay on the BitTorrent download completion time. Besides

[1] Centre National d'Études Spatiales

LEDBAT employed in µTorrent, the authors also evaluate the impact of different congestion control algorithms (Cubic, New Reno, LP, Vegas and Nice) when used with either µTorrent or mainline BitTorrent clients. The experimental results verify and extend the simulation results in [46]. Indeed, a specific congestion control algorithm has little impact on the performance of BitTorrent clients in homogeneous swarms, while peers implementing LBE algorithms have smaller download completion time in heterogeneous swarms.

YouTube

Even though their study centers on Congestion Exposure (ConEx), an IETF mechanism by which senders inform the network about the congestion encountered, Sanhaji et al. [49] also analyze the effect of long-lived LEDBAT flows on the Quality of Experience (QoE) perceived by users of YouTube, a popular video-sharing website. By implementing the YouTube server and player models in ns-2, the authors show that LEDBAT, as the congestion control mechanism for long-lived flows, improves the QoE of YouTube users.

Bufferbloat Measurement

Chirichella et al. [50] develop an approach to infer the queuing delay experienced by remote LEDBAT hosts. They then demonstrate the reliability of their technique in a local testbed. Afterwards, Chirichella and Rossi [51] conducted an 8-month Internet experiment using aforementioned technique to assess the current level of bufferbloat encountered by BitTorrent users. Their results show that while LEDBAT mitigates the bufferbloat, LEDBAT does not solve this problem completely. Actually, some BitTorrent users still have a significant queuing delay more than 1 s.

2.6.4 Historical and Recent LBE Transport Protocols

TCP Nice

TCP Nice [52] upgrades the congestion control mechanism of TCP Vegas [33] to provide an LBE service. Since TCP Vegas is not designed to be an LBE protocol, its linear decrease scheme has great impact on standard TCP even though it can detect congestion early. To be LBE-compliant, TCP Nice adds a multiplicative decrease scheme into TCP Vegas and improves the congestion detector. Indeed, TCP Nice keeps a minimum RTT ($minRTT$) and a maximum RTT ($maxRTT$). A congestion signal is fired if within a

same RTT, the delay experienced by a fixed fraction of packets exceeds a threshold:

$$currentRTT > minRTT + (maxRTT - minRTT) \cdot threshold$$

TCP Nice reacts to early congestion signal by halving the congestion window. In case of packet loss, TCP Nice reacts like TCP NewReno. Otherwise, TCP Nice acts like TCP Vegas. Furthermore, TCP Nice has the ability to reduce the congestion window below one by sending a packet after several RTTs. The ns-2 simulations and real-life experiments [52] show that TCP Nice achieves its goal of being a low-priority protocol.

TCP-LP

Unlike LEDBAT and TCP Nice, TCP-LP [53] is a loss-based LBE protocol. To infer early congestion, TCP-LP maintains a minimum OWD ($minOWD$) and a maximum OWD ($maxOWD$). Then it computes an Exponentially Weighted Moving Average (EWMA) of measured OWDs with smoothing parameter α. Early congestion is detected if the EWMA of measured OWDs is greater than a threshold within the range of the minimum and maximum OWD:

$$ewmaOWD(k) = (1 - \alpha) \cdot ewmaOWD(k-1) + \alpha \cdot ewmaOWD(k)$$
$$ewmaOWD(k) > minOWD + (maxOWD - minOWD) \cdot threshold$$

In the presence of early congestion indication, TCP-LP halves its congestion window and enters an inference phase by initializing an inference timeout timer. During the inference phase, if there is no other early congestion indication, TCP-LP maintains its congestion window until the timeout. Otherwise, TCP-LP decreases its congestion window to one packet. After the inference phase, TCP-LP increases its congestion window by one per RTT. In case of loss, TCP-LP reacts like TCP New Reno. The ns-2 simulations and real-life experiments [53] show that TCP-LP can provide a potentially good LBE service.

NF-TCP

Network-Friendly TCP (NF-TCP) [54] uses a network-assisted approach to offer an LBE service. It employs Explicit Congestion Notification (ECN) [58] to make delay-insensitive flows yield to delay-sensitive flows under congestion. In order to operate, NF-TCP requires that at least one router along the path deploys a modified version

of RED [59] with ECN. In addition, NF-TCP integrates a novel bandwidth estimation mechanism, called ProECN, to quickly exploit any spare link capacity. While simulation results show that NF-TCP is a promising LBE protocol, its deployment is challenged by the need for the extra support from routers.

2.7 Conclusion

An extensive review on the state of art demonstrates the appeal of LEDBAT in providing the LBE service. However, the review also points out some troublesome flaws of LEDBAT, especially the aggressiveness and latecomer unfairness issues. Most importantly, we observe that while most proposed solutions concentrate on the latecomer unfairness issue, there is only one hint for the aggressiveness issue. We stress the severity of this research gap because, from our point of view, the aggressiveness of LEDBAT can seriously harm other important traffic and thus is more serious than the latecomer issue. Indeed, the latecomer issue only affects low-priority LEDBAT flows and is solved in the presence of TCP flows.

In light of the gravity and the lack of a solution for the aggressiveness issue, we propose an in-depth study of this issue in the next chapter. And then in Chapter 4, based on the insight gained from our study, we design a new LBE protocol aiming to solve both the aggressiveness and latecomer issues by employing fuzzy control theory.

3 LEDBAT Analysis

A misconfiguration can make LEDBAT more aggressive than TCP. Therefore, this chapter explores the set of optimal parameters allowing LEDBAT to effectively perform as an LBE protocol. We first present the motivation of this study in Section 3.1. Then, using ns-2, we evaluate in Section 3.2 the impact of different buffer sizes on the behavior of LEDBAT, in the presence of TCP NewReno connections. In Section 3.3, we assess whether there exists a combination of target and decrease gain values that let LEDBAT have an LBE behavior in different network configurations. Next, in Section 3.4, we evaluate the performance of the optimal combination. Section 3.5 tests the optimal combination of target and decrease gain of LEDBAT in a highly loaded network. We finally conclude this chapter in Section 3.6.

3.1 Motivation

As previously described in Chapter 2, LEDBAT is mainly defined by two parameters, namely target queuing delay and gain, that strongly impact LEDBAT behavior in terms of fairness with other protocols. Furthermore, there are two types of gain: increase gain and decrease gain which are used in the increase and decrease phases of controlling the congestion window. RFC 6817 [10] provides guidelines to configure both parameters. Specifically, it recommends a default target of 100 ms, while the default settings for both types of gain are 1. However, these guidelines are questioned by several studies as they may lead to the generation of non-LBE traffic. Indeed, the tuning of LEDBAT is difficult and highly depends on network conditions. As an illustration, the authors of [12] conclude that the parameter target of LEDBAT should not be higher than 5 ms in a high BDP network. In addition, the authors of [44] show that LEDBAT can greatly

increase the network latency, making its impact on the network no longer transparent.

Consider:

1. LEDBAT is currently deployed over BitTorrent network;
2. The misconfiguration of LEDBAT can make other TCP flows back off and significantly increase the network latency;
3. LEDBAT may exacerbate the bufferbloat issue [36];
4. Several projects attempt to reduce the Internet latency, such as RITE [60] or the Workshop on Reducing Internet Latency [61].

We believe that a deep study on the impact of LEDBAT internal parameters would allow to assess a right set of parameter settings to reduce its potential negative effect on the network. In particular, we question the optimal setting of two LEDBAT key parameters, the target and the gain, considering various buffer sizes. We also study the impact of LEDBAT on TCP in a highly loaded network. The main objective of such study is **to close the discussion** about the feasibility of integrating LEDBAT on a large scale, **before diving into possible improvements** of its algorithm such as proposed in [5].

3.2 LEDBAT Behavior

In this section, we evaluate by means of simulation the impact of different buffer sizes on the behavior of LEDBAT in the presence of a TCP NewReno connection (hereinafter simply denoted TCP).

3.2.1 Simulation Setup

Simulations have been performed using the network simulator ns-2. We use the LEDBAT module developed by Valenti et al. [6] based on RFC 6817.

As the reference network topology, we use a dumbbell topology where a TCP flow and a LEDBAT flow share a single bottleneck link. Both LEDBAT and TCP sources send packets size of $P = 1500\,\mathrm{B}$. The bottleneck link has a capacity set to $C = 10\,\mathrm{Mb/s}$ and a one-way propagation delay $d = 50\,\mathrm{ms}$. The bottleneck router uses a FIFO drop-tail queue with a size of B packets. For convenience, we explore the set of bottleneck buffers B being a ratio to the bandwidth-delay product (BDP) in terms of packets. Hence, we

have:

$$B = \lceil n \cdot BDP \rceil = \left\lceil \frac{n \cdot C \cdot 2d}{8P} \right\rceil \tag{3.1}$$

where the ratio $n \in [0.1, 0.2, \ldots, 1.9, 2.0]$ 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 explore the set of target queuing delays $\tau \in [25, 50, 100]$ ms and the set of decrease gains $\gamma_{dcr} \in [1, 10]$. We also express B as a ratio to the target τ in terms of packets as with the BDP:

$$B : \tau = \frac{B \cdot 8P}{\tau \cdot C} \tag{3.2}$$

To analyze the behavior of LEDBAT with respect to different buffer sizes, we consider the LEDBAT congestion window, the queue length of the bottleneck buffer, and the link utilization η of LEDBAT and TCP flows.

We consider only long-lived TCP and LEDBAT flows. Every simulation lasts 200 seconds, and the link utilization is computed over the last 150 seconds of the simulation. We denote by t_X the time at which the flow using protocol X starts to transmit data. The following scenarios are simulated:

- Scenario A: $t_{\text{LEDBAT}} = t_{\text{TCP}} = 0\,\text{s}$;
- Scenario B: $t_{\text{LEDBAT}} = 0\,\text{s}$, $t_{\text{TCP}} = 20\,\text{s}$;
- Scenario C: $t_{\text{LEDBAT}} = 20\,\text{s}$, $t_{\text{TCP}} = 0\,\text{s}$.

In the following, we present only the scenario A as similar results are observed for the scenarios B and C.

3.2.2 Endangered TCP — LEDBAT Misconfiguration

This subsection presents a misconfiguration leading to a full utilization of the link capacity by LEDBAT flows whatever the TCP load. Similar situations are also reported by other authors [4, 5]. In this work, we take a closer look at this problem and its relationship with buffer sizing. Figure 3.1 and 3.2 show the time evolution of LEDBAT and TCP congestion windows as well as the queue length, in cases where LEDBAT does not fulfill its primary objectives. The target is $\tau = 100\,\text{ms}$ ($\simeq 83.3$ packets), and the decrease gain is $\gamma_{dcr} = 1$.

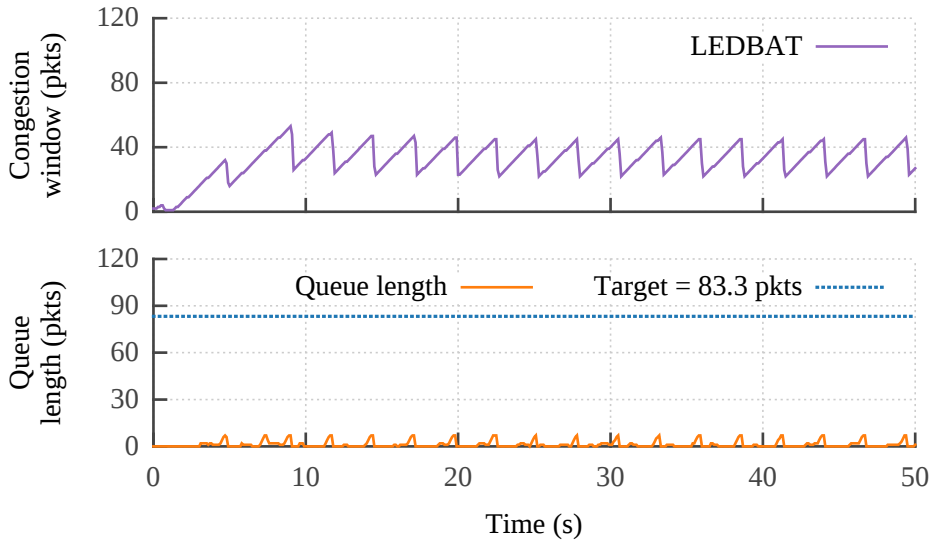
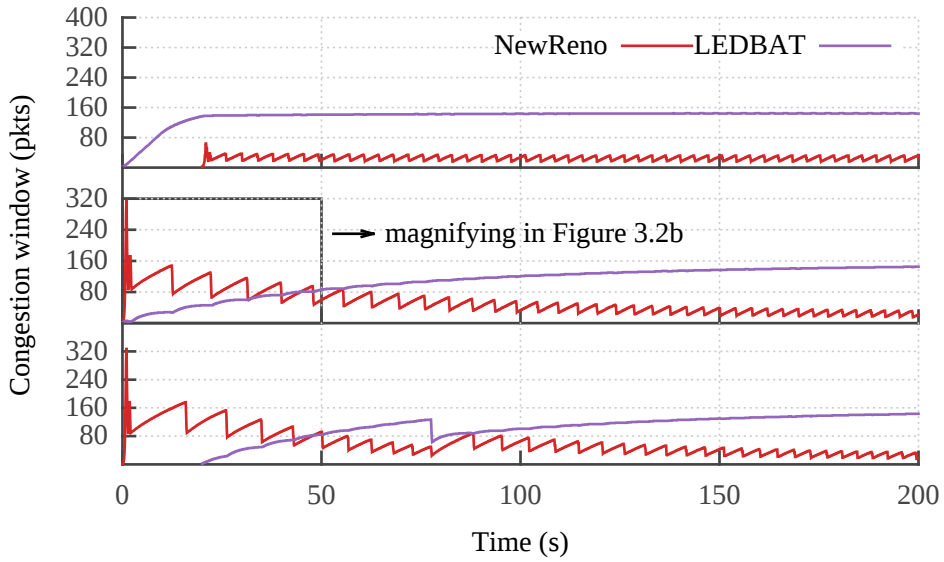


Figure 3.1: Loss-based behavior of LEDBAT.

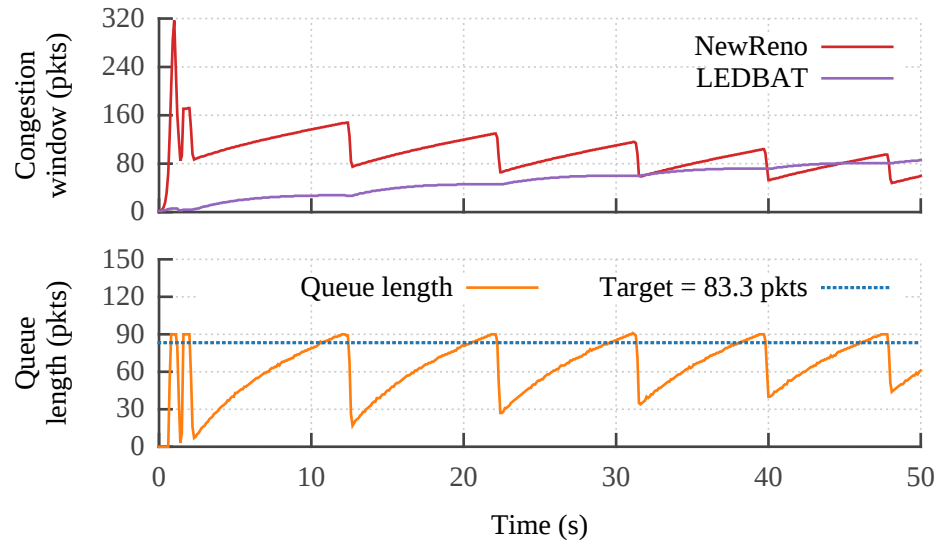
RFC 6817 [10] states that, if a compromised target is set to infinity, "the algorithm is fundamentally limited in the worst case to be as aggressive as standard TCP". Actually, it corresponds to the case where the buffer size is too small in comparison to the target, as shown in Figure 3.1. In this case, the buffer size is 9 packets, so the ratio of the buffer size to the target is 0.1. 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 RFC 6817 worst case" in which hostile LEDBAT makes TCP back off. Figure 3.2a illustrates three hostile scenarios with a same network configuration but with a buffer size of 92 packets. Even in an unfavorable situation for LEDBAT, when it starts after TCP, LEDBAT succeeds to increase its congestion window more than TCP. Whatever the scenario presented in this subsection, TCP cannot compete against LEDBAT.

To understand this hostile behavior of LEDBAT, we inspect the scenario where two flows TCP and LEDBAT start at the same time (middle plot in Figure 3.2a). In Figure 3.2b, we magnify the results of the first 50 seconds of the simulation. In the slow-start phase, TCP exponentially increases its congestion window. As a consequence, the buffer fills up immediately and then limits the LEDBAT congestion window to one packet. After the slow-start phase, from $t = 3\,\mathrm{s}$ to $t = 5\,\mathrm{s}$, as the queuing delay is small compared to the target, LEDBAT and TCP congestion windows conjointly grow at the



(a)



(b)

Figure 3.2: Aggressive behavior of LEDBAT towards TCP

same speed. As the queue continues to increase, LEDBAT reduces the increasing speed of its congestion window. Meanwhile, TCP continues to linearly increase its congestion window. After $t = 11$ s, when the queuing delay is higher than the target, LEDBAT decreases slowly and slightly its congestion window. Although the buffer size is bigger,

it remains relatively small to the target. The ratio of the target to the buffer size is 1.1 in this case. Thus, LEDBAT does not have enough time to react to queuing delay before TCP causes a buffer overflow at $t = 15$ s. 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.

3.2.3 LEDBAT Working Regions

The previous subsection illustrates cases where LEDBAT is too aggressive. In this subsection, we identify different working regions of LEDBAT. We denote as "working regions" areas where LEDBAT behaves correctly as an LBE protocol. For each combination of target and decrease gain, we plot the link utilization of TCP flow and the total link utilization of two flows against the buffer size.

Figure 3.3 presents only the simulation results from the scenario where two TCP and LEDBAT flows start at the same time (Scenario A). We do not present the results of Scenario B and C as similar behavior is observed. Each plot corresponds to a combination of target and decrease gain. From top to bottom, the target slides from 100 ms to 25 ms. The decrease gain is set to 1 in the first column and 10 in the second column. The first (bottom) x-axis is the ratio of the buffer size to the BDP. The second (top) x-axis represents the same buffer size but as the ratio to the target. The y-axis is the link utilization.

Impact of the Target

Fixing the decrease gain to 1, we study the impact of the target setting on the behavior of LEDBAT. As shown in Figure 3.3, we distinguish three operating regions of LEDBAT. First, in the leftmost region, LEDBAT behaves like TCP. In other words, as previously explained in Subsection 3.2.2, LEDBAT always increases its sending rate until a loss event is reported. In this case, the buffer size is small in comparison to the target. Hence, TCP and LEDBAT fairly share the link capacity ($\eta_{\text{TCP}} \simeq \eta_{\text{LEDBAT}}$). Then, in the middle region, LEDBAT becomes hostile to TCP. In consequence, the link utilization of TCP reduces progressively to a certain point ($\eta_{\text{TCP}} \ll \eta_{\text{LEDBAT}}$). As the buffer size still increases, the aggressiveness of LEDBAT decreases. Finally, in the rightmost region, LEDBAT works correctly as a LBE protocol ($\eta_{\text{TCP}} \gg \eta_{\text{LEDBAT}}$). To have enough time to react to queuing delay variations, we measure that LEDBAT needs a large buffer size in comparison to its target (more than 1.5 times).

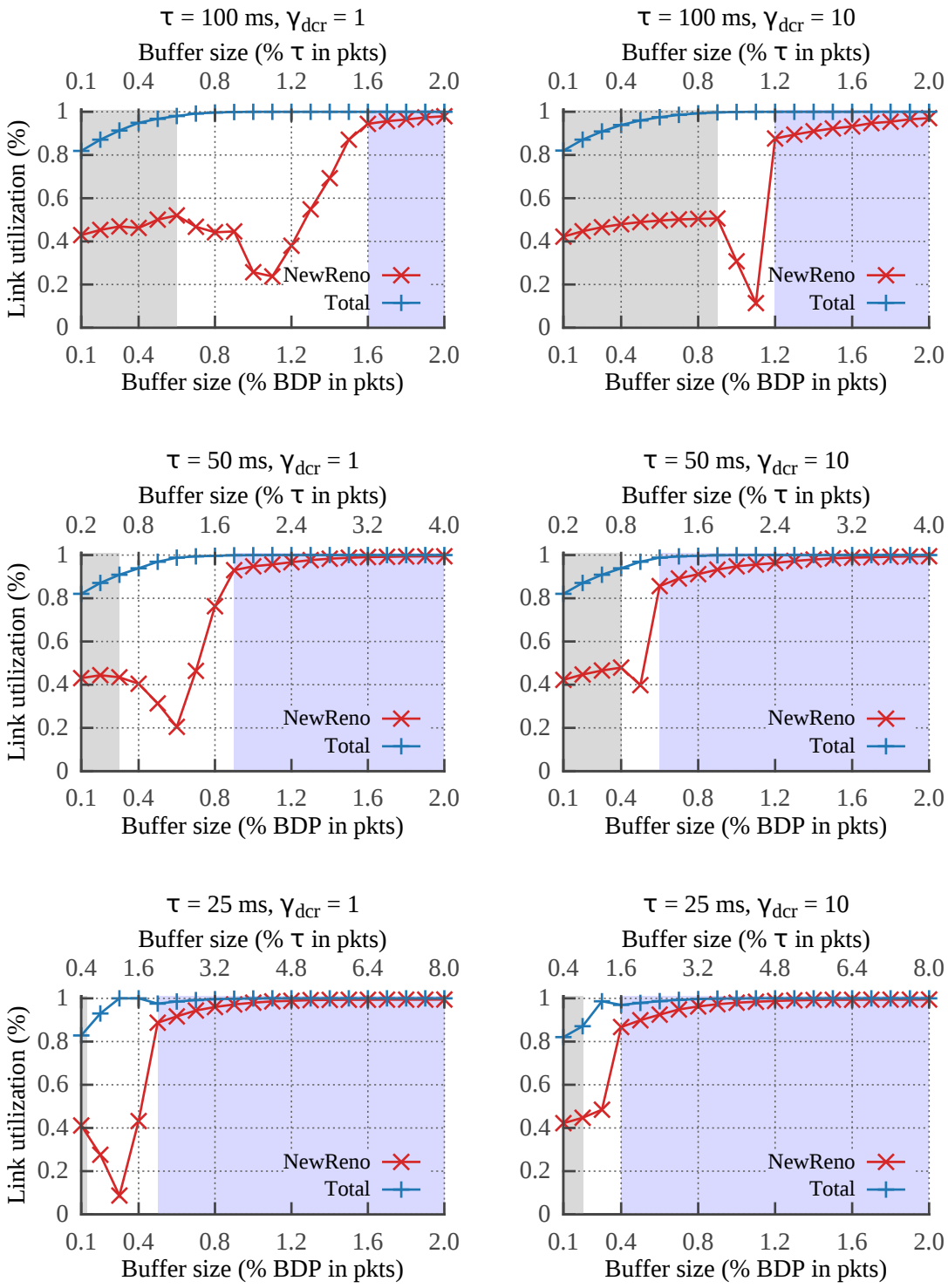


Figure 3.3: Link utilization in terms of buffer size

Considering now the buffer size in terms of the ratio to the BDP, we see that a target of 100 ms requires a buffer size as large as 1.6 times the BDP in order to perform in low-priority mode. This excess need of buffer introduces **an excessive queuing delay** known as the **bufferbloat phenomenon** [36]. A smaller target mitigates this issue. As seen in Figure 3.3, the smaller the target, the smaller is the size of the buffer.

Impact of the Decrease Gain

We now study the impact of the decrease gain setting on LEDBAT behavior. Figure 3.3 shows that, for each target, increasing the decrease gain shrinks the hostile region from both sides. As a consequence, the LBE region expands to the left, meaning a reduction in the need for buffer size of LEDBAT for low-priority mode. However, a higher decrease gain also amplifies the loss-based region.

As a conclusion, to increase the LBE working region of LEDBAT, we can reduce the target or raise the decrease gain separately or at the same time. We also highlight that setting LEDBAT parameters is a tradeoff as a target too small could make LEDBAT sensible to network conditions and a decrease gain too high could reduce significantly its performance.

3.3 Optimal Combination of Target and Decrease Gain

In this section, we assess if there exists a combination of target and decrease gain allowing LEDBAT to behave as an LBE protocol regardless of the network configuration.

3.3.1 Network Configuration

We use the same dumbbell topology as in Section 3.2. We consider different capacities $C \in [1, 5, 10, 20, 50]$ Mb/s and different one-way propagation delays $d \in [10, 50, 100, 150, 200, 250]$ ms of the bottleneck link. As in Section 3.2, we express the bottleneck buffer B as the ratio to the BDP. For each network setting C, d, and B, we explore the set of target queuing delays $\tau \in [5, 25, 50, 75, 100]$ ms and the set of decrease gains $\gamma_{dcr} \in [1, 10]$. We run about 60 simulations for each setting of target and decrease gain. We drove fewer simulations for small BDP networks as we exclude all cases with duplicate values of B caused by the ceiling function and where B is equal to 1.

In each simulation, a LEDBAT flow starts at $t = 0$ s and then a TCP flow starts later at $t = 200$ s. The simulation lasts 1000 seconds. We calculate the link utilization η of each flow over the last 500 seconds.

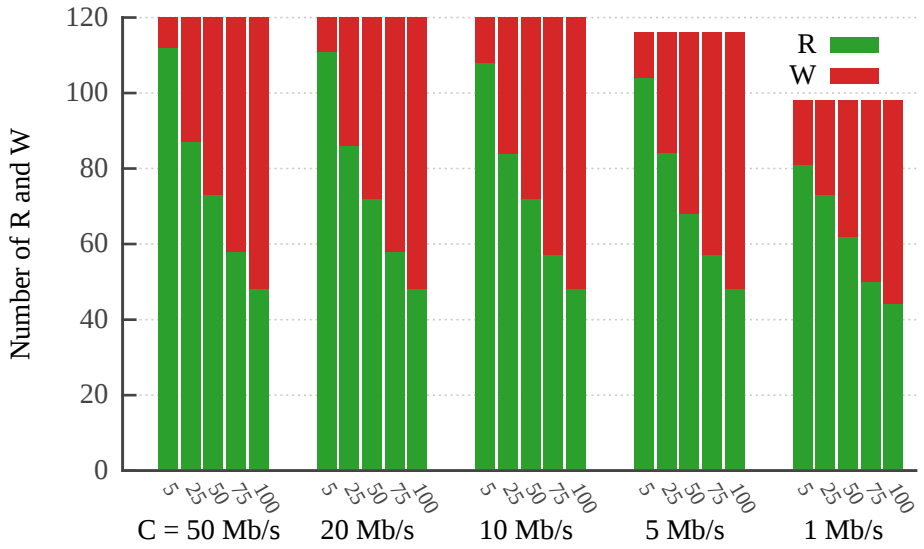


Figure 3.4: Number of R and W cases for each target setting

We use the following clustering method to classify our data and represent the set of working combinations. For each combination of target and decrease gain, when $B = BDP$, if the link utilization of TCP flow $\eta_{\text{TCP}} \geq 0.8$, then we choose the link utilization of LEDBAT flow as reference $\eta_{\text{ref}} = \eta_{\text{LEDBAT}}$. Then, for every other value of B, we calculate $\Delta = |\eta_{\text{LEDBAT}} - \eta_{\text{ref}}|$. If $\Delta \leq \varepsilon$ and $\eta_{\text{TCP}} \geq 0.8$, then we classify B in the cluster "Right" (denoted R), otherwise, in the cluster "Wrong" (denoted W). In our study, we use $\varepsilon = 0.15$. So, for different buffer sizes, if η_{TCP} is always ≥ 0.8 and the difference between η_{LEDBAT} and η_{ref} is within a limit of 15%, then we say that the combination of target and decrease gain works well for these buffer sizes.

Finally, we use statistical analysis to find which combination of target and decrease gain works well in most network configurations.

3.3.2 Choosing the Target

We begin our analysis by finding the optimal target value. In Figure 3.4, using histogram, we group the simulation results into different categories of network capacities and then, into subclasses of target values. For every target value, the stacked column represents the number of R and W cases. The column height corresponds to the total simulations including all decrease gain settings for a target value. For each setting of target, we have 120 simulations for high capacity networks and fewer simulations in case of small capacity networks.

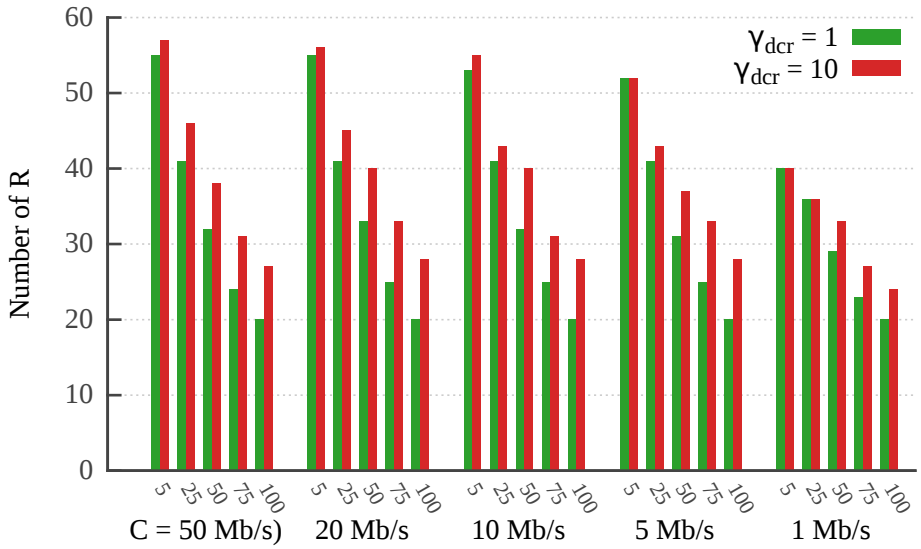


Figure 3.5: Number of R cases when $\gamma_{dcr} = 1$ and $\gamma_{dcr} = 10$ for each target setting

In Figure 3.4, we observe that the number of R cases of a target decreases as the network capacity decreases. Actually, the buffer size, expressed in ratio to the BDP, reduces when the network capacity is smaller. Therefore, the ratio of the buffer size to the target also becomes smaller. In consequence, LEDBAT behavior falls back into either hostile region or loss-based region.

As expected, Figure 3.4 shows that a target value of 5 ms works in most cases. These experiments allow us to conclude that setting the target to 5 ms is optimal.

3.3.3 Choosing the Decrease Gain and the Optimal Combination of LEDBAT Parameters

Having the optimal target value, we still use the histogram representation to find the optimal decrease gain. For each target setting, we present the number of R cases for each corresponding decrease gain value. As shown in Figure 3.5, setting the decrease gain to 10 is better as it increases the number of R cases.

3.4 Testing the Optimal Combination

3.4.1 Impact on TCP NewReno

In this subsection, we now evaluate the impact of the optimal combination of target and decrease gain on TCP using the network configuration in Section 3.2. For reference purpose, we also simulate the scenario where TCP is the only protocol in the network. In Table 3.1, we report results of two cases where $B = \text{BDP} = 84$ packets and $B = \frac{\text{BDP}}{2} = 42$ packets. Table 3.1 also shows the gain (in green) or loss (in red) of the link utilization of TCP and of the whole. For comparison, we are only interested in combinations of parameters that work in low-priority mode with either buffer sizes, that is, $(25\,\text{ms}; 1)$ and $(25\,\text{ms}; 10)$ (see Figure 3.3 in Section 3.2).

Buffer (pkts)	**Combination** $(\tau; \gamma_{\text{dcr}})$	η_{NewReno} $(\%\, C)$	η_{LEDBAT} $(\%\, C)$	η_{Total} $(\%\, C)$
84 (= BDP)	Only NewReno	99.99	–	99.99
	$(25\,\text{ms}; 1)$	98.04 (-1.95%)	1.95	99.99 $(+0\%)$
	$(25\,\text{ms}; 10)$	98.02 (-1.97%)	1.97	99.99 $(+0\%)$
	$(5\,\text{ms}; 10)$	99.13 (-0.86%)	0.86	99.99 $(+0\%)$
42 $(= \frac{\text{BDP}}{2})$	Only NewReno	96.04	–	96.04
	$(25\,\text{ms}; 1)$	88.77 (-7.57%)	8.82	97.59 $(+1.61\%)$
	$(25\,\text{ms}; 10)$	89.87 (-6.42%)	8.02	97.89 $(+1.93\%)$
	$(5\,\text{ms}; 10)$	94.21 (-1.91%)	3.53	97.74 $(+1.77\%)$

Table 3.1: Optimal LEDBAT sharing bandwidth with TCP NewReno

At first glance, Table 3.1 shows that introducing a LEDBAT flow always takes a portion of network capacity achieved by TCP when TCP is the only transport protocol on the network. When the buffer size equals to BDP, TCP alone exploits almost full capacity of the network (99.99%). LEDBAT employing the optimal combination of parameters introduces least impact on the TCP performance (-0.86%) while keeping the total throughput same as in the case of TCP alone. However, in this case, LEBDAT also obtains the lowest throughput (0.86%).

On the other hand, when the buffer size equals to $\frac{\text{BDP}}{2}$, TCP alone obtains less network capacity (96.04%). But in this case, introducing LEDBAT using the optimal combination of parameters increases the total network capacity exploited ($+1.77\%$) while causing least impact on TCP performance (-1.91%). The combination $(25\,\text{ms}; 10)$ allows to obtain a better network capacity ($+1.93\%$), but also makes more impact on TCP in the same time (-6.42%).

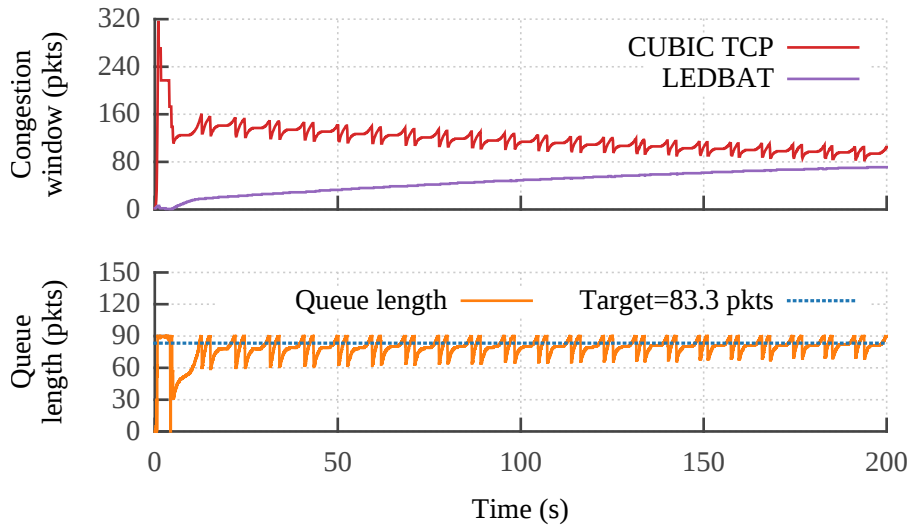


Figure 3.6: Aggressive behavior of LEDBAT towards CUBIC TCP

The results from the two showcases above indicate that LEDBAT traffic always has some impact on TCP traffic. The same observation are given by the authors of [12]. Finally, we conclude that the optimal combination of target and decrease gain is $(5\,ms; 10)$. This result is consistent with our discussion in Section 3.2.

3.4.2 LEDBAT Behavior in the Presence of CUBIC

To deal with the aggressiveness of LEDBAT, one may attempt to use a more aggressive TCP variant such as CUBIC, hoping that it will behave better than standard TCP in the hostile region. Unfortunately, in this case, it takes longer for LEDBAT to get a larger share of link capacity, but CUBIC will totally yield to LEDBAT in the end. Figure 3.6 exhibits the aggressiveness of LEDBAT towards CUBIC TCP. In this example, we use the same network configuration in Section 3.2 with a buffer size of 92 packets.

3.4.3 Global Optimal Combination Is Not Local Optimal Combination

The combination of target and decrease gain $(5\,ms; 10)$ is globally optimal, that is, optimal over a wide range of network configurations. However, it is not necessarily a local optimal combination for a specific network configuration. Moreover, the local optimal combination depends on the variant of TCP used. To demonstrate it, in this section, we heuristically find the local optimal combinations for the network configuration

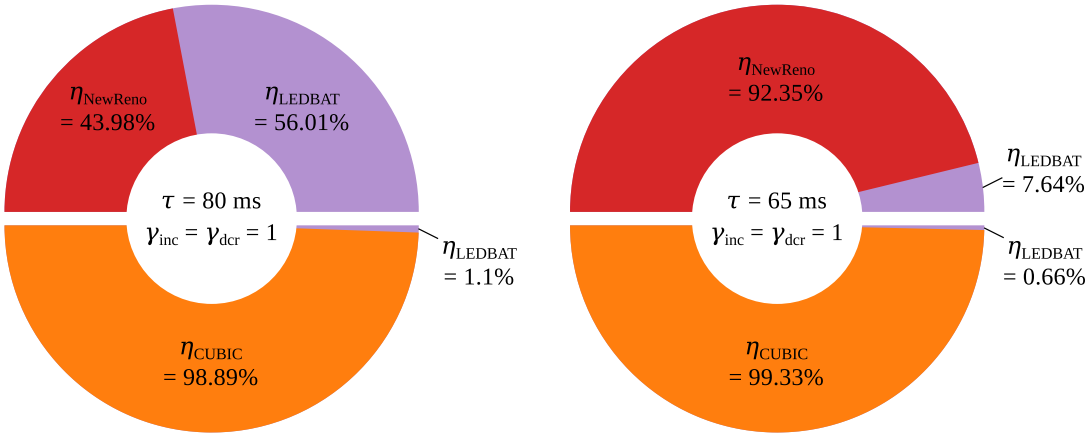


Figure 3.7: Throughputs of TCP variants and LEDBAT with different local optimal combinations of parameters.

in Section 3.2 with a buffer size of 84 packets, which equals to the BDP. We roughly define a local optimal as a combination of parameters in which the gain is fixed to one and the target is the maximum target queuing delay that still keeps LEDBAT in its working region. The optimal combinations are $(65\,ms; 1)$ for LEDBAT in the presence of standard TCP and $(80\,ms; 1)$ for LEDBAT in the presence of CUBIC.

Figure 3.7 shows the difficulty of tuning LEDBAT parameters. Indeed, the local optimal combination of LEDBAT parameters in the presence of CUBIC $(80\,ms; 1)$ is clearly not the local optimal one in the presence of standard TCP because CUBIC is more aggressive than standard TCP. And the local optimal combination of LEDBAT parameters in the presence of standard TCP $(65\,ms; 1)$ degrades LEDBAT performance in the presence of CUBIC.

In Table 3.2, we compute the gain and loss of throughputs when passing from the global optimal combination $(5\,ms; 10)$ to the local optimal combinations. We can easily observe that LEDBAT greatly increases its throughputs while making little more impact on TCP throughputs.

The results in this section demonstrate that the tuning of LEDBAT is very difficult and depends on different conditions. Moreover, even with the global optimal combination of parameters $(5\,\text{ms}; 10)$, the latecomer unfairness problem of LEDBAT still persists, as illustrated in Figure 3.8.

	Combination $(\tau; \gamma_{\text{dcr}})$	η_{TCP} $(\% C)$	η_{LEDBAT} $(\% C)$
NewReno	(5 ms; 10)	99.13	0.86
	(65 ms; 1)	92.35	7.64
	Gain/Loss	−6.84%	+778.37%
CUBIC	(5 ms; 10)	99.34	0.65
	(80 ms; 1)	98.89	1.10
	Gain/Loss	−0.45%	+69.23%

Table 3.2: Optimal combinations for a specific network configuration.

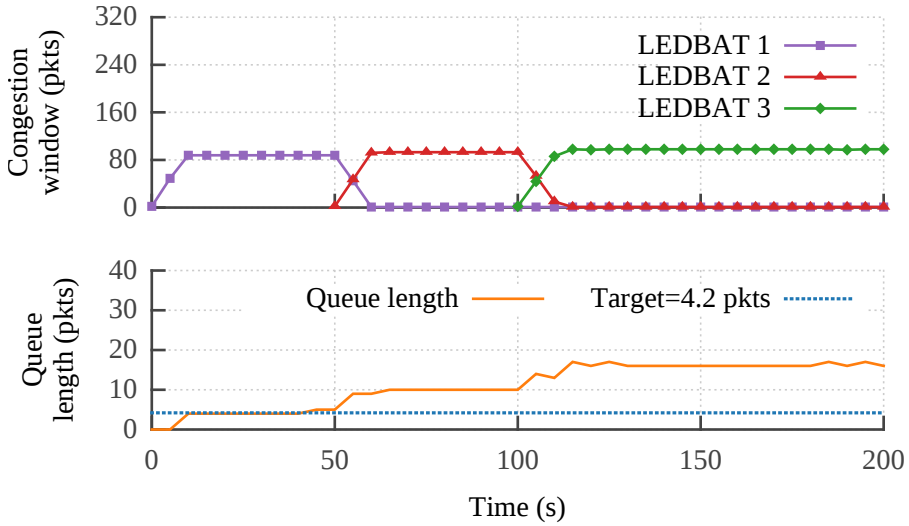


Figure 3.8: Latecomer unfairness of LEDBAT arises even with the global optimal combination of parameters (5 ms; 10).

3.5 LEDBAT in Highly Loaded Networks

In this section, we test the optimal combination of target and gain of LEDBAT found in Section 3.3 in a highly loaded network. We also take this opportunity to evaluate other LBE protocols such as TCP Nice [52] and TCP-LP [53]. TCP-LP implementation is available in ns-2 TCP-Linux. For TCP Nice, we use the module proposed in [4].

As in Section 3.2, we use a dumbbell topology with a bottleneck capacity $C = 10\,\text{Mb/s}$ and a one-way propagation delay $d = 50\,\text{ms}$. The buffer size is set to 84 packets and equals to the BDP, according to the classical "rule-of-thumb". In each simulation,

we consider N LBE sources and N TCP sources, where $N \in [1, 10, 20, 40, 60, 80, 100]$. The starting time of a TCP flow is uniformly distributed between 0 and 200 seconds. To assess the impact of LBE protocols on TCP, LBE flows begin to transfer data in an unfavorable situation, that is, only after all TCP flows have started. Therefore, the starting times of the LBE flows are uniformly distributed between 300 and 500 seconds. It should be noticed that the latecomers problem of LEDBAT is negligible when there is interaction between LEDBAT and TCP [6]. Thus, in our study, we introduce LEDBAT at random time neglecting any effects on our results. We also simulate only N TCP flows without other competing LBE flows as a baseline.

As performance metrics, apart from the aggregate average throughput, we also define the TCP bandwidth released ($TCP_{released}$) as the link utilization of TCP when it shares the bottleneck link with the tested LBE protocol over the link utilization of TCP when it is the only protocol on the same link:

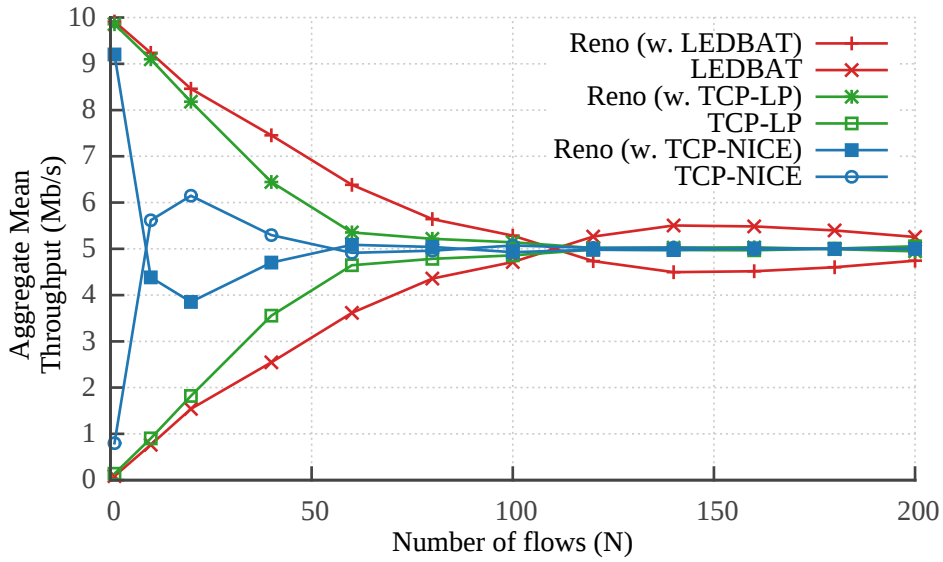
$$TCP_{released} = \frac{\eta_{(\text{TCP+LEDBAT})}}{\eta_{\text{TCP}}} \tag{3.3}$$

Every simulation lasts 1200 seconds, and all metrics are measured over the last 600 seconds.

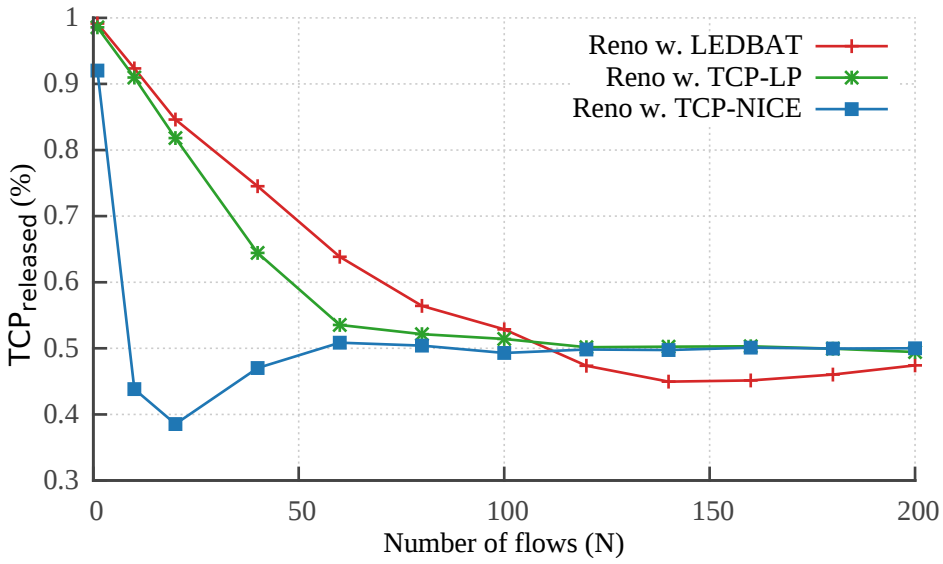
3.5.1 Results and Discussions

Figure 3.9a presents the aggregate average throughput of N TCP flows and N LBE flows sharing the bottleneck link. To stress the impact of LBE flows, we plot in Figure 3.9b the $TCP_{released}$ depending on the network load, that is, the direct impact of the introduction of LBE flows on TCP performance. As seen in Figure 3.9, when the number of TCP and LBE flows is small, LBE protocols fulfill their role. However, increasing the number of LBE flows also increases their impact on TCP traffic, even though an equivalent number of TCP flows are also introduced into the network. The more we introduce LBE flows, the less capacity is left for TCP traffic. For instance, 100 LEDBAT flows compete as aggressive as 100 TCP flows for the full capacity. In this case, TCP flows must yield to LEDBAT flows 47 % of the bandwidth obtained when they are alone in the network.

When the number of flows is above 100 (Figure 3.9a), LEDBAT is even more aggressive. This is explained by the misestimation of the OWD. Indeed, LEDBAT keeps track of the minimum One-Way Delay D_{min} to estimate the queuing delay variation. Thus, if LEDBAT runs long enough and the queue is never empty, D_{min} is always nearly equal to D_{ack} and to the maximum queuing delay allowed by the bottleneck buffer. As a result, the queuing delay sensed by LEDBAT always approximates to zero ($D_{ack} - D_{min} \simeq 0$).



(a)



(b)

Figure 3.9: Impact of N LBE flows on N TCP flows

In consequence, LEDBAT tends to raise its congestion window to reach the target queuing delay. This limitation of LEDBAT algorithm is studied in detail in [44]. As shown in Figure 3.9, if we do not limit the number of LBE flows, an aggregate of LBE flows do not behave as a LBE traffic anymore.

To better understand this problem, we measure the average congestion window of a LEDBAT flow, in the case where the number of flows of each protocol TCP and LEDBAT is 100. As a result, the average congestion window of a LEDBAT flow is 1.3 packet. Thus, a LEDBAT flow sends data at an average rate of 1.3 packet per RTT. For 100 LEDBAT flows, the router always receives on average a burst of 130 packets. An exponential backoff mechanism, as proposed in RFC 6817 [10], could help to reduce the burst size at a given time. Note that this mechanism is already implemented in the LEDBAT module used in our simulations. Unfortunately, exponential backoff does not resolve this issue in the long term as we must impose a maximum backoff time. Although the RFC 6817 states that a maximum value MAY be placed on backoff time, we argue that this maximum value MUST be always set. Supposing that a LEDBAT flow can backoff in unlimited time, the impact of LEDBAT flows on long-lived TCP traffic will then reduce significantly. But this could lead to a case where long-lived TCP flows cut off LEDBAT traffic in long periods of time and make LEDBAT useless. Moreover, unlimited backoff time could raise a potential intra-unfairness of LEDBAT flows as a side effect.

Our results in this section indicate that when LEDBAT is deployed at large scales, we must use additional strategies to limit the number of LEDBAT flows.

3.6 Conclusion

Recent studies point out cases that question the ability of LEDBAT to carry out Less-than-Best-Effort service. Misconfiguration of this protocol may result in a significant delay in the network. This motivated us to perform a deeper analysis of the impact of LEDBAT's internal parameters on its performance.

In this chapter, we propose a global optimal parametrization for the internal parameters of LEDBAT. We found that a target of 5 ms and a decrease gain of 10 are globally optimal. The optimal target is far from the guideline in the RFC, which says that the target must be lower than 100 ms. However, it is worth pointing out that in all simulation cases, LEDBAT is not fully LBE and borrows some capacity of the primary flows. We also measure the alarming impact of the increasing number of LEDBAT flows. We further demonstrate that the global optimal combination is not a local optimal combination. Moreover, the latecomer unfairness issue of LEDBAT still persists even with the global optimal combination of parameters. Therefore, we conclude that it is necessary to find a solution solving both the aggressiveness and latecomer unfairness of LEDBAT.

4 FLOWER The Design

In this chapter, we present the design of FLOWER (**F**uzzy **LOW**er-than-Best-**E**ffo**R**t) transport protocol employing the fuzzy control concept. This new delay-based transport protocol aims at providing an LBE service while solving two main LEDBAT problems — the aggressiveness towards TCP and the latecomer unfairness.

4.1 FLOWER Motivation and Goals

4.1.1 Why Fuzzy Control Matters?

As studied in Chapter 3, 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 or gain to the change of network conditions [11, 12]. To this end, we first need to derive a mathematical model of the network and then construct a mathematical model of the controller. However, the network is an increasingly complex system because of its heterogeneity. Thus, it is not trivial to derive a fine-grained network model required by the adaptive control scheme. The fuzzy control here comes into play to relieve difficulties in developing and analyzing complex systems. Main advantages of the fuzzy control are:

- A mathematical model is not required to develop a fuzzy control system. Indeed, the fuzzy control allows us to incorporate our heuristic knowledge about how to control the system directly into the fuzzy controller. Such an approach is particu-

larly interesting when the model is not trivial, difficult to derive or too complex to be implemented. Moreover, the fuzziness characteristic of this approach enables the development of a reasonable, yet tractable, model that approximates a complex real-world system;

- Since they are usually nonlinear, fuzzy controllers can operate in a much wider range of operating conditions than PID controllers;
- Fuzzy controllers are highly customizable as they consist of linguistic rules that are easy to understand and modify;
- Fuzzy controller operations are straightforward to understand, therefore we can easily design one for a concrete application.

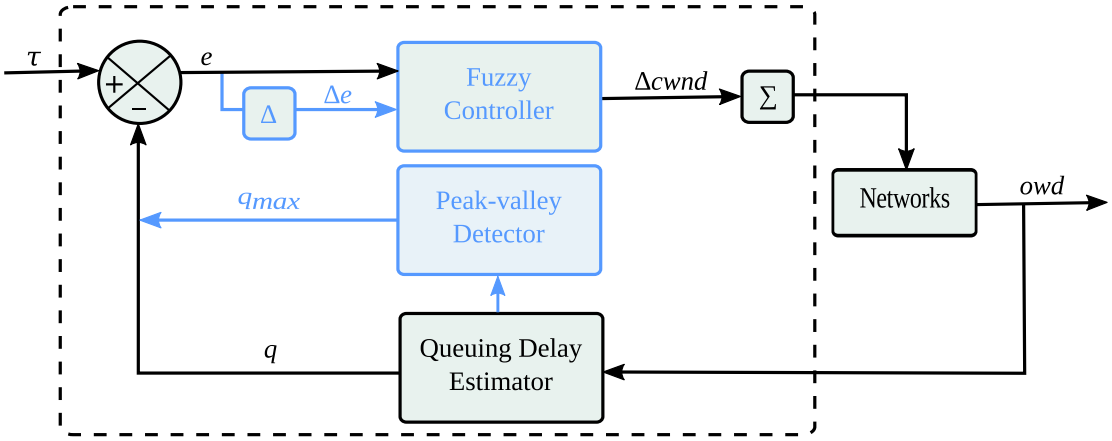
Considering the drawbacks of the adaptive control scheme and the advantages of fuzzy control as discussed above, we design a new LBE congestion protocol based on fuzzy logic. In this fuzzy approach, we can use our previous findings [26] as an entry for the fuzzy controller. Indeed, an in-depth analysis [26] gives us an insight to overcome the LEDBAT problems, or more specifically, to control the queuing delay. Hence, by means of fuzzy logic, we integrate our understanding gathered into the fuzzy controller of FLOWER. We also highlight 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.

4.1.2 The Goals

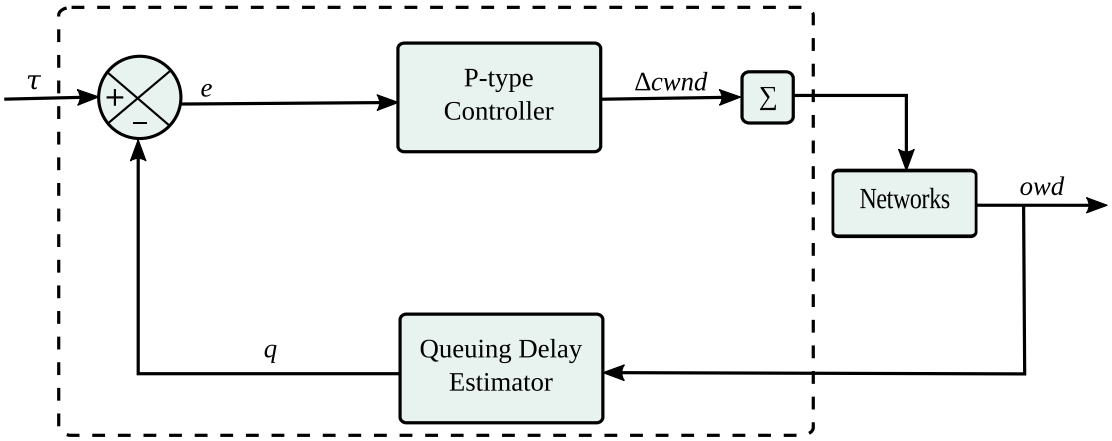
As a potential LEDBAT alternative, FLOWER must tackle its issues while keeping the same goals in terms of LBE service as listed in [10]:

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 [10]. This non-zero target queuing delay allows FLOWER to use 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.



FLOWER
(a)



LEDBAT
(b)

Figure 4.1: Block diagram of FLOWER and LEDBAT as feedback control systems.

4.2 FLOWER Design Outline

We can represent FLOWER congestion control as a feedback control system depicted in Figure 4.1a. The essential components of FLOWER are:

1. *Inputs*

   - *Target queuing delay* τ, which is the maximum queuing delay that a FLOWER flow is allowed to introduce into the network;

- *Queuing delay error* e, which is the difference between the target queuing delay and the estimated queuing delay;
- *Change of queuing delay error* Δe, which is the error trend;

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;

4. *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. The fuzzy controller also incorporates a loss detection scheme to improve congestion detection.

Basically, FLOWER operates as follows: after each RTT, FLOWER filters 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.

In the rest of this section, we give a brief comparison of LEDBAT and FLOWER, then describe the peak-valley detector component. Finally, we discuss about the slow-start mechanism which is part of FLOWER. The main FLOWER component, that is, the fuzzy controller, is further described in detail below.

4.2.1 Differences between FLOWER and LEDBAT

Figure 4.1 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.

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

On initialization:
$findingPeak \leftarrow$ **true**
$n \leftarrow 5$; $\alpha \leftarrow \frac{1}{8}$

After the RTT k:
if we have enough $(n+1)$ samples **then**
    $slidingWnd = \{q_{k-n}, q_{k-n+1}, q_{k-n+2}, \ldots, q_{k-1}, q_k\}$
    $currentValue \leftarrow q_{k-n}$
    $rightMax \leftarrow \max(q_{k-n+1}, q_{k-n+2}, \ldots, q_{k-1}, q_k)$
    $rightMin \leftarrow \min(q_{k-n+1}, q_{k-n+2}, \ldots, q_{k-1}, q_k)$
    if $findingPeak$ **then**
        if $currentValue > rightMax$ **then**
            A peak is found: $findingPeak \leftarrow$ **false**
            Calculate the new threshold S:
            $S \leftarrow (1-\alpha) \times S + \alpha \times currentValue$
            if $currentValue > S$ **then**
                A new $q_{\max}$ is found
    else
        if $currentValue < rightMin$ **then**
            A valley is found: $findingPeak \leftarrow$ **true**

Figure 4.2: Peak-valley detection algorithm.

controller. Note that FLOWER uses the same LEDBAT queuing delay estimator, which is fully described in RFC 6817 [10].

4.2.2 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 filter out the maximum queuing delay (global maximum) using a threshold S, which is computed following an exponentially weighted moving average (EWMA) of peaks. For the sake of remaining as simple as possible and not complicating our implementation, we develop a simple online peak-valley detection algorithm as shown in Figure 4.2.

To understand this algorithm, let us consider a time series of estimated queuing delays $q = \{q_k\}$ where k represents the discrete time in RTT. Basically, an element q_k is a peak/valley if it is greater/smaller than its neighbors, respectively. As our algorithm works in an online manner, at the current time k, we only need to consider

a sliding window of size $(n+1)$ consisting of q_{k-n} and its n right neighbors, that is, $\{q_{k-n}, q_{k-n+1}, q_{k-n+2}, \ldots, q_{k-1}, q_k\}$. The bigger n is, the more robust is the algorithm. We stress that there is a delay of $(n+1)$ RTTs in the detection process of $q_{\max}$ because the algorithm needs to collect enough queuing delay samples.

The algorithm alternatively identifies the peaks and valleys of queuing delays. Indeed, we have a peak/valley if q_{k-n} is greater/smaller than the maximum/minimum of its n right neighbors, respectively. Each time a peak is detected, it is then used to calculate a new threshold to filter out $q_{\max}$. Finally, when a new $q_{\max}$ is found, FLOWER discards the old value.

In our implementation, we let $n = 5$ to keep a small delay while still having a robust maximum queuing delay detection. The EWMA parameter α is set to $\frac{1}{8}$, which is the value typically used for computing the smoothed RTT for TCP.

4.2.3 On the slow-start phase

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. Using slow-start has also been reported by [8]. FLOWER uses slow-start as a synchronization signal and 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, 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.

4.3 How to Design The Fuzzy Controller for FLOWER?

A typical fuzzy control system is shown in Figure 4.3. The fuzzy controller is constituted by the following modules [62]:

A rule base contains a set of "If–Then" rules that describes how to control the system;

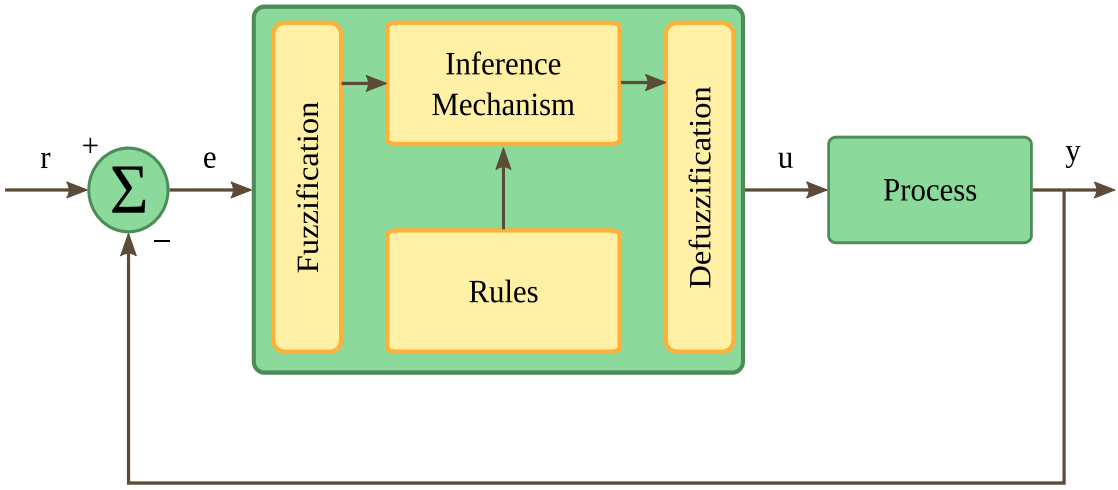


Figure 4.3: A typical fuzzy control system.

An inference mechanism emulates the human expert's decision making about how best to control the system based on the information stored in the rule base;

A fuzzification interface converts controller inputs, e and Δe, into fuzzy values that the inference mechanism can use for its fuzzy reasoning process;

A defuzzification interface converts the conclusions of the inference mechanism into numerical output $\Delta cwnd$.

The best way to have a good grasp of the fuzzy control system is through a use-case example. In the rest of this chapter, we introduce the fuzzy controller and its operations by designing one for FLOWER.

4.3.1 How to Control The Queuing Delay?

In this subsection, we use the analogy of a leaky bucket to explain the logic behind the control of the queuing delay around a target.

Let us consider a bucket with a small hole at the bottom as shown in Figure 4.4. This leaky bucket is defined by several parameters: the inflow rate, the outflow rate, the height of the bucket, and the water level in the bucket. The inflow and outflow rates are the rates at which water is poured into and leaks out of the bucket, respectively. The outflow rate is bounded by the maximum output capacity of the hole and we suppose

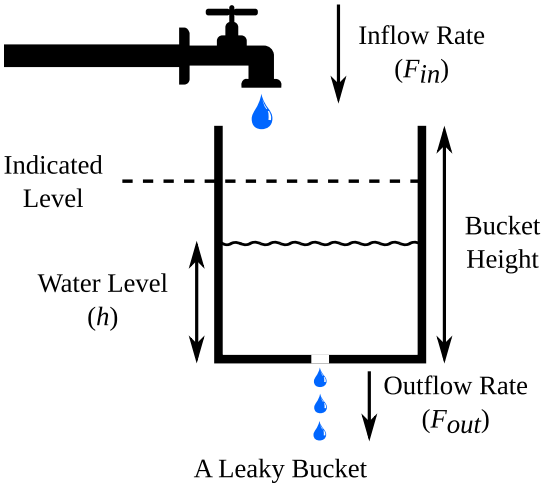


Figure 4.4: A leaky bucket.

that there is no constraint on the inflow rate. When the inflow rate is smaller than the maximum output capacity, the bucket is empty. In this case, the inflow rate equals to the outflow rate. Otherwise, the outflow rate is at its maximum capacity and remains constant regardless the inflow rate until there is no more water in the bucket. When the water level in the bucket exceeds the bucket height, water spills out.

Now suppose that we want to manually fill the leaky bucket to an indicated level lower than the bucket height. As a human-in-the-loop, we must adjust the inflow rate using the faucet to regulate the water level. Rationally, we need to alter the inflow rate based on whether the water level is above or below the indicated level, the distance between them, and the pace at which the water level is rising or falling. For example, when the leaky bucket is empty, we have to increase the inflow rate as fast as we can to quickly reach the indicated level. But when the water level is close to the indicated level, either above or below it, we must adjust the inflow rate slowly to minimize overshoot or undershoot. And if the bucket is full, we need to completely close the faucet, that is, reduce the inflow rate to zero.

The analogy of the control of the water level in the leaky bucket to the control of the queuing delay in the network is as follows: the leaky bucket models a queue at a router. The water is the fluid modeling of packets arriving and leaving the queue. The water outflow rate is the packet departure rate or the throughput, limited by the router's outgoing link capacity. Similarly, the inflow rate is the packet arrival rate at the router queue. The bucket height corresponds to the maximum queuing delay, which is in turn proportional to the maximum queue size of the router. The water level is similar to the

measured queuing delay. Lastly, spilled water corresponds to dropped packets when the router queue is full.

Assume that there is only one router on the path between two hosts. Imagine this time that our responsibility is to manually adjust the congestion window of a source flow and thus the packet arrival rate at the queue to keep the queuing delay around a fixed target. Based on the analogy between the two control problems, we can apply directly the control logic for the water level to the queuing delay problem as shown in what follows. Finally, we highlight that the queuing delays across multiple routers on the path between two hosts can be aggregated as the queuing delay at only one router.

4.3.2 Choosing the controller inputs and output

To make a decision at the sampling instance k, the controller uses as inputs the queuing delay error:

$$e(k) = \tau - q(k)$$

and the change of queuing delay error:

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

The queuing delay error is the difference between the target queuing delay and the estimated queuing delay. If the error is big, the control action must be large to quickly drive the error to zero. In contrast, if the error is small, the control action must be small to prevent oscillation. Therefore, the controller modulates its actions with the queuing delay error.

The change of queuing delay error is the error trend. For a same degree of error, the control actions should differ depending on whether the error trend is increasing or decreasing. If the error trend is increasing, the controller needs to take stronger action to correct the error, but when the error trend is decreasing, the controller must reduce the control action to avoid over-reaction. Thus, the error trend is used to amplify or dampen the actions of the controller.

The controller output is the change of congestion window $\Delta cwnd(k)$, that is, the pace at which the controller must increase/decrease the congestion window to match the queuing delay to the target queuing delay. The congestion window size $cwnd(k)$ is then calculated by:

$$cwnd(k) = cwnd(k-1) + \Delta cwnd(k)$$

4.4 Building the Rule Base

4.4.1 Linguistic Variables and Values

After having discussed the logic behind controlling the queuing delay, we need to load it into the fuzzy controller. To this end, we have to provide a linguistic description of the system dynamics by using **linguistic variables** and their **linguistic values**.

The linguistic variables describe each of the fuzzy controller inputs and outputs, so they usually are the names of inputs and outputs. For FLOWER, the linguistic variables are:

- "queuing delay error" or "$e(k)$",
- "change of queuing delay error" or "$\Delta e(k)$",
- "change of congestion window" or "$\Delta cwnd(k)$".

Each linguistic variable assumes different linguistic values that give **informative descriptions about the numeric (real) values**. The linguistic variables of FLOWER inputs and output take on the following linguistic values:

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

where the meaning is:

- S: small, M: medium, L: large,
- N: negative, P: positive,
- Z: zero, V: very.

Hence, the linguistic value PVS stands for "positive very small" and so forth. Now we can linguistically describe different states of the queuing delay, such as follows:

- The clause "$e(k)$ is Z" indicates that the queuing delay is very close to the target. Note that in this case, the value of $e(k)$ does not need to be exactly zero because of the fuzzy nature of linguistic description;
- The clause "$e(k)$ is PVL and $\Delta e(k)$ is NVS" represents the situation where the queuing delay is very close to zero (that is, the queue is nearly empty) and is increasing very slowly.

4.4.2 Rules

Up to this point, we are ready to translate our heuristic control knowledge into linguistic rules that specify how to control the system. A linguistic rule relates the inputs to the outputs of the fuzzy controller. In general, it has the following form:

If premise **Then** consequent

The premise is where we linguistically describe the inputs to inform the fuzzy controller of the system states. On the other hand, the consequent represents the fuzzy controller outputs, that is, the corresponding actions of the fuzzy controller in response to its inputs.

To illustrate how the fuzzy controller of FLOWER operates, let us take for example the following linguistic rules:

If $e(k)$ is PVL **and** $\Delta e(k)$ is Z **Then** $\Delta cwnd(k)$ is PVL

This rule describes the situation where the queuing delay is very small and does not increase. In consequence, we must increase the congestion window by a very large value.

If $e(k)$ is NVS **and** $\Delta e(k)$ is NVS **Then** $\Delta cwnd(k)$ is NS

This rule describes the situation where the queuing delay is slightly beyond the target delay and is increasing very slowly. In consequence, we must decrease the congestion window by a small value to counteract the movement.

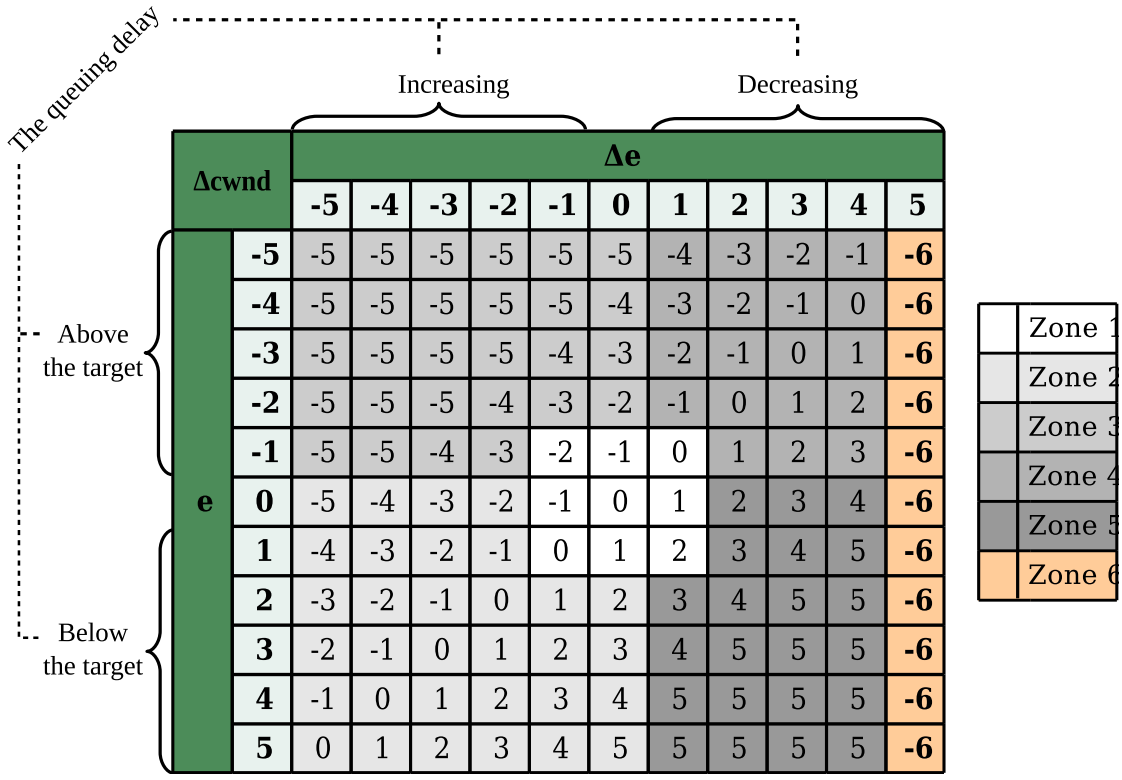
4.4.3 Rule Base

Once we establish all rules, we also finish constructing the rule base for the fuzzy controller. The rule base models the relationship between inputs and outputs of the system. It serves as a **repository** to store the available knowledge about how to solve the problem. For a fuzzy controller with two inputs and one output such as the one for FLOWER, we can list all rules in the rule base using the tabular representation as shown in Table 4.1. Note that in Table 4.1, we use **linguistic-numeric values** [62] to shorten the description of linguistic values, for example:

- -5 represents NVS or “negative very small”,
- 0 represents Z or “zero”,
- 3 represents PM or “positive medium”, and so on.

The advantage of this notation is that it quantifies the sign of linguistic variables, which is convenient for us as $e(k)$ and $\Delta e(k)$ can take on negative or positive values.

For better understanding of the fuzzy controller dynamics, we divide the rule table into six zones as follows:



Δcwnd / e	Δe = -5	-4	-3	-2	-1	0	1	2	3	4	5
-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**

Table 4.1: The rule base of the FLOWER fuzzy controller.

Zone 1: Rules of this zone maintains the steady-state queuing delay around the target. Both $e(k)$ and $\Delta e(k)$ are either negative or positive but they are very close to zero. In consequence, the fuzzy controller must slightly increase or decrease the congestion window to rectify small deviations from the target.

Zone 2: In this zone, $e(k)$ is either negative or zero, which means that the queuing delay is either **below** or **equal** to the target. In addition, since $\Delta e(k)$ is negative most of the time, the queuing delay has the tendency to **raise** and thus **moves in the direction** of the target. Therefore, based on the magnitude of the increase trend of the queuing delay, the fuzzy controller must either increase or decrease the congestion window to accelerate or decelerate the queuing delay motion to match the target, respectively.

Zone 3: In this zone, since $e(k)$ is positive, the queuing delay is **above** the target. On the other hand, the $\Delta e(k)$ is negative most of the time like in Zone 2, which means that the queuing delay has the tendency to **raise** and hence, in this case, **moves away** from the target. Consequently, the fuzzy controller must decrease the congestion window to compensate the increase of the queuing delay.

Zone 4: For this zone of rules, both $e(k)$ and $\Delta e(k)$ are positive, which corresponds to the situation where the queuing delay is **above** and **reducing towards** the target. As a result, to drive the queuing delay to the target, the fuzzy controller needs to accelerate or decelerate the queuing delay motion based on the magnitude of its decrease trend.

Zone 5: Rules of this zone represents the situation where the queuing delay is either **below** or **equal** to the target. Moreover, the queuing delay is **decreasing away** from the target. Thus, $e(k)$ is either negative or zero and $\Delta e(k)$ is negative in this case. The fuzzy controller must therefore increase the congestion window to reverse the decrease trend of the queuing delay.

Zone 6 — Loss Detection Zone: 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 implemented by rules of this special zone, called **Loss Detection Zone**. Concretely, when $\Delta e(k)$ is 5 or PVL, the fuzzy controller implies that there is a very large fall in the queuing delay because of loss. In consequence, the fuzzy controller must immediately reduce the congestion window to minimum, for example, set to one packet. This case corresponds to the following output: $\Delta cwnd(k)$ is -6 or NVVL.

4.5 Choosing Membership Functions

In this section, we look to quantify the significance of linguistic value using the fuzzy logic. For this purpose, we employ **membership functions** to define the semantic of linguistic values. Let A denote a linguistic value and X be a universe of discourse for an input or output of a fuzzy system, that is, the range of numerical values that the input and output can take as values. Each linguistic value A is associated with a membership function. This membership function quantifies the **certainty** or **membership degree** that a numerical value $x \in X$ can be classified linguistically as A. The set of numerical values of X that a membership function describes as being a linguistic value A is called

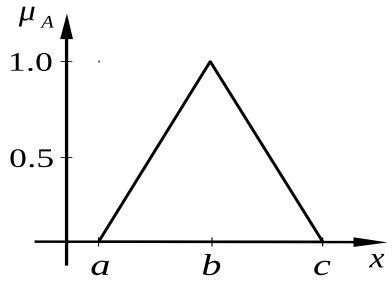


Figure 4.5: The triangular membership function.

a **fuzzy set**. When dealing with fuzzy logic, people usually confuse certainty with probability. However, we stress that they are distinct concepts because the membership function is not a probability density function. Indeed, certainty represents the accuracy of linguistic descriptions, not the probability of some random event.

In this thesis, we use the most common triangular membership function, as shown in Figure 4.5, defined by the three parameters $\{a, b, c\}$ as follows:

$$\mu_A(x) : X \mapsto [0, 1]$$

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

where $a < b < c$ and b is the center of the triangle membership function, that is, where it reaches its peak.

Consider, for instance, the membership function μ_{PVS} that quantifies the meaning of the linguistic value "positive very small" for any numerical value $x \in X$:

- if $\mu_{PVS}(x) = 0$ then we are certain that x is not PVS;
- if $\mu_{PVS}(x) = 0.5$ then we are only half certain that x is PVS. It could also be Z with some degree of certainty;
- if $\mu_{PVS}(x) = 1$ then we are absolutely certain that x is PVS.

Figure 4.6 shows all membership functions for the inputs and output of the FLOWER fuzzy controller.

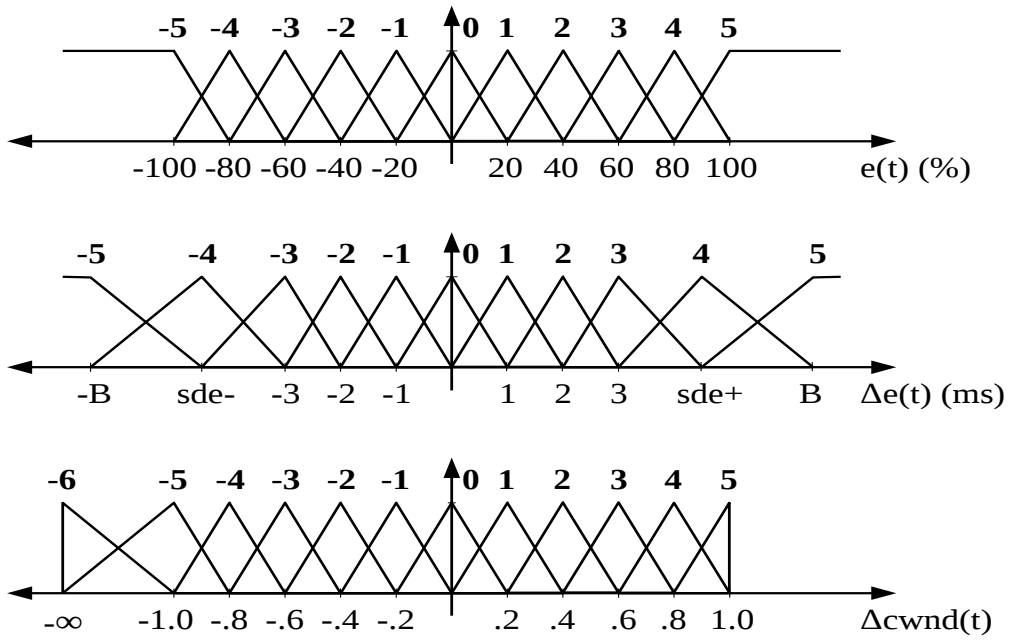


Figure 4.6: The membership function of the FLOWER fuzzy controller.

4.5.1 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 $e(k)$ are linearly distributed on the universe of discourse $[-100, 100]\,\%$.

4.5.2 Membership Functions of $\Delta e(k)$

The queuing delay ranges 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 base in Table 4.1, 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 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 emphasize that, as an effect of the loss detection zone, when $\Delta e(k) > sde_+$, even if the certainty $\mu_{PVL}(\Delta e(k))$ is small, FLOWER reduces the congestion window to its initial value.

4.5.3 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, that is, 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.

4.6 Fuzzy Controller Operations

4.6.1 Fuzzification

Fuzzification is the process of making a numerical value fuzzy so that it can be used by the fuzzy system. Whenever the fuzzification module receives a numerical value x, it converts this value into a corresponding linguistic value by associating a certainty $\mu_A(x)$.

4.6.2 Inference Mechanism

The inference mechanism derives the fuzzy outputs from the fuzzy inputs obtained by fuzzification, according to the relation defined through fuzzy rules. The main matter is how to interpret the meaning of each rule, that is, how to determine the influence produced by the premise on the consequent of the fuzzy rule. To assess this influence, the inference process generally involves in two steps:

1. The certainty of the premise is determined using the fuzzy conjunctive operator (AND);

2. The certainty of the consequent, influenced by the premise, is determined using the fuzzy implication operator.

Consider the i-th rule of the rule base:

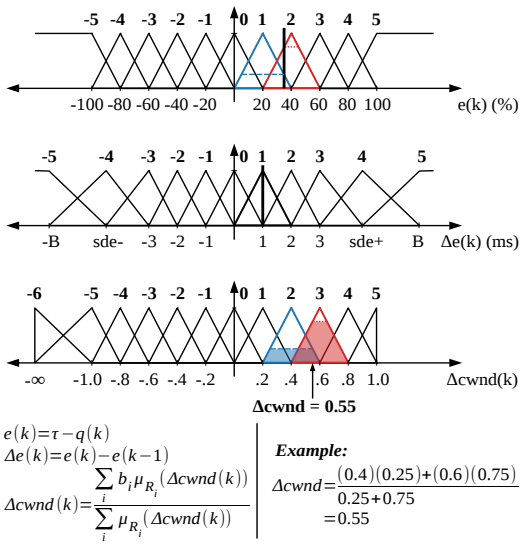
$$R_i\text{: If } u_{i1} \text{ is } A_{i1} \text{ and } \ldots \text{ and } u_{in} \text{ is } A_{in} \text{ Then } v \text{ is } B_i$$

where $A_{i1}, \ldots, A_{in}$, and B_i are the linguistic values of the linguistic variable $u_{i1}, \ldots, u_{in}$, and v in the universes of discourse $X_1, \ldots, X_n$, and Y, respectively. We use the minimum to represent both fuzzy operators. Therefore, the fuzzy conjunctive operator is defined as follows:

$$\begin{aligned}\mu_{A_i}(x_0) &= \mu_{(A_{i1} \text{ AND } \ldots \text{ AND } A_{in})}(x_1, \ldots, x_n) \\ &= min(\mu_{A_{i1}}(x_1), \ldots, \mu_{A_{in}}(x_n))\end{aligned}$$

where $x_0 = (x_1, \ldots, x_n)$. Finally, we have the following definition for the fuzzy implication operator:

$$\mu_{R_i}(y) = min(\mu_{A_i}(x_0), \mu_{B_i}(y))$$



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

Figure 4.7: Example of fuzzy controller operation.

4.6.3 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:

$$y = \frac{\sum_i b_i \mu_{R_i}(y)}{\sum_i \mu_{R_i}(y)}$$

4.6.4 Example of Fuzzy Controller Operation

Consider the example in Figure 4.7. Suppose that $e(k) = 35$ after being converted to the percentage form and $\Delta e(k) = 1$. The fuzzification process gives $\mu_{PVS}(e(k)) = 0.25$ and $\mu_{PS}(e(k)) = 0.75$, whereas $\mu_{PVS}(\Delta e(k)) = 1$. Figure 4.7 shows the certainties 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, by consulting the rule table in Figure 4.7, we have the following corresponding rules:

R_1: if $e(k)$ is PVS and $\Delta e(k)$ is PVS Then $\Delta cwnd(k)$ is PS

R_2: if $e(k)$ is PS and $\Delta e(k)$ is PVS Then $\Delta cwnd(k)$ is PM

Now, consider the first rule, shown as the blue cell in the rule table of Figure 4.7. Let $x_0 = (e(k), \Delta e(k))$, and thus, according to the inference mechanism, we have:

$$\mu_{A_1}(x_0) = \min(\mu_{PVS}(e(k)), \mu_{PVS}(\Delta e(k))) = 0.25$$

and then:

$$\mu_{R_1}(\Delta cwnd(k)) = min(\mu_{A_1}(x_0), \mu_{PS}(\Delta cwnd(k))) = 0.25$$

The certainty of the membership function for the output $\mu_{R_1}(\Delta cwnd(k))$, which is the conclusion reached by rule R_1, is shown in Figure 4.7 as the blue region of the output membership function defining the linguistic value PS.

In the same way, for the second rule (red cell in the rule table), we obtain $\mu_{R_2}(\Delta cwnd(k)) = 0.75$ as shown by the red region of the output membership function defining the linguistic value PM in Figure 4.7.

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:

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

4.7 Conclusion

In this chapter, we discussed the motivation, the goals and the design of the FLOWER congestion control using the fuzzy control. We also study in detail the fuzzy controller, which is the core of FLOWER mechanism. It is worth noting that there exist many other choices for the shape of membership functions (for example, trapezoidal or Gaussian shapes) and as well as for defuzzication methods (for example, mean of maximum or center of gravity). For FLOWER, we use the triangular membership functions and the center-average defuzzication method to facilitate the computation in real time. In the next chapter, we will evaluate the performance of our new LBE protocol using ns-2 simulator.

5 Performance Evaluation of FLOWER

In this chapter, we use the network simulator ns-2.35 to validate our new protocol. We specifically focus on the FLOWER performance and assess whether FLOWER solves the aggressiveness and latecomer unfairness issues which are the two major drawbacks of LEDBAT. We also study the coexistence between FLOWER and AQM schemes.

5.1 Simulation Setup

5.1.1 Ns-2 Implementation of FLOWER

To assess its performance, we have implemented an ns-2 prototype of FLOWER based on a LEDBAT module developed by Valenti et al. [6]. The prototype is implemented as a Linux congestion control module on top of the TCP-Linux framework [63]. Therefore, simulation results are much closer to a real implementation in the Linux kernel and would allow easily porting our implementation to the Linux kernel (this also been the case for the LEDBAT module [6]).

5.1.2 Reference Network Configuration

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 [14] for the sake of comparison. All sources send packets with a size of $P = 1500\,\text{B}$. The bottleneck link has a capacity set to $C = 10\,\text{Mb/s}$ and a one-way propagation delay $owd \in [10, 50, 100, 150, 200, 250]\,\text{ms}$. 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 $B = \lceil n \cdot BDP \rceil = \lceil n \cdot C \cdot 2 \cdot owd/(8 \cdot P) \rceil$, 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: $\tau\,(packets) = \tau\,(ms) \cdot C/(8 \cdot P)$. Therefore, a target queuing delay $\tau = 100\,\text{ms}$ corresponds to 83.3 packets and is rounded to 84 packets.

5.2 Simulation Results

5.2.1 Interaction with TCP

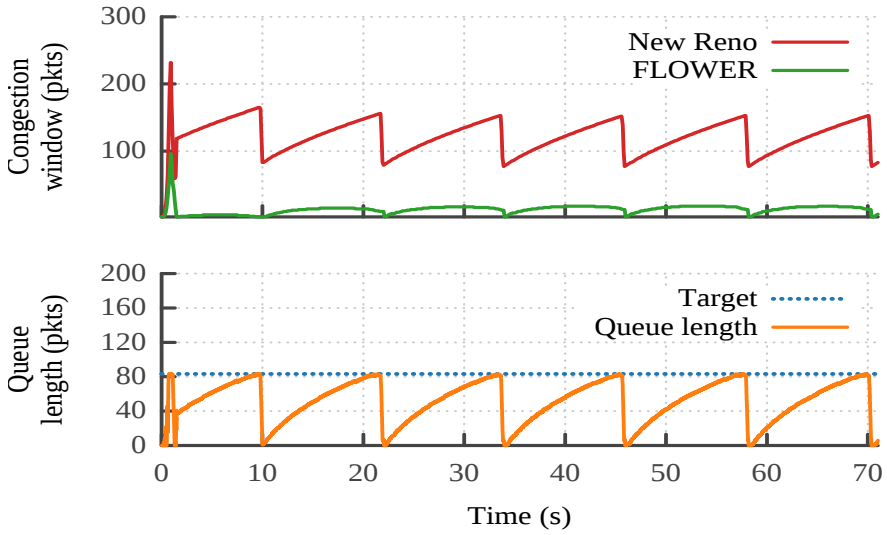
In this section, we study the behavior of FLOWER in the presence of TCP and more specifically, the interaction between the FLOWER fuzzy controller and the TCP AIMD algorithm.

Scenario and metrics

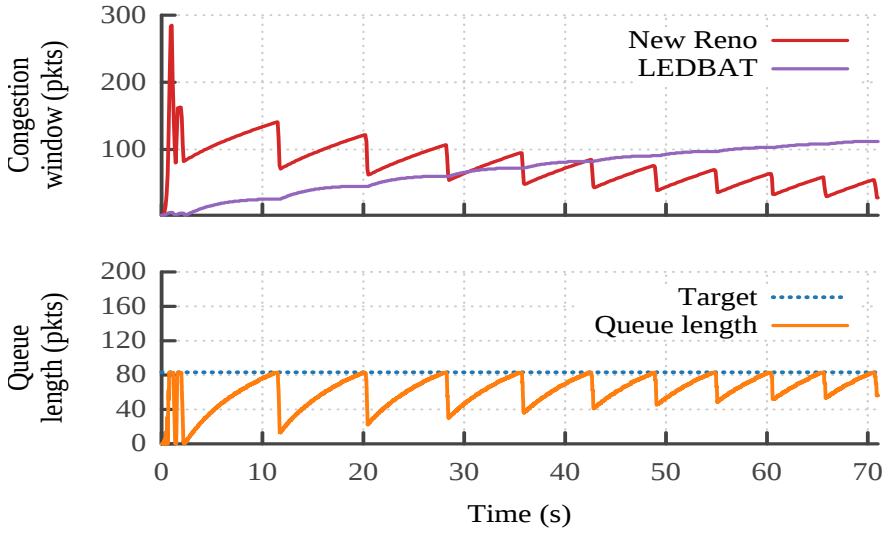
Two TCP and LBE flows start at $t = 0\,\text{s}$ and stop at $t = 75\,\text{s}$. In this scenario, $owd = 50\,\text{ms}$ and $B = BDP$. To investigate the behavior of one LBE flow in coexistence with one TCP flow, we consider their congestion windows and the queue length of the bottleneck buffer.

Results

Fig. 5.1 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. 5.1a. In the slow-start phase, TCP and FLOWER exponentially increase 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 $t = 3\,\text{s}$, 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 $t = 7.5\,\text{s}$, (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 $t = 9.3\,\text{s}$. Slightly afterwards, TCP gets losses and en-



(a) FLOWER vs. TCP NewReno



(b) LEDBAT vs. TCP NewReno

Figure 5.1: TCP and LBE congestion windows and bottleneck queue length as a function of time.

ters its recovery phase. As a consequence, TCP halves its congestion window and the bottleneck queue is drained.

TCP re-enters the congestion avoidance phase at $t = 10\,\mathrm{s}$ while FLOWER grows at

its maximum speed as the queue is not full. FLOWER prevents bottleneck overflow by reducing its sending rate before the knee phase [64] (i.e., when the rate increases gradually but slower than the delay). When TCP halves its congestion window at $t = 21.8\,\text{s}$, we observe an abrupt fall of the queuing delay. Shortly afterwards, FLOWER detects this fall with the help of the loss detection scheme, hence FLOWER drops to the minimum congestion window. Therefore, the queue is drained and FLOWER enters a new cycle. Henceforth, both FLOWER and TCP are in steady state.

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. 5.1b. This figure also illustrates that the LEDBAT P-type controller does not react correctly to congestion events. We refer the reader to previous studies [8, 26] 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 illustrate the good performance of our fuzzy controller scheme.

5.2.2 FLOWER versus LEDBAT performance in coexistence with TCP NewReno and CUBIC

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

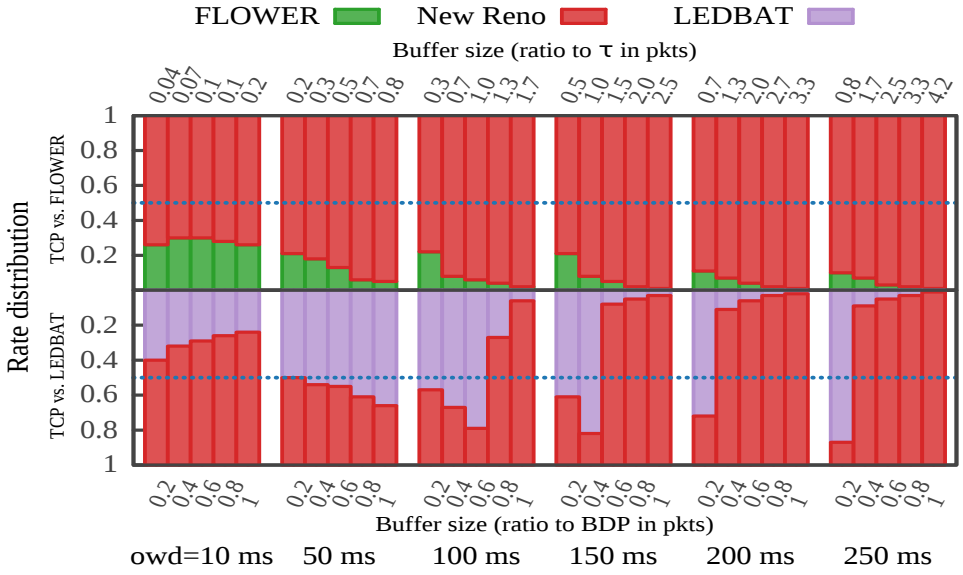
Scenario and metric

We consider 5 long-lived TCP flows with 5 LBE flows. The simulation lasts 1200 s where TCP flows start consecutively every 10 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 $t = 450\,\text{s}$ to allow the TCP flows to reach their full capacity.

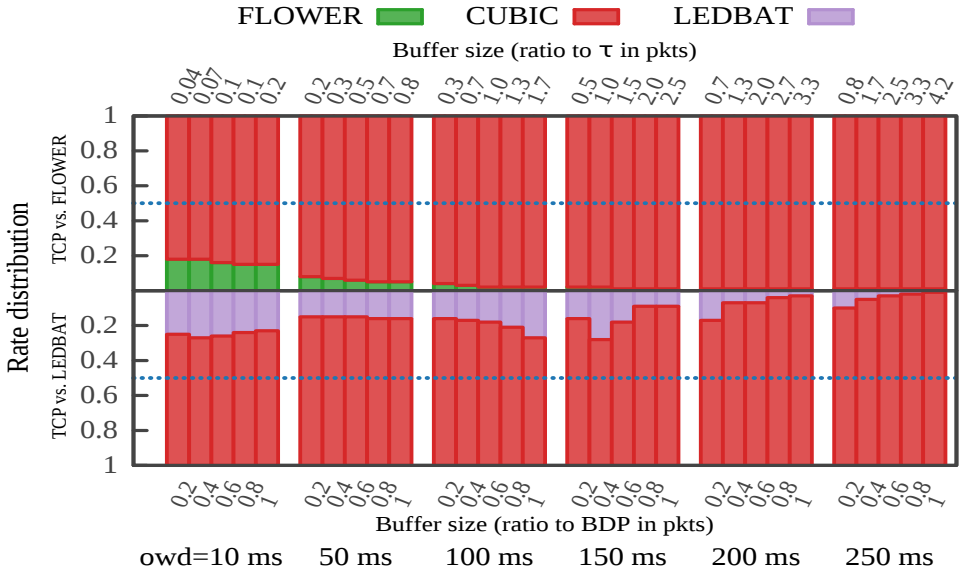
To assess the impact of LBE on TCP, we define the metric *rate distribution* (X) as the total throughput achieved by all flows F_k where $k \in \{TCP, LBE\}$ over the total throughput of all flows on the link:

$$X_k = \frac{F_k}{F_{TCP} + F_{LBE}} \tag{5.1}$$

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,



(a) LBE vs. TCP NewReno



(b) LBE vs. CUBIC

Figure 5.2: Rate distribution of TCP and LBE flows.

the mean of the 10 metric values is taken as the measured value.

Results

In Fig. 5.2, using histograms, we group the simulation results into different categories of one-way delay (denoted *owd* in Fig. 5.2), 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 5.1.2. 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. 5.2, 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 (5.1).

Fig. 5.2a shows the performance of LEDBAT and FLOWER in the presence of TCP NewReno. We have selected a set of network configurations following our previous study on the LEDBAT performance issues [26]. These network configurations illustrate a large number of use-cases where LEDBAT performs (in Fig. 5.2a 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. 5.2a, LEDBAT obtains sometimes more than TCP NewReno and crosses the fair-share line represented by a dotted line. We then compare the results obtained by FLOWER in these configuration. Fig. 5.2a 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 average cases.

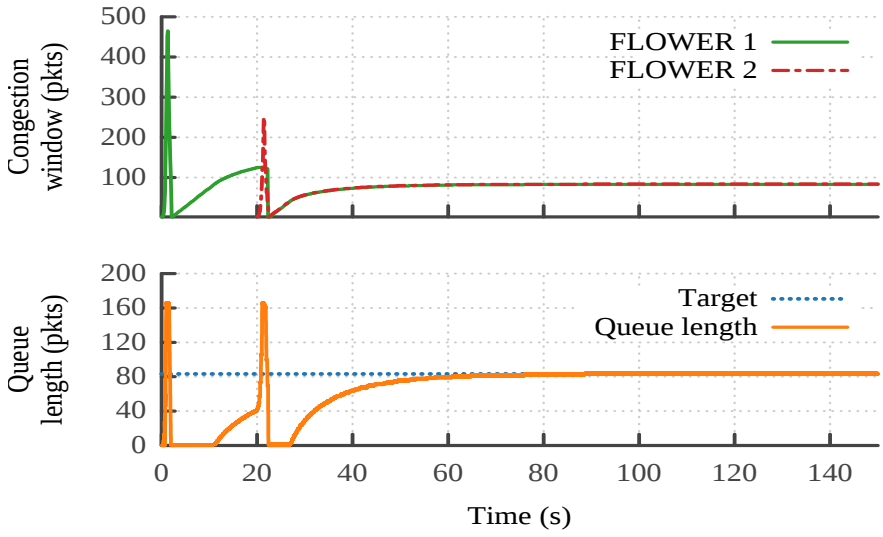
Using the same network configurations as above, we now study the performance of LEDBAT and FLOWER in coexistence with CUBIC in Fig. 5.2b. CUBIC is more aggressive than TCP NewReno but in those cases, the performance of FLOWER is far better than LEDBAT in respect of the LBE principle.

5.2.3 Intra-protocol fairness

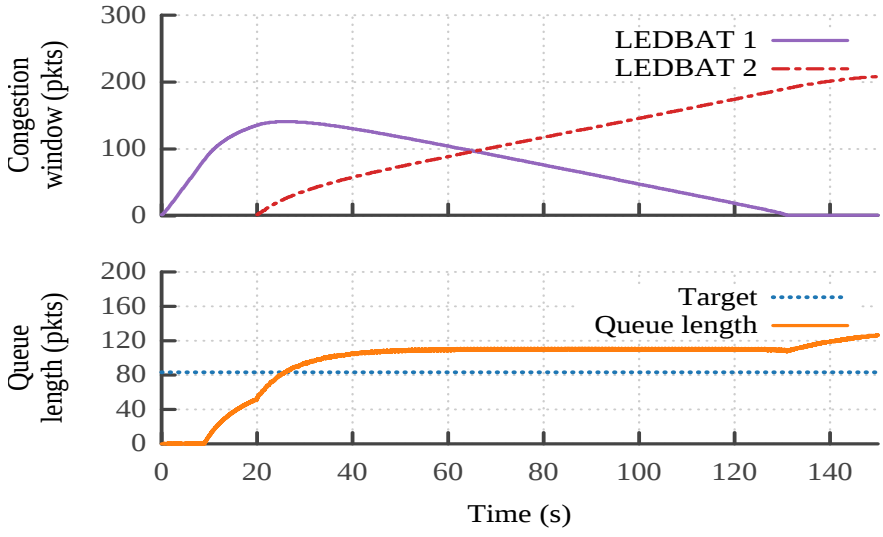
We finally study the interaction between two FLOWER flows to assess their intra-fairness and determine whether or not FLOWER is impacted by the latecomer issue.

Scenario and metric

In this scenario, the buffer size B is set to twice the BDP. This configuration is favorable to observe the LEDBAT latecomer unfairness phenomenon. The bottleneck link has a one-way delay $owd = 50\,\text{ms}$. The first LBE flow starts at $t = 0\,\text{s}$ and the second starts



(a) FLOWER



(b) LEDBAT

Figure 5.3: LBE congestion windows and bottleneck queue length as a function of time.

at $t = 20s$. Both flows last 150 s. As in 5.2.1, we draw their congestion windows and the queue length of the bottleneck buffer.

Results

Fig. 5.3b shows the LEDBAT latecomer issue [6]. 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 $t = 20\,\text{s}$, 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, the first LEDBAT flow reaches its minimum rate at $t = 131\,\text{s}$ as shown in Fig. 5.3b. On the contrary, FLOWER does not inherit this latecomer issue thanks to the loss detection scheme described in Chapter 4 as shown in Fig. 5.3a. This experiment demonstrates that two FLOWER flows can now share fairly the link capacity.

5.3 Coexistence of FLOWER and AQM

While Active Queue Management (AQM) has been a research field for over two decades with many proposals, its adoption remains very limited. However, recent concern about the excessive end-to-end delay on the Internet due to bufferbloat returns AQM as an up to date and hot topic at the IETF. Indeed, AQM is usually considered as the best solution to solve the bufferbloat problem [19]. On its part, LEDBAT is an LBE transport protocol designed to work mainly under a DropTail queuing discipline. In the existence of AQM schemes, LEDBAT loses its LBE characteristic and behaves like standard TCP as shown by the authors of [7,13,14]. LEDBAT RFC also admits this fact [10]: "If Active Queue Management is configured to drop or ECN-mark packets before the LEDBAT flow starts reacting to persistent queue buildup, LEDBAT reverts to standard TCP behavior rather than yielding to other TCP flows". Therefore, when designing a new LBE protocol (or any kind of novel transport protocol), it is important to study its coexistence with AQM schemes.

In this section, we evaluate the impact of AQM such as RED [15], CoDel [16] and PIE [17] on the LBE-compliance of FLOWER in the presence of standard TCP connections. To this end, we directly employ the scripts used by the authors of [14], which are available at [18].

5.3.1 Active Queue Management Schemes

Random Early Detection (RED)

RED randomly dropped packets with a probability p, calculated based on the Exponential Weighted Moving Average (EWMA) q_{avg} of the instantaneous queue length as follows:

$$p(q_{avg}) = \begin{cases} 0 & 0 \leq q_{avg} \leq min_{th}, \\ \dfrac{q_{avg} - min_{th}}{max_{th} - min_{th}} p_{max} & min_{th} < q_{avg} \leq max_{th}, \\ 1 & q_{avg} > max_{th} \end{cases}$$

where

min_{th}: the minimum threshold,

max_{th}: the maximum threshold,

p_{max}: the maximum probability for packet dropping at the maximum threshold.

In this study, we use the default version of RED in ns-2. In this version, the $gentle\_$ mode is activated to make RED more robust; the min_{th} and max_{th} are automatically configured in function of the target average delay $targetdelay\_$, which has a default value of 5 ms.

Controlled Delay (CoDel)

The goal of CoDel is to keep the minimum queuing delay (or sojourn delay) experienced by packets in a fixed interval (100 ms by default) below a target delay (5 ms by default). Therefore, CoDel starts to drop selected packets when the minimum queuing delay is higher than the target delay. Each time CoDel drops a packet, CoDel sets the next dropping time based on the number of drops since the beginning of the dropping state, as follows:

$$nextDropTime = lastDropTime + \frac{interval}{\sqrt{numOfDrops}}$$

For this study, we use the CoDel implementation in ns-2 in the scripts provided by the authors of [14].

Proportional Integral Controller Enhanced (PIE)

Similar to CoDel, PIE keeps the queuing delay around a target delay, which has a default value of 20 ms. However, instead of monitoring the real delay for each packet like CoDel,

PIE estimates the current queuing delay based on the queue draining rate using Little's law. To determine the dropping probability every t_{update} time units, PIE employs a PI-type controller that takes into account both the current queuing delay and its trend:

$$p = p + \alpha \cdot (queuingDelay - targetDelay) + \beta \cdot (queuingDelay - lastQueuingDelay)$$

where the factors α and β are respectively set to 0.125 and 1.25 by default. The ns-2 implementation of PIE used in this study can be found at [65].

5.3.2 Scenario and Metrics

We consider 5 long-lived standard TCP flows with 5 LBE flows. All flows start at time $t = 0$. In this scenario, $owd = 50\,\text{ms}$ and $B = 250\,\text{pkts} = 3 \times BDP$ to reproduce the bufferbloat.

To evaluate the interaction between LBE protocols (LEDBAT, FLOWER) and AQM schemes (RED, CoDel, PIE), we measure the rate distribution of TCP X_{TCP}, the average queue length $E[Q]$ in terms of packet, and the bufferbloat intensity defined as $E[Q]/B$. Note that in [14], the authors denote X_{TCP} as $TCP\%$.

For each combination of LBE protocols and AQM schemes, we run the simulation 10 times and each run lasts for 60 s. The mean of the metric values is then taken as the measured values.

5.3.3 Impact of AQM Schemes on LBE Protocols

We present the simulation results of this section in Figure 5.4 using a parallel coordinate plot. The left and right y-axes correspond to the bufferbloat intensity $E[Q]/B$ and the rate distribution of TCP, respectively. In the parallel coordinate plot, a line connecting these two metrics represents the interaction of each combination of AQM schemes and LBE protocols. The ideal interaction is illustrated by the green oblique region in Figure 5.4, in which the queuing delay is low while the LBE traffic remains low-priority.

Under DropTail, TCP continuously fills up the buffer until loss and therefore maximizes the bufferbloat intensity, as shown in Figure 5.4. As for LEDBAT and FLOWER, in this case, they are both LBE-compliant, which are represented by TCP shares approaching one. We recall that the goal of LEDBAT and FLOWER is to keep the queuing delay around a fixed target. Nevertheless, this design choice only limits the exacerbation of bufferbloat but does not solve it.

Figure 5.4 clearly shows that employing an AQM scheme solves the bufferbloat issue.

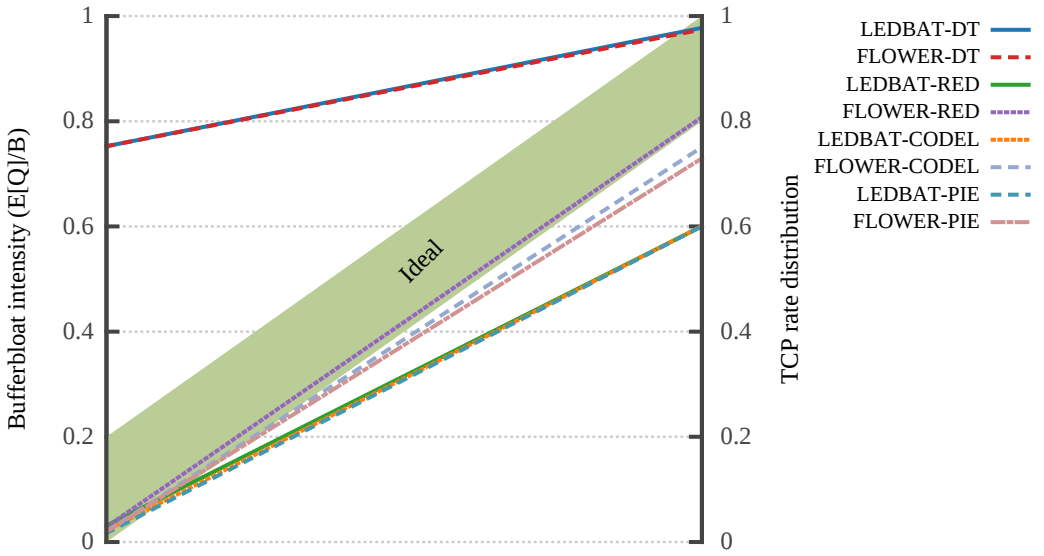


Figure 5.4: Impact of AQM on LBE protocols.

However, such an AQM scheme also compromises the low-priority characteristic of LBE protocols and raises their aggressiveness towards TCP. In this case, LEDBAT competes quite fairly with TCP. The results for LEDBAT are indeed in accordance with the study in [14]. Regarding the new protocol, in all cases, FLOWER is always more LBE-compliant than LEDBAT and tends towards the ideal region. There are two reasons behind this outcome. First, FLOWER has a loss detection zone in its fuzzy rule base that allows it to react better than LEDBAT in times of congestion. Second, FLOWER resets its congestion window to minimum in case of loss to minimize its impact on higher priority flows.

5.4 Conclusion

In this chapter, we evaluated the performance of the new LBE protocol FLOWER by means of simulation. The results show that FLOWER behaves as an LBE protocol in network scenarios where LEDBAT fails while solving the latecomer unfairness problem. We also demonstrated the good interaction between FLOWER and AQM schemes. Actually, in all testing cases, FLOWER is always more LBE-compliant than LEDBAT and tends towards the ideal interaction with AQMs, where the queuing delay is low while the LBE traffic remains low-priority. Also note that the loss detection zone in the fuzzy rule base plays an important role in the outperformance of FLOWER.

All of these simulations above demonstrate the potential of FLOWER as an alternative to LEDBAT and also highlight the power of the fuzzy control. To the best of our knowledge, FLOWER is the first solution that solves both the aggressiveness and latecomer issues of LEDBAT while still having a good interaction with AQM schemes. Finally, with the fuzzy control, it is easy to extend the rule base to upgrade FLOWER to a hybrid loss and delay-based protocol for a better reaction to congestion.

6 Conclusion

This thesis investigated the possibility to deploy the Lower-than-Best-Effort service over large bandwidth-delay product networks, in particular over long-delay link such as satellite networks. LBE service is a low-priority service class that uses only the residual network resources while minimizing the impact on other traffic classes. The motivation to use such a service is usually under question. However, LBE service gives rise to many applications (e.g. such as automatic backup, software updates, or peer-to-peer file transfers, ...) that are advantageous to both users and a network providers. Furthermore, the LBE service over satellite communications might enable a low-cost or even free Internet access in remote communities.

In this thesis, we only focused in the deployment of LBE service at the transport layer. Initially, we chose to study LEDBAT since it is the most widely deployed LBE transport protocol. However, our analysis exhibits the difficulty in tuning LEDBAT parameters and thus prompts us to design a new LBE protocol employing fuzzy control theory. All of these contributions are summarized below.

LEDBAT Analysis

In the beginning of this thesis, by means of simulation, we study the impact of LEDBAT internal parameters on its performance in the presence of TCP over a wide range of network configurations. LEDBAT employs a P-type controller to modify the congestion window with respect to the queuing delay. This queuing delay is derived by the variation of the estimated end-to-end one-way delay. Two key parameters of LEDBAT are the target queuing delay and gain (increase and decrease gain). Both parameters strongly impact on the LEDBAT behavior in terms of fairness with other protocols. Consequently,

a misconfiguration leads to the aggressiveness of LEDBAT towards TCP. LEBDAT also suffers from the intra-unfairness problem, named latecomer advantage.

We first study the impact of the target queuing delay and decrease gain settings on the working regions of LEDBAT — the areas where LEDBAT behaves correctly as an LBE protocol. Then, by exploring the set of LEDBAT parameters, we find that (5 ms; 10) is the global optimal combination of target queuing delay and decrease gain allowing LEDBAT to be LBE-compliant over a wide range of network configurations. We note that this optimal target is far from the guideline in the RFC, which recommends a target of 100 ms. Besides, in all simulation cases, LEDBAT is not fully LBE and borrows some capacity of the primary flows. As a result, the impact of the increasing number of LEDBAT flows remains an issue. Furthermore, the latecomer unfairness issue of LEDBAT still persists even with the global optimal combination of parameters. Our in-depth LEDBAT analysis highlights the necessary to find a solution to solve both the aggressiveness and latecomer unfairness of LEDBAT.

Design of FLOWER

In light of the LEDBAT analysis, we propose FLOWER, a new delay-based congestion control protocol designed to provide an 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. The main difference between both protocols is that FLOWER replaces the P-type controller with the fuzzy controller that, besides the queuing delay error, also utilizes the error trend to modulate the congestion window. While being more robust than the P-type controller, we highlight the fact that the fuzzy controller is simple to implement.

To test our protocol, we have implemented a prototype of FLOWER in the ns-2 simulator. First, we assess the impact of FLOWER flows on TCP flows in different network conditions. The results demonstrate that, when coexisting with standard TCP, FLOWER always behaves as an LBE protocol where LEDBAT fails in average use cases. On the other hand, in the presence of CUBIC, the performance of FLOWER is far better than LEDBAT in respect of the LBE principle, although CUBIC is more aggressive than standard TCP. Then, we study the interaction between two FLOWER flows to assess their intra-fairness and determine whether FLOWER is not impacted by the latecomer issue. This time, the simulation shows that FLOWER does not inherit the latecomer issue of LEDBAT thanks to the loss detection scheme. In consequence, FLOWER flows can fairly share the link capacity. Finally, we evaluate the interaction between AQM

schemes and FLOWER in the presence of standard TCP connections. In all testing cases, FLOWER is always more LBE-compliant than LEDBAT and tends towards the ideal interaction with AQMs, where the queuing delay is low while the LBE traffic remains low-priority. There are two reasons behind this outcome. First, FLOWER has a loss detection zone in its fuzzy rule base that allows it to react better than LEDBAT when congestion occurs. Second, FLOWER resets its congestion window to minimum in case of loss to alleviate its impact on higher priority flows.

All of these simulation results above demonstrate the potential of FLOWER as an alternative to LEDBAT. To the best of our knowledge, FLOWER is the first solution that solves both the aggressiveness and latecomer issues of LEDBAT while still having a good interaction with AQM schemes.

Perspectives

There still are points that this thesis do not cover and that could be explored. In the following, we present these remaining open issues:

Fixed target queuing delay Similarly to LEDBAT, FLOWER uses a fixed target queuing delay set to 100 ms by default. Therefore, a non-compliant implementation could introduce the intra-unfairness issue between FLOWER or LEDBAT flows, as discussed in Chapter 2. One possible solution is to employ an adaptive scheme to cope with different network configurations. However, how to adapt this target queuing delay remains unclear until now. Consequently, using either a fixed or adaptive target queuing delay requires a special research attention.

Loss-based reaction While it has a loss detection zone in its fuzzy rule base, FLOWER mainly uses the queuing delay as a main congestion indication. With fuzzy control, we could easily expand the rule base to upgrade FLOWER to an hybrid loss and delay-based protocol. Loss-based reaction improves the interaction between FLOWER and AQM schemes. Moreover, it also improves the FLOWER performance in sub-packet regimes, which encourages the deployment FLOWER to help wireless community network users in developing regions to efficiently share low-bandwidth backhaul Internet links.

FLOWER implementation and testing We are currently porting the ns-2 implementation of FLOWER to the Linux kernel for testing. At an initial stage, we plan

to test FLOWER performance and its interplay with AMQs on our local testbed at ISAE lab, and then over the satellite link using CESARS platform provided by CNES. Afterwards, we expect to further push this idea at IETF.

A List of Acronyms

ACK	ACKnowledgment
AIMD	Additive Increase/Multiplicative Decrease
AQM	Active Queue Management
BDP	Bandwidth Delay Product
CoDel	Controlled Delay
ECN	Explicit Congestion Notification
EWMA	Exponential Weighted Moving Average
FLOWER	Fuzzy LOWer than best-EffoRt
IETF	Internet Engineering Task Force
IP	Internet Protocol
LBDP	Large Bandwidth Delay Product
LBE	Lower-than-Best-Effort
LEDBAT	Low Extra Delay BAckground Transport
OWD	One-Way Delay
PID	Proportional-Integral-Derivative
PIE	Proportional Integral controller Enhanced

RED	Random Early Detection
RFC	Request for Comments
RTT	Round-Trip Time
TCP	Transmission Control Protocol

B

List of Publications

Published

1. **On The Existence Of Optimal LEDBAT Parameters**
   Si Quoc Viet Trang, Nicolas Kuhn, Emmanuel Lochin, Cédric Baudoin, Emmanuel Dubois, Patrick Gélard
   2014 IEEE International Conference on Communications (ICC)

2. **FLOWER — Fuzzy Lower-than-Best-Effort Transport Protocol**
   Si Quoc Viet Trang, Emmanuel Lochin, Cédric Baudoin, Emmanuel Dubois, Patrick Gélard
   2015 IEEE 40th Conference on Local Computer Networks (LCN)

Submitted to review

1. **Non-Renegable Selective Acknowledgments (NR-SACKs) for TCP**
   Fan Yang, Paul Amer, Si Quoc Viet Trang, Emmanuel Lochin

2. **FLOWER — An Innovative Fuzzy Lower-than-Best-Effort Transport Protocol**

IETF work

93rd IETF Prague presentation of FLOWER at the session of the Internet Congestion Control Research Group (ICCRG).

Bibliography

[1] A. Sathiaseelan, "Researching Global Access to the Internet for All (GAIA)," *IETF Journal*, July 2014.

[2] 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.

[3] T. Chown, T. Ferrari, S. Leinen, R. Sabatino, N. Simar, and S. Venaas, "Less than Best Effort: Application Scenarios and Experimental Results," in *Quality of Service in Multiservice IP Networks* (M. Marsan, G. Corazza, M. Listanti, and A. Roveri, eds.), vol. 2601 of *Lecture Notes in Computer Science*, pp. 131–144, Springer Berlin Heidelberg, 2003.

[4] G. Carofiglio, L. Muscariello, D. Rossi, and C. Testa, "A hands-on assessment of transport protocols with lower than best effort priority," in *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pp. 8–15, Oct 2010.

[5] M. Kühlewind and S. Fisches, "Evaluation of Different Decrease Schemes for LEDBAT Congestion Control," in *Energy-Aware Communications* (R. Lehnert, ed.), vol. 6955 of *Lecture Notes in Computer Science*, pp. 112–123, Springer Berlin Heidelberg, 2011.

[6] D. Rossi, C. Testa, S. Valenti, and L. Muscariello, "LEDBAT: The New BitTorrent Congestion Control Protocol," in *Proceedings of 19th International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–6, August 2010.

[7] J. Schneider, J. Wagner, R. Winter, and H. Kolbe, "Out of my way - evaluating Low Extra Delay Background Transport in an ADSL access network," in *Teletraffic Congress (ITC), 2010 22nd International*, pp. 1–8, Sept 2010.

[8] G. Carofiglio, L. Muscariello, D. Rossi, and S. Valenti, "The Quest for LEDBAT Fairness," in *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pp. 1–6, Dec 2010.

[9] G. Carofiglio, L. Muscariello, D. Rossi, C. Testa, and S. Valenti, "Rethinking the Low Extra Delay Background Transport (LEDBAT) Protocol," *Computer Networks*, vol. 57, pp. 1838–1852, June 2013.

[10] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)." RFC 6817, December 2012.

[11] A. Abu and S. Gordon, "A Dynamic Algorithm for Stabilising LEDBAT Congestion Window," in *Second International Conference on Computer and Network Technology (ICCNT)*, pp. 157–161, April 2010.

[12] N. Kuhn, O. Mehani, A. Sathiaseelan, and E. Lochin, "Less-than-Best-Effort Capacity Sharing over High BDP Networks with LEDBAT," in *IEEE 78th Vehicular Technology Conference (VTC Fall)*, pp. 1–5, September 2013.

[13] Y. Gong, D. Rossi, and E. Leonardi, "Modeling the interdependency of low-priority congestion control and active queue management," in *2013 25th International Teletraffic Congress (ITC)*, pp. 1–9, September 2013.

[14] Y. Gong, D. Rossi, C. Testa, S. Valenti, and M. Täht, "Fighting the bufferbloat: on the coexistence of AQM and low priority congestion control," *Computer Networks*, vol. 65, pp. 255–267, 2014.

[15] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, August 1993.

[16] K. Nichols and V. Jacobson, "Controlling queue delay," *Communications of the ACM*, vol. 55, pp. 42–50, May 2012.

[17] R. Pan, P. Natarajan, C. Piglione, M. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, "PIE: A lightweight control scheme to address the bufferbloat problem," in *IEEE 14th International Conference on High Performance Switching and Routing (HPSR)*, pp. 148–155, July 2013.

[18] http://perso.telecom-paristech.fr/~drossi/index.php?n=Dataset.LEDBATAQM.

[19] N. Khademi, D. Ros, and M. Welzl, "The new aqm kids on the block: An experimental evaluation of codel and pie," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 85–90, April 2014.

[20] "Global Internet User Survey 2012," *Internet Society*, November 2012.

[21] "Project Loon." http://www.google.com/loon.

[22] "Internet.org." https://www.internet.org.

[23] "Global Access to the Internet for All Research Group (GAIA)." https://irtf.org/gaia.

[24] A. Norberg, "uTorrent transport protocol." BitTorrent, http://www.bittorrent.org/beps/bep_0029.html, October 2012.

[25] A. Sathiaseelan and J. Crowcroft, "The free Internet: a distant mirage or near reality?," tech. rep., University of Cambridge, February 2012.

[26] S. Q. V. Trang, N. Kuhn, E. Lochin, C. Baudoin, E. Dubois, and P. Gelard, "On The Existence Of Optimal LEDBAT Parameters," in *IEEE International Conference on Communications (ICC)*, June 2014.

[27] S. Q. V. Trang, E. Lochin, C. Baudoin, E. Dubois, and P. Gelard, "FLOWER - Fuzzy Lower-than-Best-Effort Transport Protocol," in *IEEE 40th Conference on Local Computer Networks (LCN)*, October 2015.

[28] G. Maral and M. Bousquet, *Satellite Communications Systems: Systems, Techniques and Technology*. Wiley, 5th ed., February 2010.

[29] S. Floyd, "HighSpeed TCP for Large Congestion Windows." RFC 3649, December 2003.

[30] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-friendly High-speed TCP Variant," *SIGOPS Operating Systems Review*, vol. 42, pp. 64–74, July 2008.

[31] D. Leith, R. Shorten, and G. McCullagh, "Experimental evaluation of cubic-TCP," in *Proceedings of the 5th International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet)*, February 2007.

[32] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A Compound TCP Approach for High-Speed and Long Distance Networks," in *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM)*, pp. 1–12, April 2006.

[33] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, pp. 1465–1480, October 1995.

[34] D. J. Leith, L. L. Andrew, T. Quetchenbach, and R. N. Shorten, "Experimental Evaluation of Delay/Loss-based TCP Congestion Control Algorithms," in *Proceedings of the 6th International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet)*, 2008.

[35] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services." RFC 2475, December 1998.

[36] 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.

[37] R. Bless, K. Nichols, and K. Wehrle, "A lower effort per-domain behavior (PDB) for differentiated services." RFC 3662, December 2003.

[38] Y. Hayel, D. Ros, and B. Tuffin, "Less-than-best-effort services: pricing and scheduling," in *Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM)*, p. 75, March 2004.

[39] "Background Intelligent Transfer Service (BITS)." Microsoft, `https://msdn.microsoft.com/en-us/library/bb968799(v=vs.85).aspx`.

[40] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, *Feedback Control of Computing Systems*. Wiley-IEEE Press, 1st ed., August 2004.

[41] R. C. Dorf and R. H. Bishop, *Modern Control Systems*. Prentice Hall, 12th ed., July 2010.

[42] D. Rossi, C. Testa, and S. Valenti, "Yes, We LEDBAT: Playing with the New BitTorrent Congestion Control Algorithm," in *Proceedings of the 11th International Conference on Passive and Active Measurement*, PAM'10, (Berlin, Heidelberg), pp. 31–40, Springer-Verlag, 2010.

[43] M. Andreica, N. Tapus, and J. Pouwelse, "Performance evaluation of a Python implementation of the new LEDBAT congestion control algorithm," in *Automation Quality and Testing Robotics (AQTR), 2010 IEEE International Conference on*, vol. 2, pp. 1–6, May 2010.

[44] D. Ros and M. Welzl, "Assessing LEDBAT's Delay Impact," *IEEE Communications Letters*, vol. 17, pp. 1044–1047, May 2013.

[45] I. Komnios, A. Sathiaseelan, and J. Crowcroft, "LEDBAT performance in sub-packet regimes," in *11th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, pp. 154–161, April 2014.

[46] C. Testa and D. Rossi, "On the impact of uTP on BitTorrent completion time," in *IEEE International Conference on Peer-to-Peer Computing (P2P)*, pp. 314–317, August 2011.

[47] C. Testa, D. Rossi, A. Rao, and A. Legout, "Experimental Assessment of BitTorrent Completion Time in Heterogeneous TCP/uTP Swarms," in *Traffic Monitoring and Analysis* (A. Pescapè, L. Salgarelli, and X. Dimitropoulos, eds.), vol. 7189 of *Lecture Notes in Computer Science*, pp. 52–65, Springer Berlin Heidelberg, 2012.

[48] C. Testa, D. Rossi, A. Rao, and A. Legout, "Data plane throughput vs control plane delay: Experimental study of BitTorrent performance," in *IEEE Thirteenth International Conference on Peer-to-Peer Computing (P2P)*, pp. 1–5, September 2013.

[49] A. Sanhaji, P. Niger, P. Cadro, and A.-L. Beylot, "DropTail Based ConEx Applied to Video Streaming," in *The Eleventh International Conference on Networking and Services (ICNS)*, May 2015.

[50] C. Chirichella, D. Rossi, C. Testa, T. Friedman, and A. Pescape, "Passive bufferbloat measurement exploiting transport layer information," in *IEEE Global Communications Conference (GLOBECOM)*, pp. 2963–2968, December 2013.

[51] C. Chirichella and D. Rossi, "To the Moon and back: Are Internet bufferbloat delays really that large?," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 417–422, April 2013.

[52] A. Venkataramani, R. Kokku, and M. Dahlin, "TCP Nice: A Mechanism for Background Transfers," *SIGOPS Operating Systems Review*, vol. 36, pp. 329–343, December 2002.

[53] A. Kuzmanovic and E. W. Knightly, "TCP-LP: low-priority service via end-point congestion control," *IEEE/ACM Transactions on Networking*, vol. 14, pp. 739–752, August 2006.

[54] M. Arumaithurai, X. Fu, and K. Ramakrishnan, "NF-TCP: A Network Friendly TCP Variant for Background Delay-Insensitive Applications," in *NETWORKING 2011* (J. Domingo-Pascual, P. Manzoni, S. Palazzo, A. Pont, and C. Scoglio, eds.), vol. 6641 of *Lecture Notes in Computer Science*, pp. 342–355, Springer Berlin Heidelberg, 2011.

[55] N. Kuhn, E. Lochin, J. Lacan, O. Mehani, and R. Boreli, "CLIFT: A Cross-Layer InFormation Tool for latency analysis based on real satellite physical traces," in *7th Advanced Satellite Multimedia Systems Conference and the 13th Signal Processing for Space Communications Workshop (ASMS/SPSC)*, pp. 182–189, September 2014.

[56] K. Eger, T. Hoßfeld, A. Binzenhöfer, and G. Kunzmann, "Efficient Simulation of Large-scale P2P Networks: Packet-level vs. Flow-level Simulations," in *Proceedings of the Second Workshop on Use of P2P, GRID and Agents for the Development of Content Networks*, pp. 9–16, June 2007.

[57] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec, "Adding Virtualization Capabilities to the Grid'5000 Testbed," in *Cloud Computing and Services Science* (I. Ivanov, M. Sinderen, F. Leymann, and T. Shan, eds.), vol. 367 of *Communications in Computer and Information Science*, pp. 3–20, Springer International Publishing, 2013.

[58] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP." RFC 3168, September 2001.

[59] B. Braden, D. Clarkand, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance." RFC 2309, April 1998.

[60] "Reducing Interent Transport Latency (RITE)." `http://riteproject.eu`.

[61] "Workshop on Reducing Internet Latency." `http://www.internetsociety.org/latency2013`.

[62] K. M. Passino and S. Yurkovich, *Fuzzy Control*. Addison Wesley, 1st ed., September 1997.

[63] 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.

[64] 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, pp. 1–14, June 1989.

[65] `https://heim.ifi.uio.no/~naeemk/research/PIE/ns-2/`.