Performance Evaluation of Quality of Service Mecanisms in the Internet

Tigist Alemu

Académie de Montpellier

Université de Montpellier II

— Sciences et Techniques du Languedoc —

Thèse

présentée à l'Université des Sciences et Techniques du Languedoc
pour obtenir le diplôme de DOCTORAT

Spécialité :	**Informatique**
Formation Doctorale :	**Informatique**
École Doctorale :	**Information, Structures et Systèmes**

Évaluation des Performances des Mécanismes de Qualité de Service dans l'Internet

par

Tigist Alemu

Soutenue le 13 Décembre 2004 devant le Jury composé de :

Michel HABIB, Professeur, Université de Montpellier II ..... Président
Christophe DIOT, INTEL Laboratories, Cambridge ..... Rapporteur
Ken CHEN, Professeur, Université de Paris 13 ..... Rapporteur
Philippe NAIN, Directeur de recherche, INRIA Sophia-Antipolis ..... Examinateur
Martin MAY, Senior Research Assistant, Swiss Federal Institute of Technology, Zurich ..... Examinateur
Alain JEAN-MARIE, Directeur de recherche, INRIA Sophia-Antipolis/Université de Montpellier II ..... Directeur de Thèse

Académie de Montpellier

Université de Montpellier II

— Sciences et Techniques du Languedoc —

Thèse

présentée à l'Université des Sciences et Techniques du Languedoc
pour obtenir le diplôme de DOCTORAT

Spécialité :	**Informatique**
Formation Doctorale :	**Informatique**
École Doctorale :	**Information, Structures et Systèmes**

Évaluation des Performances des Mécanismes de Qualité de Service dans l'Internet

par

Tigist Alemu

Soutenue le 13 Décembre 2004 devant le Jury composé de :

Michel HABIB, Professeur, Université de Montpellier II ..... Président
Christophe DIOT, INTEL Laboratories, Cambridge ..... Rapporteur
Ken CHEN, Professeur, Université de Paris 13 ..... Rapporteur
Philippe NAIN, Directeur de recherche, INRIA Sophia-Antipolis ..... Examinateur
Martin MAY, Senior Research Assistant, Swiss Federal Institute of Technology, Zurich ..... Examinateur
Alain JEAN-MARIE, Directeur de recherche, INRIA Sophia-Antipolis/Université de Montpellier II ..... Directeur de Thèse

Université de Montpellier II (Université des Sciences et Techniques du Languedoc)
Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM)

Référence

Alemu,Tigist ■ ***Évaluation des Performances des Mécanismes de Qualité de Service dans l'Internet*** ■ Thèse de doctorat, Université de Montpellier II, 13 Décembre 2004

To Saint Mary and Rachel.

Acknowledgments

First of all, my special thought goes to my PhD advisor Professor Alain Jean Marie. I cannot find words to express to him my deep gratitude. I gratefully acknowledge his advices and constructive suggestions that drew inspiration to this work. His continuous guidance, humanity and professionalism were crucial in helping me get through the last 4 years. In particular, I will never forget all his support to help me complete this PhD thesis during the last year when I experienced the most difficult moment of my life. Without his invaluable support this thesis would not have been possible. It was an honor and enjoyable to work with such an exceptional person like him. Our collaboration enriched my experience and knowledge in computer science.

I am grateful to Christophe Diot, Martin May, Ken Chen and Philippe Nain for honoring me by being members of my jury and also for their comments and suggestions. I have also a special thought for Michel Habib and Jean Pierre Aubert who gave me the moral and support from the beginning.

I would like also to thank Séverine Bérard for her moral support, my former officemate Jérôme Palaysi for his hospitality and kindness and all other members of the LIRMM research institute that helped me carrying out my work. I would also like to thank Marie-Claire Pigière for all her support.

Special thanks to Kebadu Muluken for his encouragements and support. My successful completion of this thesis is also his.

I am also thankful to Ginette Mondoloni for her daily and valuable moral support during difficult times. She shared all these hard times with me and showed special concern more than anyone else for the successful completion of my thesis. What she has done is engraved on my heart! I owe a special dept of gratitude to Pierre Mondoloni who never stopped to help me in pursuing my studies since high school. Without his support I would never have reached this achievement.

I am also deeply grateful to Yvan Calas for all his valuable help and for almost 7 years of unforgettable good and difficult time of student life we shared. I would also like to express my happiness of experiencing the joy of being the mother of Rachel. She was my strength and one of the main sources of my motivation for the completion of this work.

Finally, I thank my parents for all they have done for me, in particular my mother for being there for me during the writing of this manuscript. I also thank my brothers and sister and friends nearby and back home in Ethiopia for directly or indirectly helping me during the course of the work.

Contents

List of Figures

List of Tables

List of Algorithms

Résumé de la Thèse en Français

Introduction

Cette thèse est consacrée à la qualité de service (QoS) dans l'Internet. Plus précisément cette thèse s'intéresse à améliorer la qualité de service offerte par des mécanismes de gestion de file d'attente au niveau des nœuds intermédiaires d'un réseau: les routeurs.

La diversité des nouvelles applications de l'Internet crée le besoin de faire évoluer le réseau vers un réseau multiservice intégrant la voix, la vidéo et les données. Or le service actuel dit "au mieux" (*best effort*) rendu par l'Internet est inadéquat pour les nouveaux types d'applications. Les services demandés par les nouvelles applications peuvent être:

- L'acheminement garanti des paquets.
- L'allocation suffisante de bande passante.
- La garantie de délai faible d'acheminement des paquets.
- La réduction des variations des délais d'acheminement des paquets (la gigue).
- La réduction du taux de perte des paquets.

La qualité de service d'un élément d'un réseau est sa capacité à offrir un certain niveau de garantie concernant la satisfaction des besoins des applications pour les services rendus.

Ainsi, pour répondre aux nouveaux besoins, IETF (*Internet Engineering Task Force*), le groupe de travail de l'Internet, a comme rôle de faire des propositions d'amélioration des protocoles actuels de l'Internet. La solution actuellement envisagée consiste à ajouter de nouvelles architectures de qualité de service (QoS) ou à modifier les mécanismes déjà existants. En plus de l'architecture dite service au mieux (*best effort*), les nouvelles architectures proposées pour satisfaire les besoins variés des applications sont IntServ (*Integrated Service*) et DiffServ (*Differentiated Service*). Néanmoins, ces architectures très complexes nécessitent l'apport des mécanismes moins complexes à granularité plus fine à savoir les protocoles de bout en bout comme TCP ou UDP, et aussi les mécanismes de QoS situés au niveau des nœuds intermédiaires du réseau. Les protocoles de bout en bout sont utilisés pour le transport des données alors que les routeurs qui servent à router contiennent

des files d'attente qui sont utilisées à absorber les rafales de données pour offrir un certain niveau de QoS. Tous ces mécanismes, allant des architectures de QoS aux nœuds du réseau, s'enchevêtrent et sont connectés les uns des autres. Ainsi, le contrôle de congestion est effectué par TCP au niveau des sources et des destinations en collaboration avec des mécanismes de gestion de file d'attente situées au niveau des routeurs. Il s'avère donc nécessaire d'améliorer les schémas de gestion des files d'attente pour que l'algorithme adaptatif courant de contrôle de congestion (TCP/IP) devienne plus efficace afin que les architectures de QoS garantissent les services qu'elles proposent face à la diversité des applications. Ainsi, le centre d'intérêt de cette thèse se situe au niveau de granularité le plus fin, c'est-à-dire au niveau des nœuds du réseau. Plus précisément, la contribution de cette thèse vise à améliorer la qualité de service fournie par les mécanismes de gestion de files d'attentes situés au niveau des routeurs.

La technique de gestion de file d'attente la plus utilisée dans l'Internet est Drop Tail. Cependant, Drop Tail a plusieurs inconvénients comme le problème du *lock-out* et le phénomène de la file (trop) souvent pleine. Le *lock-out* est le problème où peu de connexions monopolisent l'occupation de la file d'attente, et empêchent ainsi les paquets appartenant à d'autres flux de trouver de la place. Cela crée une inéquité d'utilisation des ressources entre les différents flux. L'influence de Drop Tail sur des trafics de nature en rafale illustre ce problème [75]. En effet, pendant les périodes où la file est pleine, les flux en rafale subissent des pertes consécutives entraînant ainsi une forte réduction de leur fenêtre de congestion, et diminuant alors considérablement le débit. Le deuxième problème de Drop Tail est que la file d'attente reste souvent pleine sur de trop longues périodes de temps. Cela engendre des temps d'attente très importants et mène également au phénomène de synchronisation globale, où chaque source subit des pertes de paquet et diminue par conséquent son débit d'émission sur une période supérieure à un temps de boucle (*Round Trip Time* ou RTT). Ainsi, cela produit une longue période de sous-utilisation du lien.

À l'origine de certains de ces problèmes, le trafic du réseau est de plus en plus de nature en rafale, provoquant la congestion du réseau et générant de plus grands délais et de plus grandes pertes de paquet. De grands buffers peuvent être utilisés pour absorber les rafales et maintenir un taux élevé d'utilisation du lien. Néanmoins, comme mentionné plus haut, Drop Tail génère des temps d'attente importants pour des buffers de grande taille. Cela peut induire de grands délais de gigue, ce qui n'est pas adéquat non seulement pour TCP mais aussi pour des applications audio interactives. D'autre part, des buffers de petite taille permettent de réduire les temps d'attente mais en contrepartie le lien est sous-utilisé et les pertes de paquet sont plus importantes, ce qui pénalise encore plus les flux de nature en rafale. Drop Tail fait donc apparaître un problème de compromis entre l'utilisation satisfaisante du lien et la réduction des délais d'acheminement des paquets.

Afin d'éviter l'écroulement du système de contrôle de congestion dans l'Internet, l'IETF a recommandé l'utilisation de RED (*Random Early Detection*) [43, 16]. RED est un mécanisme de gestion active des files d'attente qui permet d'obtenir un haut débit et de faibles délais (en moyenne) pour du trafic TCP, en rejetant de manière aléatoire des paquets appartenant à différents flux. Le principe de RED est basé sur une estimation, à chaque arrivée de paquet, d'une moyenne glissante de la taille de la file d'attente à l'aide de

l'algorithme EWMA (*Exponential Weighted Moving Average*) qui utilise un coefficient de pondération. Le rejet aléatoire a lieu avec une probabilité qui est fonction de la valeur de cette moyenne pondérée. Lorsque la moyenne pondérée dépasse son seuil minimal Min_{th} ("*th*" pour *threshold* ou seuil), la probabilité de rejet préventif augmente avec la moyenne pondérée jusqu'à un certain seuil de rejet maximal Max_p où la moyenne pondérée atteint son seuil maximal Max_{th}. RED rejette tous les nouveaux paquets arrivant lorsque la moyenne pondérée dépasse Max_{th} (voir Figure 1.1(a) page 29).

Le mécanisme RED a beaucoup été étudié dans la littérature. Certaines études ont proposé des modifications ou des alternatives à l'algorithme de RED afin de résoudre ses problèmes (et donc améliorer ses performances) et assurer son large déploiement dans l'Internet.

Le principal inconvénient de RED est le problème de la configuration de ses paramètres. Par exemple, [75] a montré que le choix des paramètres de RED était une science inexacte car la détermination de la meilleure combinaison des valeurs de ces paramètres est difficile. De la même manière, [34] a montré qu'il n'y a pas d'unique ensemble de valeurs des paramètres convenant pour différents types de trafic, et affirme que la bonne configuration des paramètres implique une paramétrisation globale, ce qui est difficile voire impossible à effectuer. C'est pourquoi [75, 24] se sont interrogés sur le déploiement réel de RED dans l'Internet. Par ailleurs, plusieurs études ont montré la dépendance entre les performances de RED et la bonne configuration de ses paramètres.

Il existe deux approches visant à réduire la sensibilité de RED vis à vis de ses paramètres. La première approche est basée sur des modèles quantitatifs comme dans [105, 37, 51, 52, 66, 8] permettant la prédiction de certains paramètres de RED. Pour cela, ces modèles nécessitent en entrée la distribution du RTT, le nombre de flux actifs et la valeur de la bande passante du lien faisant goulot d'étranglement. Cependant, il est très difficile d'extraire ou de déduire précisément ces informations de nature variable à partir des observations locales. En pratique, les caractéristiques du réseau ne sont pas connues à partir des routeurs RED.

La deuxième approche est une approche adaptative. Des travaux comme [32, 34, 41] ont utilisé cette approche appelée ARED (*Adaptive RED*), qui effectue une re-configuration permanente des paramètres de RED selon la charge du trafic.

Nous décrivons brièvement dans la section suivante l'approche adaptative de RED.

État de l'art sur le mécanisme RED adaptatif

Dans ce paragraphe, nous faisons un bref rappel du mécanisme de ARED décrit dans [34] et [41]. L'idée directrice de ce mécanisme est d'améliorer les performances de RED en adaptant le paramètre Max_p (le seuil maximal de la probabilité de rejet) en fonction de la charge du trafic. L'idée d'adapter Max_p est basée sur des observations concernant l'impact de la charge du trafic sur la détection précoce de la congestion (*early detection*). En effet, pour un grand nombre de connexions TCP, si les notifications de congestion ne sont pas envoyées à un nombre suffisant de connexions TCP, alors on observe une augmentation du

taux de perte de paquet car RED se comporte alors comme Drop Tail (phénomène de *buffer overflow*). Dans ce cas, quelle que soit la valeur de Max_p, le lien est grandement utilisé. Cependant, la détection précoce devrait s'effectuer de façon plus agressive. Pour cela, il faudrait augmenter la valeur de Max_p. De cette façon, les connexions TCP réduiraient leur débit de transmission, ce qui diminuerait la charge offerte et le taux de perte de paquets. Pour un petit nombre de connexions TCP, si les notifications de congestion sont envoyées à un trop grand nombre de connexions TCP, alors la charge offerte est excessivement réduite, causant une sous-utilisation du lien qui est le goulot d'étranglement. Dans ce cas, afin d'obtenir un plus grand taux d'utilisation du lien, la détection précoce doit être plus conservatrice. Pour cela, il faudrait diminuer la valeur de Max_p.

L'algorithme de ARED prend en compte ces observations en utilisant la taille moyenne de la file d'attente pour déterminer si le comportement de la détection précoce de la congestion doit être agressif ou conservateur. Une taille moyenne proche de Min_{th} indique un mécanisme de détection précoce trop agressif. Dans cette situation, les auteurs de [34] proposent de faire décroître la valeur de Max_p d'un facteur constant α. D'autre part, si la taille moyenne est proche de Max_{th}, cela signifie que la détection précoce est trop conservatrice. Dans ce cas, la valeur de Max_p est augmentée d'un facteur constant β. Cependant, si la taille moyenne oscille bien entre Min_{th} et Max_{th}, les auteurs n'adaptent pas Max_p car ils considèrent que le mécanisme de détection précoce se comporte comme ils le souhaitent (cela permet d'éviter une trop grande augmentation du taux de perte de paquet et une sous-utilisation du lien). Cette configuration de Max_p fait en sorte que la taille de la file d'attente oscille entre Min_{th} et Max_{th}. Floyd *et al.* dans [41] ont modifié cet algorithme de façon à ce que la taille moyenne oscille à proximité de la taille moyenne cible dans un intervalle bien spécifié afin d'obtenir un délai moyen prévisible. Contrairement à [34] où Max_p est adapté à chaque arrivée de paquet, [41] adapte Max_p à chaque intervalle de temps, en utilisant une augmentation additive et une diminution multiplicative (*Additive Increase, Multiplicative Decrease* - AIMD) au lieu d'une augmentation et d'une diminution multiplicatives (*Multiplicative Increase, Multiplicative Decrease* - MIMD) utilisé dans l'algorithme ARED original de [34]. Si la taille moyenne dépasse la borne supérieure de l'intervalle cible, alors Max_p est adapté par une augmentation additive une seule fois par intervalle de temps. D'autre part, si la taille moyenne est inférieure à la borne inférieure de l'intervalle cible, alors Max_p est diminué d'un facteur multiplicatif. Enfin, si la taille moyenne est dans l'intervalle cible, alors Max_p n'est pas adapté.

Tout comme dans [34], Floyd *et al.* dans [41] ont utilisé un taux de changement constant pour Max_p. En adaptant Max_p d'un facteur constant comme dans [34, 41] ARED améliore les performances du mécanisme RED original. Cependant l'idée que nous avons souhaité tester est que si nous adaptons Max_p en utilisant un taux de changement dynamique qui sera fonction des variations de la taille moyenne de la file d'attente (indiquant ainsi la charge du trafic), il devrait être possible d'améliorer non seulement les performances de RED mais également celles de ARED.

Contributions

L'algorithme PSAND

Notre travail s'est basé sur l'approche adaptative de RED décrite ci-dessus car cette approche ne nécessite aucune hypothèse sur le type de trafic et par conséquent réduit la dépendance de RED vis-à-vis des paramètres concernant le scénario de trafic (*i.e. la bande passante, le RTT et le nombre de connexions actives*). Notre travail diffère d'Adaptive RED original sur la façon dont les paramètres sont ajustés. Notre but est d'améliorer les performances de RED adaptatif du point de vue du temps d'attente et de la gigue sans pour autant sacrifier le taux de perte. Pour cela, nous proposons un nouvel algorithme nommé PSAND qui configure les paramètres de RED. Cet algorithme consiste à adapter le paramètre Max_p selon un taux de changement variable. Contrairement à [34, 41] où un facteur constant est utilisé, notre mécanisme adapte Max_p selon un taux de changement dynamique qui est fonction des changements de la taille moyenne de la file d'attente et de sa valeur par rapport à la taille moyenne cible spécifiée.

Nous avons testé notre mécanisme d'ajustement des paramètres de RED à l'aide de simulations sous `ns` [94] qui ont montré une amélioration des performances par rapport à RED et par rapport aux deux précédentes versions de RED adaptatif. En effet, les résultats ont montré une réduction de la taille moyenne et de la variance de la taille instantanée, stabilisant ainsi le taux d'occupation de la file indépendamment du nombre de flux TCP, sans augmenter et parfois dans certains cas (pour un grand nombre de flux par exemple) en diminuant le taux de perte des paquets. De plus, notre mécanisme maintient l'occupation de la file éloignée des phénomènes de *buffer overflow* et de *buffer underflow*.

Par ailleurs, nous avons observé que comme la fonction de rejet suit étroitement la dynamique du trafic, les pertes aléatoires doivent commencer le plus précocement possible. En effet, nous avons remarqué que la valeur de Min_{th} doit être la plus faible possible, et que la valeur de Max_{th} doit être grande.

Enfin, nous avons comparé avec d'autres mécanismes récemment proposés comme le *PI controller* de Hollot *et al.* [51] et montrons que notre proposition montre des résultats compétitifs et parfois meilleurs.

Interaction de FEC avec RED

Le chapitre 5 présente une étude de l'interaction de RED avec FEC (Forward Error Correction). FEC est un mécanisme consistant en l'envoi par la source de paquets portant de l'information redondante. Cette information redondante permettra, dans certains cas, de réparer les paquets perdus sans nécessiter la retransmission de ces paquets perdus.

Nous avons comparé à l'aide simulations les résultats obtenus par RED en combinaison avec FEC (RED/FEC), avec ceux obtenus par Drop Tail (DT) et FEC (DT/FEC). À notre connaissance, une telle étude n'a jamais été effectuée jusqu'à présent. En effet, FEC a seulement été étudié avec le mécanisme Drop Tail.

Nous pensons que RED pourrait améliorer les performances obtenues par des sources

UDP implémentant FEC puisque RED disperse de manière plus uniforme les paquets perdus, réduisant ainsi les pertes consécutives de paquets pour un flux donné, et rendant par conséquent les pertes plus indépendantes. Cette propriété rend *a priori* RED plus compatible avec l'utilisation de FEC. Il pourrait être par conséquent plus intéressant d'ajouter une faible quantité de redondance dans les flux FEC en présence d'une file d'attente RED afin de réduire le taux de perte de paquets pour les sources UDP. Bien évidemment, cela devra être fait sans pénaliser les sources TCP. En effet, l'ajout de redondance augmente la charge globale du réseau, ce qui a un effet néfaste sur les sources TCP puisqu'elles répondent à l'augmentation de congestion en diminuant leur debit sans pour autant que les sources UDP fassent de même.

Afin de mener à bien cette étude, nous avons considéré plusieurs mesures de performance concernant le traffic agrégé (taille moyenne de la file représentant le délai moyen passé par un paquet dans la file, variance de la taille instantanée représentant la gigue, débit agrégé) et pour un flux spécifié (taux de perte de paquet avant et après correction par FEC, longueur des rafales de perte de paquets).

Nous avons ainsi pu montrer que, bien que RED expérimente plus de pertes que Drop Tail avant correction par FEC, RED peut être plus avantageux que Drop Tail du point de vue du taux de perte de paquet après correction par FEC. Cependant nos résultats ont également montré que la combinaison RED/FEC n'améliore pas toujours la combinaison DT/FEC. Cela dépend en effet d'un certain nombre de paramètres comme le nombre de flux TCP constituant le trafic transverse, la taille des blocs FEC et le taux de redondance dans un bloc FEC.

À l'aide de la métrique du taux de perte de paquet après correction par FEC, nous avons montré que RED/FEC était plus efficace que DT/FEC pour un faible nombre de flux TCP. En effet, un grand nombre de flux TCP augmente la "rafalité" des flots et également la charge du réseau, ce qui induit une augmentation du taux de perte de paquet pour les sources UDP utilisant FEC. Nous avons également observé que si le taux de perte avant correction est trop grand, RED/FEC obtient de moins bonnes performances que DT/FEC car FEC n'est capable de réparer suffisamment de paquets perdus quelles que soient la taille des blocs FEC et la quantité de paquets de redondance. Les résultats ont également montré que le nombre de flux FEC pour lequel RED est avantageux augmente avec la quantité d'information redondante ajoutée dans un bloc FEC. Dans la situation où RED est avantageux, nous avons également vérifié que les gains en performance pour RED tels que le délai dans la file et la gigue sont conservés.

Par ailleurs, nos resultats ont montré que pour une quantité fixe d'information redondante dans un bloc FEC, l'avantage (respectivement l'inconvénient) de RED/FEC sur DT/FEC est plus important (respectivement moins important) pour de faibles tailles de bloc que pour de grandes tailles de bloc.

Dans le cas où RED/FEC obtient de moins bonnes performances que Drop Tail, la différence de performance entre RED/FEC et DT/FEC croît avec la taille des blocs FEC. Les performances de RED/FEC se dégradent avec la quantité relative de redondance contenue dans un bloc FEC.

Tous ces résultats suggèrent que si un flux UDP implémentant FEC a la connaissance

de son taux de perte avant correction, et si le traffic généré par ce flux traverse une file d'attente RED, alors il est préférable de suivre les recommandations suivantes afin d'obtenir pour RED/FEC de bonnes performance. Pour cela, nous devons être dans le cas où le taux de perte de paquets avant correction n'est pas trop grand, c'est-à-dire dans le cas où le nombre de flux TCP est faible. Nous pouvons alors augmenter raisonnablement la quantité relative de redondance contenue dans le bloc FEC sans augmenter la charge du réseau et pénaliser les sources TCP. De plus, augmenter raisonnablement la taille des blocs FEC sans augmenter le délai de réception du bloc et le temps nécessaire pour le codage et le décodage améliore les performances de RED/FEC. Nous remarquons que le taux de redondance et la taille des blocs FEC au dessus duquel RED perd son avantage est grand. Puisqu'il n'est pas pratique de choisir d'aussi grandes valeurs, il est par conséquent préférable de choisir RED/FEC que DT/FEC quand le taux de perte de paquets avant correction par FEC est faible.

Les conclusions énumérées ci-dessus ont été obtenues à l'aide de simulations sous `ns`. Elles ont également été confirmées la plupart du temps à l'aide d'un modèle mathématique.

Conclusion

Les propositions d'amélioration des schémas de gestion des files d'attente sont fondamentales pour que l'algorithme TCP/IP actuel puisse supporter la croissance spectaculaire du nombre d'utilisateurs et pour qu'il puisse se comporter efficacement en présence de nouveaux services nécessités par différentes applications. C'est pourquoi l'IETF a recommandé l'utilisation de mécanismes de gestion active des files d'attente, ce qui constitue toujours un domaine actif de la recherche dans l'Internet. Le plus connu et le plus étudié des gestionnaires de file d'attente recommandé par l'IETF pour le déploiement dans l'Internet est RED. Pour toutes ces raisons, le schéma de gestion active de file d'attente RED est le sujet de cette thèse.

D'une part, puisque le principal inconvénient de RED est la dépendance de ses performances au travers de ses paramètres, cette thèse a proposé des améliorations de l'algorithme de RED en ayant pour but d'éviter ce problème et d'améliorer la qualité de service des connexions TCP. Le choix d'un trafic TCP est motivé de la façon suivante. Une grande part du trafic dans l'Internet est encore basé sur TCP. De plus, il y a une tendance à classifier les trafics pour supporter la différentiation de service dans la prochaine génération de l'Internet et l'utilisation du protocole approprié pour la classe de service. C'est pourquoi l'étude d'un flux basé sur TCP est utile. Mais la raison principale, c'est que RED est prévu pour fonctionner avec TCP, grâce à son interaction avec le mécanisme d'adaptation de la fenêtre.

D'autre part, cette thèse a étudié le schéma RED en présence du mécanisme FEC (Forward Error Correction) en ayant pour perspective que si RED est déployé dans l'Internet, alors il serait nécessaire d'étudier son comportement lorsque l'on utilise une technique succeptible d'être utilisée pour le transport, comme FEC au dessus du protocol UDP.

De manière plus spécifique, l'amélioration de RED que nous avons proposée dans cette

thèse et que nous avons aussi présenté dans [3, 1, 2] est basée sur une approche adaptative de RED (ARED) puisque nous pensons que l'idée de base pour une solution résolvant les problèmes de RED repose sur le RED adaptatif original proposé dans Feng *et al.* [34] et Floyd *et al.* [41]. En effet, cette approche est un proposition sérieuse pour le déploiement dans l'Internet actuel car elle adapte dynamiquement les paramètres de RED sans se baser sur des hypothèses concernant le type de trafic, alors que les autres approches se basent sur les valeurs des variables du réseau données *a priori*, telles que le temps de boucle, le nombre de connexions et la bande passante du lien. Cependant, comme ces variables ne sont pas connues en pratique par les nœuds RED, il est encore difficile d'extraire ou de déduire des informations exactes de ces variables à partir d'observations locales.

Cette thèse s'est penchée sur la probabilité de rejet maximum Max_p puisque le choix de ce paramètre affecte significativement les performances de RED. Les références [34, 41] ont adapté la valeur de ce paramètre en utilisant des facteurs d'ajustement fixés qui ne reflètent pas le taux de changement de la charge du trafic. Cette thèse a montré que, en utilisant des ajustements dynamiques plus élaborés et orientés vers une valeur cible qui sont fonction de la distance vers un objectif de performance et en autorisant des modifications des autres paramètres de RED, cela offre une amélioration des performances de ARED original. Les analyses statistiques et qualitatives de cette nouvelle proposition nommée PSAND ont montré que, comparé au RED original adaptatif :

- PSAND améliore la stabilité de la taille de la file en réduisant la variance de la taille instantanée. PSAND répond plus rapidement à un changement soudain de la congestion et montre une meilleure adaptation au changement en ramenant la taille de la file vers sa taille cible.

- PSAND réalise une taille moyenne de la file plus prévisible et plus faible.

PSAND réalise l'objectif de cette thèse qui est de minimiser la variance de la taille instantanée de la file avec comme contraintes de fournir une taille moyenne cible spécifiée, sans augmenter substantiellement le taux de perte de paquet. PSAND n'augmente pas le taux de perte et même dans certains cas (lorsqu'il y a un grand nombre de flux), il le diminue. De plus, PSAND améliore le taux d'utilisation du lien : il réduit la probabilité que la file soit vide et laisse la taille instantanée de la file loin de l'état où la file est vide ou pleine.

Toutes ces améliorations de performance sont obtenues quel que soit le nombre de flux, montrant ainsi une performance plus robuste que le RED adaptatif original.

Cette thèse étudie également le choix des valeurs des paramètres seuil de la file (Min_{th} et Max_{th}) pour PSAND. D'une part, les études effectuées avec des valeurs fixes de Min_{th} et Max_{th} ont montré que :

- De faibles valeurs de Min_{th} donnent une amélioration globale des performances concernant la variance de la taille instantanée, le délai dans la file, le taux de perte de paquet et le taux d'utilisation du lien.

- Des valeurs suffisamment grandes de Max_{th} autour de $2\hat{K}_T$ ($\hat{K}_T$ étant la taille cible de la file) donnent également de bonnes performances globales. Max_{th} ne doit pas être choisi trop grand pour éviter de grands délais dans la file d'attente.

Ces résultats ont suggéré l'utilisation d'une configuration avec Min_{th} positionné à 0 et Max_{th} égal à $2\hat{K}_T$.

D'autre part, cette thèse a décrit plusieurs méthodes d'adaptation de Min_{th} et de Max_{th}, et a étudié si ces méthodes d'adaptation pouvaient améliorer encore plus les performances de PSAND. Les résultats ont montré que la configuration précédemment décrite ($Min_{th} = 0$ et $Max_{th} = 2\hat{K}_T$) donnait les meilleures performances et qu'il n'était par conséquent pas nécessaire d'adapter les paramètres Min_{th} et Max_{th}.

Cette thèse a proposé une amélioration du schéma ARED. Nous avons également comparé notre schéma avec d'autres schémas bien connus ou récemment proposés comme PI, LRED, AVQ, REM et BLUE. Cette comparaison a montré que :

- Avec une complexité moindre, PSAND donne un compromis général désiré et de bonnes performances telles qu'un faible délai dans la file, une faible gigue et une plus grande stabilité en présence d'un fort trafic. En effet, les autres mécanismes peuvent obtenir de meilleures performances que PSAND pour une certaine métrique mais perdent leur performance pour d'autres métriques. Pour une configuration du réseau avec un RTT variable, PSAND a montré de meilleures performances globales à l'exception de la métrique mesurant l'équité en terme de débit entre les flux. De plus, contrairement à tous les autres schémas, PSAND a montré de bonnes performances non seulement pour un grand nombre de flux mais également pour un faible nombre de flux.

Cette thèse ne prétend pas que PSAND est la proposition la meilleure ou optimale pour résoudre les problèmes de RED mais montre plutôt qu'il est possible d'obtenir des performances compétitives et même meilleures en gardant l'esprit de base de RED et son schéma d'origine, permettant ainsi peu de changement à son algorithme. Des schémas comme PI, REM, AVQ, BLUE et LRED ont abandonné le schéma d'origine de RED sans pour autant améliorer substantiellement (ou même parfois ne pas améliorer) les performances obtenues par PSAND. De plus, remarquons que PSAND n'a pas besoin de connaître les valeurs des paramètres du réseau.

Les résultats de cette thèse ont également montré que PSAND pouvait encore être amélioré. Nous travaillons actuellement sur la façon de choisir dynamiquement certains de ses paramètres, en particulier le choix des valeurs pour $coef$ et γ qui donneraient les meilleures performances. Cependant, nous avons observé que ces améliorations dépendaient du nombre de flux actifs dans le réseau. Ces résultats montrent que nous avons essayé d'aller le plus loin possible dans l'amélioration de l'efficacité de RED mais aller plus loin nécessite la connaissance des variables du réseau. Par conséquent, un prochain travail qui utiliserait un modèle nécessitant la valeur de ces variables en entrée améliorera PSAND.

Une autre contribution de cette thèse est l'étude de l'interaction de FEC avec RED. Pour cette étude, un flux UDP implémentant FEC est multiplexé avec de nombreuses connexions TCP au niveau d'un lien faisant office de goulot d'étranglement, et où Drop Tail ou RED seraient utilisés comme schéma de gestion de la file. Notre intuition était que FEC combiné avec RED pourrait donner de meilleurs résultats que Drop Tail avec FEC car le processus de perte de paquet de RED expérimente un nombre plus faible de pertes consécutives. Les résultats ont montré que notre intuition n'était pas toujours confirmée mais que cela dépendait plutôt de différents paramètres comme le nombre de flux TCP actifs, le taux de redondance et de la taille des blocs FEC. Selon la valeur de ces paramètres, parfois RED ou parfois Drop Tail donne de meilleures performances. Un modèle analytique ayant confirmé les résultats obtenus par simulations a également été présenté.

Il existe une myriade de propositions concernant les schémas de gestion des files d'attente. Nous ne prétendons pas que RED est meilleur que Drop Tail même si il n'y a aucun doute que RED présente dans certaines situations une amélioration évidente des performances comparé à Drop Tail. Nous ne prétendons pas non plus qu'une variante de RED est meilleure que les autres mais nous pensons plutôt que le choix d'un schéma de gestion de file d'attente doit être basé sur le besoin de service d'une application et sur le protocole de bout en bout utilisé. En effet, il n'y a pas de schéma de file d'attente qui offre de bonnes performances globales pour différentes conditions de réseaux. Chaque schéma a ses propres avantages et inconvénients. Dans la prochaine génération de l'Internet où la différenciation de service est attendue et où le trafic doit être classifié selon les nécessités de service, différents schémas de gestion de file d'attente pourraient être associés à différentes classes de service. Chaque classe de service sera assigné à une gestion de file d'attente qui conviendra le mieux à ses besoins.

Part I

Introduction

This thesis is dedicated to the QoS (Quality of Service) in the Internet. More specifically, it focuses on the improvement of the QoS offered by queue management schemes that operate inside routers in conjunction with end-to-end protocols. We will give a general description related to this issue and propose methods and guidelines to address this problem.

Since its first step in 1969 with the ARPANET network that connected 4 distant machines located at the university of Utah, the university of California at San Barbara and Los Angeles and the Stanford Research Institute, and since its spreading to Europe, Japan and Oceania in 1980, Internet has experienced a tremendous growth. the Internet Protocol (IP). In 1978, when they split TCP into two protocols TCP (Transmission Control Protocol) and IP, Cerf and Postel predicted in [23] that the simplicity of IP would allow to build fast and inexpensive gateways. IP is based on a simple concept that allows datagrams to traverse routers from the source to destination without the help of the sender and the receiver, and that only puts intelligence at end points of the network (at the sender and receiver).

Because of this success, network traffic has experienced not only a tremendous increase as the number of users and applications has increased but also has changed in nature with the emergence of new applications that impose new service requirements. Different applications have different requirements in delay, bandwidth and jitter. Some of these applications are :

- Delay-sensitive applications: real-time and interactive applications like telephony and conferencing, on-line games, multimedia. These applications have low-latency requirements but are loss-tolerant. However, even if data applications can recover from loss via retransmission, losses above 5% lead generally to very poor effective throughput.

- Loss-sensitive applications: file transfer (FTP : *File Transfer Protocol*), email (SMTP : *Simple Mail Transfer Protocol*) and web-browsing (HTTP: *Hyper Text Transfer Protocol*) applications, delivery of mission-critical information. These applications are delay-tolerant to some extent but do not tolerate unpredictable losses.

- Continuous media applications (streaming audio and video) require a fixed, usually large, bandwidth.

However, there is a price to pay for this simplicity. The reason IP is simple is because it does not provide many services. As a result, even if IP technology has shown success for traditional applications such as web, email and file transfer, it faces some limitations since it can not fulfill all the service requirements of these new emerging applications.

Indeed, even if IP relies on the upper layer transport protocol (TCP) in order to assure a reliable delivery of a packet, it can not rely on TCP to assure a timely delivery or provide any guarantees about data throughput. The basic service offered by IP has been named the “best effort” service.

To address this issue, the IETF is working on proposals to upgrade the current Internet in order to make it converge towards another network that integrates data, voice and image and provides a certain level of quality of service (QoS). However, the implementation of these mechanisms of QoS is complex which is paradoxical as compared to the concept of simplicity of IP.

A quality of service is the ability of a network element (an application, a host or a router) to offer some level of guarantee that the service requirements can be satisfied. Service requirements of applications can be :

- Reliable delivery of packets.
- Sufficient allocation of bandwidth.
- Small delay of packet delivery.
- Small delay jitter.
- Small packet loss rate.

Nevertheless, transmitting data, voice and images on the same network implies different and contradictory characteristics. For instance, as noted above, voice transmission can tolerate some transmissions errors. But the variation of the amount of data transmitted can lead to a poor audio quality and consequently to an incomprehensible audio message. On the other hand, applications such as FTP tolerate the variation of the amount of data transmitted on the network but are sensitive to transmissions errors. A network should then propose different characteristics according to the application's requirements.

In addition to the current best effort service, the different type of QoS architecture proposed to accommodate new applications demands are : the integrated services (IntServ) [54, 65] and the differentiated services (DiffServ) [30, 62].

An implementation of a QoS architecture can be composed by one of the three architectures or by a combination of the three. However, these architectures do not work by themselves. They need the cooperation of mechanisms with finer granularity. In order to deliver their services, these complex QoS architecture need to cooperate with others less complex mechanisms, such as the end-to-end protocols and intermediate routers.

End-to-end protocols such as TCP, UDP and FEC are used for data transportation. Core routers are used to improve the performance of end-to-end protocols by absorbing bursts of data and delivering QoS. All these mechanisms, from the three QoS architectures to routers, are connected to each other and depend on each other. Among all these interesting mechanisms, the focus of this thesis is located at the finest granulation, that is at core routers level. More precisely, the contribution of this thesis deals with the improvement of the QoS provided by routers. Consequently, the contribution can improve end-to-end protocols performances as well as the QoS delivered by the best effort service and others QoS architectures such as IntServ and DiffServ.

In order to locate our contribution precisely in its context, we shall begin with a general presentation of QoS architecture such as Best Effort, IntServ and DiffServ. We shall then present end-to-end protocols like TCP, UDP and FEC, since it is the interaction of these protocols with router mechanisms which has to be taken into account for a good QoS. Finally, we describe different functionalities of a router, related to the delivery of QoS.

For this introduction, we use reference materials such as [49, 59, 80, 95, 96, 100].

The best effort service

The best effort service does not propose any QoS characteristics. Indeed, it does not guarantee when nor how much data can be delivered. The traditional IP network uses this service model which is convenient for applications that do not have time constraints (and for low priority applications) like FTP and SMTP.

The integrated services (IntServ)

It is a service model [54, 65] based on resource reservation for each flow between the sender and the receiver, according to an application's QoS request. This reservation is performed by using the Resource reSerVation Protocol (RSVP) [17] that works in parallel with TCP or UDP.

The IntServ architecture offers two different levels of service:

- A *guaranteed service* (GS or IntServ guaranteed) which proposes the delivery of data with a guarantee to bound the end-to-end delays and packet loss rate, and also to ensure the availability of bandwidth.

- A *controlled load* (CL or IntServ controlled) which proposes a service which is better than the best effort service. Under network conditions with small load, it is equivalent to the best effort service. It does not provide guarantees on end-to-end delay and bandwidth.

For both guaranteed and controlled load service, a non conforming (out-of-specifications) traffic is treated like non-QoS best effort traffic.

The IntServ service model is used for high priority applications with strong time constraints and applications that are highly sensitive to end-to-end delay and delay jitter such as real-time applications, video conference, voice, etc.

For example, the IntServ service can be compared to the postal service that gives a guarantee to deliver a mail within a certain amount of time (e.g. DHL, Chronopost, Fedex). IntServ can be used by service providers networks that interconnect private companies networks.

Even though IntServ provides QoS guarantees for individual application sessions and ensures a high resource utilization, its implementation is complex since it relies on resource reservation that needs routers to maintain state information of allocated resources for each

individual application session. It also necessitates the exchange of signalling information between routers. Router manufacturers are reluctant to implement this type of evolution. Moreover, the router should identify the flow corresponding to every packet passing through the router. Maintaining states about the network resources seems infeasible for a large network interconnecting a very large number of nodes. This scalability problem of IntServ has lead the IETF to propose another QoS architecture called DiffServ.

The differentiated services (DiffServ)

This service model [30, 62] can be used for applications having standard time constraints (e.g. transactional or interactive applications) and less priority than those that need the IntServ service model. This service model can be convenient to private companies networks. The differentiated service can be compared to a postal service where a an express mail is delivered before a normal mail which is less costly. However, both mailing systems do not provide a guarantee for a correct delivery and for the time needed for delivery.

In the case of congested networks, where some packets have to be eliminated, this service model tends to make a decision on which packet should be dropped by providing a way to specify packets priority. The priority can be established according to the importance of the data or according to the offered price for the service. Those who pay more can get better services. This allows to vary the billing of a service according to the service offered. For this purpose, DiffServ classifies the network traffic. Every class of traffic is identified by a value of the field TOS (*Type Of Service*) of the IP packet header or a combination of this value and that of other fields of the header (adresses, protocol..). This classification that assigns a level of priority is a complex operation and is performed at boundary (or edge) nodes. The task of the central routers is only to deliver packets according to the priority tags contained in the TOS field of the IP header. The field TOS of the IP packet header also called DS (*Differentiated Service*) is composed by two fields. The DSCP (*Differentiated Service Code Point*) coded with 6 bits and the CU (*Currently Unused*) field of 2 bits currently unused. The DSCP field determines the PHB (Per-Hop Behavior) that a packet will receive. The PHB corresponds to the description of the routing behavior of a router. A simple example of PHB consists in guaranteeing that a certain class of traffic gets more than x % of outgoing link bandwidth or packets belonging to a certain class are first served before others.

These edge router's task is to perform per-flow traffic conditioning (policing, marking, dropping) and to tag packets as in-profile and out-profile according to the *Service Level Agreement* (SLA) between the ISP (*Internet Service Provider*) and the application. The traffic is measured and it is checked if it is conforming to the service level agreement (meter). The traffic is then marked, that is, a DSCP is assigned to it. The next step is to shape it, that is to control if the throughput exceeds the one specified in the SLA. The excess is dropped by the dropper. The interior routers (core routers) perform per-class traffic management. They buffer and schedule incoming packets based on the tag of the packets. In-profile packets get preference (priority).

There are currently two standard PHB's representing the service levels (traffic classes):

- The *Expedited Forwarding* (EF, premium) offers a premium service that accelerates the treatment of packets and guarantees to minimize delay, jitter and packet loss rate. It provides the highest level of aggregate quality of service. In order to ensure this service,

  The aggregate traffic should be conditionned (via policing and shaping) so that its arrival rate at any node is always less than that node's configured minimum departure rate. Router should have limited buffer size and perform priority queueing. The output traffic should also be shaped so that it meets the SLA. Moreover, this kind of traffic should only represent a small portion of the total traffic.

- The *Assured Forwarding* (AF): this service delivers packets with a high probability. But excess traffic is not delivered with a high probability as compared to in-profile traffic which is in compliance with the SLA. The AF traffic has four services classes. Within each class, there are three levels of drop precedences for packets. For example, the "Olympic Games" service is composed of three classes: the AF_1 class (bronze), the AF_2 class (silver) and the AF_3 class (gold). For each class priority, levels 1, 2 and 3 can be assigned.

  A router should allocate a minimum of forwarding resources, buffer space and bandwidth for each class. A packet of a lower level of drop precedence has a higher probability of being delivered. In case of congestion, the router discards in priority packets with a higher level of drop precedence and keeps those with a lower level of drop precedence. In order to reduce the congestion for each class of traffic, an appropriate queue management mechanism is used by the router. There exists several queue management (QM) schemes. We will discuss about them briefly below and describe in detail well-known QM schemes. We will further study them in the following chapters since it is the main subject of this thesis.

The DiffServ architecture is less complex than the IntServ architecture since it does not need to store per flow requirement informations, and since the complex operation of traffic classification is performed at edge routers. The task of interior routers is only to forward packets according to their priority tag.

Even though there is a need to introduce differentiated services in the Internet, the best effort service still continues to be dominant in the current Internet and might be useful in the next generation Internet. Hence, enhancement proposals to the current Internet, such as the efforts to improve queue management schemes are necessary so that the current TCP/IP algorithm behaves efficiently in presence not only of best effort services but also of new emerging services needed by the Internet community. Indeed, queue management mechanisms operating inside routers, play an important role in controlling network congestion, in conjunction with the adaptive algorithm of the TCP protocol used at network end-points.

In the following we will describe briefly the end-to-end protocols such as TCP and UDP. We also describe another mechanism called FEC (Forward Error Correction) that can be used to increase reliability of a protocol.

End-to-end protocols

The transmission control protocol (TCP)

The Transmission Control Protocol/Internet Protocol (TCP/IP) was first referenced in 1973 in the notes of Cerf [22] and was specified in detail in subsequent works. The TCP protocol was split into two protocols, the TCP and IP protocols in 1978 by Cerf and Postel [23]. TCP and IP were adopted by the U.S. Department of Defense (DOD) in 1980. In 1983, a number of different networks connected to the Advanced Research Projects Agency (ARPA) net switched to TCP/IP. The network of networks so connected split to MILNET which interconnects various military-related networks and to the Internet for other sites. Internet became an international wide area network that uses TCP/IP to connect government and educational institutions across the world. After 1995, when the National Science Foundation (NSF) who was in charge of Internet started to give authority to private companies, TCP/IP began to be used widespreadly on commercial and private networks. Initially offering few basic services such as file transfer, electronic mail and remote log-on, TCP is currently the most predominantly used transport protocol in the IP-based communication networks. IETF RFC 793 [88] defines TCP. It is used for application like SMTP, Telnet, HTTP and FTP.

TCP/IP is composed of the following layers:

- The IP layer: its function is to route each data packet from node to node, based on the IP number, a four bytes destination address. Ranges of IP numbers are assigned to different organizations by the Internet authorities. The organizations assign groups of their numbers to departments. IP operates on gateway machines that forward data from department to organization to region and then around the world.

- The TCP layer: its function is to guarantee the correct delivery of data to the destination by detecting, retransmitting lost packets and performing flow control.

Certain applications like FTP need to transmit a large amount of data. Since packets can be lost or duplicated, transferring an important volume of data using a non connection-oriented service can become a tedious task. Indeed, implementing error detection and correction mechanisms for every application can be a very complex operation. At a lower level, the IP layer offers a non connection-oriented service as it only forward IP packets to the destination through multiple routers. In conjunction with this service, the use of a reliable packet delivery service is therefore a necessity. TCP is designed to offer such services.

TCP is a connection-oriented protocol that offers a reliable, byte streaming and full-duplex services :

- Before any exchange of data between two end hosts, a connection must be established. Once the connection is established, data can be transfered between the client and

the server in full duplex allowing a simultaneous flow of data in both directions. The sender writes a stream of bytes into the send buffer and sends it through the connection. The receiver reads this stream of bytes from its receive buffer. At the TCP layer this stream of bytes is assembled to constitute a packet called TCP segment.

- TCP gives a guarantee for the delivery of the sent data by using sequenced acknowledgments and retransmitting unacknowledged data after the expiration of some timeout interval or after the reception of a triple duplicate ACK (see below). TCP also guarantees that packets will be delivered in the same order in which they were sent. For this purpose, it uses the sequence number of each byte transmitted and stored in its receive buffer, to reorder the data (segments) that are out of order and eliminate duplicate segments.

Acknowledgment and retransmission mechanisms

In order to guarantee the communication, TCP uses acknowledgment and retransmission mechanisms. The TCP source waits for an acknowledgment (ACK) from the destination confirming the correct delivery (without loss or corruption) of the sent segment. It is only after then that the source sends the next TCP segment or gives room for the other data in its buffer. If it receives a duplicate ACK confirming that the destination has received a corrupted TCP segment, it retransmits the copy of the segment. In a case where the sent segment or the ACK or the duplicate ACK are lost, the TCP source might wait for an answer indefinitely. To avoid this situation, TCP maintains a retransmission timer for each segment sent. It detects the loss of the sent segment if the destination did not send an ACK to the source before the expiration of the timer. The timer expires if it exceeds the retransmission timeout (RTO). In this case, TCP retransmits a copy of the lost segment with a new timer. Hence, TCP continues to retransmit this way the lost or corrupted segment until the reception of a corresponding ACK. The value of the RTO is based on the estimation of the round trip time (RTT). It should be set greater than an RTT in order to accommodate delays due to the link propagation delays, the transmission delay, the ACK generation time, the header processing time and therefore avoid premature retransmissions. On the other hand, the RTO should not be set too large as this would lead to large delays in particular when the loss rate is high. Hence, the computation of the RTO should be accurate and responsive to variable delays in networks. It is initially described in [88] and refined in [61] and [55].

Its computation is based on the estimation of the RTT by the sender for each connection:

$$RTT_{\text{estimated}} \leftarrow (1-\alpha) \times RTT_{\text{estimated}} + \alpha \times RTT_{\text{sampled}} \ .$$

$$RTO = RTT_{\text{estimated}} + 4 \times \text{deviation} \ ,$$

where:

$$\text{deviation} \leftarrow (1-\alpha) \times \text{deviation} + \alpha \times |RTT_{\text{sampled}} - RTT_{\text{estimated}}| \ ,$$

and where a typical value of α is 0.125 .

Flow control and sliding window

Under the simple acknowlegment technique, the sending host will only send one segment of data after each acknowledgment, even if the receiver's buffer size is far larger. In order to increase the data throughput, TCP uses the sliding window mechanism that allows the sending of multiple segments before waiting for an acknowledgment and without the need for each segment to be individually acknowledged. Upon the reception of one ACK, the sender can send a number of additional segments depending on the size of receiving host's buffer size.

Another role of the sliding window mechanism is to perform a flow control that avoids a slow receiving host being swamped by a faster sending host by regulating the amount of data being sent to the destination. The size of the window is dynamically adjusted depending on the buffer space of the receiver. TCP maintains a limited amount of buffer space for each connection: the send buffer that stores data waiting to be sent and the receive buffer that stores the data delivered to the destination but not yet read by the application.

Congestion control

The flow control mechanism addresses the problem of buffer overflow at the receiver level but not at intermediate routers. Indeed, due to an increase of traffic (congestion), intermediate routers experience high queueing delay. In this case, mechanisms of retransmissions by timeout can react by retransmitting the highly delayed or lost packets. This will make worse the situation. Thus, to overcome the problem of buffer overflow at intermediate nodes as well as at the receiver level, TCP performs a congestion control by adjusting the transmission window (*cwnd* for congestion window) of the sender whenever a congestion is detected *i.e.* when the retransmission timer expires.

The window size W is defined as follows:

$$W \ = \ \min(cwnd, advertizedWindow) \ ,$$

where $advertizedWindow$ is the window size advertised by the destination using the ACKs.

To adjust this congestion window, the TCP algorithm is composed of three phases:

- The slow start phase: after a loss detected by timeout.
- The fast recovery phase: after a loss detected by fast retransmit (triplicate ACK).

- The congestion avoidance phase: in all other cases.

RFC 2581 [4] describes different congestion control algorithms proposed.
The original algorithm is described as follows :

- When a TCP connection begins, the slow start algorithm initializes the congestion window (*cwnd*) to 1 and the slow start threshold (*ssthresh*) is initialized to 64 kbytes by default.

  While there is no loss event or *cwnd* does not exceed *ssthresh*, cwnd increases by one segment for each reception of an ACK:

  $$cwnd \;=\; cwnd + \text{segment size}\ .$$

  This actually results in an exponential increase for *cwnd*.

  After the return of the first ACK, the window can sent 2 segments. The reception of an ACK for these 2 segments increases the window size to 4 and so on until it reaches the advertised window size of the receiver. In the absence of congestion and in case of lower queueing delay, the congestion window increases very quickly. Figure 1 illustrates this increase. After each RTT the window size can double its size. This exponential growth can lead to congestion.

  For the first TCP implementation (TCP Tahoe), losses are detected by timeout only. For the TCP Reno losses are detected by three duplicated ACK. When a loss is detected or *cwnd* exceeds *ssthresh* then the slowstart phase ends and the congestion avoidance phase starts.

- In the congestion avoidance phase, if *cwnd* is greater or equal to *ssthresh* then *cwnd* increases linearly in an additive manner after every ACK received. Note that in the congestion avoidance phase *ssthresh* and *cwnd* remain equal.

  Additive increase of *ssthresh*: for every useful acknowledgment received, compute:

  $$\begin{aligned} ssthresh &= ssthresh + (\text{segment size})^2/ssthresh\ , \\ ssthresh &= \min(ssthresh, \text{maximum window size})\ . \end{aligned}$$

  However, once a timeout is detected, *ssthresh* is divided by two and *cwnd* is set to 1:

  $$\begin{aligned} ssthresh &= cwnd/2\ , \\ ssthresh &= \max(ssthresh, 2 \times \text{segment size})\ . \end{aligned}$$

  As a combination of these two rules, the slowstart phase is re-entered in order to avoid burst of retransmissions.

  TCP uses an AIMD (Additive Increase for the congestion window and Multiplicative Decrease for *ssthresh*) algorithm to control congestion.

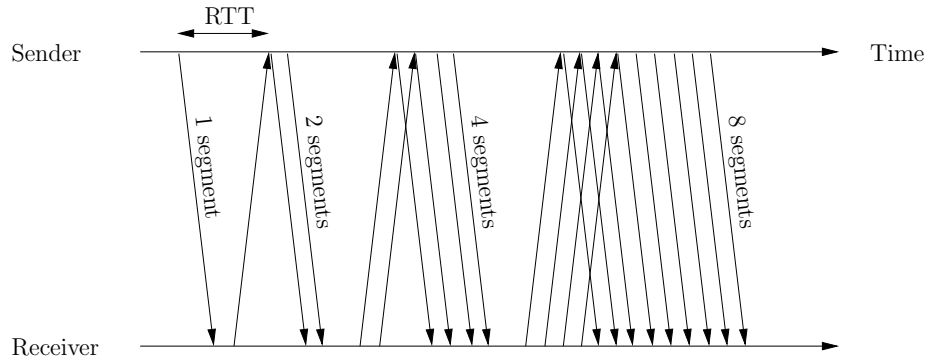


Figure 1: TCP slow start phase

The full specification for TCP involves fast recovery. The fast retransmit algorithm has first appeared for the Tahoe release whereas the fast recovery has appeared for Reno. Therefore, a different mechanism is added for every loss detected by fast retransmit:

- When a loss is detected by fast retransmit (triplicate ACK), then apply the multiplicative decrease for $ssthresh$ *i.e*:

$$\begin{aligned} ssthresh &\leftarrow cwnd/2 \ , \\ ssthresh &\leftarrow \max(ssthresh, 2 \times \text{segment size}) \ . \end{aligned}$$

- Then enter in fast recovery phase until the loss is repaired. When entering this phase, temporarily keep the congestion window high in order to keep sending. Indeed, since an ACK is missing, the sender is likely to be blocked, which is not the desired effect:

$$\begin{aligned} cwnd &\leftarrow ssthresh + 3 * \text{segment size} \ , \\ cwnd &\leftarrow \min(cwnd, 64KB) \ . \end{aligned}$$

- For every received ACK, at least until the lost is repaired, the exponential increase mechanism is run *i.e*:

$$\begin{aligned} ssthresh &= ssthresh + (\text{segment size})^2/ssthresh \ , \\ ssthresh &= \min(ssthresh, \text{maximum window size}) \ . \end{aligned}$$

Several subsequent proposals of TCP variants like TCP Vegas [18, 71], TCP SACK [72], TCP NewReno [4] modified the slow start, the congestion avoidance algorithms or the ACK format required in order to improve the congestion control mechanism of the original TCP Tahoe.

Mathematical models of TCP

The reactive nature of TCP makes it difficult to predict the throughput of a TCP connection. Several authors [21, 52, 71, 72, 73, 79, 83] have proposed models resulting in a formula for computing the throughput as a function of network parameters: the "square root formula" described in [73].

Consider a TCP connection with constant round trip time RTT and constant packet size MSS. Assume that the network is stationary, that the transmission time is negligible compared to the RTT, that losses are rare (less than 2%) and that the time spent in slow start or fast recovery is negligible. Then the average throughput BW computed by the following relation:

$$BW \;=\; \min\left(\frac{MSS \times c}{RTT \times \sqrt{p}}, \text{link capacity}\right) \;, \tag{1}$$

where:

- BW is the bandwidth/throughput of the connection.
- MSS is the maximum segment size.
- RTT is its round trip time.
- p is the packet loss probability.
- c is a constant that depends on the acknowledgment strategy used (delayed or every packet) as well as on whether packets are assumed to be lost periodically or at random [73]. It is often taken as $\sqrt{3/2}$.

Further reference materials on TCP modeling can be found in [28, 46, 49, 84, 98, 99].

UDP protocol

The *User Datagram Protocol* (UDP), defined by RFC 768 [87], provides a simple, but unreliable message service for transaction-oriented services. UDP gives a datagram-oriented service *i.e.* it does not transmit a stream of data but rather already segmented data into datagrams. Another difference with TCP, is that it does not use acknowledgments, retransmission and flow control mechanisms. As a result, datagrams can be lost without the sender knowing it, and the receiver can experience buffer overflow.

The motivations factors for the use of UDP in networks are:

- It is the simplest transport protocol since there is no need to keep states about the source and destination.

- It is not a connection-oriented protocol, therefore it does not have delays for connection establishment and termination like TCP. The gain can be substantial when the amount of data is small, or the RTT is large. In particular, this is the case for long distance communications such as satellite TCP/IP networks.

- Since UDP does not perform the retransmission of lost packets and avoids as a result long delays, it is convenient for real-time multimedia applications with strong time constraints.

- It contains a small header containing the source port number that identifies the application on the sending machine, the destination port number that identifies which application is to receive data, the length field indicating the amount of the data and the checksum field that allowed to identify corrupted packets. A TCP segment has 20 bytes of header overhead per segment, whereas UDP only has only 8 bytes of overhead.

- It can transmit data as fast as it desires since there is no feedback due to congestion control. UDP segments can be lost and can be delivered to the destination while being out of order.

- UDP does not have the scalability problem since it is connectionless. An application that uses UDP can send data to a large number of receivers. For this reason, UDP can be more suitable for multicasting.

For applications that can tolerate a small fraction of packet loss and that reliable data transfer is not absolutely necessary, UDP can be chosen instead of TCP. Hence UDP is commonly used for interactive real-time applications, such as video conferencing, for applications such as NFS (*remote file server*), Internet telephony, SNMP (*network management protocol*), DNS (*domain name translation*) and RIP (*routing protocol*).

Forward error correction (FEC)

As an alternative to TCP, FEC (*Forward Error Correction*) has been proposed. FEC is an open-loop technique which consists in adding to packets a redundancy which is exploited by the destination to recover from losses without requiring packet retransmission. This extra information is computed using error correcting codes. FEC is well suited for real-time applications which can tolerate few losses. However, FEC has two antagonistic effects: on the one hand the redundancy generated by the source increases the overall load of the network resulting in an increase of the loss rate; on the other hand the redundant information generated by FEC helps to recover from a part of the losses. Therefore, the use of FEC is interesting only if the loss rate increase can be compensated by a greater correction of losses.

In conjunction with these end-to-end mechanisms, intermediate routers should participate in improving the offered QoS.

QoS inside intermediate routers

One goal of Internet QoS is to control packet loss. Packet loss occurs mainly for two reasons. The first reason is the damage of packets in transit. However packet loss due to this phenomenon is rare($\ll 1\%$). The second cause for packet loss, network congestion, occurs more frequently.

In order to control network congestion, the adaptive algorithm of the TCP protocol is used in conjunction with queue management mechanisms operating inside routers. Our work focuses on improving the quality of service offered by intermediate routers. Routers that support QoS have the following four functions:

Classification

The classification of packets can be done based only on the TOS field of the IP packet header or on additional fields such as the IP address and port number of the sender and receiver or the type of document. The use of classification is to determine the incoming packets output interface as well as the particular buffer needed for storage, and to allocate an output bandwidth.

Policing and marking of traffics

These policies are used to check if the input traffic is in compliance with the expected profile. The expected profile is determined by the SLA. If it is a non-conform traffic, then packets issued from this traffic are discarded or marked. The marking function allows to transmit non conform packets in absence of congestion. In case of heavy traffic load, these packets are discarded. Another method is to discard among these packets those that have less priority. The traffic shaping policy delays the excess traffic as compared to the expected traffic so as to forward it later. This method shapes the traffic but introduces additional delay. There exit two methods, namely the leaky bucket and token bucket mechanisms, to measure a traffic and to determine its compliance with a SLA. More precisely, these two algorithms shape or mark the traffic.

Once the classification, policing and shaping of a traffic are performed, the traffic is directed to the buffer where queueing disciplines take a decision on the selection of the next packet to be dropped. We describe in the next section some queueing disciplines.

Queue management

The TCP flow control and congestion avoidance algorithms adjusts the source sending rate according to the destination reception capacity. This end-to-end congestion control is not sufficient since congestion can also occur when the traffic bandwidth requirements exceeds the transmission capacity of the gateway. Buffers are used to absorb this excess of traffic. But due to the buffer space limitation, routers drop or mark the excess of

traffic and suppose that the upper layer end-to-end protocols like TCP will detect the congestion and will respond by limiting their transmission rate. Routers can not leave all the congestion control task to TCP. Because without queue management schemes, packet drops and consequently lost packets retransmissions increase, leading to a higher level of congestion.

By dropping packets when it is necessary or appropriate and thus controlling the queue length to absorb bursts, queue management schemes are forced to play an important role in the congestion avoidance and control along with end-to-end TCP congestion control mechanism. Queue management schemes differ on their methods of selection of the next packet to be dropped or marked. Two categories of queue management mechanisms can be distinguished:

- The passive queue management (PQM) schemes decide to drop packets only when the queue reaches its capacity or a certain specific value.

- The active queue management (AQM) schemes decide to drop packets preventively without waiting for a buffer overflow.

We describe in the following some schemes classified in these two categories.

Passive queue management (PQM) schemes

Drop Tail, Drop-from-Front and Push-Out are well known passive queue management schemes:

- The *Drop Tail* mechanism accepts all incoming packets until the buffer reaches a certain threshold (eg: the buffer capacity). Once the threshold is reached, Drop Tail drops from the tail of the queue all incoming packets. Buffered packets wait for their service time without a risk of being dropped.

- The Drop-from-Front mechanism [69] differs from the Drop Tail scheme by dropping the packet queued in the front of the queue when the queue reaches a certain threshold. By dropping the packet buffered earlier, Drop-from-Front allows the TCP congestion avoidance mechanism to detect the packet drop earlier than can does Drop Tail.

- The Push-Out mechanism [93] differs from the Drop-Tail and Drop-from-Front schemes by deciding to drop the latest packet buffered and replacing it by the newly incoming packet. Since the latest packet buffered is pushed out from the queue, this scheme wastes the router resources. This scheme is also more complicated than the other two schemes.

All these schemes present some limitations in presence of heavy congestion since they do not sent early congestion notification to the TCP-based applications at the source so that these sources reduce their sending rate. It is only when the buffer overflows that TCP senders are notified of congestion. Moreover, since all incoming packets are dropped, all TCP senders are constraint to back-off their transmission rate which leads to a global synchronization of the sources. Another drawback of these schemes is the unfairness problem between connections. Certain applications with high bandwidth requirements can monopolize the buffer space. For instance, when UDP and TCP flows compete for the same bandwidth and buffer space, the UDP flows monopolize the buffer space since they do not respond to packet drops whereas TCP sources respond by reducing their sending rate. Another example is the unfairness observed between highly bursty and less bursty traffics.

The full queue problem is another drawback of PQM schemes. As packets are not dropped in a preventive manner, the queue tends to be usually full. This leads to an important queueing delay and therefore to an important end-to-end delay, and also diminishes the routers burst absorption capacity. These performances are unacceptable to certain types of applications, in particular to delay-sensitive applications.

In order to solve the problems of the PQM schemes and improve the QoS of connections, the links must play an *active* role in the congestion control. Active queue management schemes are designed under this perspective [16].

Active queue management (AQM) schemes

Unlike PQM, active queue management (AQM) schemes drop an incoming packet in a preventive manner with a probability that depends on the congestion level. Instead of detecting the congestion by buffer overflow, AQM schemes detect congestion based on the queue length, the packet arrival rate or packet loss rate, and increase the drop probability with the congestion measure. These preventive packet drops induce implicit congestion notification to senders that react by reducing their TCP congestion window. This way AQM control and reduce the queue length providing a low queueing delay. AQM reduce also the number of consecutive packets and achieve fairness among sources with different burstiness by avoiding the problem of lock-out observed for PQM schemes. We will describe in Chapter 1 different active queue management schemes.

In order to enhance AQM schemes, *explicit congestion notification* has been proposed in [89, 90].

Explicit congestion notification

Explicit congestion notification (ECN) [89, 39] is a technique that marks packets instead of dropping them as some AQM schemes like RED (Random Early Detection). Indeed, packet dropping generates more retransmissions that increase the queueing delay. The basic idea of ECN is to avoid this situation.

Packet that should be dropped are marked. The router can mark the two bits CU in the IP type of service (TOS, renamed to DS in DiffServ) header of the packet [89]. This

marked packet will notify the sender about the detection of an incipient congestion. The source acts as if the packet is dropped and reduces its sending rate. By applying an ECN-type policy, the router manages to save the packet, thereby enhancing the goodput while still conveying congestion notification to the sources.

Active queue management (AQM) can choose algorithms to mark (ECN type) or drop (like RED) packets. However, in case of implicit congestion notification induced by packet drops and sent by the receiver could not arrive to the sender in an appropriate time. Indeed, acknowledgment packets can be lost on their way. On the other hand, explicit congestion notification induced by ECN-type policies are sent faster and in a more reliable manner by the router to the sender. The sender could then react in time to a congestion increase.

In combination with active queue management and end-to-end congestion control, ECN is able to provide a network with practically no packet drop. This considerably increases the performance of every source.

Queue management schemes are primordial in improving the TCP end-to-end performance. They have gained more and more importance with the growth of Internet as they are necessary to improve QoS such as throughput, delay, jitter and loss. Active queue management is still an active area of networking research. For all these reasons, our thesis focuses on this topic.

Queue scheduling

Queueing disciplines and queueing scheduling work side by side to manage network traffic at the router level. We have seen that queueing disciplines struggle to avoid congestion by dropping the selected packets. However if packets are waiting a long time for transmission, and packets arrive faster than they can be served, then the queueing delay increases introducing congestion. To avoid this situation, the queue scheduling scheme of the router executes its part of the task by selecting the next packet to be served.

Since our focus is located at the router level, in order to have a wider and complete view about router mechanisms, we will bring this introduction to an end by a review of the existing main queue scheduling mechanisms even though we do not study these mechanisms in this thesis.

This section describes how some queue scheduling mechanisms select the next packet to be dropped.

FIFO

FIFO (*First In First Out*) queueing, is the default and most widely used queue scheduling scheme. It consists in forwarding the first packet arrived at the queue *i.e.* packets are served in the same order that they have being received in the buffer. The advantage of the FIFO scheduling scheme is its implementation simplicity and its ability of giving a predictable maximum queueing delay equal to the queue capacity. Queueing delay would

be small if the queue capacity is small. FIFO does not make any service differentiation between different class of traffics. All flows experience the same treatment.

But facing this egaliterian service, since TCP controls its transmission rate itself by reducing in case of congestion, its sending rate and thus its share in bandwidth, non-responsive flows such as UDP flows, consume the entire buffer space.

Priority queueing (PQ)

The traffic is classified into different classes of traffic. Each class is associated to a particular queue according to its priority. Every queue implements FIFO. A packet is served only if there is no other packets with highest priority. A variant of this scheduling discipline, rate-controlled priority queueing, serves in priority packets of the class of traffic with highest priority if the amount of traffic in the high-priority queue stays below a certain threshold.

The PQ scheme is useful for a high priority traffic such as real-time applications, VoIP traffic. Like FIFO this scheme has the advantage of simplicity. However, if the proportion of high priority traffic is very important, lower priority traffics may experiment a very large queueing delay as well as a higher packet loss rate and may suffer from bandwidth starvation.

Even among high priority traffics sharing the same priority queue, a misbehaving high priority flows can deteriorate the QoS offered by the priority queue since the queue can observe large queueing delay and jitter.

Like FIFO, the PQ discipline does not solve the unfairness problem concerning the share of bandwidth between the TCP and UDP flows. Even if TCP flows are placed in the priority queue for protection against UDP flows, they will take in their turn, the highest proportion of the bandwidth and starve the UDP flows.

Fair queueing (FQ)

Fair queueing [29] is a flow-based queueing discipline that is designed to obtain a fair share of network resources among flows and to prevent bursty flows from grabbing all the available bandwidth. Every incoming packet is classified according to the flow it belongs to and is assigned to a specific queue dedicated to the particular flow. Every packet at the head of each flow-based queue are served by round robin scheduler. When the scheduler meets an empty queue, it jumps it and passes to the next queue. Each of the n non-empty queues receives $1/n$ of the total output bandwidth. This allocation changes dynamically with the number of active queues. The advantage of putting each flow in a separate queue is the possibility to protect well-behaving flows from misbehaving flows. Only queues of the misbehaving or highly bursty flows will be influenced. One of the drawbacks of FQ is its incapacity to fulfill the different bandwidth requirements of different flows. Indeed, FQ is supposed to allocate fairly the same amount of bandwidth to different flows. However, the amount of bandwidth received by each flow depends on the size of packets. Flows with predominately larger packet sizes receive a higher amount of bandwidth. Another drawback is its incapacity to support real-time applications like VoIP.

Class-based fair queueing (CBFQ)

The traffic is classified according to the service requirement of the flows. For each service class, a desired amount of bandwidth is allocated. Within a given class of service, packets are again classified according to the flow to which they belong. Each flow of a given service is associated to a particular queue. To each of the n queues of a particular class of service is assigned fairly $1/n$ of the total bandwidth, and served like FQ, according to a round robin algorithm.

For instance, assume 3 VoIP flows are assigned 20% of the total output bandwidth. Each of these flows receive 1/3 of the 20% of the allocated bandwidth if the other 80% of the bandwidth are allocated to 8 other IP flows, each of these flows will receive 1/8 of 80% output bandwidth.

Weighted Fair queueing (WFQ)

WFQ [85] is designed in order to improve FQ performances in particular offering a service differentiation in terms of bandwidth and satisfying different bandwidth demands of applications. To achieves this goal, WFQ shares the network resource between flows based on their bandwidth requirements.

The weighted fair queueing gives preference to less bursty traffics with less bandwidth requirements. In order to reduce the queueing delay, WFQ first serves these kind of traffics and shares fairly the rest of the bandwidth to the most bursty traffics that are big consumers of bandwidth. Each queue is assigned with a configuration share of bandwidth.

Unlike FQ, WFQ tries also to allocate bandwidth fairly to packets with variable sizes, so that larger packets do not monopolize a large amount of bandwidth. To achieve this goal, WFQ takes into account the packets length during packets service by transmitting one bit at a time from the head of the packets and from different queues according to the weighted bit-by-bit round-robin (WRR discipline. The packet reassembling is determined by the packet's finish time, that is the order in which the last bit of each packet is transmitted.

To illustrate this theory, lets consider the example given by [59]. Suppose flow 1 is assigned to queue 1 with 50% of the output bandwidth. Flow 2 is assigned to queue 2 with 25% of the output bandwidth and flow 3 to queue 3 with the rest 25% of bandwidth. The WRR scheduler serves two bits of the 600 bytes packet from queue 1, one bit of the 350 bytes packet from queue 2 and one bit of the 450 bytes packet from queue 3, and returns to queue 1 for the same operation. This causes the 600 bytes packet to finish first the transmission and reassembly before the others, and the 350 bytes packet before the 450 bytes packet.

This WRR scheduling discipline is approximated by directly assigning a finish time to each packet. Packets will be served according to the finish time. A packet with the earliest finish time will be served first. The computation of the finish time of a packet is based on the total output bandwidth, the number of active queues, the weight assigned to each queue and the size of the packet. The WFQ can be deployed at the edges of the network to provide a fair bandwidth allocation with a bounded delay for different classes

of service. However, WFQ's bandwidth allocation for variable length packets introduces a computational complexity that leads to a scalability problem, that is to the incapacity to support large number of service classes. Other variants of WFQ such as class-based WFQ, self-clocking fair queueing (SCFQ) [48] and worst-case WFQ have been proposed.

Class-based queueing (CBQ)

The class-based queueing (CBQ) scheduling discipline [44] is designed to improve the performance of FQ and PQ disciplines. Unlike FQ and similar to PQ and WFQ, CBQ classifies flows into different queues and assigns different amount of bandwidth to different queues according to the flow bandwidth requirements. Unlike PQ, CBQ does not allow bandwidth starvation of low priority queues since at least one packet from each queue should be served.

CBQ classifies packets according to the service requirements (eg: real-time or file transfer services). Each created class of service is assigned to a queue. CBQ allocates a weighted bandwidth for each queue. Higher bandwidth queues can be served several times in a single service round if only one packet is served in a round. Otherwise, they can send more packets in a single service round as compared to lower bandwidth queues. Packets are served using the round robin scheduler.

In order to control congestion, the CBQ mechanism prefers to avoid bandwidth starvation that is resource denial instead of resource reduction. One of the drawback of this mechanism is that to determine the amount of bandwidth associated to each service class, the knowledge of the mean packet size is required. When the size of the packets are variable, CBQ can not support the configured amount of bandwidth.

In the rest of this thesis, we studied the buffer management schemes we have just described, in particular the RED active queue management scheme.

Thesis organization

This thesis is organized in three parts.

Part I: An overview on queue management schemes

This part describes well-known active queue management schemes. We focused on the RED scheme and its variants.

Part II: Configuration of RED parameters

This part is composed by three chapters.

Chapter 2: Adaptation of the parameter Max_p

This chapter proposes a new active queue management scheme named $PSAND$ and shows its performance.

Chapter 3: Configuration of the parameters Min_{th} and Max_{th}

This chapter investigates different methods for setting other parameters of our proposed scheme. We investigated fixed and adaptive methods for this setting.

Chapter 4: Comparison of PSAND with actives queue management schemes.

This chapter is dedicated to the comparison of PSAND with others well known AQM schemes.

Part III: FEC under RED queue management scheme

This last part deals with the study of the RED queue management scheme in presence of the FEC end-to-end error control technique.

Part II

An Overview on Queue Management Schemes

Chapter 1

RED and some of its variants

Contents

1.1 Introduction

The most widely used queue management scheme in the current Internet is Drop Tail. However Drop Tail has two important drawbacks namely the lock-out phenomenon and the problem of maintaining full queues for long periods of time [16].

By keeping a maximum queue occupancy for long periods of time, Drop Tail causes not only a high queueing delay but creates also the problem of global synchronization where all sources experience packets drops that reduce their sending rate for a period of more than a round-trip-time. The phenomenon of global synchronization further causes alternating periods of over and under-utilization of the link.

The lock-out is a problem where a few connections monopolize the queue space and prevent other flows from getting room in the queue thus causing unfairness of resources usage among flows. The bias of Drop Tail against bursty flows [75] illustrates this problem. During periods of full queues, bursty flows suffer from several consecutive drops that cause dramatic reduction of their congestion windows and that lead to a very low throughput.

Large buffers can be used to absorb bursts and maintain a high link utilization. But the Drop Tail mechanism used with a large buffer causes large queueing delays. This in turn can induce large delay jitter which is not convenient to TCP, nor to audio interactive applications. On the other hand, small buffers reduce the queueing delay but lead to a low link utilization and a higher loss rate of packets and penalizes even more bursty flows. Drop Tail gives then rise to a trade-off problem between high link utilization and low queueing delay. Since network traffic tends to become more bursty, this is a serious flaw.

In order to prevent the Internet congestion collapse, the Internet Engineering Task Force (IETF) recommended for the routers of the next generation Internet the use of RED [16, 43], an active queue management scheme which was proposed by Floyd and Jacobson in 1993. The aim of RED is to achieve high throughput and low average delay (for TCP traffic) by spreading randomly packet's drops between flows. The RED random drop function is described in the following section by equation (1.2). A detailed description of the RED mechanism will be presented in section 1.2.1.1. Even though the RED active queue management (AQM) scheme provides performance improvement as compared to Drop Tail under certain parameter settings and network conditions, it still exhibits several problems such as high delay jitter and bandwidth unfairness. A number of researchers have proposed modifications or alternatives to RED with the aim of solving its deficiencies (and then improving its performance) and ensure its wide-spread deployment. The variants bear on the modification of the control variable and/or drop function computation. Most of the proposals concentrate on adapting the marking probability (marking = marking or dropping) but use different congestion measures. The existing AQM schemes measure

congestion differently by using the input rate, events of buffer overflow and emptiness, or a combination of these factor. The queue length (or the average queue length) is widely used in RED and most of its variants (Ared Feng, Ared Floyd, GRED), use the queue length as a congestion measure. Other schemes like SRED and BLUE use the buffer overflow event or the occurrence of empty queue event.

The traffic input rate is also used in some AQM schemes such as AVQ to make them more adaptable to instantaneous traffic variance and to achieve the desired link utility. PI, REM and SFC [47] jointly use queue length and traffic input rate.

Based on the calculation of these control variables, the variants of RED can be classified into the following main categories:

- RED with aggregate control (SRED, DSRED, REM, BLUE, ...).
- RED with per-flow control (FRED, FB-RED, SFB, XRED).
- RED with class-based threshold (CBT-RED, BRED).

For the first category, the computation of the drop probability does not differ from a flow to another or from a packet to another. The drop probability is only a function of the control variable (the queue size, the input rate... depending on the variant). Although there is no active discrimination on the drop probability, there happens to be a passive discrimination, as shown for instance by [49]. More precisely, there is a bias in the drop probability between highly bursty and less bursty flows. Highly bursty flows experience a higher drop probability. In addition, because UDP flows do not back up their sending rate by using congestion notification, they experience a higher rejection rate. This effective but not intentional discrimination, can also be observed for flows with different RTT. Indeed, flows with a larger RTT have their throughput reduced.

In the second category, the variants focus on the computation of a drop probability depending on a traffic type (for example TCP and UDP traffics). This is an active discrimination on the drop probability. In this category, for certain RED variants like FRED, packets are classified according to the flow to which they belong. Each flow experiences its own drop probability as a function of the control variable.

Different RED variants compute the drop function differently. There exists variants of RED that use a linear function, others that use a step function. We now describe briefly the functioning of some of RED variants.

1.2 RED with aggregate control

1.2.1 Proposals without network scenario parameters

1.2.1.1 RED (Random Early Detection/Drop)

The basic idea behind RED is that a router detects congestion early by using the average queue size $\widehat{K}_{cur}$ as a control variable and setting two thresholds Min_{th} and Max_{th} for

packet drop. The average queue size works as a low pass filter. Its estimation is done with an Exponential Weighted Moving Average (EWMA) computed at each packet arrival at the queue as follows :

$$\widehat{K}_{cur} \leftarrow (1-\omega)\widehat{K}_{cur} + \omega K_{cur} \ , \tag{1.1}$$

where K_{cur} is the instantaneous queue size and ω is a fixed weight parameter much smaller than 1. Because $\omega \ll 1$, $\widehat{K}_{cur}$ changes much slowly than K_{cur} does. The advantage of using this low pass filter is the ability to accumulate short term congestion and to make $\widehat{K}_{cur}$ follow the long term changes of K_{cur} that reflects persistent congestion in networks.

Once the average queue size is computed, the second computational part of the RED algorithm is the drop probability. The idea is that by computing the packet drop probability as a function of the level of congestion, RED should manage to obtain a low packet drop probability during low congestion and an increased packet drop probability as the congestion level increases.

The RED drop function $d(\widehat{K}_{cur})$ uses the three parameters Min_{th}, Max_{th} and Max_p (where Max_p is the maximum random drop probability before the 100% rejection rate occurs). An average queue size in the interval $[Min_{th}, Max_{th}[$ indicates a low level of congestion and induces a small packet drop probability. In this case, packets arriving from different sources are chosen randomly and dropped. This random drop allows the queue not to be full and to have room to certain incoming packets. By avoiding frequent occurences of full queue, packets coming from different sources are not dropped simultaneously. This avoids the global synchronization of the TCP sources. An average queue size in the interval $[Max_{th}, B]$, where B is the buffer capacity, indicates a persistent and higher level of congestion and induces a larger packet drop probability.

The drop function d is defined as:

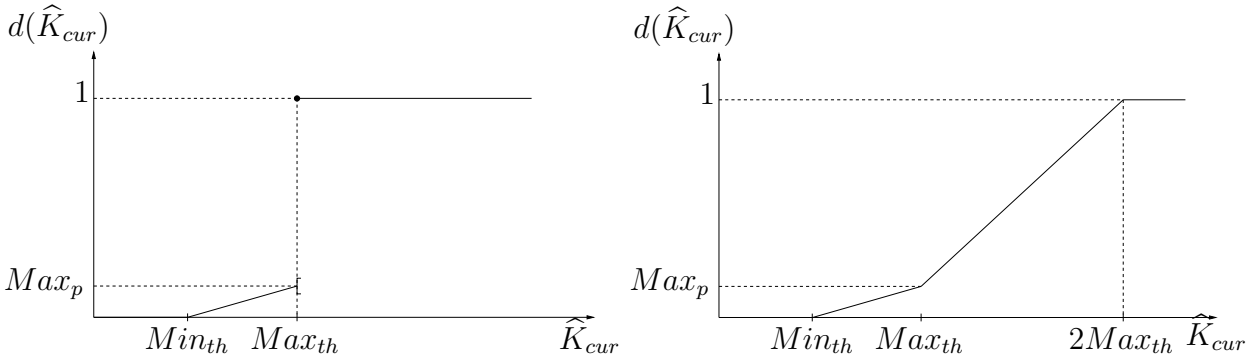
$$d(\widehat{K}_{cur}) \ = \ \begin{cases} 0 & \text{if } \widehat{K}_{cur} < Min_{th} \ , \\ \frac{\widehat{K}_{cur} - Min_{th}}{Max_{th} - Min_{th}} Max_p & \text{if } Min_{th} \leq \widehat{K}_{cur} < Max_{th} \ , \\ 1 & \text{if } \widehat{K}_{cur} \geq Max_{th} \ . \end{cases} \tag{1.2}$$

The above definition of RED represents RED in strict mode (see Figure 1.1(a)). Algorithm 1 gives the pseudo-code of RED in strict mode.

A gentle mode for RED (GRED) has been proposed. It differs from the strict mode in the following points:

- The 100% drop is triggered only when $\widehat{K}_{cur}$ is twice Max_{th}. RED in strict mode triggers a 100% drop much earlier.

- The random drop is triggered if $Min_{th} \leq \widehat{K}_{cur} < 2Max_{th}$ increasing the proportion of the random drop as compared to RED's proportion in strict mode.



(a) Strict RED

(b) Gentle RED

Figure 1.1: RED's drop function

RED in gentle mode is then defined as follows:

$$d(\widehat{K}_{cur}) \;=\; \begin{cases} 0 & \text{if } \widehat{K}_{cur} < Min_{th} \ , \\ \frac{\widehat{K}_{cur}-Min_{th}}{Max_{th}-Min_{th}} Max_p & \text{if } Min_{th} \leq \widehat{K}_{cur} < Max_{th} \ , \\ \frac{(1-Max_p)(\widehat{K}_{cur}-Max_{th})}{Max_{th}} + Max_p & \text{if } Max_{th} \leq \widehat{K}_{cur} < 2Max_{th} \ , \\ 1 & \text{if } \widehat{K}_{cur} \geq 2Max_{th} \ . \end{cases} \tag{1.3}$$

Figure 1.1(b) illustrates RED in gentle mode.

RED has a drawback: its performance is significantly sensitive to the setting of its parameters Max_p, Min_{th}, Max_{th} and ω. Different variants have been proposed with the aim to find a solution.

Algorithm 1 RED (strict mode)

/* Notations: */
/* – $\widehat{K}_{cur}$: average queue size. */
/* – Min_{th}: minimum threshold. */
/* – Max_{th}: maximum threshold. */
/* – $d(\widehat{K}_{cur})$: packet drop probability. */

⇒ **Upon packet arrival:**

Calculate $\widehat{K}_{cur}$.
if $Min_{th} \leq \widehat{K}_{cur} < Max_{th}$ **then**
  Calculate drop probability $d(\widehat{K}_{cur})$ using (1.2).
  Mark or drop packet with probability $d(\widehat{K}_{cur})$.
else if $Max_{th} \leq \widehat{K}_{cur}$ **then**
  Mark or drop packet.
end if

1.2.1.2 BLUE

BLUE is an active queue management scheme proposed by Feng *et al.* in [33] with the aim to improve RED performance in terms of packet loss rates and buffer size requirements in the network. For this purpose, the authors use directly the packet loss rate and link utilization to measure congestion. Indeed, the authors argue that even though the queue length indicates congestion, it gives very little information as to the degree of severity of this congestion. BLUE uses only a few configuration parameters:

- p, the packet marking/dropping probability.
- d_1, the increase step for p.
- d_2, the decrease step for p.
- $freeze\_time$, the minimum time interval between two successive updates of p.

The authors take d_1 significantly larger than d_2 because link underutilization can occur when congestion management is either too conservative or too aggressive, but packet loss occurs only when congestion management is too conservative.

Algorithm 2 BLUE

/* Notations: */
/* – p : probability which is used to mark or drop packets. */
/* – $freeze\_time$: minimum time interval between two successive updates of p. */
/* – d_1 : amount by which p is incremented when the queue overflows. */
/* – d_2 : amount by which p is decremented when the link is idle. */
/* – now : current date of the system. */
/* – $last\_udpdate$: date of the last update of p. */

⇒ **Upon packet loss event:**
if $(now - last\_update) > freeze\_time$ **then**
  $p = p + d_1$.
  $last\_update = now$.
end if

⇒ **Upon link idle event:**
if $(now - last\_update) > freeze\_time$ **then**
  $p = p - d_2$.
  $last\_update = now$.
end if

⇒ **Upon packet arrival:**
Mark or drop packet with probability p.

BLUE decouples the packet loss event (due to buffer overflow) from the link idle event. Therefore, if BLUE detects packet loss caused by buffer overflow or if the queue length exceeds a certain threshold value L, then the packet loss probability p is considered conservative. It is therefore increased by a step d_1 if the update interval is greater than $freeze\_time$. The aim of threshold L is to allow room to be left in the queue for transient bursts and allows the queue to control queueing delay.

If BLUE detects a link idle event (*i.e.* the queue becomes empty), the packet loss probability p is considered too aggressive. It is therefore decreased by a step d_2 if the update interval is greater than $freeze\_time$.

The authors claim that BLUE reduces the packet loss rate and keeps the queue stable as compared to RED.

Algorithm 2 describes the BLUE algorithm.

1.2.1.3 Stabilized RED (SRED)

SRED (Stabilized RED) is a mechanism proposed by Ott *et al.* in [82] with the aim of stabilizing the queue size. For this purpose, it uses a statistical technique to estimate the number of active flows to pre-emptively drop packets with a load dependent probability. More precisely, SRED estimates the number of active connections by using a small "zombie" list to keep information about flows that have recently sent packets.

At initialization, the list is filled with the packet flow identifier for each incoming packet. For each packet of the list, the $Count$ variable is set to zero.

Once the list is full, when a packet arrives at the buffer, it is compared with a randomly chosen zombie in the zombie list. If the incoming packet has the same flow identifier as the zombie, then a hit is declared and the $Count$ of the zombie is incremented by one. With probability p, the zombie flow identifier is replaced by the flow identifier of the incoming packet served for the comparison and the $Count$ of the zombie is set to 0. With probability $1-p$, there is no change of the zombie list.

The authors claim that the hit mechanism can be used to identify misbehaving flows without keeping per-flow state. These kind of flows are more likely to cause hits than well behaved flows and the count of a zombie coming from these flows is high. Moreover, the authors estimated the number of active flows from the average hit rate as described below. They first estimate the hit frequency $P(t)$ at the arrival time of the $t-th$ packet according to the following formula:

$$P(t) \;=\; (1-\alpha)P(t-1) + \alpha Hit(t)\;, \tag{1.4}$$

where α is a constant, and where:

$$Hit(t) \;=\; \begin{cases} 0 & \text{if } no\ hit\;, \\ 1 & \text{if } hit\;. \end{cases} \tag{1.5}$$

The authors claim that $1/P(t)$ is a good estimate for the number N of active flows until the arrival of the $t-th$ packet. Instead of using the average queue size like RED,

SRED computes the packet drop probability $d(K_{cur}, N)$ (noted p_{zap} in the original paper) from the estimated number of active flows and the instantaneous queue size K_{cur} (noted q in the original paper). First the function p_{SRED} is defined as:

$$p_{SRED}(K_{cur}) \;=\; \begin{cases} 0 & \text{if } 0 \le K_{cur} < B/6 \ , \\ \frac{1}{4} \times p_{max} & \text{if } B/6 \le K_{cur} < B/3 \ , \\ p_{max} & \text{if } B/3 \le K_{cur} < B \ , \end{cases} \tag{1.6}$$

where B is the buffer capacity of the queue and p_{max} is the maximum early-drop probability. Note that p_{SRED} has only three possible levels of dropping (0, $p_{max}/4$ and p_{max}), and that p_{SRED} does not depend of the past behavior of the instantaneous queue length K_{cur}. Figure 1.2 illustrates these levels of dropping.

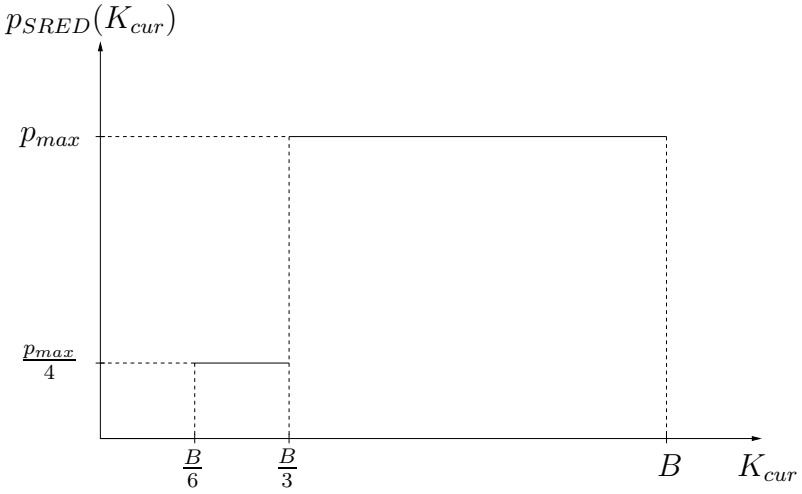


Figure 1.2: SRED drop function

The authors proposed two versions of SRED : the "simple SRED" where the drop probability $d(K_{cur}, N)$ only depends on the instantaneous buffer size and on the estimate $P(t)$:

$$d(K_{cur}, N) \;=\; p_{SRED}(K_{cur}) \times \min\left(1, \frac{N^2}{256^2}\right) \ . \tag{1.7}$$

For the second version named "full SRED", the drop probability $d(K_{cur}, N)$ depends also on whether a packet caused a hit or not, and is computed as follows:

$$d(K_{cur}, N) \;=\; p_{SRED}(K_{cur}) \times \min\left(1, \frac{N^2}{256^2}\right) \times (1 + Hit(t) \times N) \ . \tag{1.8}$$

Note therefore that $d(K_{cur}, N)$ is an increasing function of the estimated number of active flows. Algorithm 3 shows the principle of the SRED mechanism.

The simulation results have shown that SRED can stabilize, over a wide range of load levels, the buffer occupancy at a level which is independent of the number of active connections (this level is equal to a specified target value equal to $B/3$). However, SRED suffers from low throughput as shown in their simulation results.

Algorithm 3 Stabilized RED

/* Notations : */
/* – $P(t)$: represents the inverse of the estimated number of active flows.. */
/* – $d(K_{cur})$: packet drop probability. */

⇒ **Upon packet arrival:**
Compute $P(t)$ according to (1.4).
Compute $p_{SRED}(K_{cur})$ according to (1.6).
Compute $d(K_{cur})$ according to (1.7) for the *simple SRED* or (1.8) for the *full SRED*.
Mark or drop packet with probability $d(K_{cur})$.

1.2.1.4 Random exponential marking (REM)

The Random Exponential Marking (REM) active queue management scheme has been proposed by Athuraliya *et al.* in [7, 8] with the aim to achieve *high utilization* and *negligible loss and delay*. The authors attempt to match user input rates to link capacity and to stabilize the queue size around a small target queue size regardless of the number of users sharing the link. For this purpose, unlike RED which uses the mean queue length that must steadily increase as the number of users increases to determine the marking probability, REM uses a variable called price as a congestion measure in order to determine the marking probability. The price steadily increases while the mean queue length is stabilized around the target $\widehat{K}_T$ (denoted $b_l^\star$ in [7, 8]), as the number of users increases. To compute the price variable denoted $p_l(t)$ (where t is the time period), REM uses the following elements:

- γ is a constant which controls the responsiveness of REM to changes in network conditions.
- α_l is a constant set by each queue individually,which trades off utilization and queueing delay during transient changes in network conditions(α_l is a weight).
- K_{prev} (noted $b_l(t)$ in the original paper) is the aggregate queue length at queue l in period t. The queue length K_{cur} at period $t+1$ (noted $b_l(t+1)$ in the original paper) is computed as :

$$K_{cur} \;=\; \max\left(K_{prev} + x_l(t) - c_l(t), 0\right) \;.$$

- $\widehat{K}_T$ is the target queue length.
- $x_l(t)$ is the aggregate input rate to queue l in period t.
- $c_l(t)$ is the available bandwidth to queue l in period t.

The price variable is updated periodically or asynchronously based on:

- the *difference between input rate and link capacity* which corresponds to the rate mismatch and is measured by $x_l(t) - c_l(t)$.

- the *difference between queue length and target* which corresponds to queue mismatch and is measured by $K_{prev} - \widehat{K}_T$.

REM explicitly controls the update of its price. For queue l, the price $p_l(t)$ in period t is updated according to:

$$p_l(t+1) \;=\; \max\left(p_l(t) + \gamma \times \overbrace{(\underbrace{\alpha_l}_{\text{weight}} \times (\underbrace{K_{prev} - \widehat{K}_T}_{\text{queue mismatch}}) + \underbrace{x_l(t) - c_l(t)}_{\text{rate mismatch}})}^{\text{weighted sum}}, 0\right) \;, \tag{1.9}$$

In order to use only local and aggregate information and avoid the need of per-flow information, the rate $x_l(t) - c_l(t)$ is approximated by the change in queue length $K_{cur} - K_{prev}$ since $x_l(t) - c_l(t)$ is the rate at which the queue length grows and since it is usually easier to sample queue length than rate in practice. The price is therefore updated based only on the current and previous queue length. Equation (1.9) then becomes:

$$p_l(t+1) \;=\; \max\Big(p_l(t) + \gamma \times (K_{cur} - (1-\alpha_l)K_{prev} - \alpha_l\widehat{K}_T), 0\Big) \;. \tag{1.10}$$

When the number of users increases, the input rate exceeds the link capacity or there is excess backlog to be cleared from the queue. The mismatches in rate and in queue grow, *price* is then *incremented* since the weighted sum of these mismatches is positive increasing as a result the marking probability. This sends a stronger congestion signal to the sources which then reduce their rates.

When the sources rates are too small, the input rate does not exceed the link capacity and there is no excess backlog to be cleared and the mismatches will be negative. The price is then *decremented* making the marking probability decrease and raising source rates.

The mismatches can drive to zero, yielding high utilization and negligible loss and delay in equilibrium. In equilibrium, the price stabilizes and this weighted sum must be zero. This can hold only if the input rate equals capacity ($x_l = c_l$) and the backlog equals its target ($\widehat{K}_{cur} = \widehat{K}_T$). The buffer will be cleared in equilibrium if the target queue is set to zero.

Then the marking/dropping probability $d(t)$ of a packet at queue l in period t is determined by an exponentially marking probability function as :

$$d(t) \;=\; 1 - \phi^{-p_l(t)} \;, \tag{1.11}$$

where $\phi > 1$ is a constant. Algorithm 4 gives the pseudo-code of the REM scheme.

The authors claim that REM is stable for a more narrow variety of network environments than PI and LRED, although it has a quicker response than PI. However, the problem with REM is to determine the values of the three constants α, γ and ϕ. Athuraliya *et al.* [9, 8, 6] explain each of these constants and recommend how to set up these constants:

Algorithm 4 REM

⇒ **At each interval of time:**
Compute the price $p_l(t+1)$ using (1.10).
Compute the packet drop probability $d(t+1)$ using (1.11).

⇒ **Upon packet arrival:**
Mark or drop packet with probability $d(t+1)$.

- α determines the prominence given to the queue length when determining the level of congestion. It trades off utilization and queue length in transient change of network condition with a smaller α producing a higher utilization. A value of $\alpha = 0.1$ is chosen in [8].

- Concerning the parameter γ, it determines the speed of convergence of the algorithm. A larger value of γ gives a faster convergence but it also incurs a higher risk of oscillatory queue. When choosing γ, consideration should be given to the dynamics at the router. If the congestion level at the router is changing constantly, a larger value of γ should be used for the algorithm to track the changes. A value of $\gamma = 0.001$ is chosen in [8].

- The parameter ϕ determines the range of loss or marking probability, which also depends on the range of the price $p_l(t)$. Ideally, ϕ should be chosen so that the end-to-end probability observed at hosts fluctuates around 0.5. A value of ϕ equals to 1.001 is recommended in [8] and a value of ϕ equals to 1.05 or 1.1 is recommended in [9].

REM was designed to increase the link utilization at a router while maintaining a small queue length. However, REM does not provide adequate fairness which is sacrificed at the cost of higher utilization. Moreover, under heavy congestion or with large stable drop probability, REM suffers from a long response time. In addition, when the buffer size is small, the responsiveness of REM will becomes worse as well.

1.2.1.5 Double slope RED (DSRED)

Double Slope RED (DSRED) has been proposed by Zheng and Atiquzzaman in [102, 103] with the aim to improve the throughput and delay characteristics of RED. DSRED uses a two-segment drop function which provides much more flexible drop operation than RED as shown in Figure 1.3.

DSRED introduces the following additional variables:

- K_l: threshold for the average queue length to start packet dropping at the buffer.

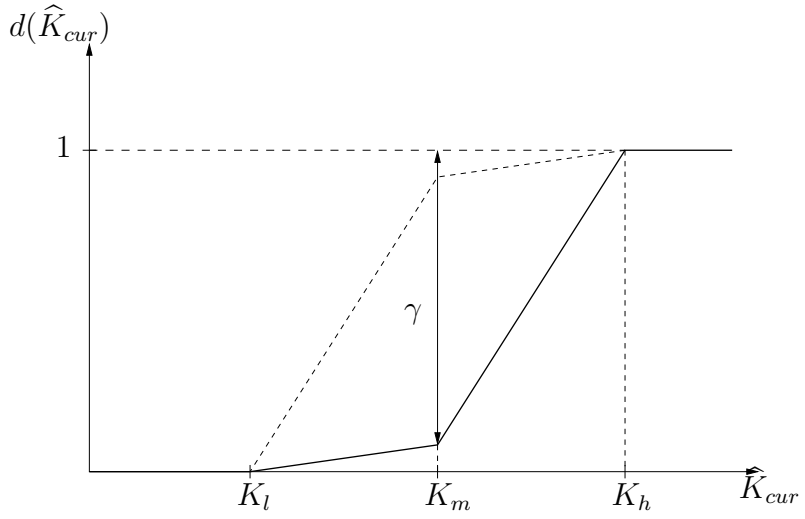


Figure 1.3: DSRED's drop function

- K_h: threshold for average queue length to change the drop function drop.
- K_m: threshold for average queue length to change the drop function drop.
- α: drop function slope for the first linear segment between K_l and K_m.
- β: drop function slope for the second linear segment between K_m and K_h.
- γ: mode selector for adjusting drop function slopes.

Like RED, DSRED uses the average queue size $\widehat{K}_{cur}$ (noted avg in the original paper) as a congestion measure. According to the value of the average queue size, DSRED uses one of the drop functions expressed as:

$$d(\widehat{K}_{cur}) \;=\; \begin{cases} 0 & \text{if } \widehat{K}_{cur} < K_l \ , \\ \alpha(\widehat{K}_{cur} - K_l) & \text{if } K_l \leq \widehat{K}_{cur} < K_m \ , \\ 1 - \gamma + \beta(\widehat{K}_{cur} - K_m) & \text{if } K_m \leq \widehat{K}_{cur} < K_h \ , \\ 1 & \text{if } K_h \leq \widehat{K}_{cur} \leq N \ , \end{cases} \tag{1.12}$$

where α and β are the slopes of the two segments and are given by:

$$\alpha \;=\; \frac{2(1-\gamma)}{K_h - K_l} \ , \tag{1.13}$$

$$\beta \;=\; \frac{2\gamma}{K_h - K_l} \ . \tag{1.14}$$

Parameter α is the slope of the drop function corresponding to an average queue size between K_l and K_m whereas β is the slope of the drop function corresponding to an average queue size between K_m and K_h. The weighted average queue size $\widehat{K}_{cur}$ is computed as for the RED scheme:

$$\widehat{K}_{cur} \;=\; (1-\omega)\widehat{K}_{cur} + \omega K_{cur}\ , \tag{1.15}$$

where K_{cur} is the instantaneous queue size.

The parameter γ adjust the operating mode of DSRED. The drop rate can be increased and decreased by adjusting γ. This allows DSRED to handle different congestion situations, that is to increase the drop rate with higher rate instead of a constant rate when congestion increases and to reduce the drop probability in case of a low congestion level.

The pseudo-code describing the implementation of the DSRED scheme is given by Algorithm 5.

Algorithm 5 DSRED

/* Notations : */
/* – $\widehat{K}_{cur}$: (weighted) average queue length. */
/* – $d(\widehat{K}_{cur})$: packet drop probability. */

⇒ **Upon packet arrival:**
Compute the average queue length $\widehat{K}_{cur}$ using (1.15).
Compute the packet drop probability $d(\widehat{K}_{cur})$ using (1.12).
Mark or drop packet with probability $d(\widehat{K}_{cur})$.

Even though DSRED shows an improved packet drop performance resulting in a higher throughput than RED, it adds several additional parameters.

1.2.1.6 Adaptive RED version Feng *et al.*

In order to increase the link utilization rate and reduce packet loss rate Feng *et al.* proposed in [34] an adaptive version of RED denoted $ARED_{Feng}$. The authors investigated in [34] the impact of traffic load on early detection mechanisms. In periods of heavy congestion, in order to avoid packet loss due to buffer overflow, RED must send congestion notifications to a sufficient number of sources so that the offered load is reduced. In order to prevent the link from being underutilized, RED must not also send congestion notifications to too many sources. In case of N sources sharing a bottleneck link, sending one congestion notification to one source correspond to a reduction of the offered load by a factor of $(1-1/2N)$ since TCP reduces the size of its congestion window by 1/2. For large N, the impact of one congestion notification decreases but for small N the impact increases. Hence, in case of large N, if RED is not more aggressive then the queue remains close to fully occupied and thus behaves like Drop Tail. In case of small N, if RED is not less aggressive, then too many sources receive congestion notifications and reduce their transmission rate provoking underutilization of the link. To optimize performance, the authors made the following recommendations: for small number of connections, early detection should be conservative

in order to achieve high link utilization rate. But in case of large number of connexions, a high link utilization is achieved whatever the value of Max_p. However, the early detection should be aggressive in order to reduce packet loss rate.

Consequently, the authors conclude that adapting the parameter Max_p according to the traffic load improves the performance of RED and propose an algorithm for this purpose. The algorithm uses the average queue length to infer the aggressive or conservative behavior of the early detection. If the average queue size is around the Min_{th} than the early detection mechanism is too aggressive. As a result, the value of Max_p is decreased by a constant factor α. Otherwise, if the average queue size is around Max_{th} then it means that the early detection is too conservative. In this case the value of Max_p is increased by a fixed factor β. However, if the average queue size oscillates between Min_{th} and Max_{th} the authors do not adapt Max_p because they consider that the early detection mechanism behaves as desired and can avoid packet loss rate and link underutilization.

Algorithm 6 describes this scheme.

Algorithm 6 Feng's *and al.* adaptive RED algorithm

```
/* – K̂_cur : current weighted average queue size. */
/* Initialization of fixed parameters */
Initialize Min_th and Max_th.
α ← 3.
β ← 2.
/* Adaptation of Max_p. */
for each K̂_cur Update do
  if (Min_th < K̂_cur < Max_th) then
    status = Between.
  else
    if (K̂_cur < Min_th and status ≠ Below) then
      status = Below.
      Max_p ← Max_p / α.
    else
      if (K̂_cur > Max_th and status ≠ Above) then
        status = Above.
        Max_p ← Max_p × β.
      end if
    end if
  end if
end for
```

1.2.1.7 Adaptive RED version Floyd *et al.*

Floyd *et al.* proposed in [41] another version of adaptive RED with minimal changes to the original adaptive RED ($ARED_{Feng}$) described in the previous paragraph. The

aim of this proposal is to achieve a predictable average queuing delay, and to alleviate the problem of variable delay and parameters sensitivity. The proposal of Floyd *et al.* (denoted $ARED_{Floyd}$) retains the basic insight of $ARED_{Feng}$ which consists in adapting automatically RED parameters. In particular, the ARED design consists in setting the parameter w_q automatically based on the link speed, and in adapting the parameter Max_p in response to the queue length so as to increase the throughput and achieve a reasonable average queue length.

However, $ARED_{Floyd}$ differs from $ARED_{Feng}$ in the following points :

- $ARED_{Floyd}$ uses a target range which is half away between Max_{th} and Min_{th} inside which the average queue size should oscillate. $ARED_{Feng}$ brings the average queue size between Min_{th} and Max_{th}.
- It uses a time interval at which Max_p is adapted instead of adapting like $ARED_{Feng}$ at every packet arrival. Hence, this way Max_p is adapted slowly and infrequently. The authors claim that this gives $ARED_{Floyd}$ a more robust performance.
- For robustness, $ARED_{Floyd}$ also chooses to use an AIMD (Additive Increase Multiplicative Decrease) approach to adapt Max_p instead of an MIMD (Multiplicative Increase Multiplicative Decrease) like $ARED_{Feng}$.
- Moreover, $ARED_{Floyd}$ bounds Max_p in order to ensure an acceptable performance of ARED during a transition period where, due to the slow adaptation of Max_p, ARED is not able to adapt Max_p to its next value after a sharp change in the congestion level.

Algorithm 7 illustrates this approach. INTERVAL is the only additional parameter. The authors set it to 0.5 seconds. At each time interval, if the average queue size exceeds its target range then Max_p is increased additively by the parameter α. If the average queue size is below the target range, then Max_p is decreased multiplicatively by the parameter β (see the Algorithm 7 for the setting of α and β). The setting of these two parameters is performed so as to avoid the oscillations of the average queue size from above to below the target range just after a single modification of the the value of the parameter Max_p. But if the average queue size remains within its target range, then no modification of the value of Max_p is performed.

The authors claim that $ARED_{Floyd}$ achieves a predictable average delay. However, experiments reported in [41] showed that the queue size still exhibits pronounced oscillations.

1.2.2 Proposals requiring network scenario parameters

1.2.2.1 GREEN

The GREEN algorithm has been proposed by Feng *et al.* in [36, 60]. This scheme is based on the estimation of the packet loss rate by using a mathematical model of the steady-state

Algorithm 7 Floyd's *and al.* adaptive RED algorithm

/* $\widehat{K}_{cur}$: weighted average queue size at the beginning of the current interval. */
/* Initialization of fixed parameters */
Set $\widehat{K}_T$ as a function of the delay target.
$Min_{th} \leftarrow \max(5, \frac{delay_{target} \times C}{2})$ where C is the link capacity.
$Max_{th} \leftarrow 3 \times Min_{th}$.
$INTERVAL \leftarrow 0.5$ seconds.
$target\_range \leftarrow [Min_{th} + 0.4 \times (Max_{th} - Min_{th}), Min_{th} + 0.6 \times (Max_{th} - Min_{th})]$.
$\alpha \leftarrow \min(0.01, \frac{Max_p}{4})$.
$\beta \leftarrow 0.9$.
/* Adaptation of Max_p. */
for each INTERVAL **do**
  if ($\widehat{K}_{cur} > target\_range$ and $Max_p \leq 0.5$) **then**
    $Max_p \leftarrow Max_p + \alpha$.
  else
    if ($\widehat{K}_{cur} < target\_range$ and $Max_p \geq 0.01$) **then**
      $Max_p \leftarrow Max_p \times \beta$.
    end if
  end if
end for

behavior of TCP [73]. This model is derived from the formula described on page 13 by Equation (1) of a connection's throughput presented by Mathis *et al.* in [73].

If L is the link capacity and N is the number of active TCP flows that share fairly the link bandwidth, then the bandwidth share of a flow is equal to L/N and the packet loss probability p, is deduced by the following steady-state TCP formula derived from equation (1) the TCP square root formula :

$$p = \left(\frac{N \times MSS \times c}{L \times RTT} \right)^2 . \tag{1.16}$$

Note however that (1.16) provides good estimates only when p is of the order of a few percent [73]. Congestion notification is more aggressive for large N and small RTT. Moreover, since this formula needs the RTT, the MSS and the number of TCP connections as an input, GREEN should estimate their values without requiring per-flow state information. The MSS parameter is estimated by the router by looking at the size of each packet.

In order to estimate RTT for flows, GREEN does not use any per-flow state. The authors have actually proposed two methods. The first approach requires TCP senders to embed their current RTT estimates within the TCP header. This estimated RTT is the difference between the sending time of a packet by the source and the reception time of its corresponding ACK by the same source. The second approach is to use the IDMaps service [45], a scalable Internet-wide service that aims to provide Internet distance

estimates. However, IDMaps is currently a theoretical service which is not yet deployed in the Internet. Note that there exists other methods as the Global Network Positioning (GNP) proposed by Ng and Zhang in [81] to estimate the round trip time.

The number of active flows is estimated by counting the number of flows passing through the router in a given time interval. However, a long time interval may cause GREEN to overestimate the number of flows and reduce the overall link utilization. A short time interval may cause GREEN to underestimate the number of flows, resulting in over-provisioning of the link bandwidth. Each flow is identified by the source and destination identification numbers contained in the IP packet header. The estimated packet loss rate p, is used as the drop probability.

GREEN regulates TCP flows over the same link to a fair sending rate by preventing shorter RTT flows from consuming more than their fair share of the bandwidth, and hence prevents them from inducing congestion.

Algorithm 8 gives the pseudo-code of the GREEN scheme described in [60] which represents an improvement of the original version of GREEN described in [36]. This improvement relies on the fact that it is designed to achieved high link utilization in the presence of short-lived or low-bandwidth flows, unlike the initial version.

For the new version of GREEN proposed in [60], the authors introduce an additional parameter $\gamma(t)$ in (1.16) in order to adapt the packet drop function according to the link utilization rate and to the number of packets lost during the last time interval. Then the drop probability becomes:

$$p \;=\; \left(\frac{N \times MSS \times c}{\gamma(t) \times L \times RTT}\right)^2 .$$

The parameter $\gamma(t)$ is adapted as follows:

- If packets are lost during the last time interval then:

  $$\gamma(t+1) \;=\; 0.95\,\gamma(t) .$$

  This means that γ decreases by a factor of 0.95. The packet loss probability increases as a result. The constant 0.95 allows the reduction of wide oscillations in link utilization. This configuration of γ allows the detection of a congestion increase.

- Otherwise, if the link utilization rate is less than 98%, then γ should be configured so that the link utilization is decreased:

  $$\gamma(t+1) \;=\; \frac{1 + currentUtil}{2 \times currentUtil}\,\gamma(t) .$$

  This means that γ increased by a factor depending on the current link utilization rate $currentUtil$. The increase of γ leads to a decrease of the packet loss probability. The authors choose this factor in such a way that a link utilization rate of 50% is increased to a new link utilization of 75%. This configuration of γ allows the detection of a congestion decrease.

Algorithm 8 GREEN

/* Notations : */
/* – RTT: estimated round-trip time for a given flow. */
/* – p: packet loss probability for a given flow. */
/* – MSS: maximum segment size. */
/* – c: constant value depending on the acknowledgment strategy used. */
/* – N: estimated number of flows. */
/* – L: capacity of the outgoing link. */
/* – $\gamma(t)$: constant at time t. */
/* – $window$: window of time used to estimate the number of flows and to estimate the current link utilization by counting the number of departures for this given window of time. */
/* – $currentUtil$: current link utilization. */
/* – $queueDrops$: number of queue drops due to overflow. */

⇒ **Upon packet arrival:**
$RTT \leftarrow obtainRTT(pkt)$.
$p \leftarrow \left(\frac{N \times MSS \times c}{\gamma(t) \times L \times RTT}\right)^2$.
Mark or drop packet with probability p.
if $currentTime() - lastUpdate \geq window$ **then**
  Update variables $currenUtil$, N and $queueDrops$.
  $lastUpdate \leftarrow currentTime()$.
  if $queueDrops > 0$ **then**
    $\gamma \leftarrow 0.95\gamma$.
  else if $currentUtil < 0.98$ **then**
    $\gamma \leftarrow \frac{1+currentUtil}{2 \times currentUtil}\gamma$.
  end if
end if

This way the authors claim to overcome the problems detected with the first version of GREEN. Nevertheless, the authors indicate the limitations of their current model due to the behavior of γ. The parameter γ oscillates between very high and low values. A large number of flows is needed to maintain a sustained high link utilization. For small number of flows, to achieve high link utilization GREEN tries to increase γ without success since there is not enough contention. The authors are considering a "smarter" GREEN that attempts to increase γ in such cases.

1.2.2.2 Adaptive Virtual Queue (AVQ)

Adaptive Virtual Queue (AVQ) has been proposed by Kunniyur and Srikant in [67, 66, 68] with the motivation to design an AQM scheme that results in low-loss, low-delay and high link utilization rate at the link. AVQ is a virtual queue-based AQM scheme that detects

congestion based on the arrival rate of the packets at the link.

AVQ uses a virtual queue with a virtual service capacity that is less than the actual capacity of the link. The virtual capacity at each link is modified such that the total flow entering each link achieves a desired utilization of the link.

It also uses as an input to the model, system parameters like the maximum round trip time (RTT), the minimum number of active connections N in order to find the fastest rate at which the marking probability is adapted and therefore to achieve the system stability. The adaptation rate that is meant to maintain the system stability, denoted by α, is computed so as to achieve a desirable link utilization rate.

The variables used by the AVQ scheme are :

- γ: desired utilization of the link ($\gamma \leq 1$).
- α: damping factor used to determine how fast the marking probability is adapted at the link to the changing network conditions.
- C: capacity of the link.
- $\widetilde{C}$: *virtual* capacity of the link.
- B: buffer size.
- $\widetilde{B}$: *virtual* buffer size.
- λ: arrival rate at the link.

The AVQ algorithm described in Algorithm 9 (page 45) works as follows:

- The router maintains a virtual queue whose link capacity is $\widetilde{C} \leq C$ and whose buffer size is equal to the buffer size of the real queue. The virtual and the real queue differ by their link capacity and the size of their queue.
- At each packet arrival:
  - The virtual queue length is computed as the difference between the previous virtual size and the number of bytes served since the arrival of the previous packet. The service of packets is performed according to the capacity of the virtual capacity of the link.
  - A fictitious packet is enqueued in the virtual queue if there is sufficient space in the buffer. No actual enqueueing or dequeueing of packets is necessary in the virtual queue. The algorithm just keeps track of the virtual queue length.

- If the new packet overflows the virtual buffer, then the packet is discarded from the virtual buffer and the real packet is marked by setting its ECN bit to 1 or dropped, depending upon the congestion notification mechanism used by the router.
- If the packet does not overflow the virtual buffer size then the virtual queue size is updated by adding the number of bytes of the arrived packets to the virtual queue.
- the virtual queue capacity is updated according to the following differential equation:

$$\dot{\tilde{C}} \;=\; \alpha(\gamma C - \lambda)\ . \tag{1.17}$$

The principle of this equation is that marking has to be more aggressive when the link utilization exceeds the desired utilization and should be less aggressive when the link utilization is below the desired utilization. This situation can be viewed as a token bucket where tokens are generated at rate $\alpha\gamma C$ up to a maximum of C and by each arrival of a packet, α times the size of packet tokens are removed from the bucket.

This algorithm is evaluated through the experiments conducted in [67, 66, 68]. The experiments show that AVQ maintains a very small queue length and the system stabilizes after a load change. Indeed, the average and the standard deviation of the queue length before and after the introduction of a load change are almost similar. AVQ manages also to achieve a link utilization rate close to the target link utilization rate. In addition, the packet loss rate experienced by AVQ is lower than other schemes like PI, REM and RED.

There are two parameters that have to be chosen to implement AVQ:

- The desired utilization γ (it determines the robustness to the presence of uncontrollable short flows).
- The damping factor α (it determines how fast one adapts the marking probability at the link to the changing network conditions).

Both parameters α and γ determine the stability of the AVQ algorithm and the authors provide a design rule to choose these parameters.

Theorem 1 *Suppose that the feedback delay R^+, the number of users N^- and the utilization $\widehat{\gamma}$ are given. Find $\alpha^\star$ satisfying:*

$$\omega R^+ + \arctan\left(\frac{\omega}{K_{11}}\right) = \frac{\pi}{2}\ , \tag{1.18}$$

Algorithm 9 Adaptive virtual queue (AVQ)

/* Notations : */
/* – B: buffer size. */
/* – VQ: number of bytes currently in the virtual queue. */
/* – s: arrival time of previous packet. */
/* – $\widetilde{C}$: virtual capacity of the link. */

⇒ **Upon packet arrival:**
$t \leftarrow$ current time.
$b \leftarrow$ number of bytes.
/* Update virtual queue size */
$VQ \leftarrow \max(VQ - \widetilde{C}(t-s), 0)$.
if $VQ + b > B$ **then**
  Mark or drop packet in the real queue.
else
  /* Update virtual queue size */
  $VQ \leftarrow VQ + b$
end if
/* Update virtual capacity */
$\widetilde{C} = \max(\min(\widetilde{C} + \alpha \times \gamma \times C(t-s), C), 0)$.
/* Update virtual capacity */
$s \leftarrow t$.

where ω is defined as:

$$\omega(\alpha^\star, R^+, N^-, \widehat{\gamma}) \;=\; \frac{1}{\sqrt{2}}\sqrt{(K_{12}^2 - K_{11}^2) + \sqrt{(K_{12}^2 - K_{11}^2)^2 + 4K_2^2(\alpha^\star)^2}}\,, \tag{1.19}$$

and where:

$$K_{11} \;=\; \frac{N^-}{\widehat{\gamma}C(R^+)^2}, \quad K_{12} \;=\; K_2 \;=\; \frac{3}{2}\frac{\widehat{\gamma}C}{N^-}\,.$$

Then, for all $\alpha < \alpha^\star$, the system is stable. Moreover, for every $\alpha < \alpha^\star$, the system remains stable for all $N > N^-$, $\gamma < \widehat{\gamma}$ and $R < R^+$.

1.2.2.3 Proportional controller (P controller)

Hollot *et al.* proposed the P controller in [52, 50] by performing a simple transformation of the RED scheme. The authors argue that even if the low pass filter (Equation 1.1) allows transient burst pass through, from a control standpoint averaging can lead to instability and low frequency oscillations of the queue size. In addition, this design of RED can lead to the sluggishness of the response time of the control system.

In order to make the system more responsive, the authors remove the low-pass filter by setting the averaging weight to one. This is equivalent to compute the loss probability by

using the instantaneous queue size K_{cur} (denoted by q in the original paper) instead of the average queue size $\widehat{K}_{cur}$ as:

$$p \;=\; Max_p \frac{K_{cur} - Min_{th}}{Max_{th} - Min_{th}} \;. \tag{1.20}$$

Algorithm 10 describes the P controller.

Algorithm 10 P controller

/* Notations : */
/* – p: probability to mark or drop a packet. */

⇒ **Upon packet arrival:**
Compute packet drop probability p using (1.20).
Mark or drop packet packet with probability p.

Given the capacity C of the bottleneck link in $packets/s$, the minimal number of TCP connections N^- and the maximum round trip time R^+, the authors compute the slope L of the drop function and deduce $Max_{th} - Min_{th}$ for which the system is stable as follows:

$$L \;=\; \left| \frac{\left(1 + \frac{i\omega_g}{p_{tcp}}\right)\left(1 + \frac{i\omega_g}{p_{queue}}\right)}{\frac{(R^+C)^3}{(2N^-)^2}} \right| \;,$$

and:

$$Max_{th} - Min_{th} \;=\; \frac{Max_p}{L} \;,$$

where the variables ω_g, p_{tcp} and p_{queue} are defined as follows:

$$\omega_g \;=\; \left(\frac{2N^-}{(R^+)^3 C}\right)^{\frac{1}{2}}, \quad p_{tcp} \;=\; \frac{-2N^-}{(R^+)^2 C}, \quad p_{queue} \;=\; \frac{-1}{R^+} \;.$$

Using simulations, the authors showed that the P controller responds much more quickly to load variations whereas RED is quite sluggish to load changes. That authors also showed that, Unlike RED, the increase of RTT do not change significantly the performance of the P controller. However, the P controller suffers from a limitation which makes it impractical to implement under certain situations [52, 50]. For stability of the system, a relatively shallow slope of the drop function is required and an increased slope of leads to instability. Buffer size limitations results in a large value of Max_p that leads to oscillations. If the buffer size is increased it could lead to large queueing delay. Hence, the authors proposed an integral controller PI to regulate the queue level to a target queue size and also decouple the average queue size from the marking probability. So that the regulated output (queue size) could become independent from the level of load or the RTT.

1.2.2.4 Proportional integral controller (PI controller)

Proposed also by Hollot *et al.* in [52, 50], the PI controller is designed to increase the link utilization while maintaining a small queue size. For this purpose, the PI scheme jointly uses queue length and input rate as a congestion measure to achieve better stability of the instantaneous queue size by trying to regulate the queue length to the expected value $\widehat{K}_T$ (denoted by q_{ref} in the original paper) using queue length mismatch and its integral. The integral of queue length mismatch is related in practice to the input rate mismatch.

At each packet arrival, the packet is dropped with a probability p equal to:

$$p \;=\; a \times (K_{cur} - \widehat{K}_T) - b \times (K_{prev} - \widehat{K}_T) + p_{old} \;, \tag{1.21}$$

where K_{cur} and K_{prev} are respectively the actual instantaneous queue size and the previous instantaneous queue size (it corresponds respectively to q and q_{old} in the original paper), and p_{old} is the previous dropping/marking probability. The dropping probability p is updated at each interval of time INTERVAL (the variable INTERVAL is equal to $1/f_s$ where f_s is the sampling frequency in the original paper). Algorithm 11 gives the pseudo-code of the PI controller scheme.

Algorithm 11 PI controller

/* Notations : */
/* – p: probability to mark or drop a packet. */

⇒ **At each interval of time INTERVAL:**
Compute packet drop probability p using (1.21).

⇒ **Upon packet arrival:**
Mark or drop packet packet with probability p.

The authors provides a design rule in [52, 50] through control-theory analysis to determine the constants a and b. Given the capacity C of the bottleneck link in $packets/s$, the minimal number of TCP connections N^- and the maximum round trip time R^+ for the considered system, the parameters a and b under which the system is stable for $N \geq N^-$ and $R \leq R^+$ are computed as follows:

$$a \;=\; K_{PI}\left(\frac{1}{2f_s} + \frac{1}{\omega_g}\right) \;, \tag{1.22}$$

$$b \;=\; K_{PI}\left(-\frac{1}{2f_s} + \frac{1}{\omega_g}\right) \;, \tag{1.23}$$

with the constants ω_g and K_{PI} equals to:

$$\omega_g \;=\; \frac{2N^-}{R^{+2}C} \;,$$

and:

$$K_{PI} \;=\; \omega_g \left| \frac{1 + \frac{i\omega_g}{-\frac{1}{R^+}}}{\frac{(R^+C)^3}{(2N^-)^2}} \right| \;.$$

Note that the following relation should be satisfied: $\omega_g \ll 1/R^+$.

Nevertheless, the PI controller has some drawbacks. For example, under heavy congestion or with a large stable drop probability, PI suffers from a long response time. In addition, when the buffer size of the queue is small, the responsiveness of PI will become worse as well. Moreover, since the marking probability is directly modified and this update has to be slow enough for system stability, the scheme exhibits sluggishness when short flows are introduced.

1.2.2.5 LRED

LRED (Loss Ratio based RED) is an active queue management scheme recently proposed by Wang *et al.* in [97]. The authors argue that since RED and most of its variants use the queue length as a congestion indicator, these approaches suffer from unstable behaviors. The PI controller use both queue length and traffic input rate as congestion indicators. Proportional controller mechanisms such as PI calculate the suitable packet drop probability p according to the instantaneous queue length K_{cur} using the following control equation formulated as:

$$\delta p \;=\; H_c \, \delta K_{cur} \;.$$

where, $\delta K_{cur} = K_{cur} - \widehat{K}_T$ and $\widehat{K}_T$ is the expected queue length under the steady state (K_{cur} and $\widehat{K}_T$ corresponds to q and q_0 respectively in the original paper). Even if such mechanisms enhance the stability of the system, as in the case of RED and most of its variants, the average queue size increases with the traffic load and causes overflow and underflow of the buffer. In order to make the queue management schemes more responsive even if the number of TCP flows varies significantly, the authors of LRED employ a closed-form relationship between packet loss ratio and the number of TCP flows as a complement to the queue length to adjust dynamically the packet drop probability. The authors argue that the packet loss ratio is an important index in designing a buffer management mechanism since:

- An increasing packet loss ratio is a clear indication that severe congestion occurs, and that aggressive packet dropping is needed. LRED dynamically increases and decreases the packet drop probability.

- A decrease of packet loss ratio can serve as a signal that congestion is receding and consequently that packet drop action can made less aggressive (moderate).

To estimate the degree of link congestion, two indications are employed by LRED:

- Packet loss ratio: it is used in large time-scale to dynamically adjust the packet drop probability according to the measured packet loss ratio and to make the scheme more adaptive and robust.

- Queue length: it is used in small time-scale, upon the arrival of each new packet, to make the scheme more responsive in regulating the length to an expected value $\widehat{K}_T$.

LRED measures packet loss ratio in the large time-scale, and updates the packet drop probability in the small time-scale at each packet arrival. The packet drop probability p is computed every mp seconds (a duration which must be less than the RTT) as:

$$p \;=\; \overline{l(k)} + \beta\sqrt{\overline{l(k)}}(K_{cur} - \widehat{K}_T) \;. \tag{1.24}$$

where $\overline{l(k)}$ is an estimate of the packet loss ratio. Based on equation (1.24), the authors give a design rule for small time scale :

1. When the queue length is equal to $\widehat{K}_T$, the packet drop probability will be equal to the packet loss ratio.

2. When the queue length is larger or smaller than $\widehat{K}_T$, the packet drop probability will be also larger or smaller than the packet loss ratio.

The packet loss ratio $l(k)$ is computed as the ratio of the number of dropped packets to the number of total arrivals of packets during the latest M measurements periods. The estimate $\overline{l(k)}$ is then obtained as an EWMA of $l(k)$ by summing:

$$\overline{l(k)} \;=\; \overline{l(k-1)} \times mw + (1 - mw) \times l(k) \;.$$

where:

$$l(k) \;=\; \frac{\sum_{i=0}^{M-1} N_d(k-i)}{\sum_{i=0}^{M-1} N_a(k-i)} \;. \tag{1.25}$$

The relationship between the different timescales is illustrated in Figure 1.4. The authors took $M = 4$ in order that the average packet drop probability is as close as possible to the packet loss ratio. Algorithm 12 shows the pseudo-code describing the implementation of the LRED scheme.

As for REM, PI, SFC or AVQ, design rules are proposed for LRED in order to set the parameter β. Given the capacity C of the bottleneck link in $packets/s$, the minimal number of TCP connections N^- and the maximum round trip time R^+ for the considered system, the parameter should satisfy the following inequality:

$$\beta \;<\; \widehat{\beta} = \min\left(\bar{\beta}, \frac{\sqrt{2\eta}(2N^-)^4}{(R^+C)^3}\right) \;, \tag{1.26}$$

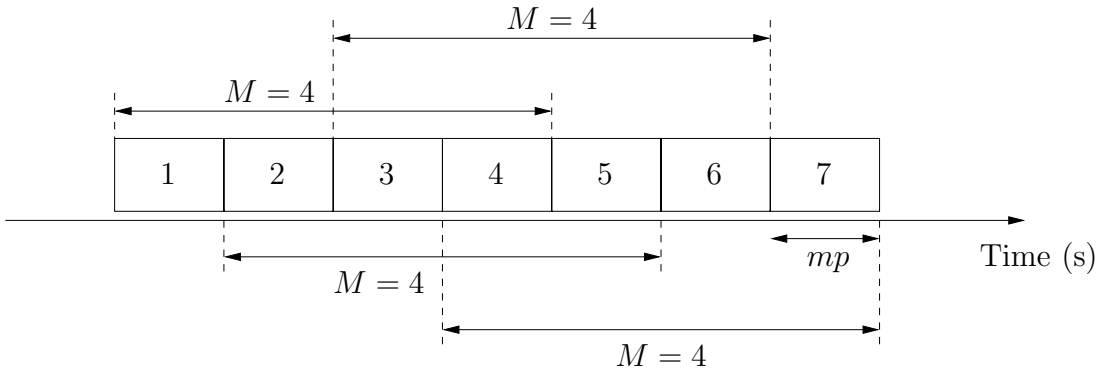


Figure 1.4: Large scale (M) and small scale (mp) time adaptation

where η is a fixed value is equal to 3/2 and where $\bar{\beta}$ is the solution of the equation:

$$R^{+}\omega + \arctan\left(\frac{\omega}{K_{11}}\right) = \frac{\pi}{2} \ .$$

The parameter ω is computed as follows:

$$\omega \ = \ \sqrt{\frac{\sqrt{K_{11}^4 + 4K_c^2H_c^2} - K_{11}^2}{2}} \ ,$$

where:

$$K_{11} \ = \ \frac{2N^{-}}{CR^{+2}}, \quad K_c \ = \ \frac{C^2}{\eta N^{-}}, \quad H_c \ = \ \bar{\beta}\sqrt{\frac{\eta N^{-2}}{C^2R^{+2}}} \ .$$

Algorithm 12 LRED

/* Notations : */
/* – $N_a(k)$: number of packets arrived in the $k-th$ period. */
/* – $N_d(k)$: number of packets dropped in the $k-th$ period. */
/* – $\underline{l(k)}$: packet loss ratio during the latest M measurements periods. */
/* – $\overline{l(k)}$: packet loss ratio. */
/* – mp: time interval between two consecutive updates of $\overline{l(k)}$. */
/* – p: probability to mark or drop a packet. */

⇒ **Update of $\overline{l(k)}$ every mp seconds:**
Compute $\underline{l(k)} \ = \ \frac{\sum_{i=0}^{M-1} N_d(k-i)}{\sum_{i=0}^{M-1} N_a(k-i)}$.
Compute $\overline{l(k)} \ = \ \overline{l(k-1)} \times mw + (1-mw) \times \underline{l(k)}$.

⇒ **Upon packet arrival:**
Compute packet drop probability p using (1.24).
Mark or drop packet with probability p.

The authors of LRED claim that compared to existing AQM schemes such as SRED, BLUE, ARED and AVQ, LRED offers more stable control of queue length around the expected value, and also achieves high link utilization, faster response time and better robustness.

1.3 RED with per flow accounting

1.3.1 Flow random early drop (FRED)

In order to address the problem of non-responsive flows, Lin and Morris have proposed in [70] a mechanism named Flow RED (FRED) for protecting routers against these non-responsive flows and thus for reducing the unfairness effect found in RED, notably the fairness problem between TCP and UDP flows. Instead of indicating congestion to randomly chosen connections by dropping packets proportionally, FRED generates selective feedback to a filtered set of connections which have a large number of packets queued. Indeed, FRED uses per-active flow accounting to compute a loss rate for each flow depending on the flow's buffer size. If a flow continually occupies a large amount of the queue's buffer space, it is detected and limited to a smaller amount of the buffer space. The cost of this per-active flow accounting (*i.e.* flows that have packets buffered) is proportional to the buffer size and independent of the total number of flows, except to the extent that buffer use depends on the number of active flows.

The authors classify the type of traffic into 3 categories:

- Non adaptive traffic: for connections that consume the bandwidth they require whatever the level of congestion (examples: audio, video applications, UDP flows).
- Robust traffic: for connections that consume bandwidth until a congestion is detected (example: bulk data transfer).
- Fragile traffic: for connections that are slow to adapt to more available bandwidth or that are sensitive to packet losses (example: telnet).

The algorithm of FRED isolates ill-behaved flows and contain attacks from these non adaptive flows by raising their loss rate, protects bursty and low speed (fragile) flows and handle robust flows as fairly as RED does.

FRED introduces the following parameters:

- Min_q: minimum number of packets each flow should be allowed to buffer.
- Max_q: maximum number of packets each flow should be allowed to buffer.
- $avgcq$: an estimate of the average per-flow buffer count. Flows with fewer than $avgcq$ packets queued are favored over flows with more.

- *qlen*: count of buffered packets for each flow that currently has any packets buffered.

- *strike*: counts for each flow the number of times the flow has failed to respond to congestion notification. FRED penalizes flow with high *strike* values.

The algorithm of FRED works as follows:

- An arriving packet belonging to flow i is definitely dropped if:
  - $qlen_i \geq Max_q$.
  - The weighted average queue size avg exceeds Max_{th} and $qlen_i > 2avgcq$.
  - $qlen_i \geq avgcq$ and $strike_i > 1$. The strike variable is incremented whenever one of these three conditions above are satisfied. These 3 conditions are used to identify and manage non-adaptive flows.
  - $avg \geq Max_{th}$ (like RED original).

- An arriving packet belonging to flow i is dropped randomly using the RED drop function if $Min_{th} \leq avg < Max_{th}$ and $qlen_i > Min_q$.

The computed packet loss probability increases with the *good run length* that is the number of packets not lost since last drop. This random drop is used for robust flows.

To protect fragile flows, an incoming packet is always accepted if the connection has fewer than Min_q packets buffered and if the average buffer size is less than Max_{th}. Otherwise arriving packets are subject to RED's random drop. This way, FRED decides whether to deterministically accept a packet from a low bandwidth connection. Unlike RED which estimates the average queue length at each packet arrival and misses the dequeue movements when no packets arrived, FRED averages at both arrival and departure of a packet. The authors gives the following example: if one packet arrives at time 0 when both the instantaneous and the average queue length have the same value equal to 500 packets and if the next packets arrives 250 packet times (service time of a packet) later, then the instantaneous queue size will be equal to 250 whereas the average queue size will remain to 500 packets. FRED avoids such miscalculation that could result in low link utilization (unnecessary packet drops). In addition, FRED does not modify the average queue size if the incoming packet is dropped unless the queue is empty. FRED computes the average per-connection queue length $avgcq$ by dividing the aggregate average queue length avg by the current number of active flows.

While FRED provides rough fairness, since it needs to track per-flow state information, it could face scalability problems [102] and also requires a sufficient buffer space in order to detect non responsive flows [35].

1.3.2 Stochastic Fair Blue (SFB)

Stochastic Fair Blue (SFB) is an active queue management scheme proposed by Feng *et al.* in [33, 35] with the aim to protect TCP flows against non-responsive flows (for example UDP flows) and therefore achieving fairness between these two kind of flows. SFB detects and rate-limits non-responsive flows to a fixed amount of bandwidth across the bottleneck link by combining two independent algorithms:

- The BLUE algorithm which uses a single marking probability to mark or drop packets when congestion occurs. BLUE is presented in Section 1.2.1.2.
- A Bloom filter [14] that allows a classification of objects through the use of multiple independent hash functions. This allows SFB to avoid the use of per-flow state information.

The algorithm of SFB is summarized as follows:

- SFB maintains $N \times L$ accounting bins which are organized in L levels with N bins at each levels.
- SFB maintains L independent hash functions. At level i, the hash function $h[i]$ $(0 \leq i \leq L-1)$ maps the flow into the j-th bin $(0 \leq j \leq N-1)$ of that level.
- The accounting bins track the queue occupancy statistics of traffic packets belonging to a particular bin. Therefore, the bins behave like virtual subqueues.
- Each subqueue or bin has a mark/drop probability p_m updated according to the bin occupancy.
- As a packet arrives at the queue, it is hashed into one of the N bins of the L levels. If the subqueue fill level is larger than a certain threshold, the p_m for the bin is increased by a fixed step of δ. If it is equal to zero, then p_m is decreased by the same fixed step of δ.

The traffic rate of non responsive flows is not reduced in case of packets drops due to the absence of the congestion avoidance algorithm for this kind of traffic. Therefore their corresponding p_m quickly reaches 1 in all the L bins into which it is hashed. Whereas for the TCP flows, the corresponding p_m remains less than 1 due to the presence of the congestion avoidance algorithm. At a given bin, if $p_m = 1$, the packet belonging to this bins is classified as originating from a non-responsive flow. Otherwise, it is classified as originating from a TCP flow.

A non-responsive traffic flow is driven into the path of bins with $p_m = 1$ at each level. Therefore, the total marking probability for non-responsive flows is 1 and their rates is limited (there is a fixed amount of bandwidth for this particular flows). For TCP traffic, the total marking probability p_{min} is less than 1. TCP traffic is, thus, protected from non-responsive flows. Packets are dropped or marked with probability p_{min}.

Algorithm 13 Stochastic Fair Blue (SFB) algorithm

/* Notations : */
/* – L: number of levels, N: number of bins per level. */
/* – $B[l][n]$: bin number $n \in \{0, \ldots, N-1\}$ belonging to level $l \in \{0, \ldots, L-1\}$. */
/* – h_i with $i \in \{1, \ldots, L\}$: hash function i. */
/* – q_{len}: instantaneous queue size of a specified bin. */
/* – p_m: probability which is used to mark or drop packets (marking probability). */
/* – p_{min}: minimum p_m value of all bins to which the flow is mapped into. */
/* – δ: amount by which p_m is incremented or decremented. */
/* – bin_size: capacity of the bins. */

⇒ **Upon packet arrival:**
Calculate hashes h_0, h_1, …, h_{L-1}.
Update bins at each level: the arriving packet is put into bins $B[0][h_0], \ldots, B[L-1][h_{L-1}]$.
for $i = 0$ **to** $L - 1$ **do**
  if $B[i][h_i].qlen > bin\_size$ **then**
    $B[i][h_i].p_m += \delta$.
    Drop packet.
  else
    if $B[i][h_i].qlen = 0$ **then**
      $B[i][h_i].p_m -= \delta$.
    end if
  end if
end for
$p_{min} = \min(B[0][h_0].p_m, \ldots, B[L-1][h_{L-1}].p_m)$.
if $p_{min} = 1$ **then**
  $ratelimit()$.
else
  Mark or drop packet with probability p_{min}.
end if

SFB can protect TCP-friendly flows from non-responsive flows without maintaining per-flow state. However, the authors indicate the limit of SFB in its ability to protect well behaved TCP flows in case of a very large number of non-responsive flows as compared to the number of bins present. In this case, the number of bins with p_m value to 1 increases, increasing the probability that a responsive flow get misclassified by being hashed into bins with p_m value to 1. Another drawback of SFB is its sensitivity to the variation of the RTTs

between flows. It gives poor fairness for flows that have widely varying RTTs [60].

Algorithm 13 describes the pseudo-code of the SFB scheme.

1.3.3 XRED

XRED is proposed by Hutschenreuther and Schill in [53] with the aim to reduce bandwidth waste for MPEG video transmission over routers using AQM. Indeed, RED drops packets without a distinction between packets. Some packets may be more crucial than the others. For example, the loss of few packets generated by a MPEG video application can make useless the entire application frame at destination. This results in a waste of the router bandwidth.

The idea of XRED is to describe a packet by three parameters: $FlowID$ for traffic flow, $ADUID$ for specified application data unit, and *Content Priority* for packet content with different priority. A list that records these three parameters is maintained at the router.

When MPEG frames are fragmented into IP packets, XRED assigns each packet with $FlowID$, $ADUID$ and *Content Priority*. When a packet is dropped, its $FlowID$, $ADUID$ and *Content Priority* are written into the list, and each arriving packet is checked by comparing its parameters with the stored parameters. The packet is discarded if its variable *Content Priority* is lower than the one in the list.

Although simulation results show that XRED reduces bandwidth waste, XRED has the following disadvantages XRED needs extra fields in the IP header. Moreover, it needs to record three parameters for each application flow, giving rise to scalability problems. Finally, it needs extra actions to write and read list and compare the three parameters, which might consume a significant amount of computer power at the router.

Among others per-flow accounting schemes that is not described in this chapter, is the Fair buffering RED (FB-RED) scheme which is proposed by Kim and Lee in [63] with the aim to address the problem of fair bandwidth sharing between flows.

1.4 RED with class-based threshold

1.4.1 Class Based Threshold RED (CBT-RED)

To solve the UDP-TCP fairness problem of RED, Class Based Threshold RED (CBT-RED) has been proposed by Parris *et al.* in [86]. CBT-RED sets the queue thresholds according to the traffic type and its priority. The UDP traffic is tagged and has its own drop threshold, which is different from that of the TCP traffic. The TCP traffic is, thus, protected from the UDP traffic. CBT-RED configures the RED parameter according to the traffic type.

1.4.2 Balanced RED (BRED)

As an extension to FRED [70], Balanced RED (BRED) has been proposed by Anjum and Tassiulas [5] with the aim to achieve fair bandwidth among TCP and UDP traffic. The principle of BRED is to regulate the bandwidth of a flow by keeping per-active flow accounting. This is done by dropping packets preventively so that the non-adaptive traffic that steal more bandwidth from the adaptive flows are penalized.

For this purpose, BRED introduces five global control variables that are used to control the bandwidth attained:

- W_m: the maximum number of packets that every flow is allowed to have in the buffer.
- l_i: minimum number of packets that flow i can have in the buffer before its packets start being dropped with probability p_i, for $i = 1, 2$.
- p_i: probability to drop a packet from a flow i when l_i is reached for this flow.

These parameters are set as follows:

$$l_1 = \beta l_2 \quad , \quad l_2 = \frac{B}{2\lceil \widehat{N} \rceil} \quad , \quad p_2 = \frac{\sqrt{\widehat{N}}}{\sqrt{\widehat{N}} + 10} \quad \text{and} \quad p_1 = \frac{p_2}{10} \ ,$$

where β is a constant ($0 < \beta \leq 1$) and $\widehat{N}$ is an estimate of the number of flows active at the gateway. It is calculated as:

$$\widehat{N} \leftarrow (1 - \omega)\widehat{N} + \omega N_{\text{active}} \ , \tag{1.27}$$

and N_{active} is the measure of the number of flows having at least one packet in the buffer. The parameter ω is set to 0.02.

The decision to drop or accept an incoming packet is mainly based on $qlen_i$ and gap_i which gives a step drop function. The variable gap_i prevents successive multiple drops that are harmful to adaptive flows. The three thresholds l_1, l_2 and W_m, on per-flow queue length $qlen_i$ divide the space of $qlen_i$ into four regions: $(0, l_1)$, (l_1, l_2), (l_2, W_m) and (W_m, ∞):

- If $qlen_i \in (W_m, \infty)$ then the packet is definitely dropped.
- If $qlen_i \in (l_2, W_m)$ and $gap_i > l_2$, then the packet is dropped with probability p_2.
- If $qlen_i \in (l_1, l_2)$ and $gap_i > l_1$, then the packet is dropped with probability p_1.
- If $qlen_i \in (0, l_1)$, then the packet is accepted.

$qlen_i$ and gap_i are increased if the packet is accepted. $qlen_i$ is decreased for each departure of a packet belonging to flow i.

Although BRED can minimize the differences in the bandwidth obtained by each flow, it needs to maintain the flow states, which means that its implementation complexity is proportional to the router buffer size.

1.4.3 RED In-Out (RIO)

Another variant of RED proposed for DiffServ (Differentiated Service) is RIO (RED In Out queue) [26, 27] where "in" stands for packets that are in compliance with the connection agreement and "out" stands for packets that are not. RIO has different dropping thresholds for each predefined traffic class. The lowest minimum and maximum thresholds are given to best effort packets. AF (Assured Forwarding) and EF (Expedited Forwarding) marked packets have lowest drop probability. However, in case of a non compliance with the SLA (Service Level Agreement), an AF packet is reclassified as best-effort class packet whereas an EF packet is dropped. EF packets have the advantage of using a separate priority FIFO queue. Whereas the AF packets share a RIO queue with best effort packets.

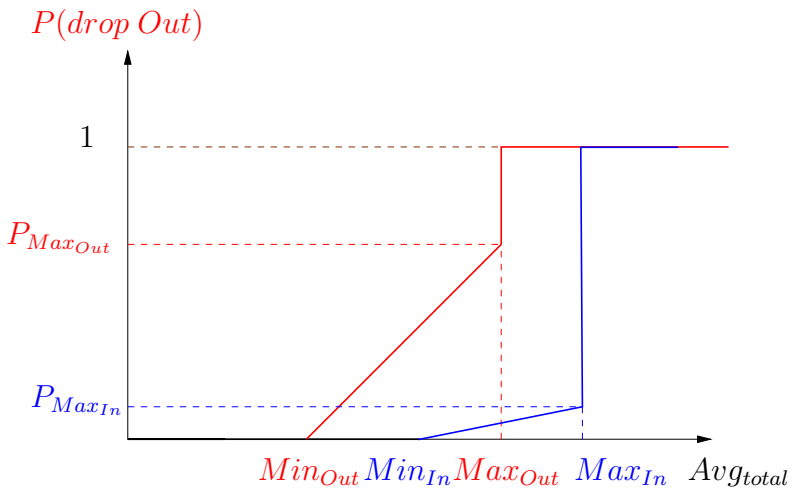


Figure 1.5: RIO drop functions

RIO uses the same mechanism as in RED and retains all the attractive attributes of RED (reduces the queue length, increases the throughput, avoids global synchronization). However, it has two different configuration thresholds for "in" and "out" packets in order to discriminates against *Out* packets in time of congestion that is to drop more aggressively "out" packets as compared to "in" packets. Hence, RIO uses twin RED algorithms for dropping packet, one for *Ins* and one for *Outs*. By choosing the parameters for respective algorithms differently, RIO is able to preferentially drop *Out* packets. There are three parameters for each of the twin algorithms. With the three parameters Min_{In}, Max_{In} and $P_{Max_{In}}$, $[0, Min_{In}]$ defines the normal phase, $[Min_{In}, Max_{In}]$ defines the congestion

Algorithm 14 BRED

/* Notations : */
/* – B: buffer size. */
/* – N_{active}: measure of the number of flows having packets in the buffer. */
/* – l_1: minimum number of packets that a flow can have in the buffer before its packets start being dropped with probability p_1. */
/* – l_2: number of packets that a flow can have in the buffer before its packets start being dropped more aggressively with probability $p_2 > p_1$. */
/* – W_m: maximum number of packets that the flow is allowed to have in the buffer. */
/* – $qlen_i$: number of packets from flow i in the buffer. */
/* – gap_i: number of packets accepted from flow i since last packet dropping from flow i. */

⇒ **At each arrival of packet from flow i:**
if it is the first packet of flow i (flow i is a new flow) **then**
  $qlen_i = 0$.
  $gap_i = 0$.
  $N_{\text{active}} = N_{\text{active}} + 1$
end if
if $qlen_i \geq W_n$ or buffer overflow **then**
  Drop packets.
else if $W_m > qlen_i > l_2$ and $gap_i > l_2$ **then**
  Drop packets with probability p_2.
else if $l_2 > qlen_i > l_1$ and $gap_i > l_1$ **then**
  Drop packet with probability p_1.
else if $qlen_i \leq l_1$ **then**
  Accept packet.
  $qlen_i = qlen_i + 1$.
  $gap_i = gap_i + 1$.
end if

⇒ **At each departure of packet from flow i:**
$qlen_i = qlen_i - 1$.
if $qlen_i = 0$ **then**
  $N_{\text{active}} = N_{\text{active}} - 1$.
  $gap_i = 0$.
end if

⇒ **At each packet drop from flow i:**
$gap_i = 0$.

avoidance phase, and (Max_{In}, ∞) defines the congestion control phase. Similarly, Min_{Out}, Max_{Out} and $P_{Max_{Out}}$ define the corresponding phases for Out packets (see Figure 1.5). The discrimination against Out packets in RIO is created by carefully choosing the parameters $(Min_{In}, Max_{In}, P_{Max_{In}})$ and $(Min_{Out}, Max_{Out}, P_{Max_{Out}})$. RIO performs this discrimination as follows :

- It drops Out packets much earlier than it drops In packets. This is done by choosing Min_{Out} smaller than Min_{In}.
- In the congestion avoidance phase, it drops Out packets with a higher probability by setting $P_{Max_{Out}}$ higher than $P_{Max_{In}}$.
- It goes into congestion control phase for the Out packets much earlier than than for the In packets by choosing Max_{Out} much smaller than Max_{In}.

Upon each packet arrival at the router, the router checks whether the packet is tagged as In or Out. If it is an In packet, the router calculates Avg_{In}, the average queue for the In packets. If it is an Out packet, the router calculates Avg_{total}, the average total queue size for all (both In and Out) arriving packets. The probability of dropping an In packet depends on Avg_{In}, and the probability of dropping an Out packet depends on Avg_{total}. The authors explain the choice of using Avg_{total} to determine the probability of dropping an Out packet as follows : If the average queue size of Out packets is used to control the dropping of Out packets, this would not cover the case where the total queue is growing due to arriving In packets. They claim that by using Avg_{total}, RIO can maintain short queue length and high throughput.

Algorithm 15 gives the pseudo-code of the RIO scheme.

1.5 Conclusion

We summarized in Table 1.1 and 1.2 the different active queue schemes presented in this chapter. In these tables, we used the following notations:

- CBC: class-based control.
- FBC: flow-based control
- IQS: instantaneous queue size.
- AQS: average queue size.
- PLR: packet loss rate.
- LU: link utilization

Algorithm 15 RIO

/* Notations : */
/* – Min_{In} and Max_{In}: minimum and maximum thresholds for In packet. */
/* – Min_{Out} and Max_{Out}: minimum and maximum thresholds for Out packet. */
/* – Avg_{In}: average queue size for the In packets. */
/* – Avg_{Out}: average queue size for the Out packets. */
/* – P_{In}: probability to mark or drop In packet. */
/* – P_{Out}: probability to mark or drop Out packet. */

⇒ **Upon packet arrival:**
if packet arrival is an In packet **then**
  Calculate the average In queue size Avg_{In}.
else
  Calculate the average queue size Avg_{total}.
end if
if packet arrival is an In packet **then**
  if $Min_{In} < Avg_{In} < Max_{In}$ **then**
    Calculate probability P_{In}.
    Mark or drop packet with probability P_{In}.
  else if $Max_{In} < Avg_{In}$ **then**
    Drop packet.
  end if
end if
if packet arrival is an Out packet **then**
  if $Min_{Out} < Avg_{total} < Max_{Out}$ **then**
    Calculate probability P_{Out}.
    Mark or drop packet with probability P_{Out}.
  else if $Max_{Out} < Avg_{total}$ **then**
    Drop packet.
  end if
end if

- IR: input rate.

All these described RED variants have improved the performance of the original RED. However, each variant has its own shortcomings and advantages. The choice of a RED variant should then be made according to the application requirements and the network situations. Hence, one can use the following guideline:

- For applications requiring fairness among flows, RED variants with per-flow control should be chosen. For example, for applications that needs fairness TCP and UDP traffics, the CBT-RED scheme would be a good choice. For a larger network, due to scalability issues, this category is not convenient.

- For applications focusing on throughput and delay performances, RED variants with aggregate control can be convenient. Among these variants, one can choose the scheme that gives the best performance according to the metric in focus (delay for REM, stability for LRED or SRED, loss rate for AVQ, ...).

In Chapter 2, we will present a new variant of RED named PSAND with aggregate control. We will compare the performances of PSAND and some of the variants presented above in Chapter 4.

Table 1.1: Comparison of different active queue management schemes (Part I)

	Scheme	Congestion index	Drop function	Target queue	Update of dropping probability	Changes from original RED
Aggregate control	RED	AQS	single linear	no	at packet arrival	–
	BLUE	PLR, LU	step function	no	time interval function, LU, PLR	step increase/decrease
	SRED	IQS, estimated number of active flows	3 segment step	yes	at packet arrival	step drop function, number of active flows, IQS
	DSRED	AQS	2 linear	no	at packet arrival	2 linear with different slopes
	P	IQS	single linear	no	at packet arrival	IQS instead of AQS
	PI	IQS, IR	$p = p + ax - by$	yes	time interval	queue and rate mismatch
	REM	IQS, IR	$p = 1 - \phi^{-c}$	yes	time interval	queue and rate mismatch
	AVQ	IR	virtual queue capacity	no	at packet arrival	completely different
	SFC	IQS, IR	$p = k_1 a + k_2 y$	yes	at packet arrival	queue and rate mismatch
	LRED	IQS, PLR	$p = p + \beta\sqrt{y}x$	yes	time interval	queue and rate mismatch
	$\text{ARED}_{\text{Feng}}$	AQS	single linear	no	time interval	adaptation of Max_p
	$\text{ARED}_{\text{Floyd}}$	AQS	single linear	yes	time interval	adaptation of Max_p
	PSAND	AQS	single linear	yes	time interval	adaptation of Max_p
CBC, FBC	FRED	AQS	single linear	no	at packet arrival	per-flow Max_{drop}
	BRED	IQS, number of active flows	4 segment step	no	at packet arrival	per-flow IQS, number of active flows, step drop function
	SFB	IQS	step function	no	at packet arrival	organize sub-queues in Bloom filter
	RIO	AQS	2 single linear	no	at packet arrival	per-class Max_p

Scheme	Policies used to detect congestion and to drop packets
RED	Uses (weighted) average queue size to calculate the packet drop probability. When the average queue size is higher than a preconfigured threshold, RED begins to drop new arrival packets with a probability proportional to average queue size and with a slope of Max_p.
BLUE	Increases p if the instantaneous queue size exceeds L and has not been updated for over $freeze\_time$. Decreases p if the link is idle for over $freeze\_time$.
SRED	Keeps a zombie list to keep track of recently seen flows, to detect misbehaving flows, and to estimate the number N of active flows. N is used in the computation of packet dropping probability.
DSRED	Uses 2 linear drop functions. Consequently defines 3 thresholds and divides the state of AQS into 4 regions.
P	Calculates the dropping probability as for RED but uses the instantaneous queue size instead of the (weighted) average queue size in the computation.
PI	Calculates the dropping probability as in Equation (1.21).
REM	Defines the price $p(k)$ as in Equation (1.9) and calculates the dropping probability according to Equation (1.11).
AVQ	Maintains a virtual queue. At each packet arrival, enqueue a fictitious packet and update the virtual queue capacity using Equation (1.17). Drops a real packet only if the virtual queue overflows.
LRED	Measures the latest packet loss ratio, and uses it as a complement to queue length in order to adjust packet drop probability according to Equation (1.24).
$\text{ARED}_{\text{Feng}}$	Keeps all the state variables of RED and adapts dynamically Max_p by using a MIMD (multiplicative increase, multiplicative decrease) scheme.
$\text{ARED}_{\text{Floyd}}$	Keeps all the state variables of RED and adapts dynamically Max_p by using a AIMD (additive increase, multiplicative decrease) scheme.
PSAND	Keeps all the state variables of RED and adapts dynamically Max_p according to the variation of the instantaneous queue size and the closeness of the instantaneous queue size from its target queue size.
FRED	Monitors the per-flow queue length and fine-tunes the dropping decision w.r.t. the per-flow queue length.
BRED	Defines 3 thresholds and divides the state of per-flow queue length into 4 regions. Fine-tunes the dropping decision w.r.t. the per-flow state.
SFB	Applies a Bloom filter to hash flows into bins. Each bin maintains queue occupancy statistics for flows that map into that bin and a corresponding drop probability. For a given flow, the dropping probability is calculated based on queue occupancy statistics of the various bins.
RIO	Uses twin RED algorithms for dropping packets: one for *in* packets and one for *out* packets. Preferentially drops *out* packets during periods of congestion.

Table 1.2: Comparison of different active queue management schemes (Part II)

Part III

Configuration of RED Parameters

Chapter 2

Adaptation of the parameter Max_p

Contents

2.1 Introduction

Achieving high link utilization and low queueing delay are two antagonistic objectives of a buffer management scheme. Small buffers can guarantee low queueing delays but often suffer from packet loss and low link utilization. In case of a highly bursty traffic, a small buffer overflows very quickly causing packet loss. As a result, TCP flows back off their transmission rate leading to low throughput. To achieve these two goals, AQM (Active

Queue Management) schemes like RED trigger packet dropping (or marking, if explicit congestion notification is enabled) before buffer overflows with a drop probability which is proportional to the degree of congestion. This way, the RED queue management scheme manages to reduce the queuing delay and increase the link utilization rate. Nevertheless, RED exhibits two main weaknesses. The first weakness is that the queuing delay varies with the network load and with the parameters settings ([41, 74, 75, 78, 82, 105]). Indeed, the average queue size oscillates around Min_{th} if the traffic load is light or the value of the parameter Max_p is very large. Otherwise, it oscillates around Max_{th} if the traffic load is heavy or Max_p has a very small value. Another drawback of RED is that the throughput is also sensitive to the traffic load and to the parameters setting([41]). In case of a highly congested network if the average queue size oscillates around Max_{th}, the resulting packets drops increases leading to a decrease in the throughput.

The RED mechanism has been widely studied in literature with the aim to alleviate these drawbacks. The different RED variants differ in the way they measure the congestion. Indeed, while most of the proposals detect congestion based on the queue lengths at the link (e.g. RED, $ARED_{Feng}$, $ARED_{Floyd}$, GRED: Gentle RED), other proposals detect congestion based on the arrival rate of the packets at the link (e.g. virtual queue-based schemes) and some use a combination of both (e.g. PI). Others use the traffic input rate, the events of buffer overflow or emptiness or a combination of different factors. Among all these works, two different approaches can be clearly distinguished.

The first approach is to build quantitative models allowing the prediction of RED parameters based on network variables like the round trip time, the number of flows and link bandwidth. For instance, works like [10, 11, 12, 47, 51, 66, 97, 104, 105] use a control theoretic analysis to model the parameter setting of RED. This modeling approach provides a better understanding of the dependency of RED on traffic conditions. However, it is still difficult to retrieve or infer accurate informations about network variables from local observations. In practice, network load characteristics are not known from RED nodes.

Therefore, several authors have advocated an adaptive approach which can be deployed in the current Internet. Indeed, this second approach does not rely on hypotheses on the type of traffic as it infers the traffic load from an average queue size. The works of Feng *et al.*, and Floyd *et al.* [32, 34, 41] used this adaptive approach of RED, named ARED (Adaptive RED), that performs a constant tuning of RED parameters according to the traffic load.

The mechanisms adopting this second approach, should carefully choose how fast the adaptation should be performed while maintaining the system stability. Since the system jointly uses the end-to-end TCP congestion control as well as the congestion control operating inside routers through active queue management schemes, the system could be sensitive to the adaptation frequency and may show an undesirable behavior. On one hand, if the adaptation process is too fast then the system responds too quickly to the changes of the network conditions leading to large oscillatory behavior and even to instability. On the other hand, if the adaptation process is too slow then the system shows a sluggish behavior leading to more packet losses and thus to a decrease of the throughput. Indeed, because

of this slow response, the average queue size could exceed Max_{th} resulting in an increased dropping rate which induces the TCP sources to slow down their transmission rate and then consequently reduce the throughput. In order to give a desirable performance, an adaptive scheme should therefore avoid these two situations.

In this chapter, we adopt the adaptive approach. Like Floyd *et al.* in [41], we believe that the basic insight for a solution to RED problems lies in this approach since it can be proposed for deployment in the Internet and does not need to know the network scenario parameters. Our point of view differs however from these works, in the way parameters are adjusted. Our goal is to achieve a performance improvement over adaptive RED for the queueing delay and the delay jitter without sacrificing the loss rate. To this end, we propose a scheme which adapts all three parameters of RED using target-oriented adjustments which are a function of the distance to the performance objective. Simulation results show that our scheme can stabilize the queue size and achieves a more predictable average queue size without substantially increasing the loss rate. Finally, it keeps also the queue size away from buffer overflow and buffer underflow independently of the number of connections.

We briefly describe in Section 2.2 the adaptive approach of RED. In Section 2.3, we propose a new algorithm that sets the RED parameters and evaluate it in Section 2.4 by extensive simulations under `ns` [94]. We conclude the chapter and propose future direction of investigations in Section 2.5.

2.2 Background on ARED and motivation

We make a brief recall of the mechanism of ARED described in [34, 41]. The idea behind this mechanism is to improve RED performance by adapting the maximum dropping probability Max_p according to the traffic load. The authors derived this conclusion based on their observations concerning the impact of traffic load on early detection. The number of congestion notifications that should be sent to the sources and that achieves a low packet loss rate and a high link utilization depends on the number of active connections [34].

Hence, an active queue management should be able to send an appropriate number of congestion notification to the TCP sources. Because of its early random drop, RED satisfies more this requirement as compared to Drop Tail. However, in case of a large number of TCP connections, if congestion notifications are not sent to enough TCP connections, then an increased packet loss due to a continual buffer overflow is observed making RED behaving like Drop Tail. In this case, whatever is the fixed value of the parameter Max_p, a high link utilization is achieved. However, early detection should be made more aggressive so that TCP connections back off their transmission rate and thus reduce the offered load and the packet loss rate. In case of a small number of TCP connections, if congestion notifications are sent to too many TCP connections, then the offered load is excessively reduced causing an underutilization of the bottleneck link. In this case, in order to achieve high link utilization, the early detection should be made more conservative.

The algorithm of ARED takes into account these observations by using the average

queue length to infer load modifications and tune the behavior of RED towards more or less aggressiveness. This way, ARED is intended to perform better than RED. According to the adaptive algorithm of Feng *et al.* [34] illustrated by Algorithm 6 (page 38), an average queue size close to Min_{th} indicates a too aggressive early detection mechanism. In this situation, the value of Max_p is decreased by a constant multiplicative factor α. On the other hand, if the average queue size is close to Max_{th} then it means that the early detection is too conservative. In this case the value of Max_p is increased by a fixed multiplicative factor β. However, if the average queue size is between Min_{th} and Max_{th}, the authors do not adapt Max_p because they consider that the early detection mechanism behaves as desired and can avoid packet loss rate and link underutilization. This setting of Max_p makes normally the queue size oscillate between Min_{th} and Max_{th}.

Floyd *et al.* in [41] modified this algorithm so that the average queue size oscillates closely to a target queue size within a target range half way between Min_{th} and Max_{th} thus achieving a predictable average delay. Unlike in [34], where Max_p is adapted at every packet arrival, [41] adapted Max_p at a slower time scale, typically every 500 ms, by using an additive increase and a multiplicative decrease instead of the multiplicative increase and a multiplicative decrease of the original ARED algorithm of [34]. If the average queue size exceeds the specified target range, then Max_p is adapted by an additive increase once every time interval. Otherwise if the average queue size is below the target range then Max_p is adapted by a multiplicative decrease. Otherwise if the average queue size is within the target range then Max_p is not adapted. Algorithm 7 illustrates this approach.

By adapting Max_p with a fixed adjustment factors (additive or multiplicative) as in [34, 41] and as our simulation results will confirm afterward, ARED improves the performance of the original RED while still exhibiting a certain amount of queue oscillations. Note that if the traffic load shows a slight change there is no need of applying a sharp change on Max_p. These sharp changes of the value of Max_p as compared to the traffic load changes can in themselves be the cause of oscillations of the queue size. By using a fixed adjustment factor on Max_p, both schemes do not adapt Max_p at a rate that reflects the effective change rate of the traffic load. Hence, our basic idea is to test whether there is a possibility of performance improvement not only on the original RED but also on ARED, if we adapt Max_p by using a more elaborate dynamic adjustments reflecting the effective change rate of the traffic load, and if we allow more modifications to all of the three parameters (Min_{th},Max_{th},Max_p).

On the other hand, an improper control may as well lead to performance deteriorations. We describe in the next section an algorithm that adapts the parameter Max_p by using a dynamic change rate which is a function of the changes in the average queue length that infer the traffic load changes. In Section 2.4, we show that this algorithm can indeed improve the performance of ARED. In Chapter 4, we will compare its performance with that of other well known or recently proposed active queue management schemes found in the literature.

2.3 RED parameters tuning

We present an alternative adaptive algorithm (named PSAND) that aims at improving the performance of ARED. Our principal objective is to minimize the variance of the instantaneous queue size and improve the stability of ARED under the following constraints:

- achieve a specified target average queue size.
- avoid an excessive increase of the packet loss rate as compared to the original RED and ARED.

The decrease of the variance of the instantaneous queue size leads to a decrease of a delay jitter and therefore more predictable queueing delay.

Following the approach of [41], we leave the network operator to choose the average target queue size as a trade-off between link utilization and delay. Hence, as it has been stated in [41], QoS should give a certain level of guarantee to offer a sufficiently low delay for a certain type of applications and traffics, such as interactive voice communications. To sign a Service Level Agreement (SLA) by stating that no packets will experience a delay of more than a specified target delay, network operators would like to have a rough estimation of the average delays in their congested routers.

This section explains how to adapt Max_p with a multiplicative factor which is computed dynamically as a function of the changes in the average queue length in order to achieve our goal. For this purpose, we follow the main idea of the algorithms of ARED [34, 41] which is to infer whether Max_p should be made more or less aggressive by observing the variations in average queue length. As we have seen, both methods leave Max_p constant when the average queue size stays in some interval, and change it by a constant factor otherwise. The difference between our approach and the schemes presented in [34, 41] is that in our approach, for a given time interval, the decision that should be taken on the level of aggressiveness or conservativeness of Max_p depends directly on the queue size change and the position of the queue size as compared to its target value.

Hence, our approach differs from this algorithm in the following points:

- Instead of using a threshold mechanism as in [34, 41], we infer a load increase from the increase in the average queue size: an increase (respectively a decrease) in the average queue size by a certain factor is directly translated into an increase (respectively a decrease) of Max_p by the same multiplicative factor. The multiplicative factor used to reflect the traffic load variation is the ratio of the value of the average queue size at the beginning of the current interval (denoted by $\hat{K}_{cur}$) over the value of the average queue size at the beginning of the previous interval (denoted by $\hat{K}_{prev}$). We denote by *change_rate* this ratio that measures the queue length change:

$$change\_rate \;=\; \frac{\hat{K}_{cur}}{\hat{K}_{prev}} \;.$$

- In addition, instead of keeping the average queue size between Min_{th} and Max_{th} as in [34], we follow the approach of Floyd *et al.* in [41] that tends to bring the average queue size to a specified target value that reflects the desired average queueing delay. Our aim is to bring the average queue size closer to its target value more quickly. For this purpose, we measure the gap between the current average queue size and its target value and increase or decrease Max_p proportionally so as to reduce this gap. Hence Max_p is adapted by a multiplicative factor which is a ratio of the average queue size at the beginning of the current interval ($\hat{K}_{cur}$) over the target average queue size (denoted by $\hat{K}_T$). We denote by $prox\_rate$ this ratio that measures the proximity of $\hat{K}_{cur}$ as compared to $\hat{K}_T$:

$$prox\_rate \;=\; \frac{\hat{K}_{cur}}{\hat{K}_T}\ .$$

  An average queue size lower (respectively larger) than its target value triggers a decrease (respectively an increase) of Max_p.

- We adapt Max_p with a factor which is a function of these two ratios representing a trade-off between the two increasing/decreasing trends. We denote this rate by r and compute it as follows:

$$\begin{aligned} r &= prox\_rate \times change\_rate \\ &= \frac{\hat{K}_{cur}^2}{\hat{K}_T \times \hat{K}_{prev}}\ . \end{aligned} \tag{2.1}$$

The variation of r and its influence on the parameter Max_p is summarized as follows:

- If $\hat{K}_{cur}$ is close to $\hat{K}_T$ and $\hat{K}_{cur}$ close to $\hat{K}_{prev}$ then Max_p will be almost constant.

- If $change\_rate > 1$ and $prox\_rate > 1$, then Max_p will be highly increased.

- If $change\_rate < 1$ and $prox\_rate < 1$ then Max_p will be highly decreased.

- if $change\_rate < 1$ and $prox\_rate > 1$ (i.e. the traffic tends to decrease but the average queue size remains above its target value.) and if the traffic decrease rate is higher ($change\_rate$ very low) than the change rate of the average queue size as compared to its target value, than Max_p will decrease.

- if $change\_rate > 1$ and $prox\_rate < 1$ (i.e. the traffic tends to increase but the average queue size remains below its target value.) and if the traffic increase rate is higher than the change rate of the average queue size as compared to its target value, than Max_p will increase otherwise it will have a decrease tendency.

For these two last cases, if *change_rate* and *prox_rate* are compensated i.e the value of one of these two ratios is the inverse of the other one, then Max_p will remain constant. Hence, for these last two cases, the behavior of Max_p will depend on the compensation of *prox_rate* and *change_rate*. Max_p will be more or less increased or decreased depending on the respective value of these two ratios.

The factors *change_rate* and *prox_rate* are illustrated in Figure 2.1. Figure 2.1 presents the evolution of the weighted average queue size obtained from a 10 seconds simulation under NS. These two ratios are measured once every time interval corresponding to the time interval at which Max_p is adapted (0.5 seconds in this example). The target average queue size ($\hat{K}_T$) is 10 packets. *change_rate* measures the queue length change by comparing the average queue size at the beginning of the current interval ($\hat{K}_{cur}$) to its previous value at the beginning of the previous interval ($\hat{K}_{prev}$). The value of $\hat{K}_{cur}$ evolves during the interval whereas $\hat{K}_{prev}$ remains constant. *prox_rate* measures the gap between the current average queue size $\hat{K}_{cur}$ and its specified target value $\hat{K}_T$. Figure 2.1 shows a decrease of both ratios at time 9 seconds and an increase at time 9.5 seconds.

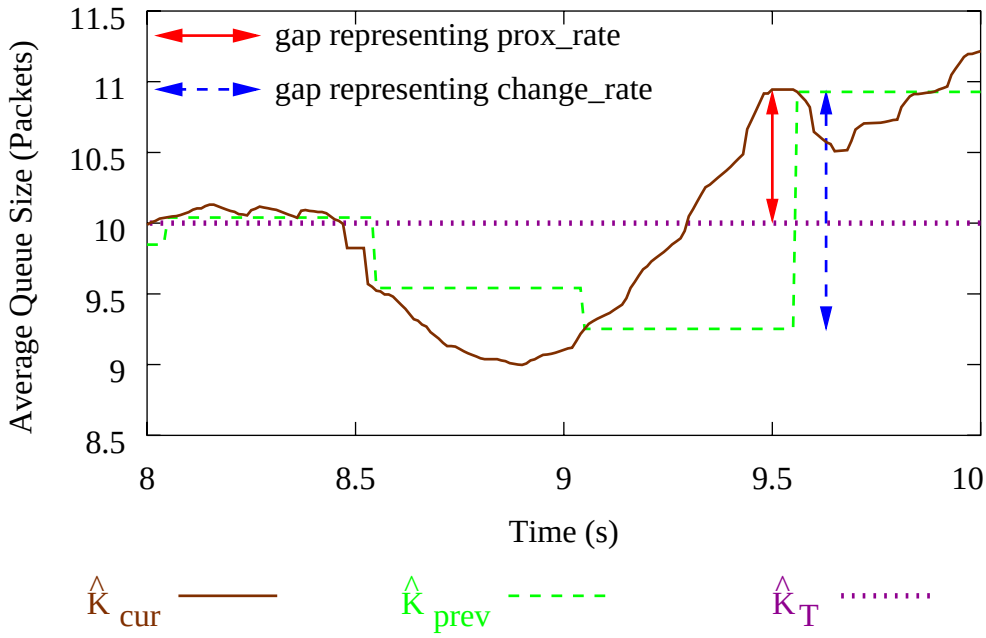


Figure 2.1: Measuring the traffic change

According to the results of our experiments, if we used directly r as a multiplicative factor of Max_p, our mechanism would not be aggressive enough in case of a severe congestion. In order to make the change rate more or less aggressive and perfect our model, we modified the influence of r by using a simple functional form. We set:

$$\beta \;=\; coef \times r^{\gamma} \;,$$

and let the adaptive scheme be:

$$Max_p \;=\; Max_p \times \beta \;. \tag{2.2}$$

Indeed, the queue length does not allow alone to determine the severity of congestion that is the number of flows [33]. For instance a single highly active source can cause a persistent queue as well as large number of flows can. Hence, since the mechanism that uses directly r, measures the queue length change, it can not be able to detect efficiently the degree of severity of the congestion and can not therefore respond aggressively enough. Moreover, as stated in [34], congestion notification does not directly depend on the number of connections multiplexed across the link. Nevertheless, congestion notification should be given at a rate which is high enough to avoid packet loss due to buffer overflow and low enough to avoid underutilization of the link. The functional form (2.2) should be able to adapt Max_p with the aim of sending congestion notifications at a rate that reflects better the change of the congestion level.

Algorithm 16 Algorithm PSAND

$\hat{K}_T = C \times \text{Delay}_{\text{target}}$.
$INTERVAL = 0.5$ seconds.

/* *Adaptation of* Max_p. */
for each INTERVAL **do**
  prox_rate $= \hat{K}_{cur}/\hat{K}_T$.
  change_rate $= \hat{K}_{cur}/\hat{K}_{prev}$.
  $\beta \leftarrow coef \times (\text{prox\_rate} \times \text{change\_rate})^{\gamma}$.
  $Max_p \leftarrow \max(\underline{Max_p}, \min(Max_p \times \beta, \overline{Max_p}))$.
  $\hat{K}_{prev} \leftarrow \hat{K}_{cur}$.
end for

/* Fixed parameters configured by the network operator */
/* based on his objectives of quality of service. */
C : The link bandwidth.
$\text{Delay}_{\text{target}}$: Target queueing delay.

/* Fixed parameters of PSAND algorithm. */
INTERVAL : time interval between two consecutive adaptations of Max_p.
$\hat{K}_T$: target queue size.
$\overline{Max_p}$: upper bound of Max_p ($\in [0.5, 1]$).
$\underline{Max_p}$: lower bound of Max_p (fixed to 0.01).

/* variable parameters */
$\hat{K}_{cur}$: Weighted average queue size at the beginning of the current interval.
$\hat{K}_{prev}$:Weighted average queue size at the beginning of the previous interval.

Algorithm 16 summarizes this scheme. Increasing (respectively decreasing) $coef$ means making Max_p respond more aggressively (respectively less aggressively). The use of γ allows to make r less aggressive if the queue size is low and more aggressive if the queue size is high.We illustrate the influence of the parameters $coef$ and γ on our adaptive RED performance in Section 2.4.

As in [41],our algorithm uses a lower bound of Max_p which is equal to 0.01 and an upper bound of Max_p chosen from the interval $[0.5, 1]$ as we will describe in section 2.4.5. Floyd *et al.* [41] used 0.5 as an upper bound of Max_p since the authors claimed that they do not try to optimize RED for a rejection rate exceeding 50%. By taking RED in *gentle* mode, *i.e.* when the drop rate increases from 0 to Max_p, the average queue size increases from Min_{th} to Max_{th}. And when the drop rate increases from Max_p to 1, than the average queue size increases from Max_{th} to $2 \times Max_{th}$. Hence, a loss rate of 25% implies an average queue size equal to $\hat{K}_T$. Therefore, if the drop rate reaches 100%, then the average queue size exceeds its target value by a factor of 4. This allows to bound the increase rate of the average queue size. The lower bound of Max_p fixed to 0.01 is motivated by the desire to bound the lowest values that can Max_p take.

The configuration of the parameters Min_{th}, Max_{th} used by our algorithm is discussed in Chapter 3. Section 2.4.3 and 2.4.4 deal with the seeting of the parameters $coef$ and γ.

2.4 Stability and performance measures

2.4.1 Simulation settings and metrics

We evaluate our algorithm (PSAND) through simulations that use the same network configuration as in [34, 41] for a better comparison with these two versions, respectively denoted $ARED_{Floyd}$ and $ARED_{Feng}$ (see Figure 2.2).

Every 0.1 seconds, long-lived TCPSack connections that generate FTP traffic originate from the leftmost node S_1 and end at the rightmost node S_3. Another forward FTP traffic is generated by node S_2 and addressed to node S_3. Finally a reverse FTP traffic is started from node S_3 and send to the node S_1. The bottleneck link located between nodes R_1 and R_2 has a capacity of about 1.5Mbps. The capacity of the other links is about 10Mbps. The bottleneck link for forward traffics is provided with a RED queue of limited capacity of 35 packets. In the same manner, a RED queue with the same capacity is assigned to the reverse FTP traffic at the bottleneck link. The other links are provided with a queue of infinite capacity. We collect the statistics from the RED queue every 0.01 seconds.

For all simulations, we follow the approach of [56, 105, 41] to set the average queue size weight (ω_q) as a function of the link bandwidth. The average queue size weight (ω_q) should be configured so that the RED mechanism is able to detect the initial stages of congestion. For this purpose, ω_q should not be set too low in order to avoid that the average queue size responds too slowly to changes in the actual queue size. Indeed, even though the low pass filter characteristics of the average queue size is able to accumulate short term congestion and allows the weighted average queue size $\hat{K}_{cur}$ follows the long term changes

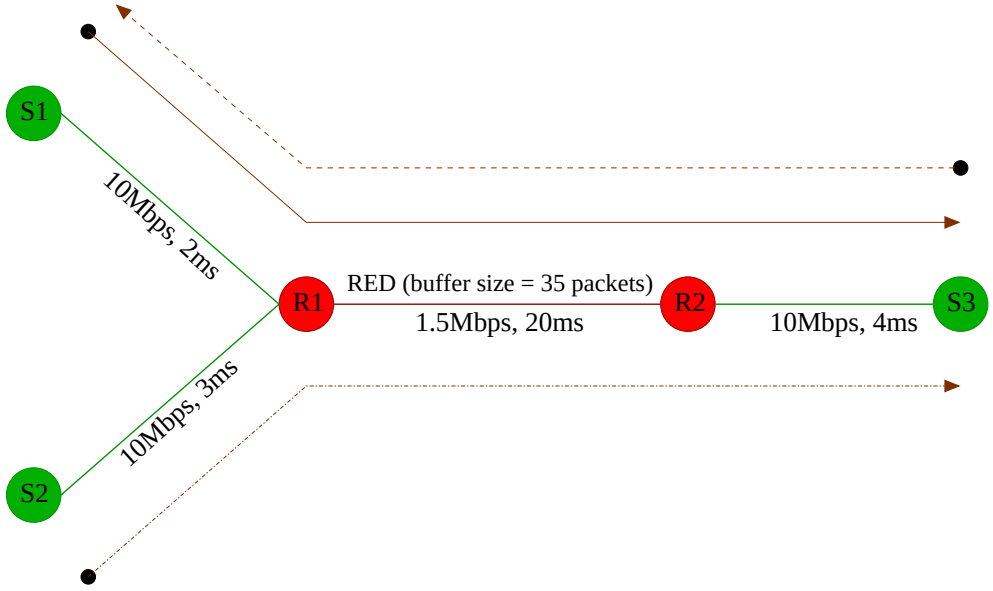


Figure 2.2: Network topology

of the instantaneous queue size K_{cur} that reflects persistent congestion in networks, it can lead to slow responses to the changes in the long-term congestion. For example, after a long-term congestion, $\hat{K}_{cur}$ can still remain high even if the instantaneous queue size K_{cur} has decreased to zero. RED continues to drop packets even after the end of congestion. This slow response to the congestion decrease leads to a low throughput. Larger value of ω can improve the response time but traces short-time congestion that is ω_q should not be set too high in order to avoid the average queue size responds too quickly to the changes of the actual queue size and follow very closely the evolution of the instantaneous queue size.

Assuming that :

- The average queue size is initially equal to 0 and that the instantaneous queue size jumps from 0 to 1 and remains constant to 1,
- Packets arrive and depart at the same rate,

then $-1/\ln(1-\omega_q)$ packets arrivals are necessary so that the value of the average queue size jumps from 0 to $0.63 = 1 - 1/e$ [43]. For example, for $\omega_q = 0.002$, 500 packet arrivals are necessary. Hence, the value $-1/\ln(1-\omega_q)$ is the time constant (expressed in packets arrivals per second) necessary for the average queue size to reach 63% of its new value.

Assuming that the link bandwidth is equal to C packets per second, and knowing that the service time is equal to the packets arrival rate , then $-1/(C\ln(1-\omega_q))$ seconds are necessary to serve the $-1/\ln(1-\omega_q)$ packets. We would like this time to be constant and equal to $t^\star$. The parameter ω_q is then computed as :

$$\omega_q \;=\; 1 - \exp\left(-\frac{1}{C \times t^{\star}}\right) \;.$$

As in [41], we configured ω_q in automatic mode such that the time constant for the average queue size estimator is equal to 1 second (which is equivalent to 10 RTT for a value of an RTT equal to 100ms). Therefore, ω_q is finally configured as follows:

$$\omega_q \;=\; 1 - \exp\left(-\frac{1}{C}\right) \;.$$

For PSAND, the parameters Min_{th} and Max_{th} are configured as described in Algorithm 16 ($Min_{th} = 0$, $Max_{th} = 20$) and the parameter Max_p is adapted every 0.5 seconds like in [41]. Whereas for ARED$_{Feng}$, Max_p is adapted frequently at a shorter time scale, every time the weighted average queue size is updated. We also set $coef = 1.75$ and $\gamma = 1.5$ for PSAND in order to obtain a desirable performance independently of the number of flows. For the setting of the other simulation's parameters, for ARED$_{Feng}$ and ARED$_{Floyd}$, we took the values they recommended in [34, 41] (refer to Table 2.1).

Table 2.1: Setting of the simulations parameters

	Min_{th}	Max_{th}	$InitialMax_p$	$\hat{K}_T$	α	β	$coef$	γ
ARED$_{Feng}$	5	15	0.02	–	3	2	–	–
ARED$_{Floyd}$	5	15	0.1	10	0.01	0.9	–	–
PSAND	0	20	0.1	10	–	–	1.75	1.5

For all the simulations, one connection starts at time 0 second from node S_1, another starts at time 1.0 second from node S_3, and another one starts at time 3 seconds. For these three connections, we set the TCP congestion window size to 15 packets and the size of the packets is 1600 bytes. In addition to these connections, we added a number of connections according to the simulation. For the additional connections, we set the TCP window size to 20 and the size of the packets to 1000 bytes.

For Figures 2.3 to 2.6, additional flows started each every 0.1 seconds from node S_1 at time 50 seconds. The number of additional flows for Figures 2.3, 2.4, 2.5 and 2.6 are respectively 20, 50, 70 and 100.

Figures 2.3 to 2.6 present the evolution of the instantaneous queue size and the weighted average queue size obtained from one simulation during 100 seconds. Whereas for Figures 2.9 and 2.10, one point on the curve is the average of 100 independent simulations, and one point on the curves of Figure 2.11 corresponds to an average of 200 independent simulations. For these figures, in addition to the basic three connections, a number of connections depending on the total number of flows desired is started at time 3.5 seconds each every 0.1 seconds from node S_1 and stopped at time 100 seconds.

We have measured the following metrics:

- the delay jitter represented by the variance of the instantaneous queue size (Figure 2.9(a)).

- the average queueing delay inferred by the true average queue size, *i.e.* the average of the instantaneous queue size (Figure 2.9(b)).
- the packet loss rate that is computed as the ratio of the number of lost packets over the number of packets arriving at the queue (Figure 2.9(c)).
- the distribution of the instantaneous queue size. We specially focus on the probability that the queue is empty or full and the probability that the queue is very close to the target queue size (Figures 2.10 and 2.11).

We will present subsequently the results of these measurements.

2.4.2 Performance results

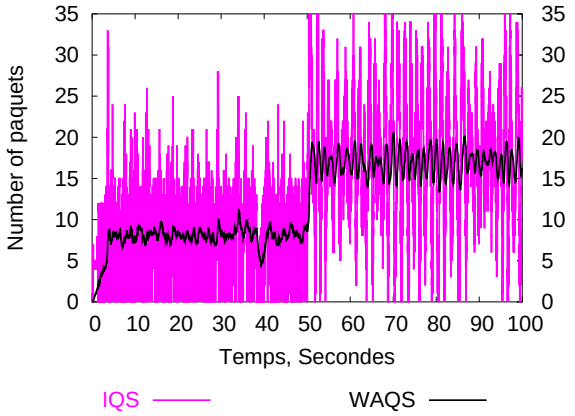
2.4.2.1 Qualitative observations

We compare our approach by using the metrics presented in the previous section with other queue management mechanisms: the original RED, ARED$_{Feng}$ and ARED$_{Floyd}$ [34, 41]. We first make a visual and qualitative comparison of the evolution of the instantaneous and weighted average queue size through Figures 2.3 to 2.6. These figures allow us to observe not only the behavior of each mechanism in presence of a high and a low traffic load but also how each mechanism reacts in response to a change of the congestion level. Figures 2.3 to 2.6 show the response to a congestion increase whereas Figures 2.7 to 2.8 show the response to a congestion decrease.

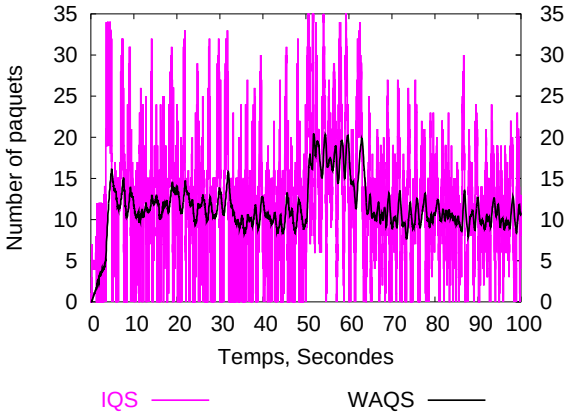
2.4.2.1.1 Traffic increase

Let us first consider the congestion increase. We can observe in Figure 2.3(d) that our mechanism responds more quickly to a rapid change in the congestion level and shows a better adaptation to the traffic increase as compared to RED (Figure 2.3(a)) and the other versions of ARED (Figures 2.3(b) and 2.3(c)). After a sharp change in the average queue size at time 50 seconds, it takes less than 5 seconds for our scheme to bring the average queue size back down to its target value (10 packets) whereas ARED$_{Feng}$ in Figure 2.3(b) takes roughly 15 seconds and ARED$_{Floyd}$ in Figure 2.3(c) takes 10 seconds. We can see that after this short transition period, all three ARED schemes (Figures 2.3(b) to 2.3(d)) bring back the average queue size back to its target value. This is not the case for RED (Figure 2.3(a)). However, we can observe in Figure 2.3(d) that PSAND differs from the other ARED schemes (Figures 2.3(b) and 2.3(c)) in the fact that the oscillations of the average queue size after 50 seconds are very close to the one observed before the congestion increase. This shows that our scheme has responded more successfully to the congestion increase induced by 20 additional flows.

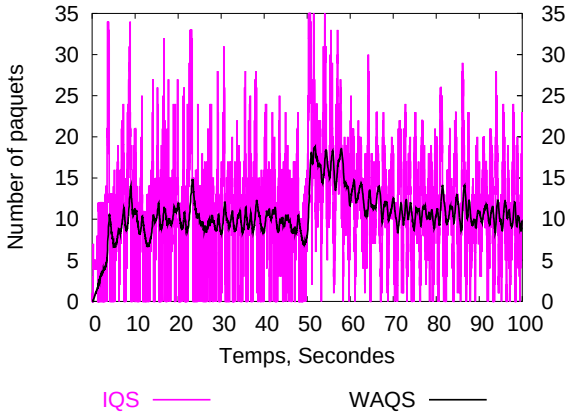
For a higher traffic load increase (50 additional flows), ARED$_{Feng}$ in Figure 2.4(b) is unable like RED (Figure 2.4(a)) to bring back the average queue size back to its target range. ARED$_{Floyd}$ in Figure 2.4(c) attempts unsuccessfully to bring it back but shows a succession of higher and lower oscillations every 15 seconds, whereas PSAND in Figure 2.4(d) brings



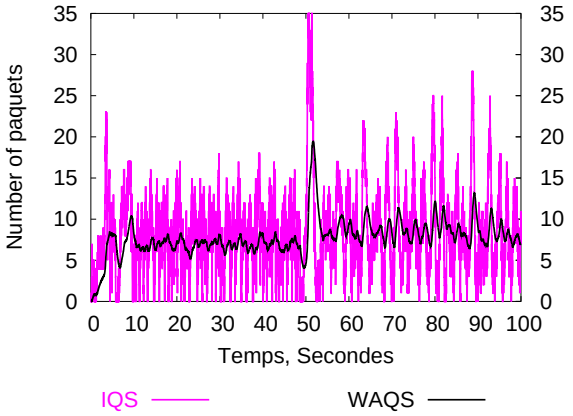
(a) RED



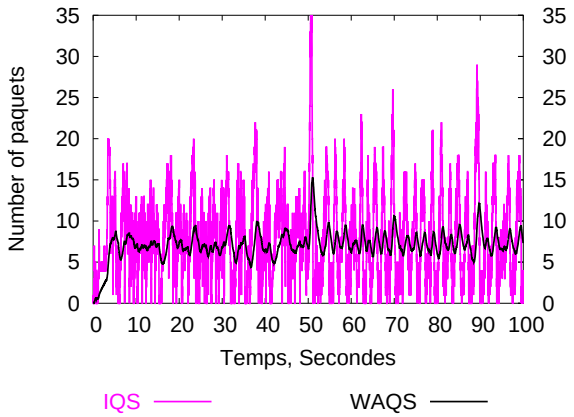
(b) ARED version Feng *et al.*



(c) ARED version Floyd *et al.*

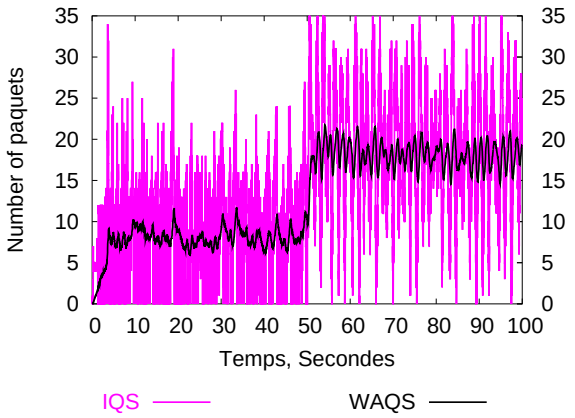


(d) PSAND (upper $Max_p = 0.5$)

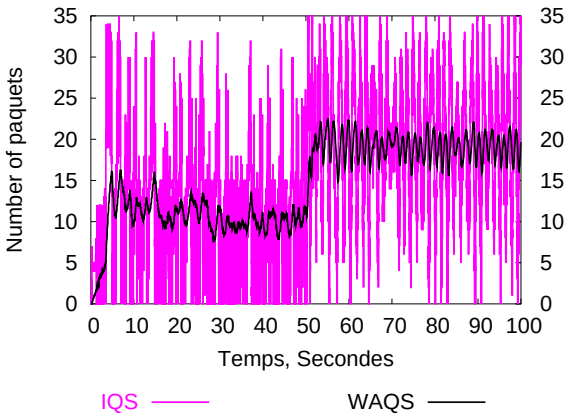


(e) PSAND (upper $Max_p = 0.75$)

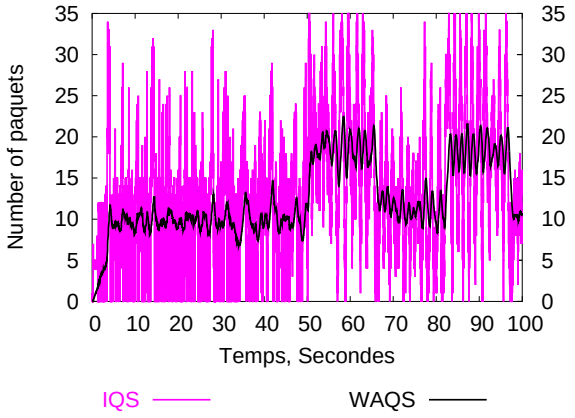
Figure 2.3: Evolution of the instantaneous (IQS) and the weighted average queue size (WAQS) for 20 additional flows)



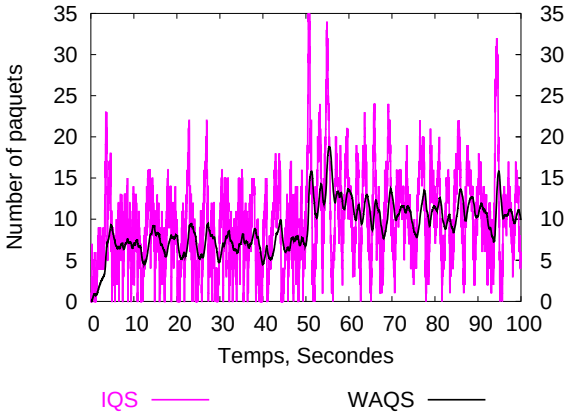
(a) RED



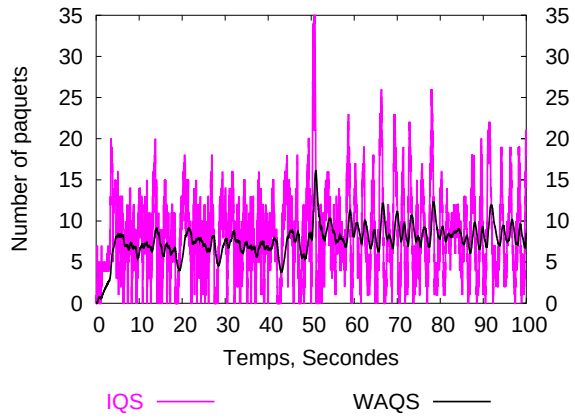
(b) ARED version Feng *et al.*



(c) ARED version Floyd *et al.*

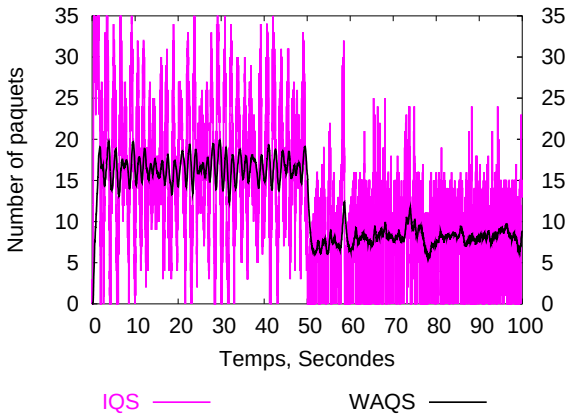


(d) PSAND (upper $Max_p = 0.5$)

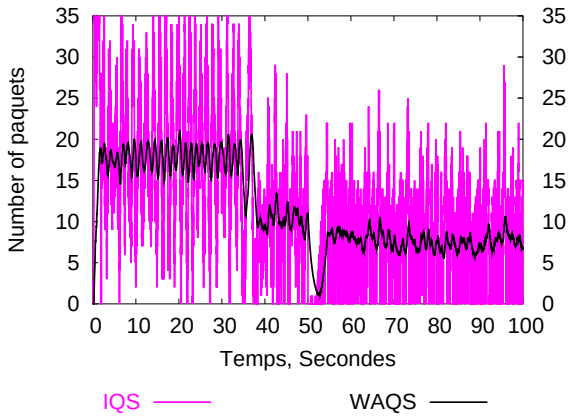


(e) PSAND (upper $Max_p = 0.75$)

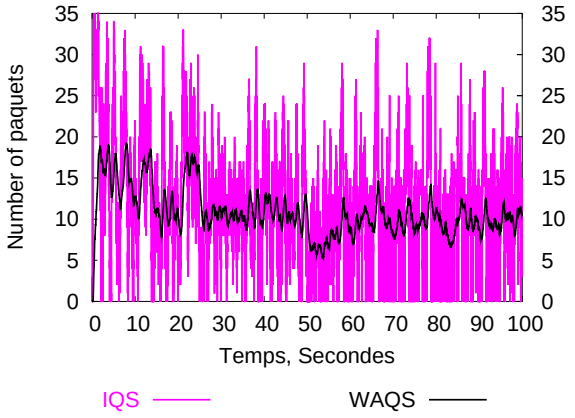
Figure 2.4: Evolution of the instantaneous (IQS) and the weighted average queue size (WAQS) for 50 additional flows



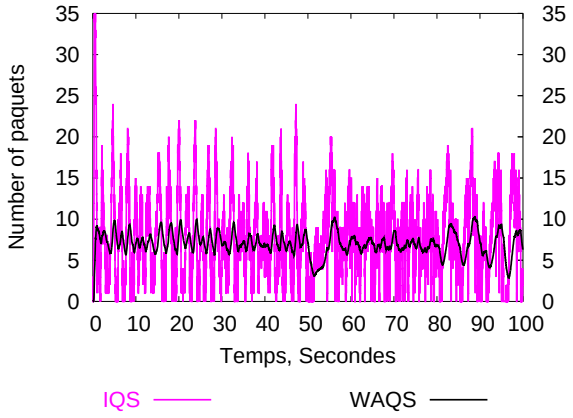
(a) RED



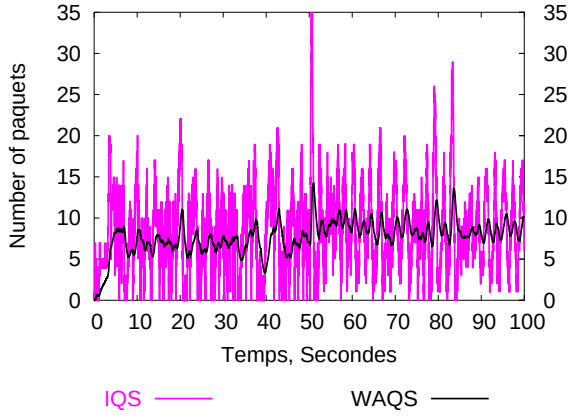
(b) ARED version Feng *et al.*



(c) ARED version Floyd *et al.*

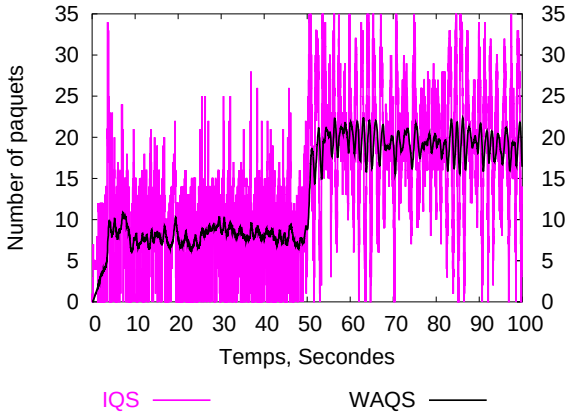


(d) PSAND (upper $Max_p = 0.5$)

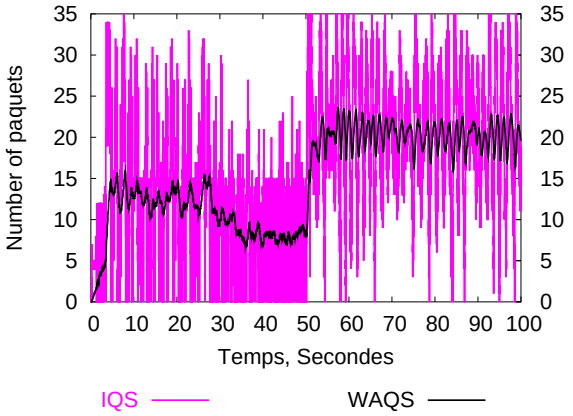


(e) PSAND (upper $Max_p = 0.75$)

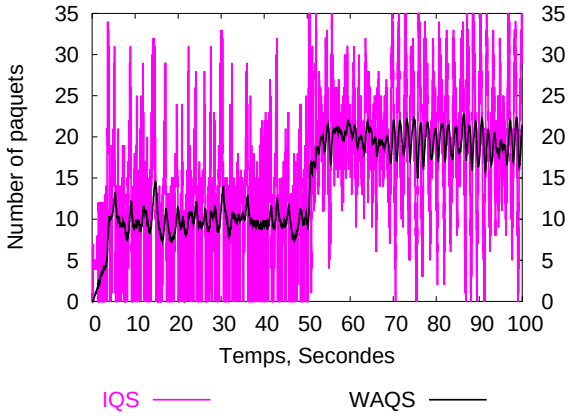
Figure 2.5: Evolution of the instantaneous (IQS) and the weighted average queue size (WAQS) for 70 additional flows



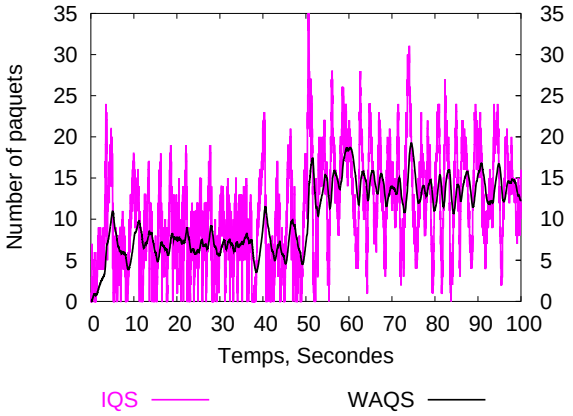
(a) RED



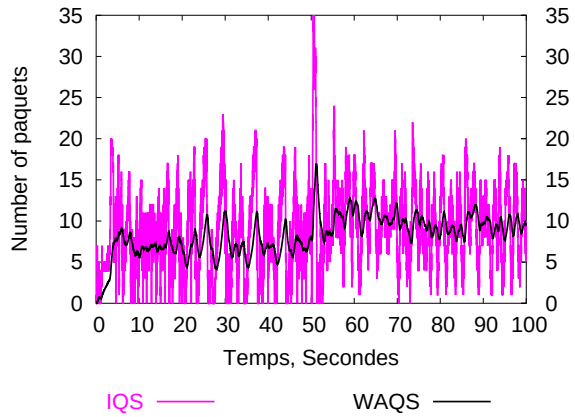
(b) ARED version Feng *et al.*



(c) ARED version Floyd *et al.*



(d) PSAND (upper $Max_p = 0.5$)



(e) PSAND (upper $Max_p = 0.75$)

Figure 2.6: Evolution of the instantaneous (IQS) and the weighted average queue size (WAQS) for 100 additional flows

back the queue size to a range which is very close to the target queue size after less than 10 seconds.

Moreover, the oscillations of the instantaneous queue size for PSAND for Figure 2.4(d) are roughly contained in a range between 5 and 15 packets, which is not the case for the other mechanisms in Figure 2.4(a) to 2.4(c).

If the amount of the traffic increase is very high, then ARED_{Feng} and ARED_{Floyd} are unable to bring the average queue size back to its target value. This is observed in Figures 2.4 to 2.6 for a number of additional flows above 20. In this case, PSAND attempts to bring the average queue size to a value that is closer to the target average queue size even if this value increases with the number of flows. For example, after a congestion increase, the weighted average queue size oscillates around 10.7 packets for 50 additional flows in Figure 2.4(d), around 12 packets for 70 additional flows in Figure 2.5(d) and around 18 packets for 100 additional flows in Figure 2.6(d).

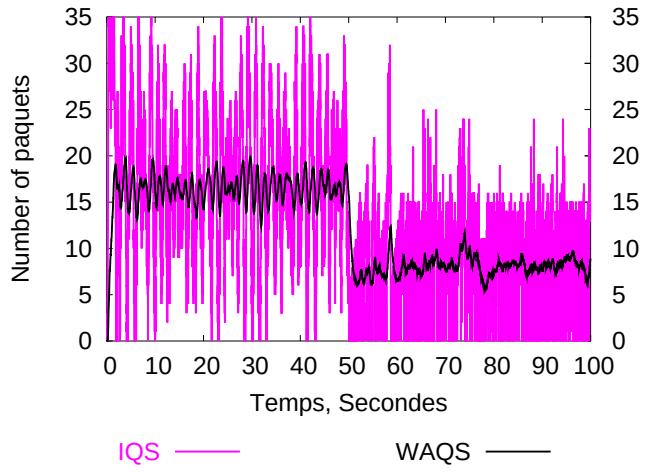
For the three ARED approaches, the more the traffic load is increased the less the instantaneous queue size goes down to 0 packets. However, whatever the number of additional flows, the instantaneous queue size for PSAND goes down less frequently to 0 packets as compared to the other ARED mechanisms. In addition, whatever the number of flows, PSAND exhibits a less frequent oscillations of the instantaneous queue size between 20 and 35 packets. After a quick adaptation to the congestion increase, PSAND never reaches the buffer capacity (35 packets) whereas for ARED_{Feng} and ARED_{Floyd} the queue size reaches frequently the maximum buffer capacity when the number of additional flows exceeds 20. As for RED, after an increase of the load, we observe a more frequent oscillations of the queue size around 35 packets whatever the number of flows.

In summary, the observations of Figures 3.6 to 2.6 show that our mechanism exhibits a robust performance by responding quickly to a rapid change in the congestion increase and shows a better adaptation to this change by bringing back the average queue size close to its target value. Moreover, it offers a more reduced amplitude of oscillations of the instantaneous queue size as compared to RED, ARED_{Feng} and ARED_{Floyd}. It also shows a less frequent occurrences of an empty queue, and also less frequent visits between 25 and 35 packets (queue size close to the maximum buffer capacity).

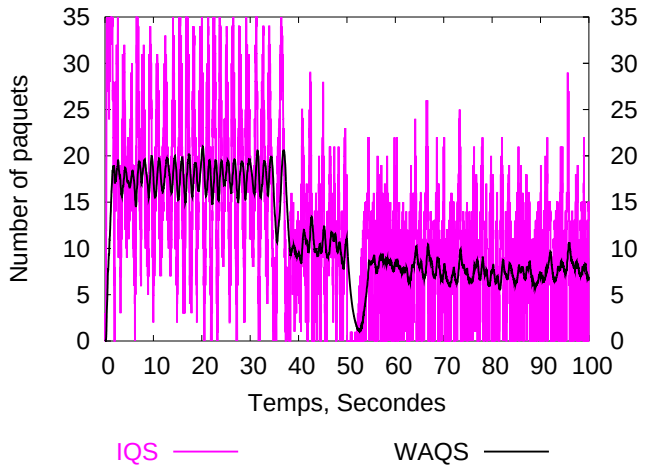
2.4.2.1.2 Traffic decrease

Let us now consider the case of a traffic decrease where 20 FTP connections in Figures 2.7 and 50 FTP connections in Figures 2.8 started at time 0 and stopped at time 50 seconds. From 50 to 100 seconds only 3 FTP connections are active.

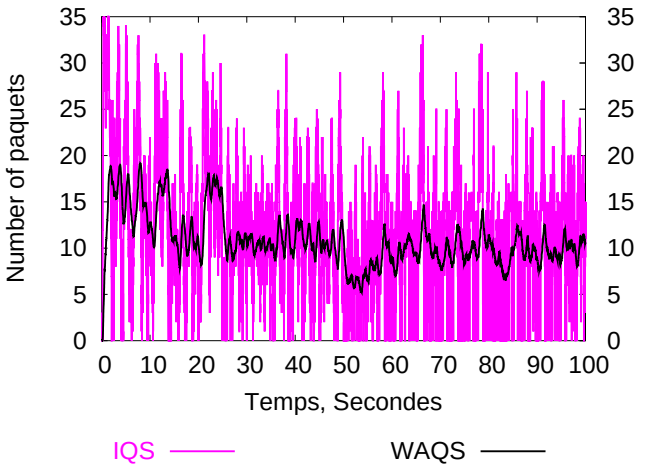
In Figure 2.7, we observe a decrease of the queue size for RED, $ARED_{Feng}$ and $ARED_{Floyd}$ when the congestion level decreases at time 50 seconds. However, RED exhibits a sharper decrease of the queue size as compared to $ARED_{Feng}$ and $ARED_{Floyd}$. In addition, during this period of a light load, RED shows a more frequently empty queue than the others queue management schemes. Even if $ARED_{Feng}$ and $ARED_{Floyd}$ show a better performance as compared to RED, they still exhibit more oscillations around 0 packets than PSAND do. Indeed, PSAND reduces highly the phenomenon of buffer un-



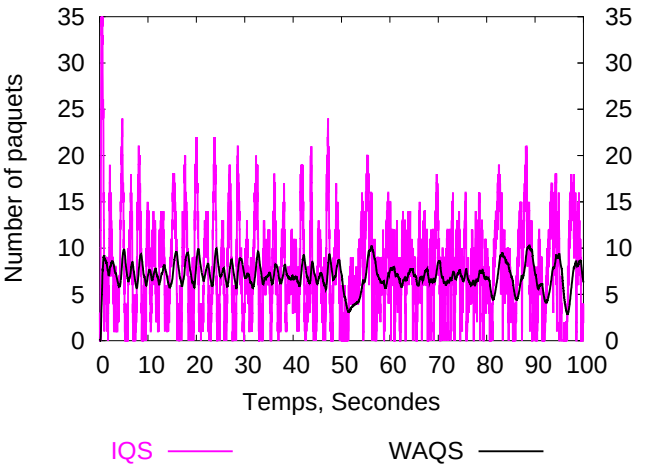
(a) RED



(b) ARED Feng

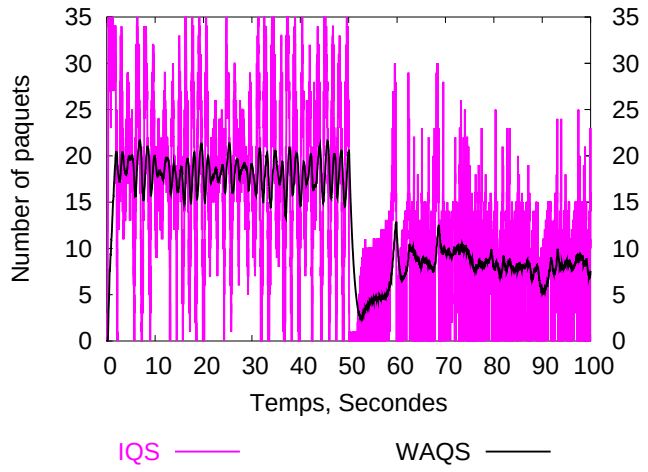


(c) ARED Floyd

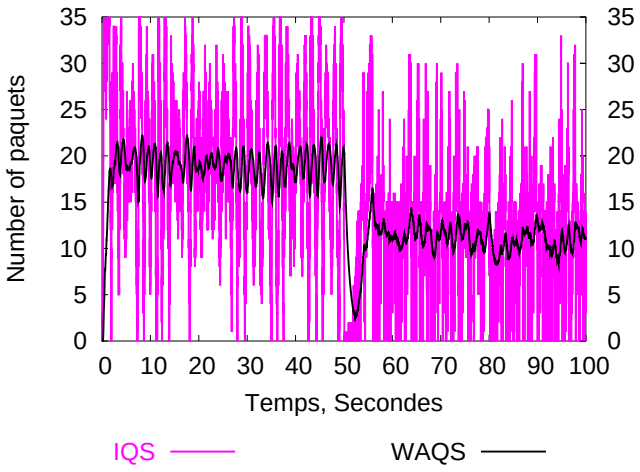


(d) PSAND

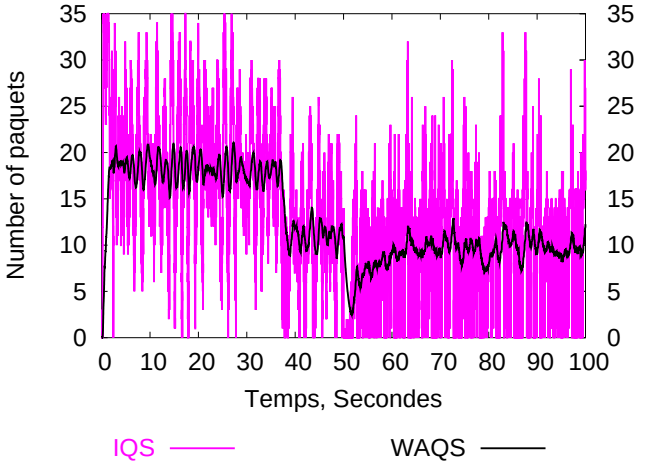
Figure 2.7: Evolution of the instantaneous (IQS) and the weighted average queue size WAQS) for 20 additional flows before the traffic decrease



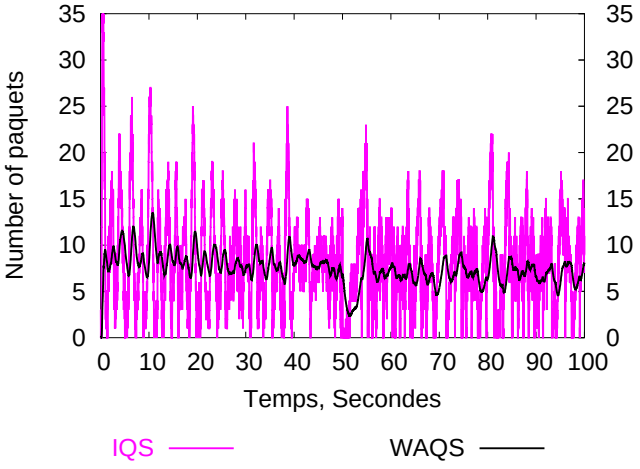
(a) RED



(b) ARED Feng



(c) ARED Floyd



(d) PSAND

Figure 2.8: Evolution of the instantaneous (IQS) and the weighted average queue size WAQS) for 20 additional flows before the traffic decrease

derflow. Unlike these schemes, the rapid change in the congestion level is less visible on the queue size evolution for PSAND. For the first 50 seconds where a higher traffic load is generated, PSAND shows an average queue size close to the target queue size. When a rapid congestion decrease occurs, PSAND responds quickly to this change of the congestion level by bringing back the average queue size to its target value. PSAND needs less than 3 seconds to stabilize the average queue size around its new value whereas the other mechanisms need about 5 seconds. The new value after the congestion decrease around which PSAND stabilizes its queue size is not too low as compared to the value around which the average queue size oscillated during the first 50 seconds of a high traffic load. Whereas for the other mechanisms we observe that the average queue size oscillates around a new value which is much lower than the one observed before the congestion decrease. Moreover, the amplitude of the queue size oscillations for PSAND is lower than the one observed for the other schemes.

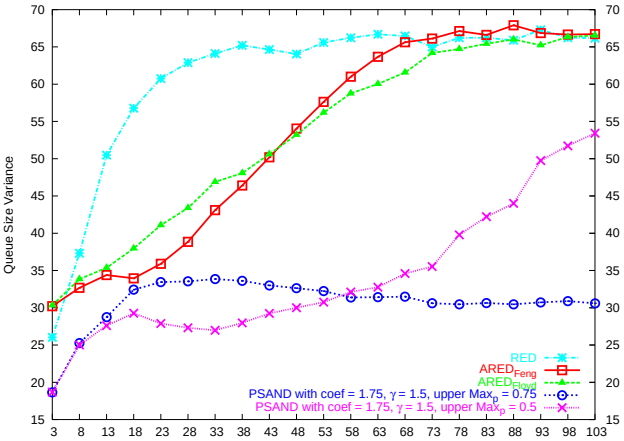
When beginning the simulation from a higher traffic load, a traffic generated by 50 FTP connections for instance as showed by Figure 2.8, RED, $ARED_{Feng}$ and $ARED_{Floyd}$ show more difficulty to reduce the amplitude of the queue size oscillations and to bring the average queue size close to its target value for the first 50 seconds where the load is high. In the same manner, after the sharp load decrease all these mechanisms show also a difficulty to avoid the queue from being frequently empty. PSAND rather shows a better adaptation of the queue size during the period of a high traffic load in the first 50 seconds as well as during the period of a low traffic load. During the two period, it bring the average queue size back to its target value and show no significant difference of the queue size oscillation when passing from a higher load to a very low traffic load..

In summary, the observations of Figures 2.7 to 2.8 show that our mechanism exhibits a robust performance since it shows a better adaptation to the traffic decrease by bringing back the average queue size close to its target value and reducing the phenomenon of buffer underflow comparing to RED, ARED$_{Feng}$ and ARED$_{Floyd}$. Moreover, unlike these schemes, the traffic decrease does not seem to affect significantly the evolution of the queue size of PSAND. Note that the observations of Figures 2.3 to 2.6 also showed that PSAND is less sensitive to the traffic increase concerning the queue size evolution as compared to the other mechanisms.

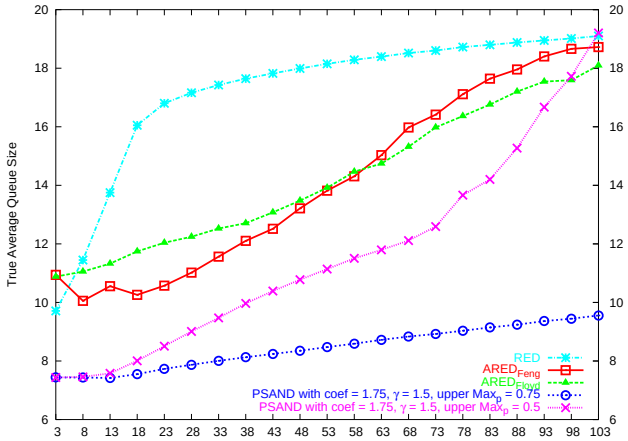
2.4.2.2 Statistical observations

To consolidate all these observations, we have compared in Figures 2.9 to 2.11 our scheme with the other mechanisms using the more rigorous statistics. As shown by Figure 2.9(a), PSAND reduces the queue size variance and thus increases the stability of the queue size independently of the number of flows as compared to RED and the other versions of ARED. Furthermore, Figure 2.9(b) illustrates also that PSAND reduces the queueing delay.

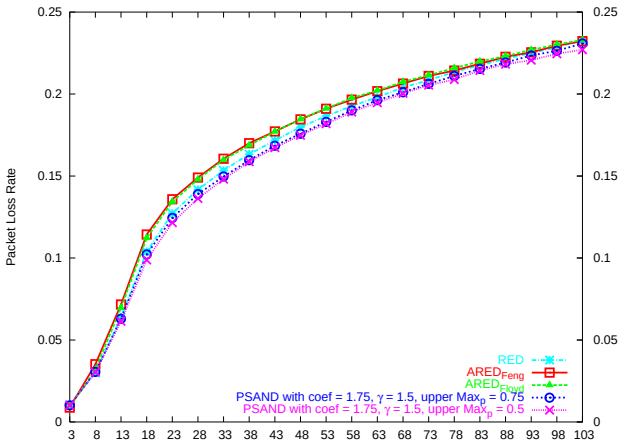
In addition to these performance gains, as the number of flows increases, we observe in Figure 2.9(c) a slight reduction of the packet loss rate. We observe for 25 flows a reduction of about 1.2% as compared to both ARED and about 0.6% as compared to RED. PSAND did not sacrifice the loss rate for the performance gain obtained in the average queue size



(a) Queue size variance for different numbers of flows



(b) True average queue size for different numbers of flows



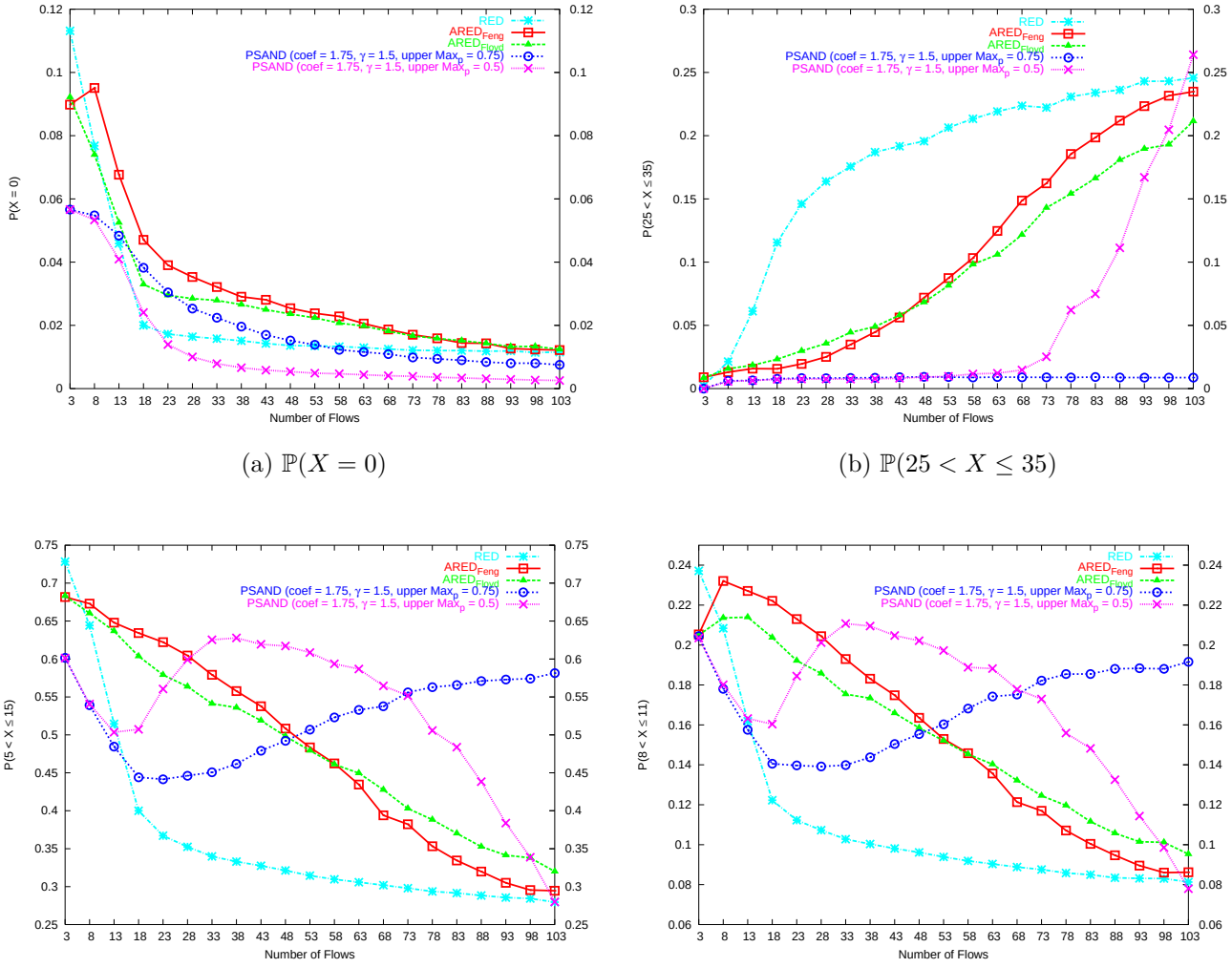
(c) Packet loss rate for different numbers of flows

Figure 2.9: Comparison of performances for different traffic load

and in the variance of the queue size. Note that the performance gain obtained by both versions of ARED as compared to RED has been acquired with a slight increase of the packet loss rate.

Moreover, as illustrated by Figure 2.10(a), for all number of flows, the probability that the queue might be empty ($P(X = 0)$) is also reduced by at least 2% as compared to the other mechanisms. This result suggests that since the link utilization can be computed as $1 - P(X = 0)$, our scheme increases the link utilization rate as compared to RED, $ARED_{Feng}$ and $ARED_{Floyd}$. Figure 2.10(b) shows also that the probability that the queue might be full ($P(25 < X \leq 35)$) is also reduced whatever the number of flows.

In Figures 2.10(c) and 2.10(d), we also observe for PSAND a higher probability of



(a) $\mathbb{P}(X = 0)$

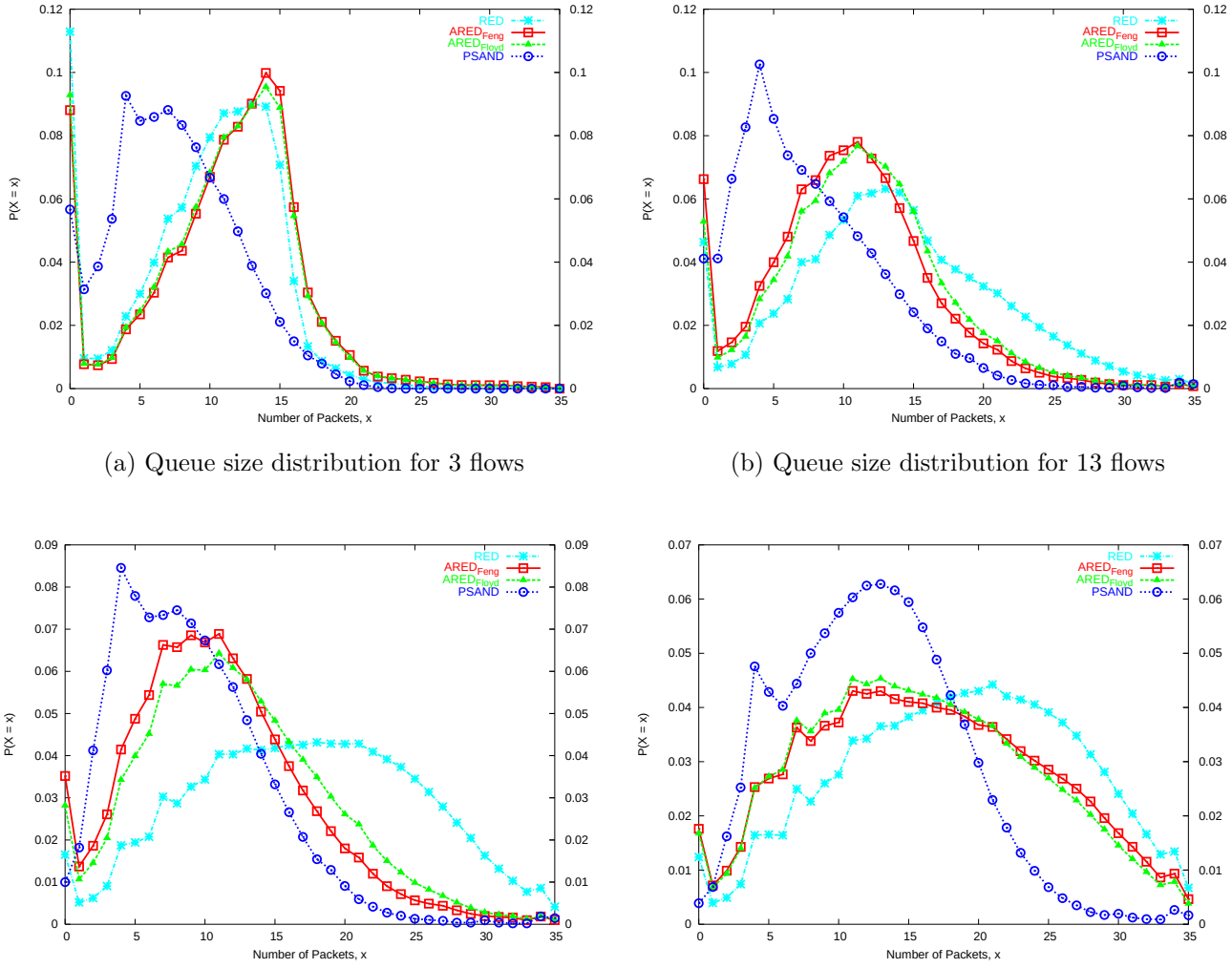
(b) $\mathbb{P}(25 < X \leq 35)$

(c) $\mathbb{P}(5 < X \leq 15)$

(d) $\mathbb{P}(8 < X \leq 11)$

Figure 2.10: Distribution of the instantaneous queue size

maintaining the queue size close to its target value (10 packets). Figure 2.10(d) presents a more restricted and closer range to the target value as Figure 2.10(c). Both figures show the same behavior even if the probability $P(5 < X \leq 15)$ is higher than $P(8 < X \leq 11)$. We observe for PSAND a lower probability than ARED$_{Feng}$ and ARED$_{Floyd}$ for a number of flows below 28 and a higher probability above 28 flows. This does not mean that concerning the predictability of the average queue size, PSAND gives a good performance for a number of flows above 28 flows and poorer performance for a number of flows below 28 as compared to ARED$_{Feng}$ and ARED$_{Floyd}$. Indeed in the case of a number of flows below 28, PSAND gives a lower probability because even if the average queue size is not close enough to its target value, it is located below this target value. Figure 2.9(b) illustrates this. Figures 2.11(a) to 2.11(c) consolidate this observation by showing that for 3, 13 and



(a) Queue size distribution for 3 flows

(b) Queue size distribution for 13 flows

(c) Queue size distribution for 28 flows

(d) Queue size distribution for 73 flows

Figure 2.11: Queue size distribution for different mechanisms.

28 flows, the highest probabilities are concentrated around the values of the queue size ranging from 5 to 10 packets. This means that for a number of flows below 28, PSAND offers an average queueing delay less than the target delay, which is more interesting than having a delay close to the target one as in the case of ARED$_{Feng}$ and ARED$_{Floyd}$ as long as this is not payed in return by an increase of the packet loss rate. But as PSAND showed us in Figure 2.9(c) a slight decrease of the packets loss rate for PSAND for all number of flows, this observed lower and less closer average queue size as compared to its target value for a number of flows below 28, appears to be more advantageous than the one observed forARED$_{Feng}$ and ARED$_{Floyd}$.

As Figures 2.10(c) and 2.10(d) showed, for a number of flows above 28, PSAND offers an average queue size closer to its target value as compared to ARED$_{Feng}$ and ARED$_{Floyd}$.

Figure 2.11(d) confirms this result by showing that for PSAND the highest probabilities are concentrated around values ranging from 5 to 15 packets whereas for ARED$_{Feng}$ and ARED$_{Floyd}$ they are concentrated around values ranging from 10 to 20 packets. For a higher number of flows, ARED$_{Feng}$ and ARED$_{Floyd}$ give an average queue size that goes far above its target value whereas for PSAND it remains in a reasonable range which is closer to the target value (see Figure 2.9(b)).

Finally, the qualitative and statistical analysis of all the results show that our approach offers a desirable overall improvement of ARED performances and achieves our objective to reduce the queue size variance while keeping the average queue size close to its target value without sacrificing the packet loss rate.

2.4.3 Parameters $coef$ and γ for different number of flows

In this section, we investigate the influence of the parameters $coef$ and γ on the performance of our adaptive RED algorithm. We showed that our scheme improves overall performances such as the variance and the average of the instantaneous queue size and also the packet loss rate. Figures 2.12 and 2.13 depict the measurement of these metrics for different values of $coef$ and γ.

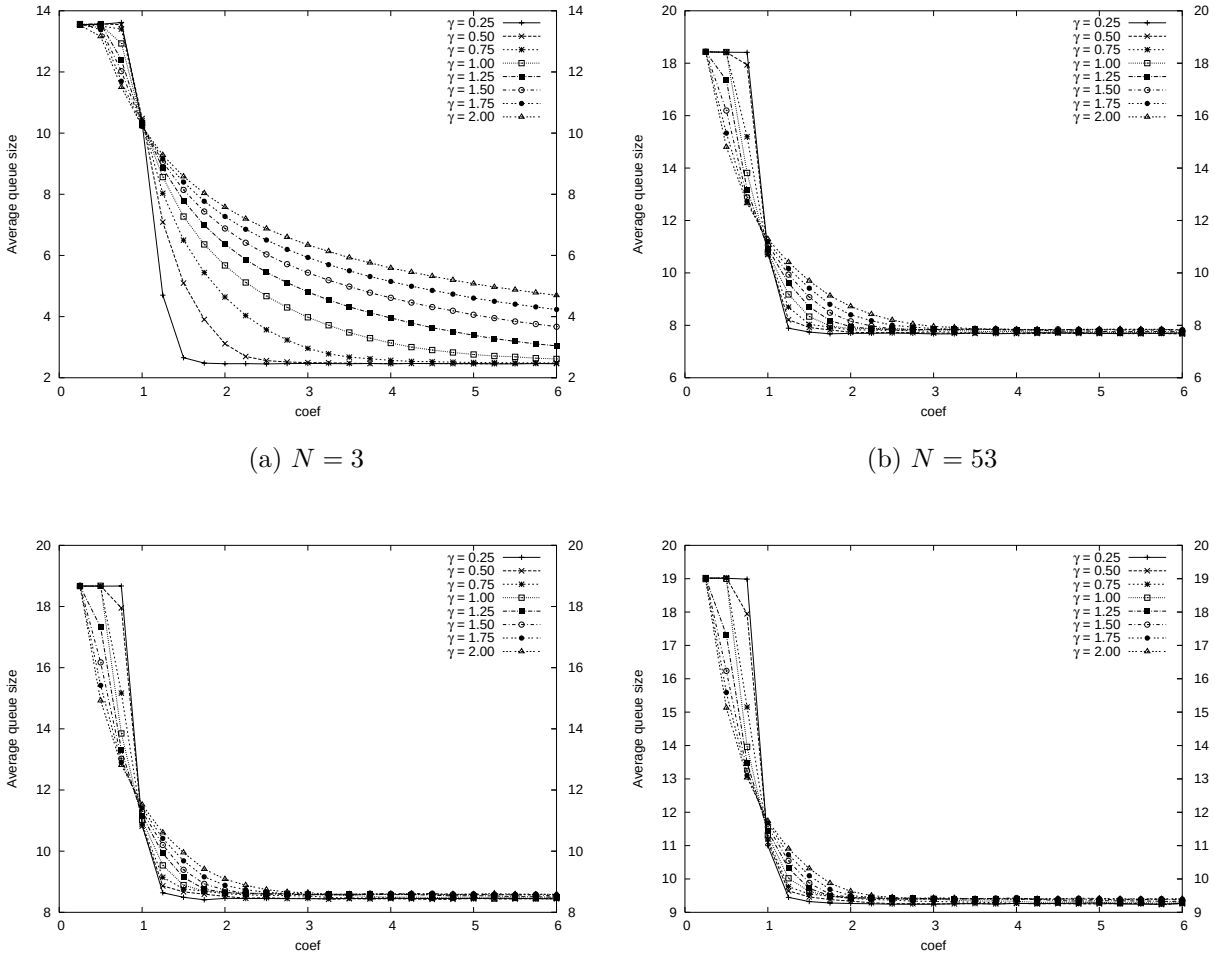
One point on the curves presented in this figure corresponds to an average of 100 independent simulations. Every simulation is run for 100 seconds. We considered different traffic loads generated by 3, 53, 73 and 103 FTP flows. The bound on Max_p is set to 0.75.

The results presented in Figures 2.12 and 2.13 showed that for any number of flows, the average queue size and the queue size variance decrease when the value of $coef$ increases. On the other hand, the increase of the value of γ increases the average queue size as well as the queue size variance.

However, Figure 2.14 shows that the packet loss rate increases as $coef$ increases but decreases as γ increases. For $coef > 1$, whatever the value of γ and whatever the number of flow, we observe an improvement of the average queue size and the queue size variance as compared to the observed performance of $ARED_{Feng}$ and $ARED_{Floyd}$ in Figure 2.9 (page 87). More performance improvements are observed when γ decreases. That is, as $coef$ increases, the average queue size as well as the queue size variance decrease and converge to a certain value. The convergence is reached more quickly for smaller values of γ.

Even though the decrease of γ can improve the average queue size and the queue size variance, it increases the packet loss rate. This increase is more important for a small number of flows. In addition, for small number of flows, large values of $coef$ increase the packet loss rate as compared to the performance of the original ARED (e.g. 1.8% instead of 0.9% for 53 flows). This increase in the loss rate is translated into an excessive decrease of the average queue size (Figure 2.12).

As this rate is less than 2%, it can be affordable since it can be recovered by TCP retransmissions. Moreover, for small values of $coef$, the packet loss rate is reduced but the average queue size increases. For a large number of flows (53 for instance) and for $coef > 1$ we observe an improvement of the average queue size as well as the loss rate.



(a) $N = 3$ (b) $N = 53$

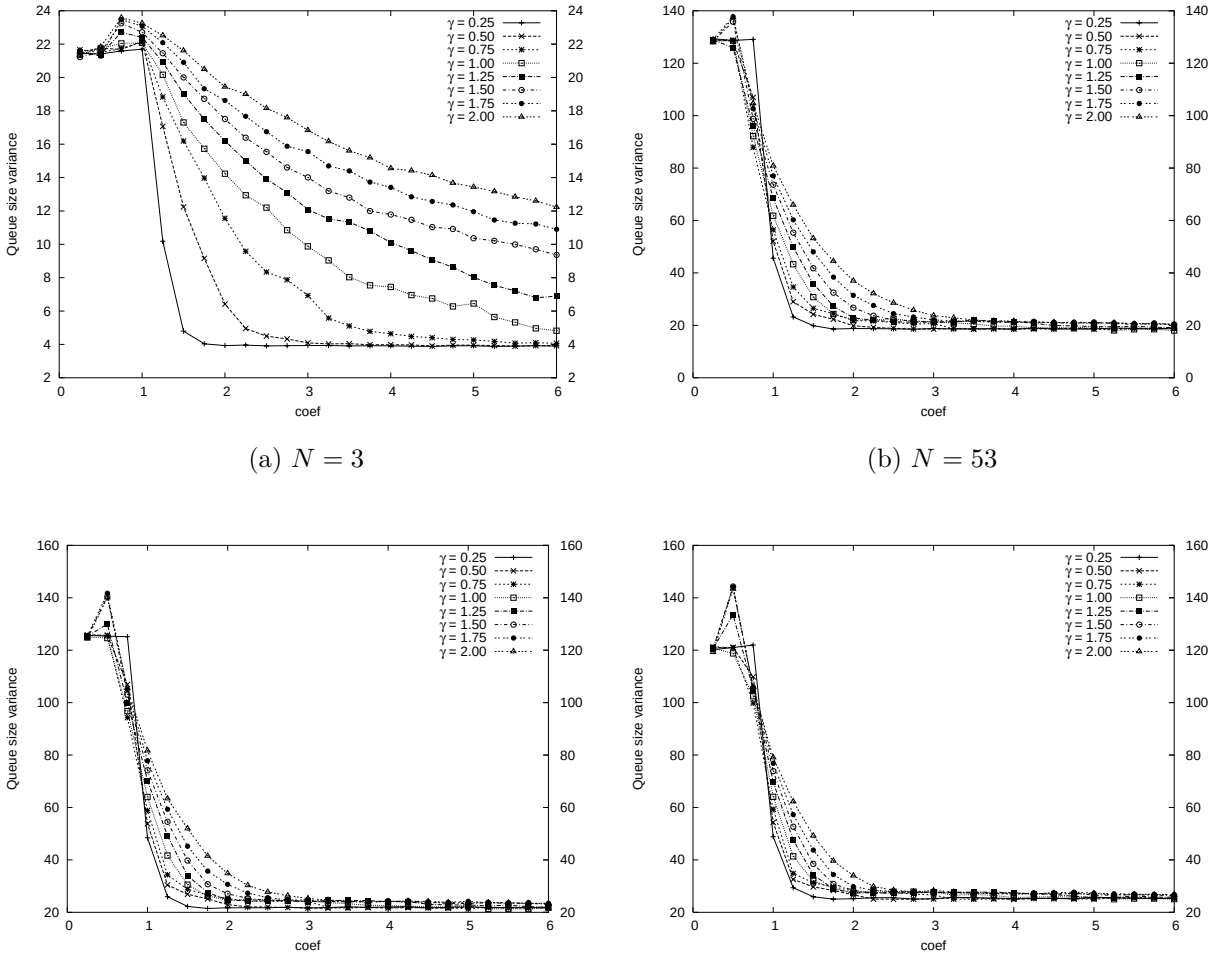
(c) $N = 73$ (d) $N = 103$

Figure 2.12: Average queue size for different values of $coef$ and γ

In conclusion, for few flows (3 flows), a value of $coef$ close to 1 and $\gamma > 1$ gives overall good performances by improving the variance and the average of the instantaneous queue size without a substantial increase of the loss rate. For large number of flows, whatever $coef > 1$, we observe overall good performances and the loss rate converges to a value less than the one observed for $ARED_{Feng}$ and $ARED_{Floyd}$ (Figure 2.9).

2.4.4 Setting of $coef$ and γ

The results of the experiments of the previous section suggest to choose a range of values of $coef$ and γ which is appropriate to small and large number of flows. Hence, based on extensive simulations for different number of flows and different upper bound of Max_p (we also conducted the same experiments as in the previous section with an upper bound of



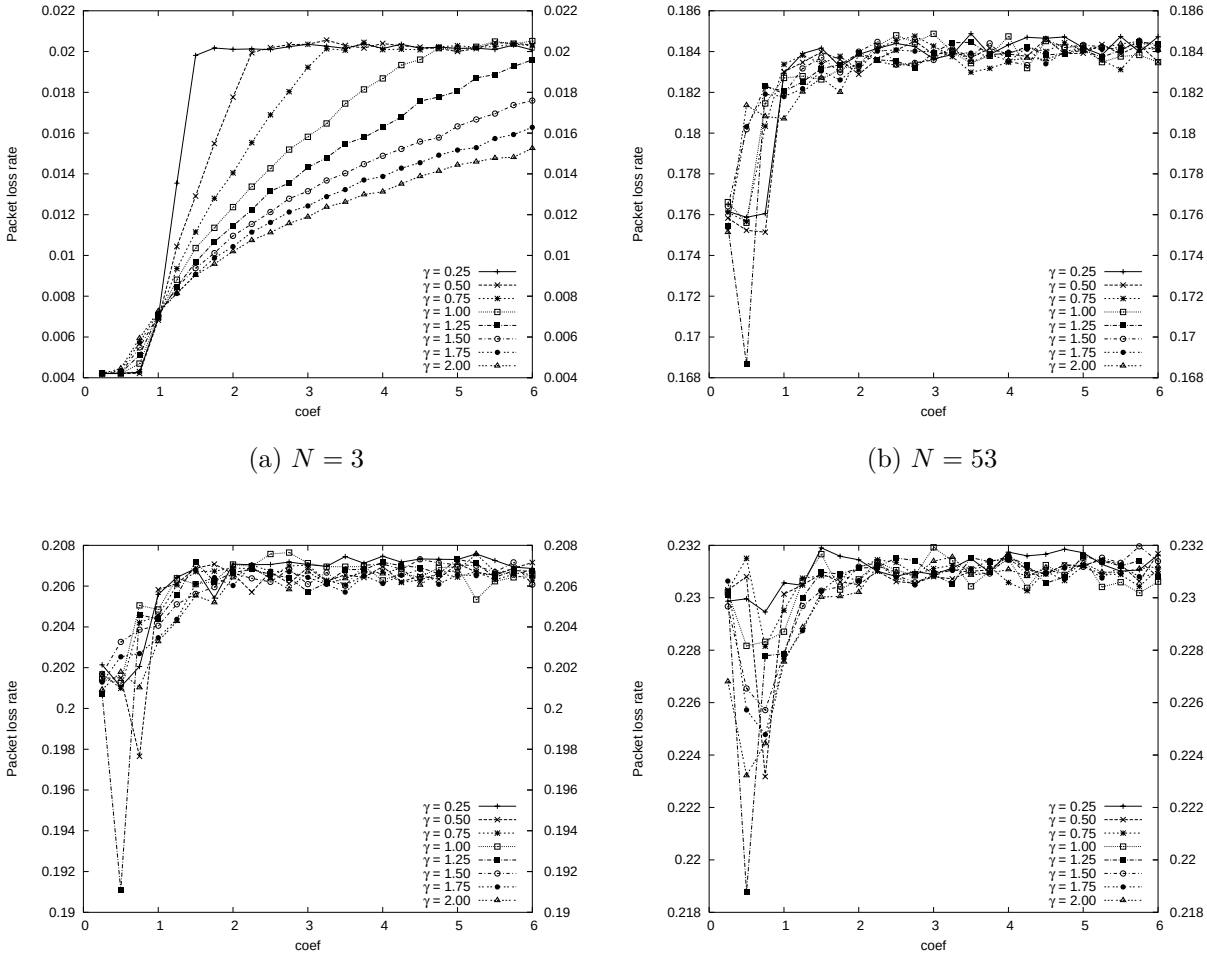
(a) $N = 3$ (b) $N = 53$

(c) $N = 73$ (d) $N = 103$

Figure 2.13: Queue size variance for different values of $coef$ and γ

Max_p equal to 0.5), we set $coef = 1.75$ and $\gamma = 1.5$ as this setting brings about reasonable overall improvement on ARED for small and large number of flows. Tables 2.2 and 2.3 show this improvement if we compare the performance of PSAND2 ($coef = 1.75, \gamma = 1.5$) with PSAND1 ($coef = 1, \gamma = 1$), ARED$_{Feng}$ and ARED$_{Floyd}$.

PSAND1 reduces slightly the loss rate by 0.2% for 3 flows and by almost 2% for 28 flows as compared to RED and the other ARED mechanisms. It also achieves an average queue size close to its target($\hat{K}_T = 10$ packets) and reduces the probability that the queue might be empty : in the case of 3 flows, 2.5% instead of 8.8% for $ARED_{Fend}$, and 9.1% for $ARED_{Floyd}$; in the case of 28 flows, 1.8% instead of 3.7% for $ARED_{Feng}$ and 2.9% for $ARED_{Floyd}$. Moreover, the queue is never almost fully occupied for 3 flows. However, for 28 flows, the probability that the queue might be almost fully occupied is greater than the one observed for the two versions of ARED (4% instead of 2.1% for $ARED_{Feng}$ and



(a) $N = 3$ (b) $N = 53$

(c) $N = 73$ (d) $N = 103$

Figure 2.14: Packet loss rate for different values of $coef$ and γ

3.2% for $ARED_{Floyd}$). In addition, even if Table 2.2 shows for PSAND1 a decrease of the variance for 3 flows, an increase of variance is observed for 28 flows as compared to the others versions of adaptive RED (60.00 as compared to 38.85 for $ARED_{Feng}$ and 43.38 for $ARED_{Floyd}$). The choice of $coef = 1.75$ and $\gamma = 1.5$ for PSAND2 improves overall performances as compared to PSAND1, $ARED_{Feng}$ and $ARED_{Floyd}$ by reducing both the variance and the probability that the queue is full for higher number of flows. This observation leads us to conclude that the adaptation of Max_p with r using (2.1) is not aggressive enough for large number of flows and shows the use to enrich the behavior of r by adapting Max_p using (2.2). The resulting performance improvement obtained can be more or less significant depending on the selected values of $coef$ and γ in relation to the traffic load. This is illustrated by the comparison of PSAND2 with PSAND3. By using a configuration of $coef$ and γ appropriate to the traffic load ($coef = 1$, $\gamma = 0.25$ for 3 flows,

Table 2.2: Comparison of Average Queue Size (AQS), Variance of Queue Size (VQS) and Packet Loss Rate (PLR) for different values of $coef$, γ and N (number of flows)

	AQS		VQS		PLR	
	$N = 3$	$N = 28$	$N = 3$	$N = 28$	$N = 3$	$N = 28$
ARED_{Feng}	10.94	11.01	30.21	38.85	0.87	14.90
ARED_{Floyd}	10.88	12.24	30.33	43.38	0.92	14.79
PSAND1 ($coef = 1$, $\gamma = 1$)	10.34	10.89	22.31	60.00	0.69	13.57
PSAND2 ($coef = 1.75$, $\gamma = 1.5$)	7.43	7.87	18.65	33.54	1.01	13.90
PSAND3 $coef = 1$, $\gamma = 0.25$ (3 flows) $coef = 2$, $\gamma = 0.25$ (25 flows)	10.6	6.20	21.92	13.74	0.66	13.82

Table 2.3: Comparison of the queue size distribution for different $coef$,γ and N (number of flows)

	$P(X = 0)$		$P(5 < X \leq 15)$		$P(25 < X \leq 35)$	
($\times 10^{-2}$)	$N = 3$	$N = 25$	$N = 3$	$N = 25$	$N = 3$	$N = 25$
PSAND1 ($coef = 1$, $\gamma = 1$)	2.55	1.88	68.86	41.41	0	4.00
PSAND2 ($coef = 1.75$, $\gamma = 1.5$)	5.93	1.47	60.05	57.60	0	0.77
PSAND3 $coef = 1$, $\gamma = 0.25$ (3 flows) $coef = 2$, $\gamma = 0.25$ (25 flows)	2.74	1.91	71.06	62.96	0	0.24

and $coef = 2$, $\gamma = 0.25$ for 28 flows), PSAND3 shows a further performance improvement as compared to PSAND2. It decreases the loss rate for small number of flows (3 flows for instance). Indeed, as showed by Table 2.2, when reducing the variance and the average queue size for 28 flows, PSAND2 had increased the packet loss rate for 3 flows as compared to PSAND1.

This shows that by using an automatic configuration of $coef$ and γ according to the traffic load, the performance of our adaptive scheme can be further improved.

2.4.5 Upper bound of the parameter Max_p

The original proposition of RED has suggested to set the parameter Max_p to 0.1. But afterwards several works have shown the significance of the parameter Max_p because of its important influence on RED performance. As stated in [49], an "optimal" Max_p should depend on network parameters like the RTT, the bandwidth and the number of active connections and is bounded as:

$$Max_p \leq \frac{N \times SS \times c}{C \times RTT},$$

where N is the number of connections, C is the total bandwidth, SS is the segment size, RTT is the round trip time and c is a constant. This shows that the optimal value of Max_p and its upper-bound depend on network characteristics. The adaptive approach of [34], an approach that we have followed, is to address this problem by avoiding the use of network parameters. However, the range of values in which Max_p should evolve, more specifically, the upper bound of this parameter can also have an impact on the performance of adaptive RED.

In this section, we conducted several experiments illustrated in Figures 2.15 and 2.16, in order to investigate the influence of the upper bound of Max_p on the performance of PSAND.

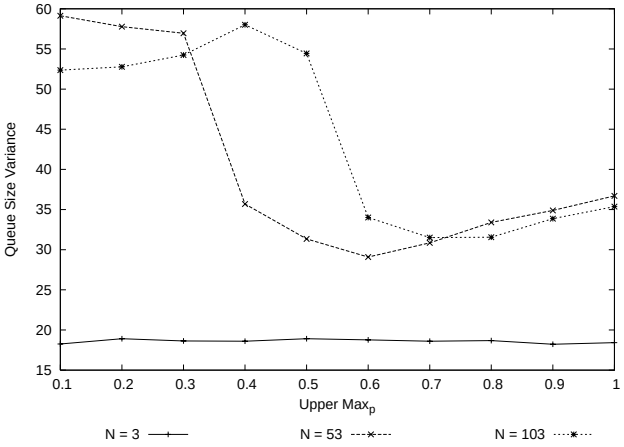
For these series of experiments, we varied the value of the upper bound of Max_p and considered different number of flows so as to investigate the impact of the upper bound of Max_p on different traffic load: low traffic load with 3 flows and high traffic load with 103 flows. For the experiments in Figure 2.15 we set $coef = 1.75$ and $\gamma = 1.5$.

The results observed in Figure 2.15(b) showed that the average queue size decreases with the upper bound of Max_p. This decrease is more spectacular for higher traffic loads. For instance, for 103 flows, the average queue size decreases from 25 to 8 packets when the upper bound of Max_p increases from 0.1 to 1. This shows that bounding the evolution of Max_p to a small value does not allow the PSAND scheme to increase aggressively its drop function as required in order to respond more efficiently to the congestion level. Hence, small upper bounds of Max_p increase excessively the queueing delay specially for large traffic load. Whereas larger upper bounds give small queueing delays for high traffic load.

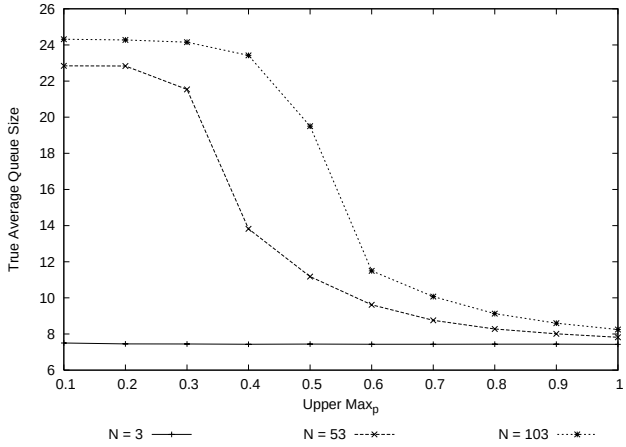
On the other hand, small traffic loads do not seem to be significantly influenced by the upper bound of Max_p. Small loads give small queueing delay, small queue size variance as well as small packet loss rate whatever the value of the upper bound of Max_p. This is because since the traffic is not sustained, the average queue size is always small giving a small drop probability that does not change much when the upper bound of Max_p (*i.e.* drop function) increases.

Figure 2.15(a) shows the effect of the upper bound of Max_p on the queue size variance. We observe the same results as above, that is for small traffic load, no significant influence is observed. But for a higher traffic load, the queue size variance decreases when the upper bound of Max_p increases. But after a certain value of the upper bound of Max_p (0.7 for instance), the queue size variance increases slowly. Since small upper bounds of Max_p are not aggressive enough to avoid a quick build up of the queue, the average queue size can often exceed the maximum threshold Max_{th}.

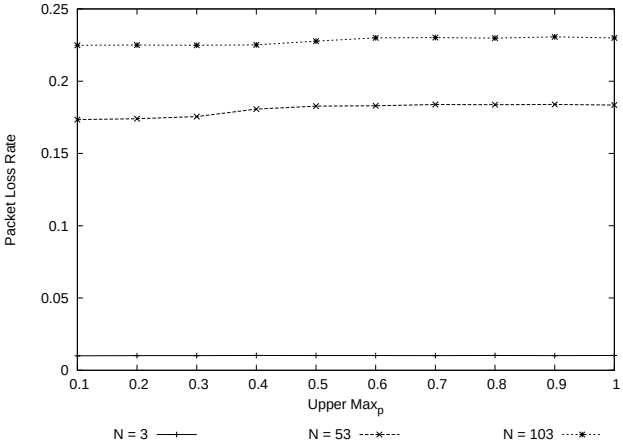
By increasing the upper bound of Max_p, the proportion of random drops of packets increases as compared to 100% drops. Consequently, since there is no rapid build up and a rapid draining of the queue, the queue size shows less oscillations. However, if the upper bound of Max_p is increased to larger value, more than 0.8 for instance (see Figure 2.15(a)), the proportion of randomly drop increases reducing the queue size. These changes in the queue size are at the origin of these light increase in the queue size variance. Concerning the packet loss rate, we can observe a very slight increase for high traffic loads when the upper bound of Max_p increases.



(a) Queue size variance



(b) True average queue size

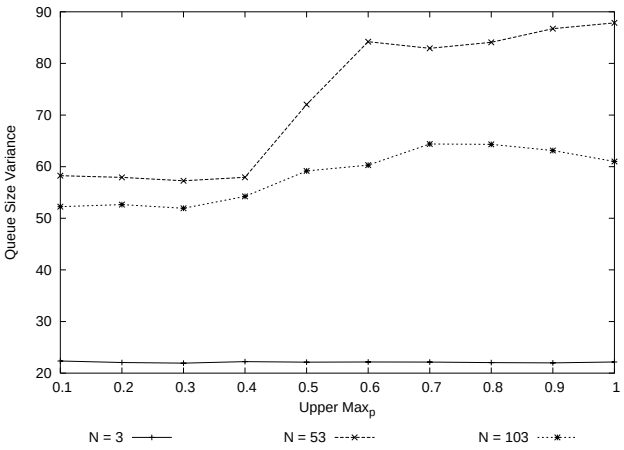


(c) Packet loss rate

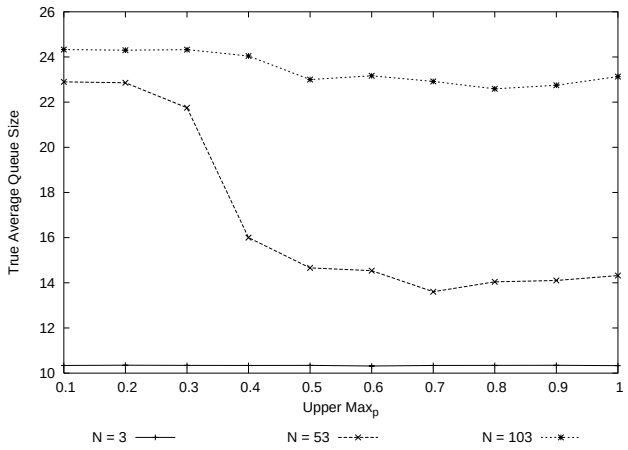
Figure 2.15: Comparison of performances for different upper bound of Max_p for $coef = 1.75$ and $\gamma = 1.5$

For the experiments presented in Figure 2.16, we set $coef = 1$ and $\gamma = 1$. Note that in previous sections, we have showed the use of $coef$ and γ in order to make the PSAND scheme more aggressive in presence of high traffic load and thus improve its performances. For this purpose, we have set $coef = 1.75$ and $\gamma = 1.5$. The reason why we set $coef = 1$ and $\gamma = 1$ in these experiments is to see if without the use of $coef$ and γ, the aggressiveness introduced by a higher upper bound of Max_p suffices to improve the PSAND scheme. The results of Figures 2.16(a) and 2.16(b) show that the upper bound of Max_p does not decrease significantly the queueing delay and increases the queue size variance as compared to the results of Figure 2.15. In addition, even decreased, the queue size remains large.

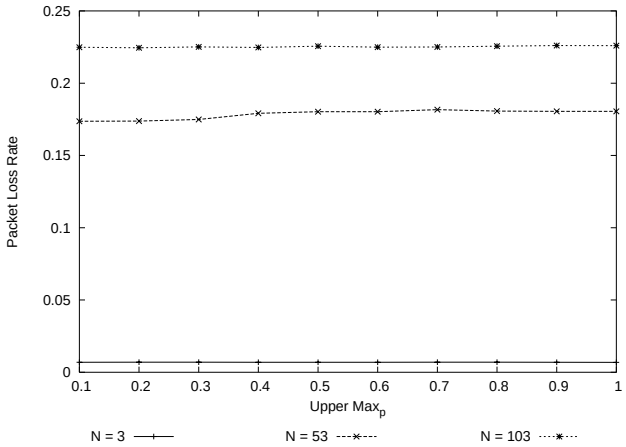
All these results show that without a proper choice of $coef$ and γ, the upper bound of



(a) Queue size variance



(b) True average queue size



(c) Packet loss rate

Figure 2.16: Comparison of performances for different upper bound of Max_p for $coef = 1$ and $\gamma = 1$

Max_p can not give a sufficiently aggressive behavior of the drop function. By using the appropriate configuration of $coef$ and γ, larger values for the upper bound of Max_p can give good overall performances.

2.5 Conclusion

This chapter has described a new mechanism for RED parameters setting in order to enhance the effectiveness of RED. We have also described this enhancement in [3, 1]. Our work is based on the adaptive approach of RED described in [34, 41] since this approach

does not require any hypothesis on the type of traffic and therefore reduces the dependency of RED on traffic scenario input parameters. We do not claim to present an optimal setting of RED parameters since as it has been shown by several works like [31, 34, 74, 76], that the optimal value of the parameter Max_p depends on the network and traffic conditions. We rather show among other possible alternatives, a way to improve the performances of Adaptive RED.

We described in Section 2.3 a new mechanism that, unlike in [34, 41] where a constant factor is used, adapts Max_p with a dynamic change rate which is a function of the change in the average queue size and the distance of the average queue size to the specified target queue size.

In addition we conclude that since the dropping rate follows closely the traffic dynamics then the random drop should start as early as possible. Indeed, we conclude from simulations that Min_{th} should be set as small as possible, whereas Max_{th} should be set as large as feasible. This will be explained in detail in Chapter 3.

In Section 2.4, we tested our parameters adjustment mechanism by extensive simulations and statistics which showed performance improvements as compared to RED and Adaptive RED. Indeed, the results showed a reduction of the variance of the instantaneous queue size as well as the average queue size independently of the number of flows, without increasing and even in some cases (for large number of flows for instance) while decreasing the loss rate. Moreover, our mechanism keeps the queue size away from buffer overflow and buffer underflow.

Our results also showed that $PSAND$ can further be improved by selecting appropriate values for the parameters $coef$ and γ. In particular, selecting dynamically a value for $coef$ and γ that gives best performances for different traffic load is an important issue. It is also interesting to analyze the performance of our scheme in presence of an UDP traffic, a mixed UDP and also web-like traffic.

Chapter 3

Configuration of the parameters Min_{th} and Max_{th}

Contents

3.1 Introduction

In this chapter, we study the influence of the parameters Min_{th} and Max_{th} on our adaptive scheme. The aim of this study is to investigate the necessity of adapting the two parameters in order to obtain performance improvements and to make thereby a proposal for their configuration. For this purpose, we use two types of experiments. For the first type of experiments, we fix Min_{th} and vary the value of Max_{th}. In the same manner, we fix Max_{th} and vary the value of Min_{th}. This is done for different traffic loads. For the second type of experiments, we propose among others possible methods, different adaptive mechanisms of Min_{th} and Max_{th} and compare the results with the one obtained from a configuration using a fixed value of Min_{th} and Max_{th} that showed the best performances for the first group of experiments. Finally, we examine the influence of Min_{th} and Max_{th} on $ARED_{Feng}$

and $ARED_{Floyd}$ schemes and compare the performance of our configuration with the best performance offered by the configuration proposed by $ARED_{Feng}$ and $ARED_{Floyd}$.

The $ARED_{Floyd}$ adaptive scheme configured Max_{th} to three times Min_{th} in automatic mode as recommended in [40]. $ARED_{Floyd}$ set also Min_{th} in automatic mode as a function of the link bandwidth. They took five packets as a lower bound which they claim works well for low-speed links. For high-speed links, they chose the value of :

$$\frac{delay_{target} \times C}{2} ,$$

as a trade-off between throughput and delay. Hence they set Min_{th} as :

$$Min_{th} \; = \; \max\left(5, \frac{delay_{target} \times C}{2}\right) .$$

As for $ARED_{Feng}$, no proposals are made for the configuration of Min_{th} and Max_{th}.

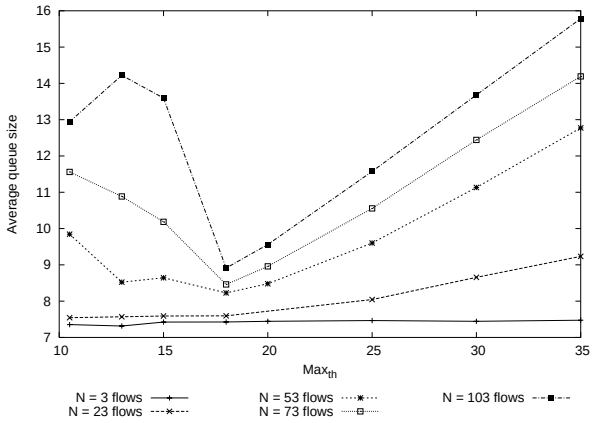
3.2 Fixed values of Min_{th} and Max_{th}

In this section, we investigate the necessity of adapting the parameters Max_{th} and Min_{th} according to the network load in order to improve the PSAND mechanism. For this purpose, we varied the values of Min_{th} and Max_{th} by considering different number of flows. We considered different traffic loads generated by 3, 23, 53, 73 and 103 TCP flows. The principle we wish to apply is: if the value of Max_{th} that gives best performances differs according to the number of flows then Max_{th} should be tuned according to the network load. In the same manner, if the best performance obtained for different number of flows are observed for different values of Min_{th}, Min_{th} should be tuned according to the network load. We wish also to test the following ideas: is it best to keep the interval ($Minth$,$Maxth$) symmetrical with respect to $\hat{K}_T$? Is it better to select $Minth$ different from 0, as in $ARED_{Floyd}$?

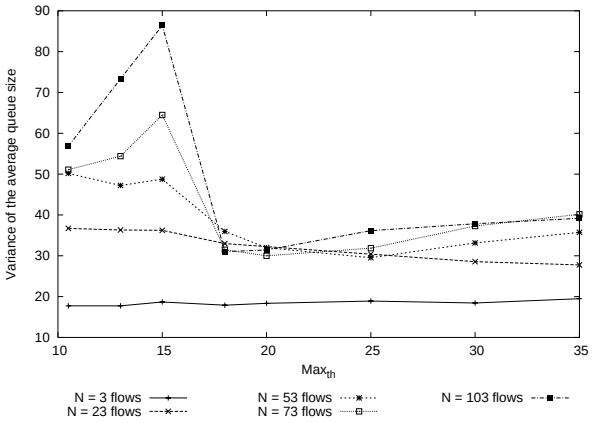
In order to test these conditions, we conduct the following experiments reported in Figures 3.1 to 3.4. We took the same experimental environment as in the previous chapter. We fixed $coef$ to 1.75 and γ to 1.5. The upper bound of Max_p is also set to 0.75. For Figures 3.1 and 3.2, we fixed the value of Min_{th} and varied the value of Max_{th}. For Figures 3.3 and 3.4, we fixed the value of Max_{th} and varied the value of Min_{th}. Since the results of Figures 3.1 to 3.4 show that the performance concerning the packet loss rate is similar for different values of Max_{th}, we only discuss afterwards about performances such as the queue size variance, the average queue size and the loss run length.

3.2.1 The variance and the average of the queue size

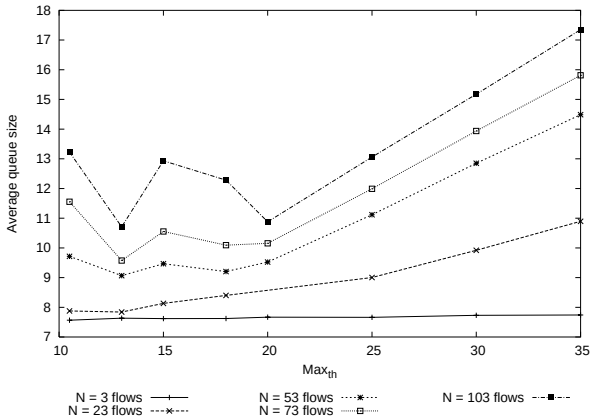
We first focus on the variation of the parameter Max_{th} (*i.e.* Figures 3.1 and 3.2). The results show on the first hand that for small number of flows (3 and 23 for instance), the influence of Max_{th} on the variance and the average of the queue size is not visible. This is



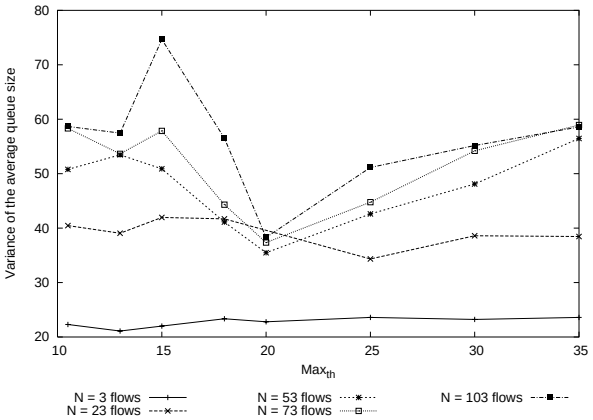
(a) Average queue size ($Min_{th} = 0$)



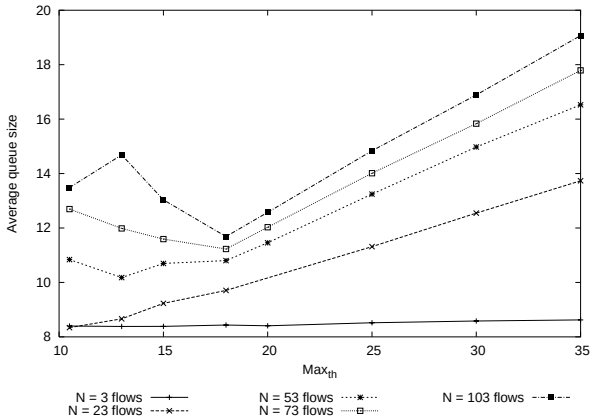
(b) Queue size variance ($Min_{th} = 0$)



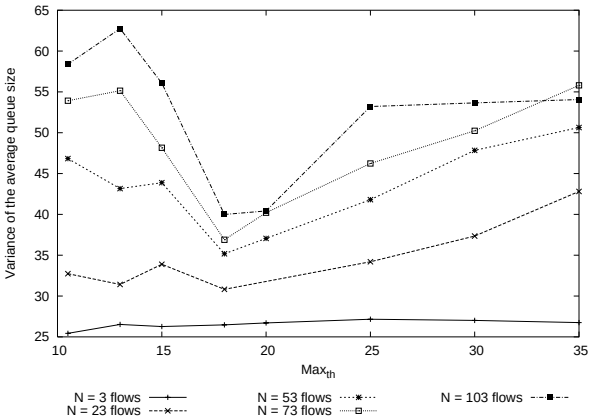
(c) Average queue size ($Min_{th} = 2$)



(d) Queue size variance ($Min_{th} = 2$)

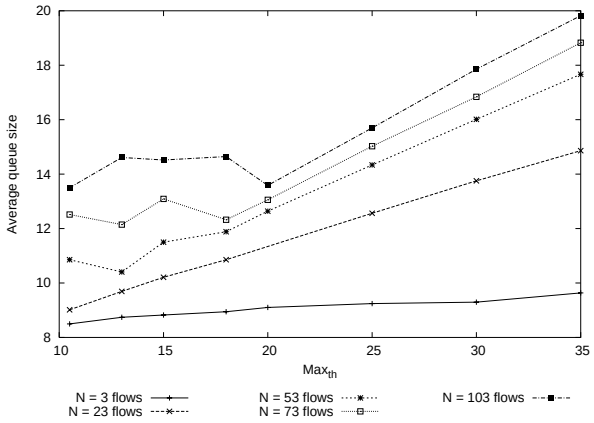


(e) Average queue size ($Min_{th} = 5$)

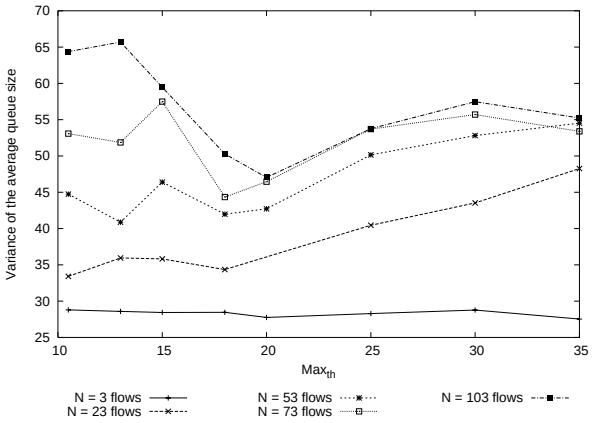


(f) queue size variance ($Min_{th} = 5$)

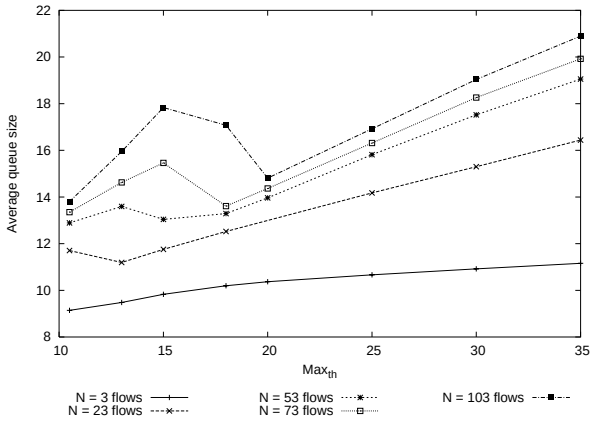
Figure 3.1: Influence of Max_{th} for different values of $Min_{th} \in \{0, 2, 5\}$



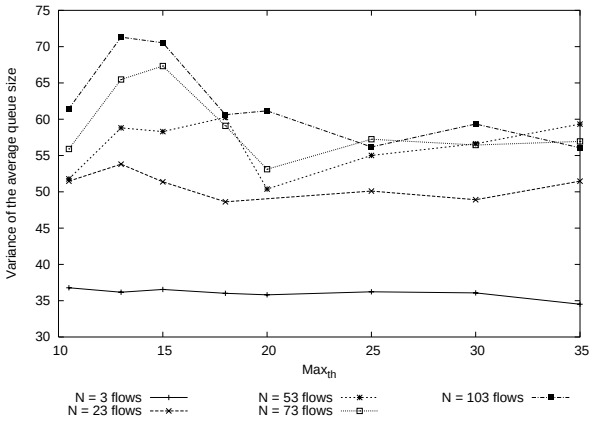
(a) Average queue size ($Min_{th} = 7$)



(b) Queue size variance ($Min_{th} = 7$)



(c) Average queue size ($Min_{th} = 9.5$)



(d) Queue size variance ($Min_{th} = 9.5$)

Figure 3.2: Influence of Max_{th} for different values of $Min_{th} \in \{7, 9.5\}$

due to the fact that in presence of light traffic load the average queue size exceeds rarely Max_{th}. We rather observe an average queue size less than the target value. Nevertheless, if the value of Min_{th} is increased, Max_{th} shows an influence: when Max_{th} increases, the average queue size grows up to a point where it exceeds the target value. And the minimum values of the average queue size are observed for small values of Max_{th}.

On the second hand, we observe that the larger the traffic load is, the more significant is the performance difference obtained for different values of Max_{th}. In case of a heavy traffic load, for small and large values of Max_{th}, the queue size exceeds its target value. Nevertheless, given a number of flows, the minimum average queue size is obtained for a value of Max_{th} that does not exceed $2\hat{K}_T$. This "optimal" value of Max_{th} increases with the traffic load. For instance, in Figure 3.1(a), $Max_{th} = 10.5$ gives the lowest average queue size for 3 flows, and $Max_{th} = 18$ is optimal for 53, 73 and 103 flows. The increase

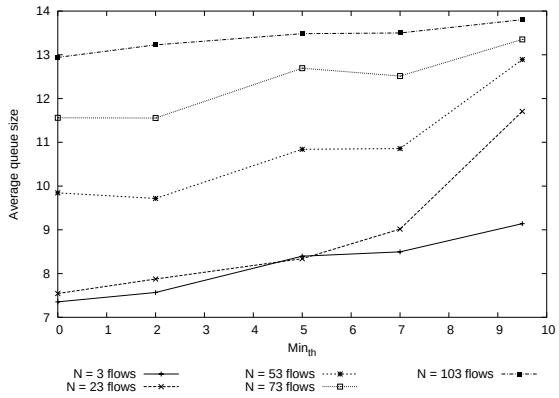
of this value with the number of flows is more visible when Min_{th} is larger. For instance, in Figure 3.2(c), for $Min_{th} = 9.5$, the optimal value of Max_{th} for 103 flows is 20, for 73 flows it is 18, for 53 flows it is 15 and for 23 flows it is 13.

Hence, the value of Max_{th} that gives the lowest average queue size depends on the number of flows and does not always match the symmetric value of Min_{th} as compared to $\hat{K}_T$. However, the average queue size obtained for $Max_{th} = 2\hat{K}_T$ is never far from the lowest average queue size observed. That is whatever the number of flows, the average queue size obtained for $Max_{th} = 2\hat{K}_T$ is tolerable. But note that given a number of flows, the values of the lowest average queue size observed increases with Min_{th}. This is because when Min_{th} is too large, the random drop is not started early enough allowing the queue size to build up. For instance, for $Min_{th} = 0$, the lowest value of the average queue size observed is around 9 for 103 flows. For $Min_{th} = 2$, we obtain an average queue size around 11, for $Min_{th} = 5$ around 12, for $Min_{th} = 7$ around 15.5 and for $Min_{th} = 9.5$ around 14.

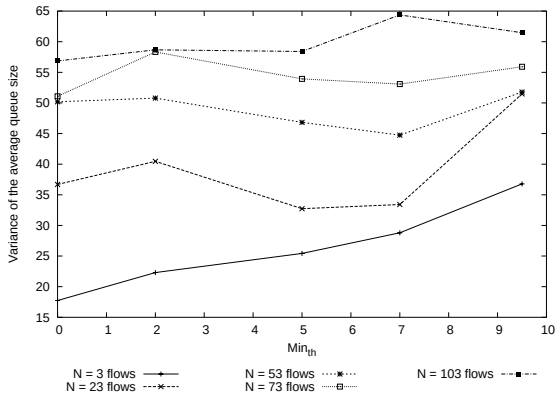
Given these observations, we search for the values of Max_{th} for which the queue size variance is minimized. Like the average queue size, for small number of flows (3 flows for instance), no significant changes in the variance is observed as the value of Max_{th} differs. Moreover, if Min_{th} is increased, then the variance is increased with Max_{th}. This is observed for instance for 23 flows and for values of Min_{th} above 5 (Figure 3.1(f), 3.2(b) and 3.2(d)). On the other hand, for a large number of flows and fixed values of Min_{th}, the variance is large for small values of Max_{th} and decreases to its smallest value for values of Max_{th} around $2\hat{K}_T$. It then increases for values of Max_{th} above $2\hat{K}_T$. This last increase is more pronounced as Min_{th} is increased. However, the increase observed for large values of Max_{th} is less important as the one observed for small values of Max_{th}. As we shall discuss in Section 3.2.2, the important variance for small values of Max_{th} is caused by the successions of quick build up of the queue since the number of flows is large, followed by quick decreases of the queue size. The variance observed for large Max_{th} is generated by a larger queue size due to a continual build up of the queue. Note that like the average queue size, the lowest variance observed when varying Max_{th}, increases when Min_{th} is increased. This observation suggests again to use small values of Min_{th}, in particular $Min_{th} = 0$. In this case, for small values of Min_{th}, the results presented in Figures 3.1 and 3.2 also encourage to use the value of Max_{th} around $2\hat{K}_T$ so as to obtain the lowest variance. This result holds whatever the number of flows. Nevertheless, if a large value of Min_{th} is used then we observe that the values of Max_{th} that give the lowest queue size variance differs according to the traffic load. Figure 3.1 shows that this value of Max_{th} increases with the traffic load. For instance, for 3 flows, the lowest variance is obtained for Max_{th} equal to 10.5, and for 103 flows it is equal to 20. However, as in the case of the average queue size, a value of Max_{th} equal to $2\hat{K}_T$ is a good trade-off between the performances obtained by different numbers of flows since it offers a variance not much different from the lowest one.

Let us now consider the influence of the parameter Min_{th} on the variance and the average queue size by examining the results reported in Figures 3.3 and 3.4.

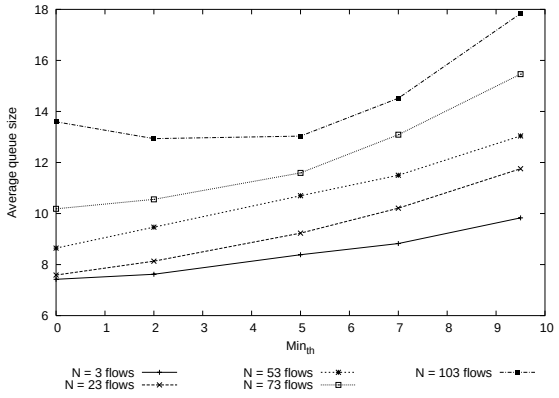
The best overall performances observed in these figures (*i.e.* the lowest average and variance of the queue size) are for small values of Min_{th}. This observation holds whatever the number of flows. This shows that for our scheme the "optimal" value of Min_{th} does not



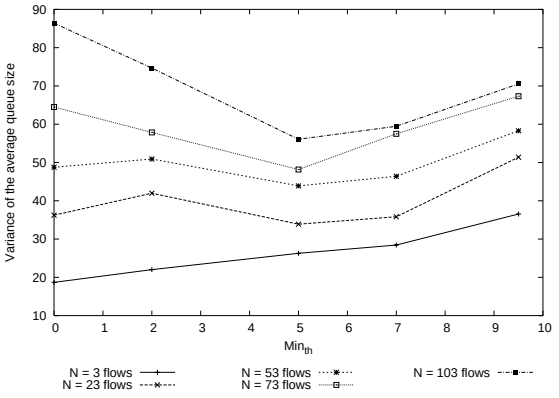
(a) Average queue size ($Max_{th} = 10.5$)



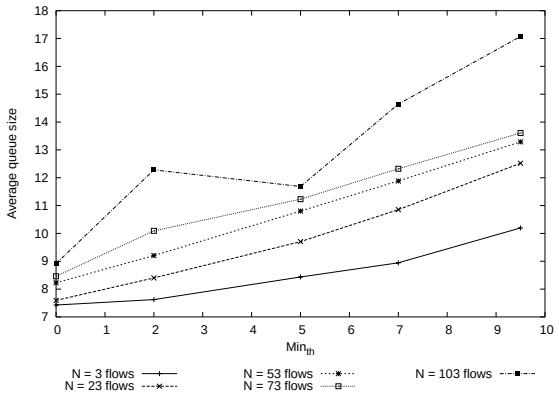
(b) Queue size variance ($Max_{th} = 10.5$)



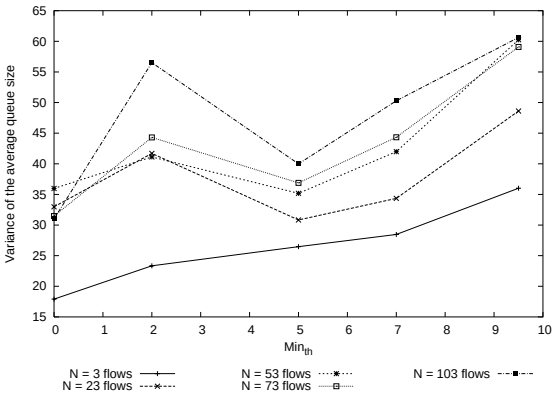
(c) Average queue size ($Max_{th} = 15$)



(d) Queue size variance ($Max_{th} = 15$)

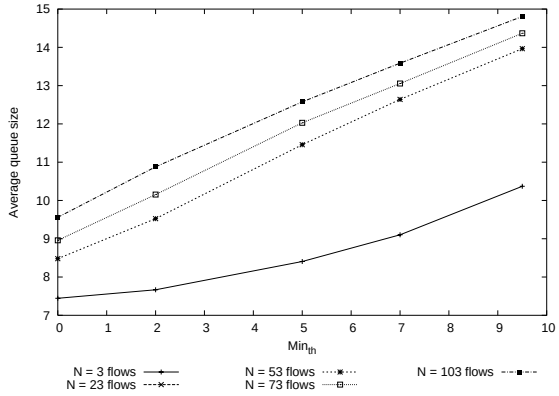


(e) Average queue size ($Max_{th} = 18$)

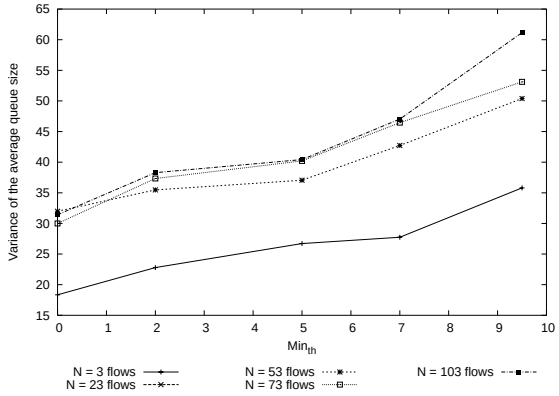


(f) Queue size variance ($Max_{th} = 18$)

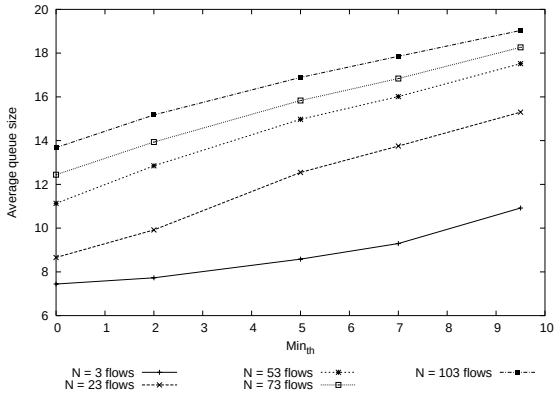
Figure 3.3: Influence of Min_{th} for different values of $Max_{th} \in \{10.5, 15, 18\}$



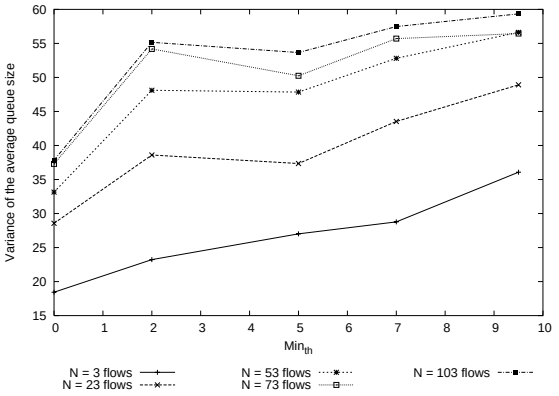
(a) Average queue size ($Max_{th} = 20$)



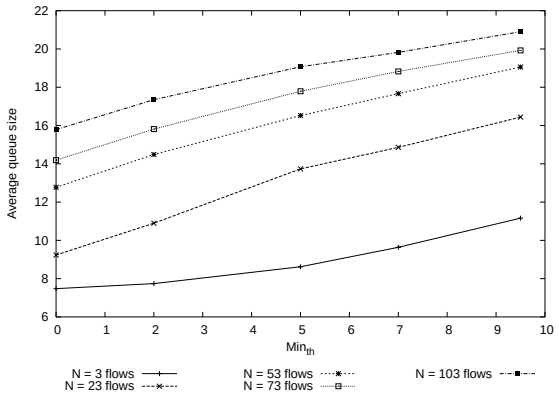
(b) Queue size variance ($Max_{th} = 20$)



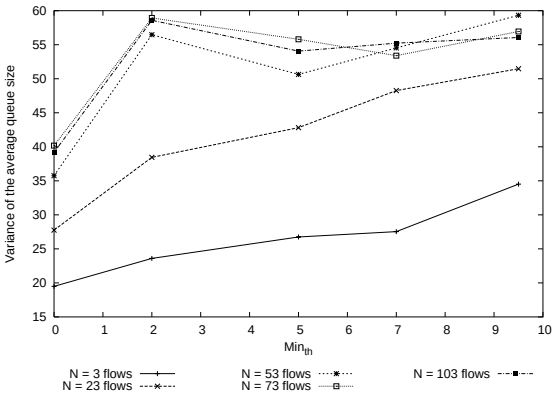
(c) Average queue size ($Max_{th} = 30$)



(d) Queue size variance ($Max_{th} = 30$)



(e) Average queue size ($Max_{th} = 35$)



(f) Queue size variance ($Max_{th} = 35$)

Figure 3.4: Influence of Min_{th} for different values of $Max_{th} \in \{20, 30, 35\}$

depend on the traffic load. Indeed, Figures 3.3 and 3.4 clearly illustrated that whatever the number of flows, the value of Min_{th} that gives the lowest average queue size are obtained for values of Min_{th} equal to 0. And for a given number of flow, the average queue size increases with the value of Min_{th}. In the same manner, for large and fixed values of Max_{th}, a value of Min_{th} equal to 0 gives the lowest queue size variance whereas for small values of Max_{th} we do not observe significant difference of the variance as Min_{th} varies. In general, the queue size variance increases with Min_{th}.

Finally, a value of Min_{th} equal to 0 gives a lower average queue size as well as a lower queue size variance. Based on the results stated above, in order to keep an average queue size less or equal to the desired target value, we need to choose a value of Min_{th} equal to 0 and a value of Max_{th} close to $2\hat{K}_T$. Even if $Max_{th} = 2\hat{K}_T$ does not always give the lowest average queue size, it gives a tolerable performance whatever the number of flows. Moreover, these same values of the parameters Min_{th} and Max_{th} allows us to minimize the queue size variance. Since all the above results are obtained for a certain fixed value of $coef$, γ and the upper bound of Max_p, we changed these parameters and tested if all these observations we made hold for other configuration of the parameters. We therefore conduct the experiments reported in Table 3.1 by setting $coef = 1$, $\gamma = 1$ and the upper bound of Max_p to 0.5. We considered $N = 23$ flows. These results allow us to derive the same observations as above.

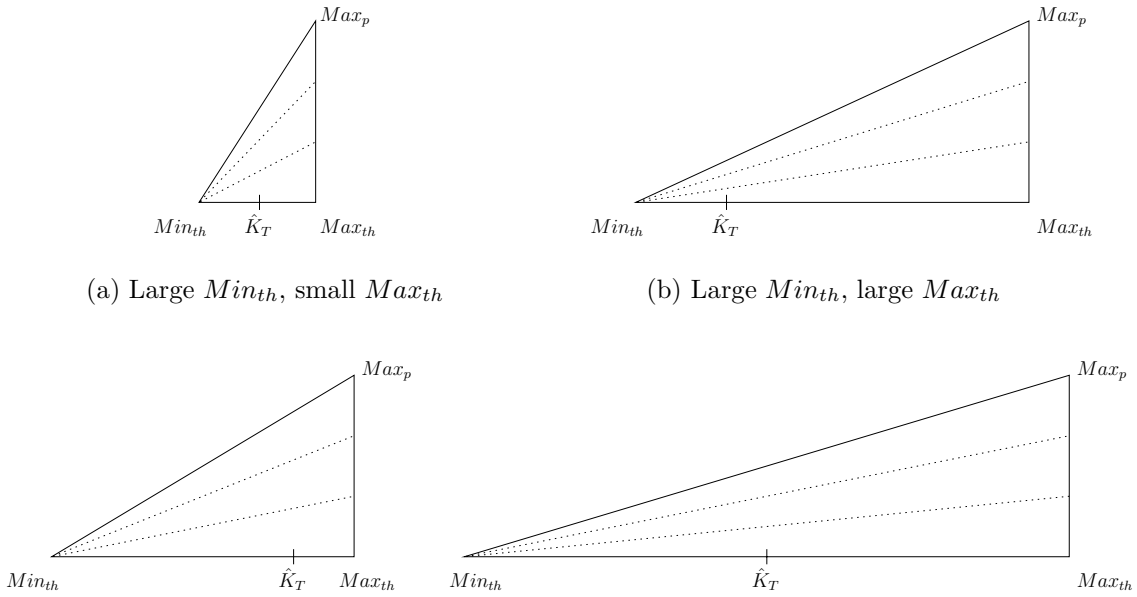
Table 3.1: Influence of Min_{th} and Max_{th} on our adaptive scheme for 23 flows, $coef = 1$ and $\gamma = 1$ (AQS = Average Queue Size, VQS = Variance of Queue Size and PLR = Packet Loss Rate, $P_{5-15} = P(5 < X \leq 15)$)

Max_{th}	Min_{th}	Row	AQS	VQS	PLR	Empty queue	Full queue	P_{5-15}
10.5	0	1	10.55	41.99	12.04%	1.52%	0.09%	49%
	5	2	11.35	43.63	13.51%	4.43%	0.049%	54.05%
	9.75	3	12.13	47.02	13.26%	3.97%	0.071%	53.34%
15	0	4	10.87	54.30	12.26%	2.01%	0.12%	40.9%
	5	5	11.49	47.64	13.58%	4.68%	0.093%	52.68%
	9.75	6	13.39	55.88	13.10%	3.78%	0.148%	45.614%
20	0	7	10.83	52.84	12.06%	1.68%	0.22%	43.4%
	5	8	12.56	47.23	13.39%	3.17%	0.1945%	53.94%
	9.75	9	14.53	52.20	13.14%	2.25%	0.233%	46.25%
35	0	10	12.93	37.78	11.68%	0.38%	0.28%	56.9%
	5	11	16.87	47.55	12.77%	1.16%	0.31%	38.07%
	9.75	12	19.24	51.04	12.52%	0.928%	0.582%	26.65%

3.2.2 Analysis

In order to give more precise explanations to the observed results, we summarize the effect of the choice of the values of Min_{th} and Max_{th} into four different extreme and important

cases. Figures 3.5(a) to 3.5(d) illustrate these cases by representing the RED drop function by a triangle.



(a) Large Min_{th}, small Max_{th}

(b) Large Min_{th}, large Max_{th}

(c) Small Min_{th}, small Max_{th}

(d) Small Min_{th}, large Max_{th}

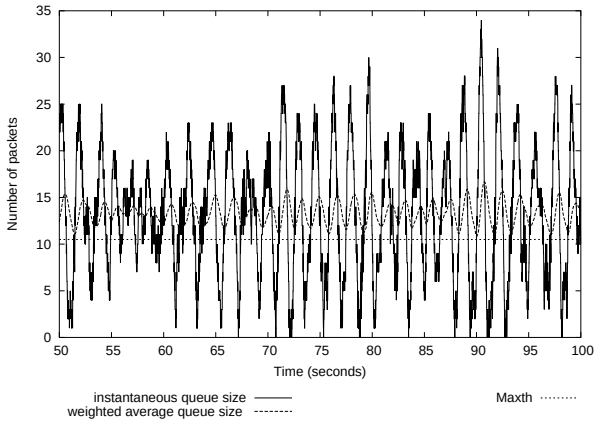
Figure 3.5: Variations of Min_{th} and Max_{th}

For the simplicity of analysis, we do not consider PSAND in gentle mode but rather in the strict mode. This means that the 100% rejection of packets starts when the weighted average queue size exceeds Max_{th}.

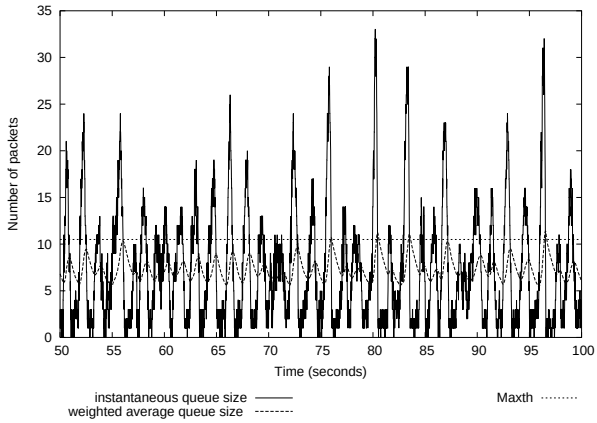
Figure 3.5(a) corresponds to the case presented in the third row of Table 3.1 where the succession of a quick increase (due to large Min_{th}) and a quick decrease of the queue size (due to small Max_{th}) gives rise to *a large oscillation of the queue size*. Moreover, an early drop due to Max_{th} gives *a large loss rate* as compared to the other cases. Even if this triangle produces the worst performances as compared to the others, it offers a more precise average queue size close to the target size.

Figure 3.5(b) corresponds to the case of the sixth and the last rows of Table 3.1. For this case, the queue experiences a quick increase (due to large Min_{th}) and continues to build up (due to large Max_{th}) leading to *an average queue size that exceeds the target queue size*. However, due to a large Max_{th}, we obtain a lower loss rate as compared to the case of Figure 3.5(a).

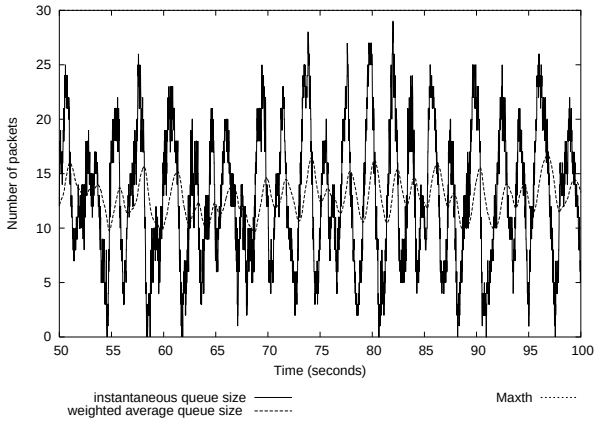
Figure 3.5(c) corresponds to the case presented in the first row of Table 3.1. For this case, due to an early drop triggered by Max_{th} in conjunction to an early random drop triggered by Min_{th}, we observe *a very small average queue size*. In addition, because of a slow built up of the queue (Min_{th} small), this case obtains the lowest variance of the queue size. However, this case experiments *one of the highest loss rates* due a small Max_{th}.



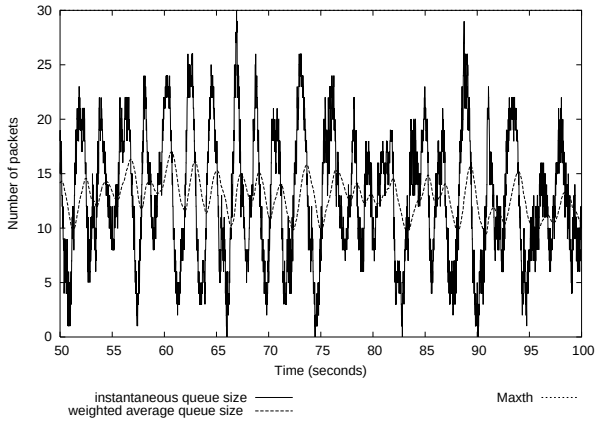
(a) PSAND with $Max_{th} = 10.5$ in gentle mode



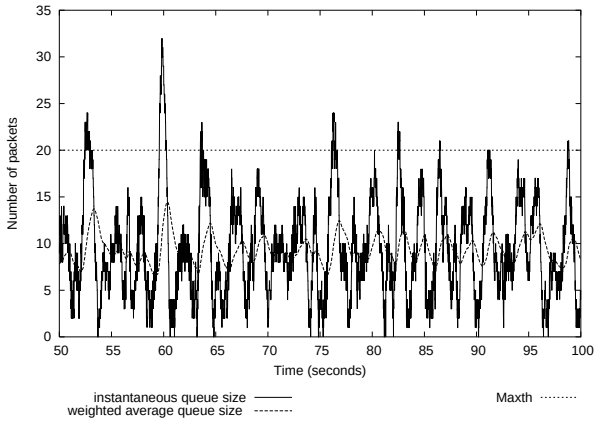
(b) PSAND with $Max_{th} = 10.5$



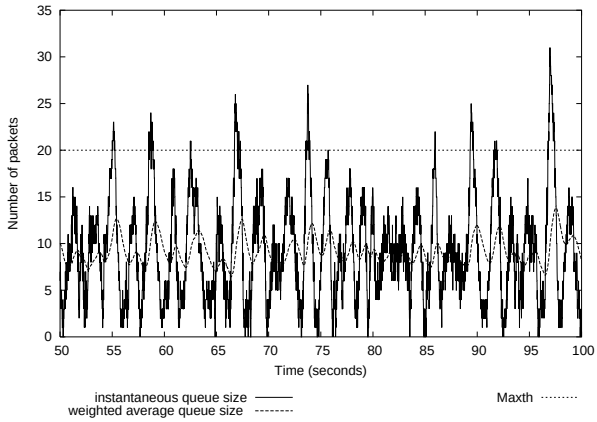
(c) PSAND with $Max_{th} = 30$ in gentle mode



(d) PSAND with $Max_{th} = 30$



(e) PSAND with $Max_{th} = 20$ in gentle mode



(f) PSAND with $Max_{th} = 20$

Figure 3.6: Evolution of the instantaneous and the weighted average queue size for 100 flows

Figure 3.5(d) corresponds to the case presented in the fourth or seventh row of Table 3.1. This is an interesting case where the best overall performance is observed. Our method of configuration of Min_{th} and Max_{th} is based on this case. By starting the random drop earlier (very small Min_{th}) at a rate which reflects closely the evolution of the queue length, our scheme scatters the packets drops over time and avoids sharp changes of the queue size. This way, it maintains a slower and gradual build up of the queue and therefore reduces the oscillations of the queue length as well as the average queue size as compared to the use of a threshold mechanism as in [34, 41]. In addition as the 100% packets drop due to Max_{th} is not triggered early (Max_{th} quite large) and as our scheme drops packets at a proportion which is a function of the queue size changes, we do not observe an increase of the loss rate as compared to [34, 41]. We rather observe a slight decrease.

In summary, large values of Min_{th} cause pronouced oscillations of the queue size since there are periods where the early drop starts late allowing a queue build up and reduces quickly the queue size. In addition, if Max_{th} is large then it leads to a queueing delay larger than the specified target queueing delay. This is illustrated by Table 3.1. For instance, we obtain in row 10 of Table 3.1, an average queue size equal to 12.93 packets exceeding the target value of 10 packets. Moreover, small values of Max_{th} cause an important packet loss rate.

All these analysis are given for PSAND in the strict mode. Let us now examine where this analysis differs when we choose PSAND in gentle mode.

3.2.3 The gentle versus the strict mode

We plot in Figure 3.6 the evolution of the queue size for a simulation run from 50 to 100 seconds with the aim of visualizing the evolution of the queueing delay observed for different values of Max_{th} and explaining the increased queueing delay observed for small and very large values of Max_{th}. These figures allows us also to visualize the difference between the gentle and strict mode of PSAND. In gentle mode, the 100% rejection rate starts only when the weighted average queue size reaches $2Max_{th}$.

Figure 3.6(a) shows that, even if Max_{th} is small, since we are in gentle mode and the 100% rejection rate starts only when the average queue size reaches $2Max_{th}$, the queue builds up quickly, in particular when the traffic is more bursty as in this case where 100 TCP flows compete for the bottleneck link. This explains the large queueing delay observed in Figure 3.1 for small values of Max_{th}. The weighted average queue size never reaches $2Max_{th}$. It oscillates between Max_{th} and $2Max_{th}$. Hence, packet losses occur only randomly. But, as the slope obtained for small Max_{th} is important, packets are dropped with a higher probability between the upper bound of Max_p and 1, and consequently decreasing the queue after a quick build up. This leads to more oscillations of the queue size as confirmed by the large variance observed for small Max_{th} in Figure 3.1.

Figure 3.6(b) considers a non gentle mode where an earlier 100% drop occurs when the weighted average queue size reaches Max_{th}. In this case, we observe a lower average queue size and a larger dropping rate due to an important slope and an earlier 100% rejection rate.

If Max_{th} is set too large as seen in Figure 3.6(c) and 3.6(d), the average queue size oscillates around a value which is larger than the target queue size. However, no significant difference is obtained between the gentle and the non gentle mode since $2Max_{th}$ is above the queue capacity and is therefore never reached. Since $Min_{th} = 0$ and Max_{th} is large, slope of the drop function is small Even for large Max_{th}, if the value of Min_{th} is increased then the slope of the drop function increases introducing hence more oscillations as see n in Figure 3.1.

Note that for all these figures, Max_p takes always the value of 0.75 because of a sustained traffic load generated by 100 TCP flows. Figure 3.6(e) shows better performance for a slope obtained from $Min_{th} = 0$ and $Max_{th} = 2\hat{K}_T$. This result can be explained as follows. A more precise control leads to a system that reacts too quickly and introduces more instability (for example a large Max_{th} increases the slope and creates more fluctuations since there is a larger dropping rate: the TCP sources reduce their congestion window and leads to more oscillations of the queue size). On the other hand, for a less precise control the system does not sufficiently react (for example a large Max_{th}). There exists a trade-off between these two controls that gives good performances.

Based on these results, we can observe that the difference between the gentle and the strict mode is only visible when Max_{th} takes small values. Since we have used the configuration of Max_{th} that gives the best performance *i.e.* for values of Max_{th} near $2\hat{K}_T$, we do not observe a performance difference between the strict and the gentle mode.

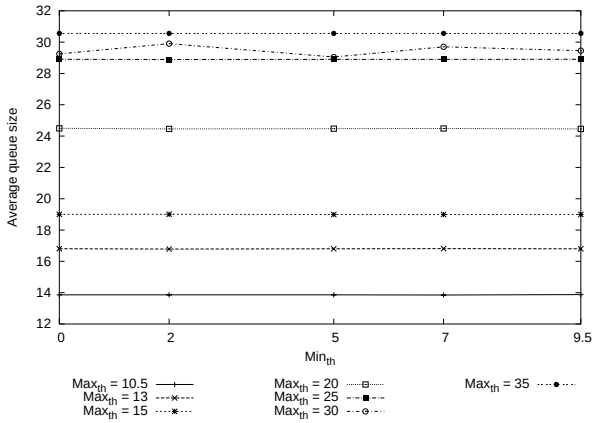
3.2.4 Recommendations

In addition to all the above experiments, we have also tested the performance of $ARED_{Feng}$ and $ARED_{Floyd}$ schemes for different values of Min_{th} and Max_{th} and present the results in Figure 3.7. If we compare the performance obtained by these figures with the performance obtained by PSAND for different values of Min_{th} and Max_{th} illustrated by Figures 3.1 to 3.4, we can observe better performance for PSAND.

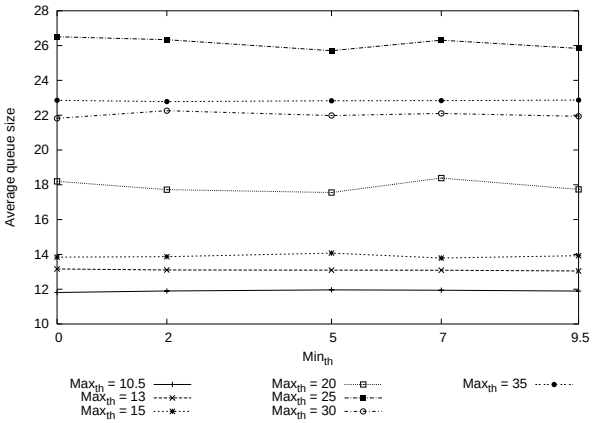
We have shown in Section 3.2.1 that a value of Max_{th} close to 20 which corresponds to $(2 \times \hat{K}_T)$ gives an overall good performance for different number of flows. Indeed, large values of Max_{th} close to the buffer capacity increase the average queue size and hence give a large queuing delay. The queuing delay increases with Max_{th} whereas the queue size variance decreases with Max_{th}. After a quick build up of the queue, small values of Max_{th} decrease sharply the queue size creating pronounced oscillations of the queue size. Moreover Max_{th} does not seem to affect significantly the packet loss rate since a negligible decrease of the loss rate is detected when Max_{th} increases. In addition, the results of Figures 3.3 and 3.4 also show that a value of Min_{th} equal to 0 gives an overall good performances.

Based on these results, we follow the guideline that suggests to

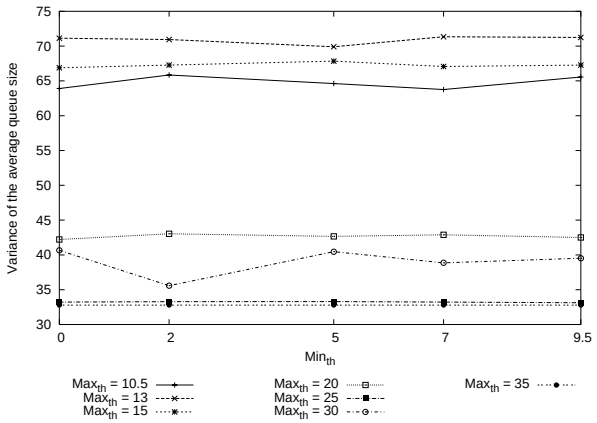
configure Min_{th} and Max_{th} so as they remain symmetrical about the target queue size $\hat{K}_T$ (*i.e.* $\hat{K}_T = (Min_{th} + Max_{th})/2$) while being as far apart as feasible depending on the value of $\hat{K}_T$ as compared to the buffer capacity B. The value of these two parameters are then chosen as follows:



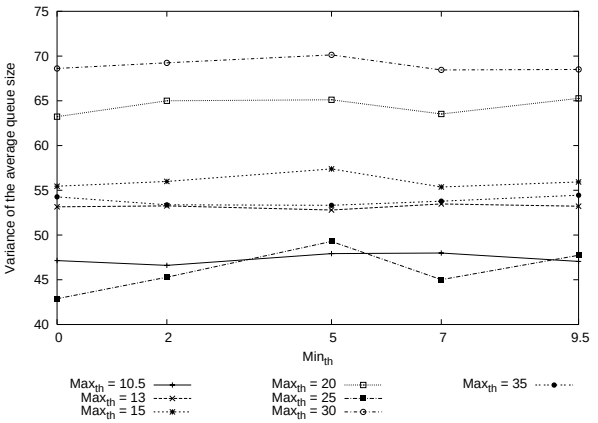
(a) Average queue size ($ARED_{Feng}$)



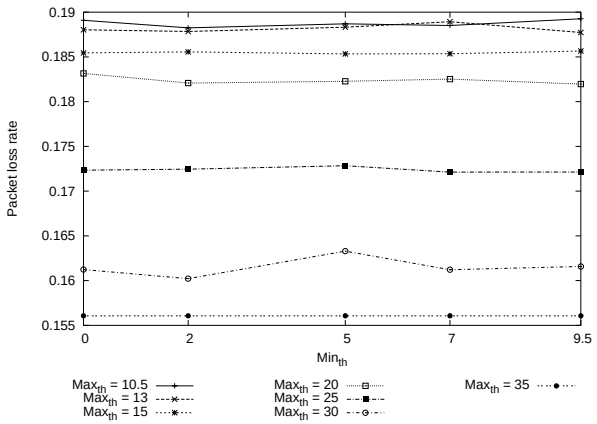
(b) Average queue size ($ARED_{Floyd}$)



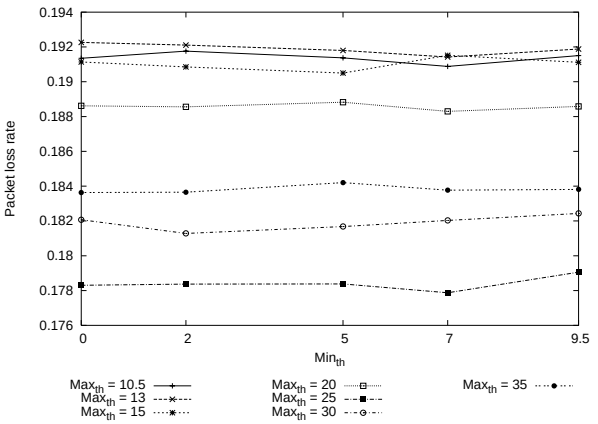
(c) queue size variance ($ARED_{Feng}$)



(d) queue size variance ($ARED_{Floyd}$)



(e) Packet loss rate ($ARED_{Feng}$)



(f) Packet loss rate ($ARED_{Floyd}$)

Figure 3.7: Varying Min_{th} and Max_{th} for $ARED_{Feng}$ and $ARED_{Floyd}$

$$\textbf{If } \hat{K}_T > \tfrac{B}{2}: \quad \begin{cases} Min_{th} = 2\hat{K}_T - B \ , \\ Max_{th} = B \ . \end{cases}$$

$$\textbf{If } \hat{K}_T \leq \tfrac{B}{2}: \quad \begin{cases} Min_{th} = 0 \ , \\ Max_{th} = 2\hat{K}_T \ . \end{cases}$$

3.3 Adapting Min_{th} and Max_{th}

In this section, we exhibit few methods among other alternatives for the adaptation of Min_{th} and Max_{th}. The parameter Max_p is adapted according to the PSAND scheme described in the previous chapter. The aim of this section is to show the drawbacks of the few enumerated methods of adaptation of Min_{th} and Max_{th}, as compared to the configuration proposed in the previous section. We do not intend neither to study nor explain the behavior of each of these mechanisms in detail but rather illustrate their performance. We consider a heavier traffic load generated by 100 TCP flows so as to test how these methods react in presence of rough traffic conditions.

Table 3.2: Different methods to adapt the parameters Min_{th} and Max_{th}

Row	Min_{th}	Max_{th}	Symmetrical evolution	Max_p change rate
1	Fixed	$Max_{th} - \delta_{\hat{K}}$	No	β
2	Fixed	$Max_{th} + \delta_{\hat{K}}$	No	β
3	$Min_{th} + \delta_{\hat{K}}$	$Max_{th} - \delta_{\hat{K}}$	Yes	β
4	$Min_{th} - \delta_{\hat{K}}$	$Max_{th} + \delta_{\hat{K}}$	Yes	β
5	$Min_{th} + \delta_{\hat{K}}$	$Max_{th} + \delta_{\hat{K}}$	No	β
6	$Min_{th} - \delta_{\hat{K}}$	$Max_{th} - \delta_{\hat{K}}$	No	β
7	$Min_{th} = \hat{K}_T$	$Max_{th} = \hat{K}_T$	Yes	–
8	$Min_{th} + \delta_{\hat{K}}$	$Max_{th} - \delta_{\hat{K}}$	Yes	$\frac{Max_{th\,i+1} - Min_{th\,i+1}}{Max_{th\,i} - Min_{th\,i}}$

There exist different ways to adapt the parameters Min_{th} and Max_{th}. We enumerate a few of them and we examine briefly their performance.

Min_{th} and Max_{th} can be updated by using a fixed constant multiplicative or additive factor like the adaptation of Max_p in [34] and [41]. The other alternative is to update these parameters with a variable factor that depends on the queue size change. There are several methods for this alternative idea. Table 3.2 describes a few of these methods.

Several questions should be answered for the adaptation of Min_{th} and Max_{th}. The first problem to be solved is to find the appropriate rate of adaptation. In all the following examples, we adapted Min_{th} and Max_{th} as a function of the changes in the average queue size in an interval as in the case of the adaptation of Max_p. Hence the change rate denoted $\delta_{\hat{K}}$ is computed as:

$$\delta_{\hat{K}} = \hat{K}_{cur} - \hat{K}_{prev} \ ,$$

Table 3.3: Performance of different adaptive methods of Min_{th} and Max_{th} ($\overline{Max_p} = 0.75$)

	AQS	VQS	PLR (%)
1	19.5	134.6	21.7
2	16.0	51.9	23.1
3	9.3	28.2	12.4
4	12.42	35.3	18.2
5	12.6	35.2	17.9
6	12.8	35.3	18.1
7	8.4	57.4	15.5
8	14	108	12.3

with the same interpretation for these values as in Chapter 2 Section 2.3.

Note that by deciding either to fix Min_{th} and Max_{th} at a constant value, either to adapt them using $\delta_{\hat{K}}$, we only explore a part of the universe of adaptation mechanisms. For instance, alternative solutions could be adapting Min_{th} and Max_{th} using an intermediate value that can be more or less aggressive. The moving weighted average that belongs to the category of adaptive mechanisms could be used in order to compute these intermediate values. For instance, the new value of Min_{th} could be adapted as follows:

$$Min_{th}(i+1) \;=\; Min_{th}(i) \times \alpha + (Min_{th}(i) \pm \delta_{\hat{K}}) \times (1-\alpha)\ .$$

The two extreme cases are obtained for $\alpha = 0$ and $\alpha = 1$. If $\alpha = 0$, then Min_{th} is adapted directly with $\delta_{\hat{K}}$. If $\alpha = 1$, then Min_{th} is fixed. For $0 < \alpha < 1$, we obtain alternative values of adaptation.

The optimal performances obtained could be from these intermediate alternatives and not from the two extreme cases. Nevertheless this method would increase once more the number of RED parameters and we transform the problem by a new problem of finding the optimal value of α.

The second question is how to make Min_{th} and Max_{th} evolve. Should we choose necessarily a symmetric evolution with respect to $\hat{K}_T$? Should Min_{th} and Max_{th} be increased or decreased as the traffic load increases, that is, the average queue size increases?

The methods we will enumerate below and describe in Table 3.2 differ in the way they answer these questions. The performance of these methods is described in Table 3.3. Each performance of a row of Table 3.3 corresponds to the method of the same row of Table 3.2.

In the first and second rows of Table 3.2, we present two methods that adapt Max_{th} and fixed Min_{th} to 0 since the above simulations results showed that this value of Min_{th} shows good performances whatever the number of flows. For the method of row 1, Max_{th} is decreased when the weighted average queue size is increased whereas for the row 2, Max_{th} is increased when the weighted average is increased. Both methods do not keep the symmetry that exists between Min_{th} and Max_{th} as compared to $\hat{K}_T$.

From row 3 to 8, both parameters Min_{th} and Max_{th} are adapted. All of these methods keep the symmetry between Min_{th} and Max_{th} except the methods presented in rows 5

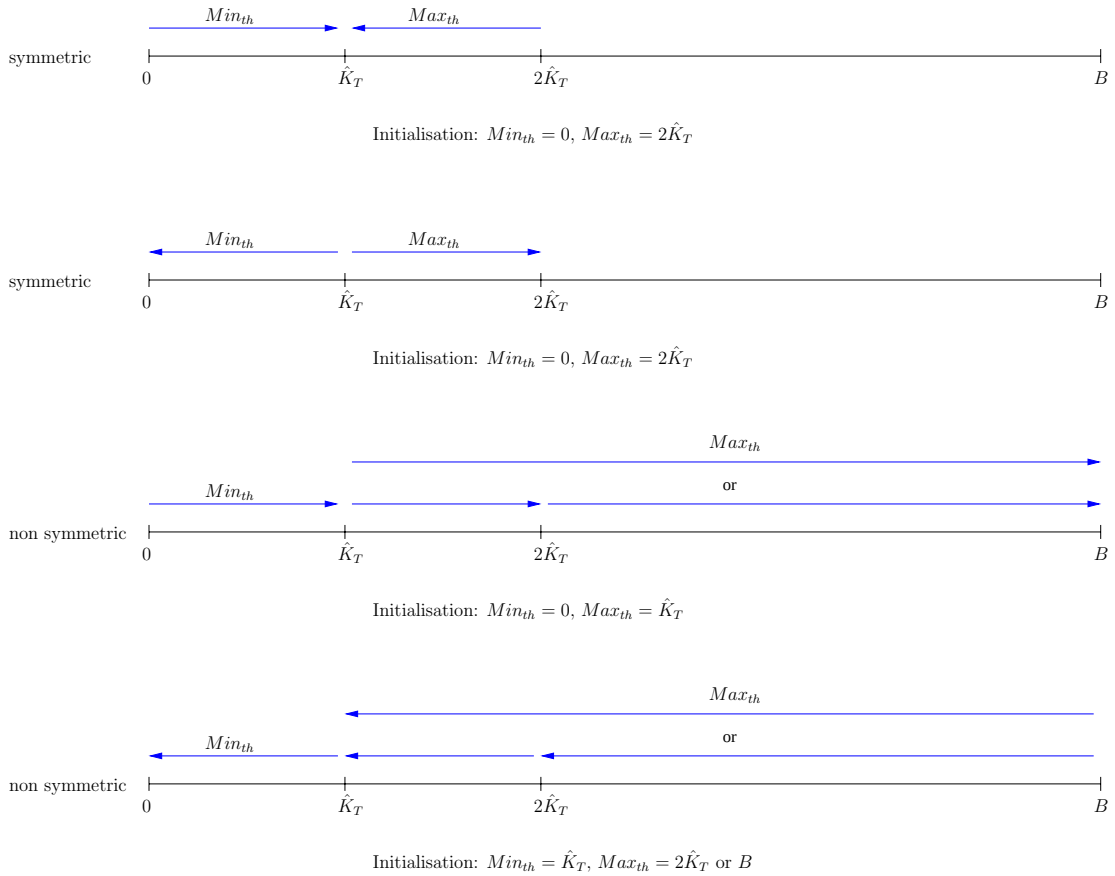
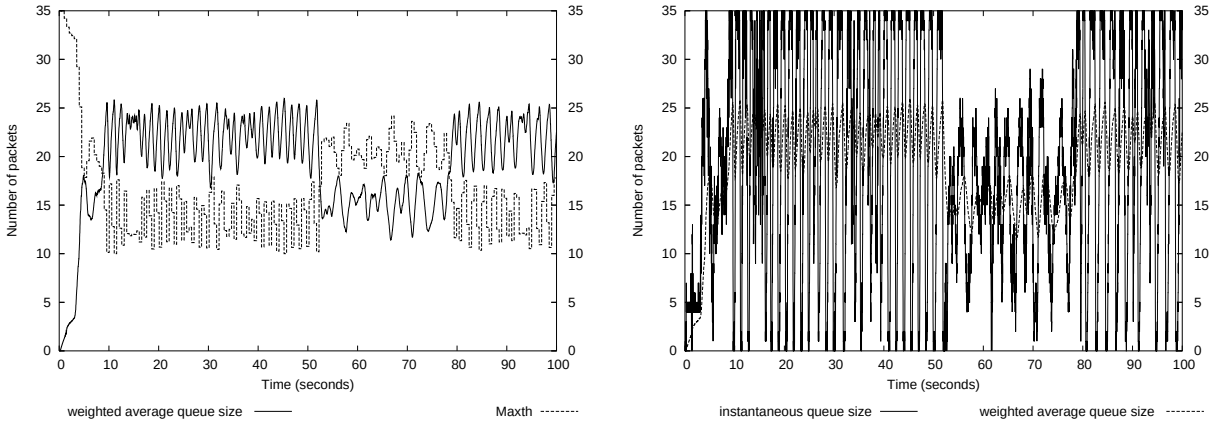


Figure 3.8: Evolution of Min_{th} and Max_{th} corresponding to rows 3 to 6 in Table 3.2

and 6.

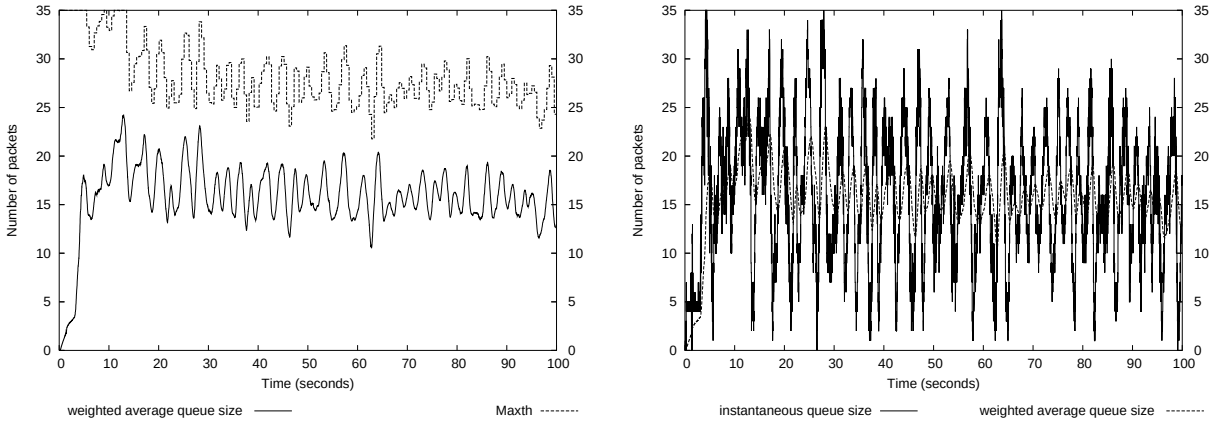
In the following, we present the simulation results of some of the methods presented in Table 3.2. Figure 3.9 to 3.14 display these results. Each figure is the result of one simulation run for 100 seconds with the same network configuration as in the previous sections.

For the mechanisms of rows 1 and 2, Min_{th} is fixed to 0 allowing the start up of the random drop earlier. However, in row 1, since Max_{th} is increased when $\hat{K}_{cur}$ increases, Max_{th} oscillates around small values close to $\hat{K}_T$. Whereas for row 2, since Max_{th} is decreased when $\hat{K}_{cur}$ increases, Max_{th} oscillates around large values between $2\hat{K}_T$ and the buffer capacity. The values around which Max_{th} evolves has an influence on the oscillations of the queue size as illustrated by Figures 3.9 and 3.10. For small Max_{th} in Figure 3.9, we observe pronounced oscillations of the queue size and an important queueing delay. Because of a sustained traffic (100 flows for instance) the average queue size exceeds Max_{th} but the dropping of packets continues at a higher probability between the upper bound of Max_p and 1 until the reduction of the average queue size below Max_{th}. This reduces the queue size oscillations observed from 50 to 80 seconds of the simulations of Figure 3.9. Since



(a) (b)

Figure 3.9: $Min_{th} = 0$, when $\hat{K}_{cur}$ increases Max_{th} decreases between $\hat{K}_T$ and buffer capacity (mode `gentle`, $\overline{Max_p} = 0.75$)

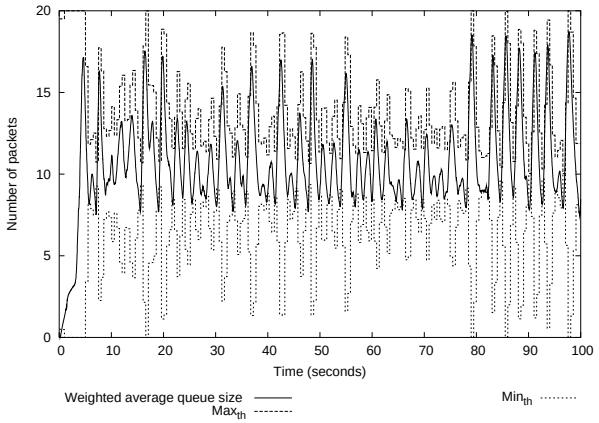


(a) (b)

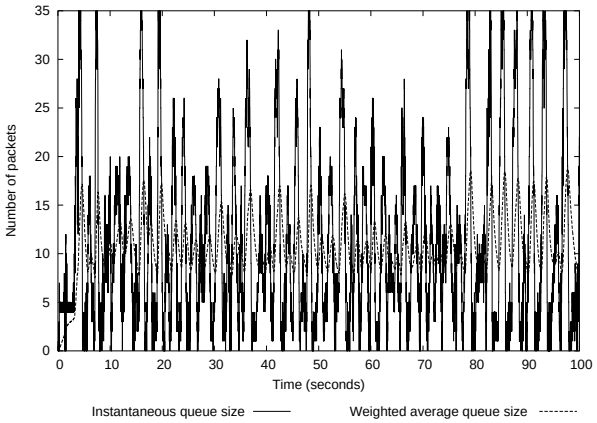
Figure 3.10: $Min_{th} = 0$, when $\hat{K}_{cur}$ increases Max_{th} increases between $\hat{K}_T$ and buffer capacity (mode `gentle`, $\overline{Max_p} = 0.75$)

the traffic load is important the average queue size increases above Max_{th} and the cycle continues. However for lower traffic load we observe less pronounced oscillations where the average queue size is below Max_{th}.

As for the mechanism presented in Figure 3.10, because of large Max_{th}, we observe less pronounced oscillations of the queue size as compared to the case of Figure 3.9. However, as observed also in the previous section, large values of Max_{th} cause an important average queue size. These two figures confirm the results observed in the previous section. That



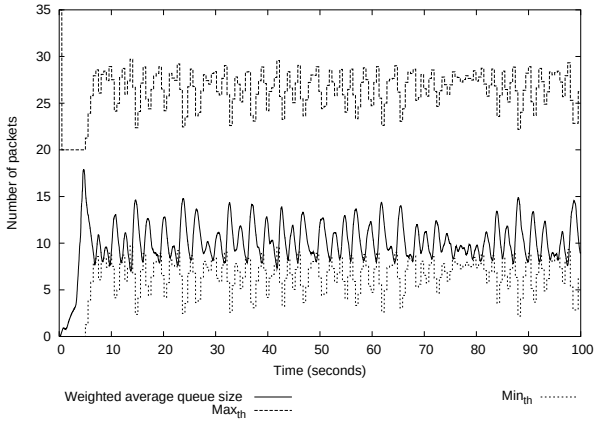
(a) Min_{th} and Max_{th}



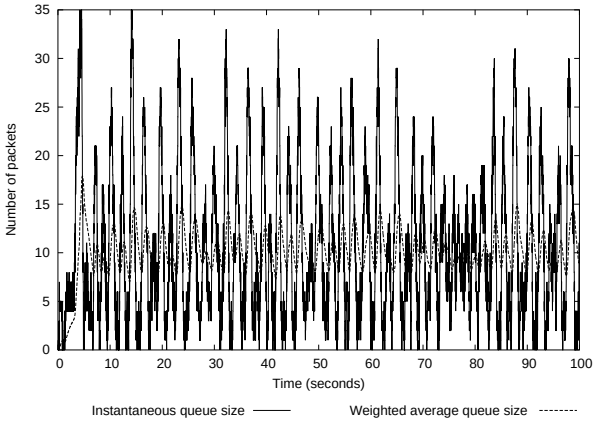
(b) Queue size evolution

Figure 3.11: Symmetric evolution, when $\hat{K}_{cur}$ and Min_{th} decrease, with 20 sources

is, large and very small values of Max_{th} causes important queueing delay, and large values of Max_{th} gives less oscillations of the queue size than very small values of Max_{th}.



(a) Min_{th} and Max_{th}



(b) Queue size evolution

Figure 3.12: Non symmetric evolution, when $\hat{K}_{cur}$ and Min_{th} decrease, with 20 sources

The evolution of Min_{th} and Max_{th} from rows 3 to 6 are illustrated in Figure 3.8. The methods presented from rows 3 to 6 give more or less similar behavior. We only consider the case of row 4 and 6 to illustrate these methods. Both cases decrease Min_{th} when the average queue size increases. Their difference is that case 4 considers a symmetric evolution of Min_{th} and Max_{th} as compared to $\hat{K}_T$ (Figure 3.11). This figure shows that in case of a symmetric evolution (row 4), the weighted average queue size oscillates between Min_{th} and Max_{th} whereas for the non symmetric case (Figure 3.12) the average queue size oscillates

around the value of Max_{th}, giving a large queueing delay. Indeed, since the difference of both methods is on the evolution of Max_{th}, for the non-symmetric case, when the average queue size increases, Max_{th} decreases. Because of a gentle mode and a heavy traffic load, the queue size oscillates frequently above Max_{th} giving then poor performances.

The row 7 corresponding to the Figure 3.13 presents a method where both Min_{th} and Max_{th} are set to the target queue size. This means that all arriving packets are rejected when the weighted average queue size reaches its target value. In this case, there is no random drops (the parameter Min_{th} is eliminated). This method called Pseudo Drop Tail (PDT) is much more strict than the Drop Tail mechanism. It therefore increases the dropping rate and thus underutilize the link. Even if the weighted average queue size oscillates closely to the target queue size, the instantaneous queue size shows more oscillations between 0 and 20 packets. This mechanism depends on the responsiveness of the parameter ω_q. If ω_q is set to 1, then the mechanism reacts too quickly by rejected all packets when the instantaneous queue size reaches the target queue size. This becomes a Drop Tail with a buffer capacity set to the value of the target queue size. If ω_q is set to 0, then the weighted average size does not follow the evolution of the instantaneous queue size. In this case, the instantaneous queue size builds up and shows more oscillations since the weighted average is too slow to reach the target queue size. Figure 3.10 illustrates the evolution of the queue size for this mechanism.

The row 8 corresponding to Figure 3.14 presents also another mechanism (called Sandwich) where unlike all the other mechanisms that uses the PSAND adaptation scheme of Max_p, Max_p is adapted differently. The multiplicative factor of Max_p is computed so as to maintain the same slope of the drop function when Min_{th} and Max_{th} evolves. The evolution of Min_{th} and Max_{th} is a symmetric evolution as compared to the target queue. If the weighted average queue size increases, then Min_{th} increases and Max_{th} decreases. If the average queue size decreases, then Min_{th} decreases whereas Max_{th} increases. The change rate of Max_p is therefore calculated as:

$$\begin{aligned} Min_{th}(i+1) &= Min_{th}(i) + \delta\hat{K} \ , \\ Max_{th}(i+1) &= Max_{th}(i) - \delta\hat{K} \ , \\ \beta &= \frac{Max_{th}(i+1) - Min_{th}(i+1)}{Max_{th}(i) - Min_{th}(i)} \ , \\ Max_p(i+1) &= \beta \times Max_p(i) \ , \end{aligned}$$

where i is the interval of adaptation of Max_p.

The drawback of this method is that the change rate computed so as to keep a constant angle does not follow the principle of adaptive RED. This principle consists in increasing Max_p when the average queue size increases, and decreasing when the average queue size decreases. In this case, Max_p is decreased when the average queue size increases. Indeed, since Min_{th} increases and Max_{th} decreases when $\hat{K}_{cur}$ increases, the new interval $Max_{th} - Min_{th}$ decreases leading to a $\beta < 1$. Since Max_p is decreased when the traffic load increases, we initialize Max_p to 1. Otherwise Max_p would be too low and packets are

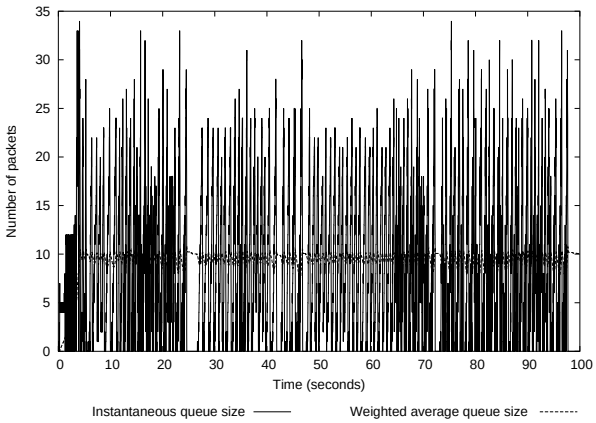
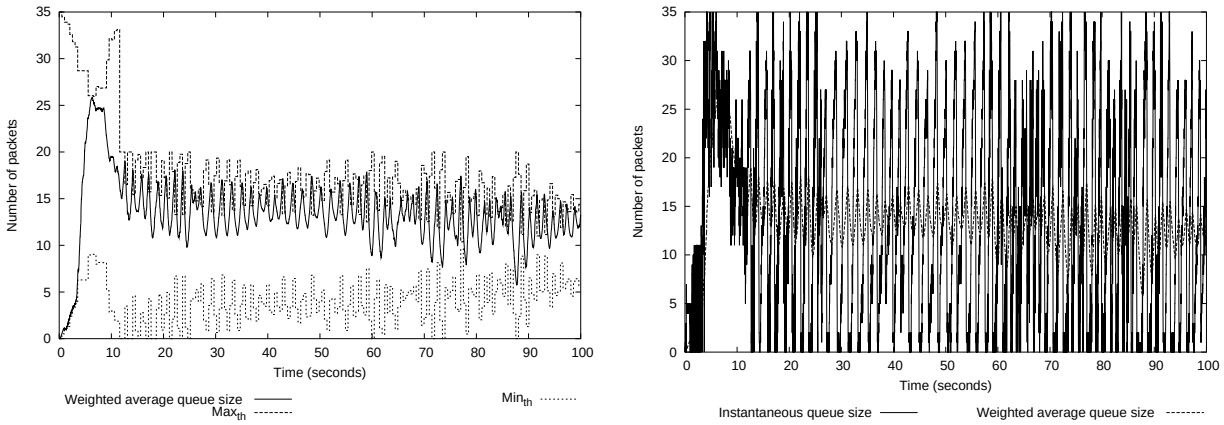


Figure 3.13: $Max_{th} = Min_{th} = \hat{K}_T$ (Pseudo Drop Tail) with 20 sources



(a) Min_{th} and Max_{th}

(b) Queue size evolution

Figure 3.14: Constant angle of the drop function (Sandwich) with 20 sources

not rejected at a rate that reduce the congestion. This behavior of this mechanism creates a large queue size where the average queue size oscillates around Max_{th}.

Min_{th} and Max_{th} can be adapted in several ways. However, we can resume all these ways as follows. If the adaptation of Max_{th} is around a very large or small values then the queue size oscillations and the average queue size are increased. However, if Max_{th} evolve in between *i.e.* around $2\hat{K}_T$, then we avoid large variance and important amplitude of the oscillations of the queue size. These results confirm the observations we made in the previous section for fixed Min_{th} and Max_{th}.

3.4 Conclusion

The results of all these experiments suggests that a sufficiently larger interval $[Min_{th}, Max_{th}$ that maintains the symmetry improves further the performances. Hence, by using small values of Min_{th} and large values of Max_{th}, our adaptive scheme of Max_p described in Algorithm 16 improves overall performances as compared to the threshold mechanisms as in [34, 41]. The described results show that for some fixed Max_{th}, the smaller Min_{th} is, the better is the overall performance. A small value of Min_{th} close to zero decreases the variance of the queue size, the average queue size, the loss rate and the probability that the queue is fully occupied and that it is empty. In the same way, for a fixed Min_{th}, small values of Max_{th} do not give the best performances. Moreover, the value of Max_{th} should not be chosen too large (in particular for a large number of flows) in order to avoid a large queuing delay as the average queue size could excessively exceed its target value.

Hence these results obtained in Section 3.2 for fixed Min_{th} and Max_{th} suggested to consider a configuration with Min_{th} set to 0 and Max_{th} set to $2\hat{K}_T$. If we compare the results of this configuration with the results obtained by several adaptive methods (refer to Table 3.3), we can notice that the configuration of a fixed Min_{th} and Max_{th} gives better performances. This reinforces our choice of fixing these two parameters without the need of adapting them.

Chapter 4

Comparing PSAND with other active queue management schemes

Contents

4.1 Introduction

We have proposed in Chapter 2 (page 67) an active queue management scheme called PSAND that follows the basic spirit of the adaptive RED presented in [34, 41] but differs in the method used to adjust the parameters Max_p, Min_{th} and Max_{th}. In particular, the parameter Max_p is adjusted as a function of the distance to the performance objective,

whereas in [34, 41], Max_p is adjusted with the same fixed value. We have compared our new method of RED parameters adjustement with the original RED and Adaptive RED. The results of these comparisons showed that PSAND gives a lower average queue size, achieving as a result the target queueing delay. It also reduces the queue size variance whatever the traffic load. These performance gains have been obtained without the sacrifice of the packet loss rate. Even though PSAND shows such performance improvement as compared to ARED, it is interesting to situate its performance among other mechanisms that are well-known or that have recently been proposed. We have described these mechanims in Chapter 1. The comparison is restricted to five of these mechanisms : BLUE [33], PI [52, 50], AVQ [67, 66, 68], LRED [97], REM [7, 8]. The reasons why we have chosen these five mechanisms are:

- BLUE : it is an adaptive mechanism like ARED and PSAND, it but does not use the queue length as a congestion measure. It uses instead the link utilization and the packet loss rate. Several works have compared their proposals with BLUE.

- PI, AVQ, LRED and REM since these mechanisms rely on a solid theoretical analysis and/or need scenario parameters as input to their model. In addition, these mechanisms are widely referenced.

- LRED since it is a robust mechanism that stabilizes the queue size whatever the traffic load: recall that our primary objective for PSAND was to reduce queue size variance. So, it could be interesting to see how far PSAND can stabilize the queue length as compared to LRED. Moreover, LRED is a very recent proposal (2004).

We present in the next section the two different topologies and in Section 4.4 the metrics used for the comparison of these mechanisms. The selected metrics allow to illustrate the strength or the weakness of a mechanism. Finally, we present like in Chapter 2 the results of the simulations we conducted under `ns` simulator [94] by making a qualitative analysis in Section 4.5 and a quantitative analysis through statistics in Section 4.6.

4.2 Topologies

Since we have compared PSAND with $ARED_{Feng}$ and $ARED_{Floyd}$ by using the topology presented in Section 2.4.1, we use first this same network topology in order to compare PSAND with the other mechanisms enumerated just above. This topology is composed by four different propagation delays: $2ms$ for the link between node S_1 and R_1, $3ms$ for the link between node S_2 and R_1, $20ms$ for the link between node R_1 and R_2 and $4ms$ for the link between node R_2 and S_3. In addition to the three basic flows as illustrated by Figure 2.2 (page 76), all others additionnal flows start from node S_1. This means that most of the connections use a propagation delay of $2ms$ on the link S_1 and R_1. This topology can

therefore be simplified into a topology where all flows experience almost the same round trip time. We designated this topology by $\text{Topo}_{\text{same}}$.

Several works have studied the case where different flows experience almost different round trip times. Likewise, for the comparison of PSAND with the others active queue management schemes, we also use a network topology where different flows experience different round trip times. For this purpose, instead of attaching the additional N flows to node S_1, we create N different nodes from node S_4 to S_N and N links with different propagation delays. Each of these nodes is attached to node R_1 through a different link. The propagation delays for the N additional flows vary linearly from $1ms$ to $200ms$. The propagation delay of the i^{th} link is then computed as follows :

$$\text{propagation delay}(i) \;=\; 1 + \frac{199(i-4)}{N-4}\, ms \;.$$

We designate this topology by $\text{Topo}_{\text{diff}}$. Figure 4.1 illustrates this topology.

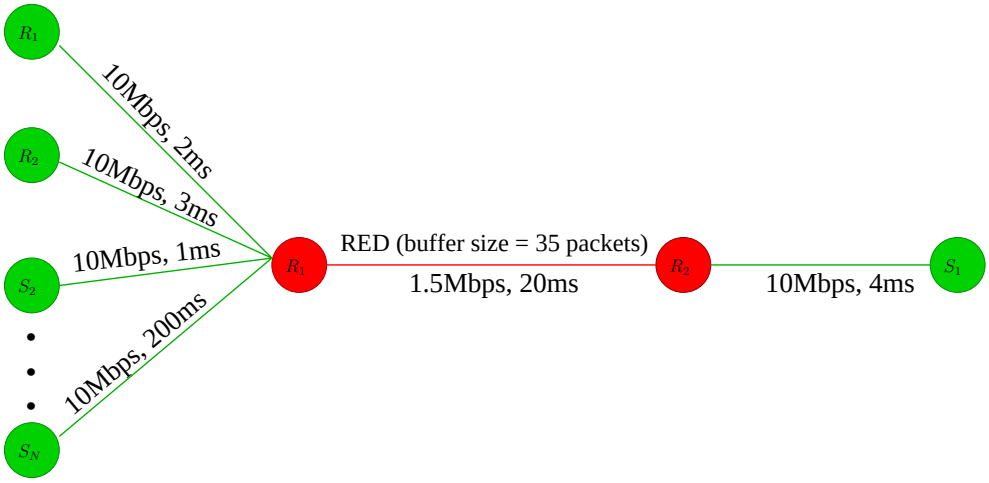


Figure 4.1: Network topology with different Round Trip Times (RTT).

4.3 Network parameters

Among the active queue management schemes selected for comparison in this chapter, AVQ, PI and LRED require as an input to their model:

- The maximum round trip time (RTT) for the considered network topology denoted by R^+.

- The minimum number of the TCP connections denoted by N^-.

In all of these three works, N^- is a given data which is considered known by the network operator. Note that other works like [82, 36, 60] have advocated some methods to estimate the number of active connections. On the other hand, R^+ is estimated as follows:

$$R^+ = \frac{B}{C} + T_p \ , \tag{4.1}$$

where B is the buffer capacity of the bottleneck link in packets, C is the bottleneck link capacity in $packets/s$ and T_p is the maximum propagation delay (in seconds). T_p is the sum of the propagation delays of the links located between two nodes. To obtain T_p, we make the sum of the propagation delays of the links located between two nodes and multiply it by two for the forward and backward paths. T_p takes the value of the maximum sum obtained.

For instance, for the topology $\text{Topo}_{\text{same}}$, $T_p = 2(3+20+4) = 54ms$ and for the topology $\text{Topo}_{\text{diff}}$, $T_p = 2(200 + 20 + 4) = 448ms$.

Table 4.1: Maximum round trip time R^+ according to the considered topology

Topology	TCP packet size	Link capacity (C)	Propagation delay (T_p)	R^+
$\text{Topo}_{\text{same}}$	1000 bytes	1.5 $Mbps$	54 ms	241 ms
$\text{Topo}_{\text{diff}}$	1000 bytes	1.5 $Mbps$	448 ms	634 ms

Once T_p is calculated, given the bottleneck link capacity $C = 1.5Mbps = 1.5 \times 10^6/(8 \times 10^3) = 187.5 packets/s$ where 10^3 is a packet size in bytes, and given the RED buffer capacity $B = 35 packets$, the value of R^+ is computed using equation (4.1). The result of this computation is presented in Table 4.1.

Once the values of N^- and R^+ are determined, the AVQ, PI and LRED schemes use these values in order to derive the system stability condition. Each of these mechanism determines the values of a certain variable for which the system becomes stable.

For AVQ, Equation (1.18) page 44 gives the values of the parameter α for which the system is stable. In the same manner, for PI, Equations (1.22) and (1.23) page 47 give the values of the parameters a and b. For LRED Equation (1.26) page 49 gives the value of the parameter β. Hence, for these values of the considered parameter, for a round trip time less than R^+ and for a number of TCP connections greater than N^-, the system is said to be stable. For the simulations conducted in this chapter, we use the values of R^+ presented in Table 4.1.

For the values of N^-, we have considered two alternatives for AVQ and LRED. For the first one, we set N^- to 3 flows since it is the smallest number of flows we consider. For the second one, we use the actual and considered number of flows as N^-. We have denoted these versions as AVQ* and LRED*. This way we can observe the performance improvement obtained if the exact number of flows is known. This case is very difficult to obtain in practice since the exact number of active flows is not known from the RED node. It is rather practical for the network operator to assume that there is a minimum number of TCP connections in the network and use this number as N^-.

Hence, we use the following values of the parameters used for each active queue management scheme:

- For BLUE: we use the values recommended by Feng et al. in [33] for the parameters d_1 and d_2, *i.e.* $d_1 = 0.0025$ and $d_2 = 0.00025$. The parameter $freeze\_time$ is set to 0.241, so that it corresponds to a typical RTT of the considered topology. For the topology $\text{Topo}_{\text{same}}$ this corresponds to $241ms$ according to Table 4.1.

- For PI: the values of the parameters a and b are set as $a = 0.000344519$ and $b = 0.000343499$. The interval of adaptation of PI parameters $INTERVAL$ (or $1/f_s$) is set to 170 seconds.

- For REM: we use the values recommended by Athuraliya et al. in [7, 8] for the parameters γ and ϕ, *i.e.* $\gamma = 0.001$ and $\phi = 1.1$.

- For AVQ: the size of the virtual queue $\widetilde{B}$ is set equal to the size of the real queue (*i.e.* 35 packets) and the desired link utilization rate is set to 1. Based on (1.18) and (1.19), we set the "optimal" value of α for every number of flows for AVQ*, and $\alpha = 0.251$ for AVQ.

- For LRED: based on (1.26), we set the "optimal" value of β for every number of flows for LRED*, and $\beta = 0.021$ for LRED.

Note that for all the results of PSAND presented subsequently unlike AVQ*, LRED* and PI, we do not give a favorable configuration for PSAND. Indeed, we do not give an "optimal" setting of the parameters $coef$ and γ according to the number of flows. We rather set their values to $coef = 1.75$ and $\gamma = 1.5$, a configuration that gives an appropriate performance whatever the number of flows as we have shown in Chapter 2.

4.4 Metrics

In this chapter we use the three major metrics presented in Chapter 2 such as the queue size variance, the average queue size and the packet loss rate. We have seen that PSAND has an advantage over ARED concerning these metrics. In order to investigate if PSAND maintains its advantages concerning these three metrics without showing weakness for other metrics, and also for a better comparison of PSAND with other active queue management schemes, we measure the following additional metrics:

- The loss run length.

- The link utilization rate.

- The fairness between the TCP flows.

The loss run length as defined in [92] is the number of lost consecutive packets between two not lost packets. We consider a per-flow loss run length. For the per-flow loss run length, we count the number of lost consecutive packets belonging to the same flow.

The link utilization rate is computed as the total number of bytes delivered by the link during the running time of the simulation over the total possible bytes that could have left the link in the same running time.

Fairness is a metric widely used to evaluate TCP congestion control algorithms when two or more TCP connections compete for scarce resources in a congested router.

There is fairness when there is a fair allocation of resources among several TCP connections that share a gateway which has a lower capacity than the total demand of the TCP connections in terms of resources. The fairness issue can be raised for bandwidth sharing among TCP flows or between TCP and non TCP flows (e.g. UDP traffic). In this chapter we only consider the fairness metric between TCP flows.

The fairness problem for RED gateways has been studied in litterature. The authors of [49] stated that the fairness performance under RED gateways depends on the parameter Max_p. Hence according to the authors, Max_p should be associated to the delay-bandwidth product of a connection. The FB-RED algorithm described in [63] and the FRED algorithm described in [70] are per-flow management schemes that intended to improve fairness between TCP flows for RED gateways. The class-based algorithms such as Balanced RED (BRED) [5, 86], stochastic fair Blue (SFB) [35] and [86] are intended to solve the unfairness problem between TCP and non TCP flows.

There exist several definitions of fairness [42, 64, 101]. These studies showed the difference that may exist between long-term fairness and short-term fairness in some systems. A congestion control algorithm can show a fair allocation of resources in a larger time-scale while showing unfairness in a smaller time-scale.

The most known definition is Jain's fairness index [57] that is used to determine the fairness among all TCP flows. This fairness index takes a value ranging from 0 to 1. A value close to 0 indicates poor fairness whereas a value close to 1 indicates a higher fairness between flows. Look at references [58, 38] for further details.

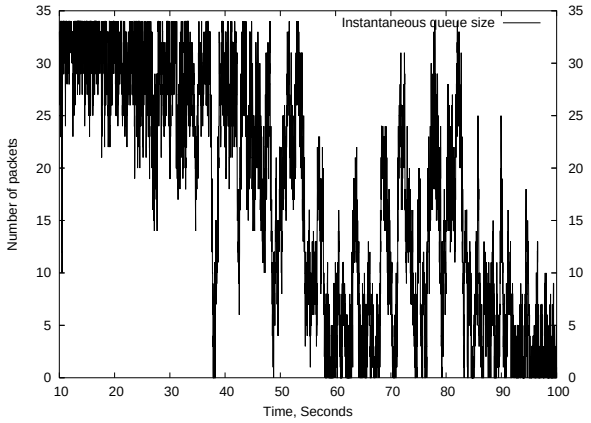
Given x_i the network resource (e.g. throughput, buffer space, etc.) allocation for flow i, and N the total number of TCP flows, the Jain's fairness index F is computed as follows:

$$F(x_1, x_2, \ldots, x_N) \;=\; \frac{\left(\sum_{i=1}^{N} x_i\right)^2}{N \sum_{i=1}^{N} x_i^2} \;.$$

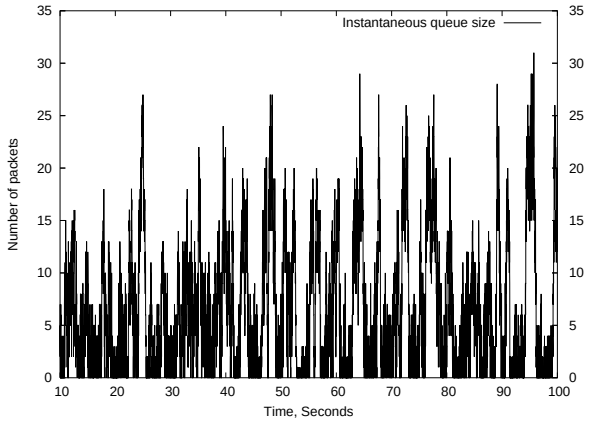
In this chapter, we use Jain's fairness index to measure the degree of similarity between all flows bandwidth given x_i the throughput of flow i. We only consider the fairness problem between TCP flows.

4.5 Qualitative analysis

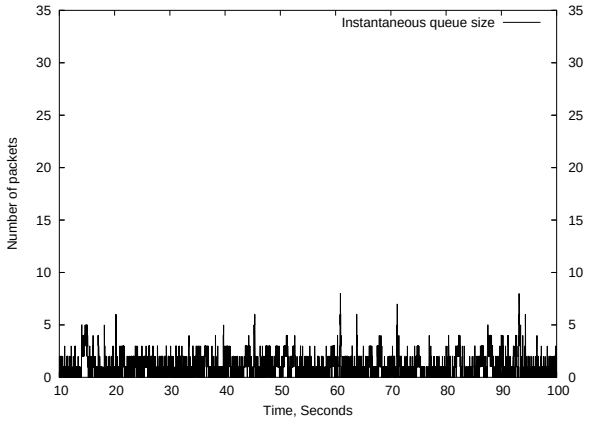
In this section, we make a visual and qualitative comparison of the evolution of the instantaneous queue size of schemes like BLUE, REM, AVQ, PI, LRED AND PSAND through



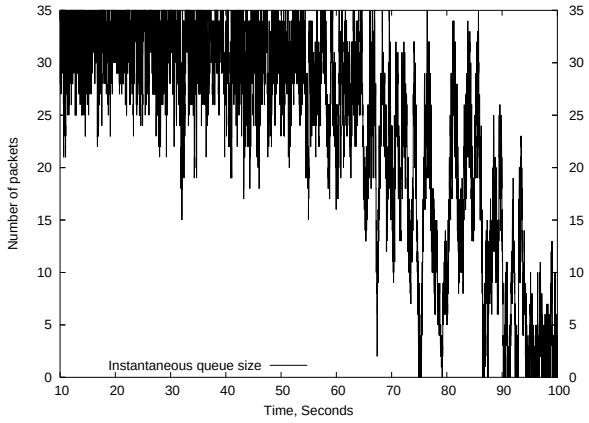
(a) BLUE



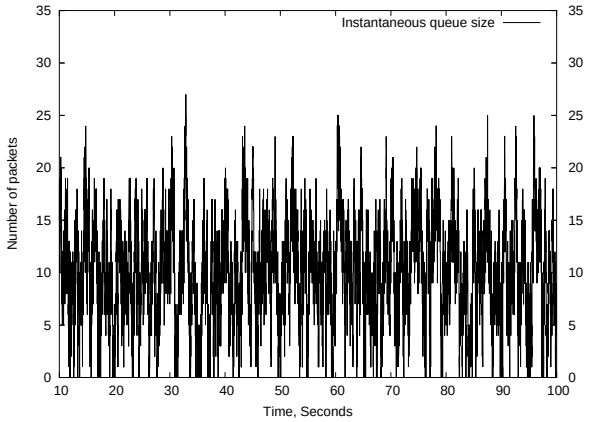
(b) REM



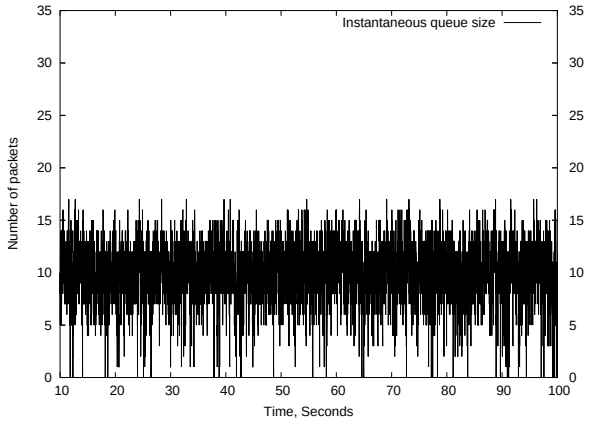
(c) AVQ*



(d) PI



(e) LRED



(f) LRED*

Figure 4.2: Comparison of different active queue management schemes for a constant traffic for 103 sources (same RTT)

Figures 4.2, 4.3 and 4.4. This curves presented in these figures are each the result of one simulation run for 100 seconds. In Figures 4.2 to 4.3 we only observe the behavior of every mechanism in presence of a heavy traffic load generated by 103 TCP flows. We do not consider a change in the congestion level like in Chapter 2 , Section 2.4.2.1. All the TCP flows start at the beginning of the simulation every 0.1 seconds.

Figure 4.2(a) shows that in presence of a heavy traffic load, BLUE exhibits a full queue for a long period of time and takes 60 seconds to reduce the queue length. The PI scheme (Figure 4.2(d)) shows the same behavior but takes more than 60 seconds to reduce the queue length. After the transient period has ended, both schemes exhibit a higher amplitude of oscillations of the queue length as compared to the other schemes. The other schemes (REM, AVQ*, LRED and PSAND) reduce the queue length almost at the beginning of the simulation (10 seconds after). However these schemes show more frequent occurences of an empty queue than PI and BLUE. Among these schemes, AVQ* shows the smallest amplitude of oscillations of the queue length with a small queue size. Nevertheless, AVQ* experiences more frequent occurences of an empty queue. If we observe Figure 4.2(b), REM also shows a more frequent occurences of an empty queue while experiencing a larger queue size with higher amplitude of oscillations as compared to AVQ*. LRED and LRED* avoid this frequent occurences of empty queue while maintaining the oscillations of the queue length around the target queue size (10 packets). LRED* improves the performance of LRED by reducing these occurrences and giving more concentrated oscillations of the queue length around the target queue size. Recall that LRED* is better tuned than LRED since for the computation of the parameter β that gives the system stability condition by using (1.26), LRED* uses $N^- = 103$ flows whereas LRED uses $N^- = 3$ flows.

If we compare PSAND with all these schemes (see Figure 4.3), we can classify it with the second group (containing REM, AVQ*, LRED and LRED*) where a reduced queue length is observed.

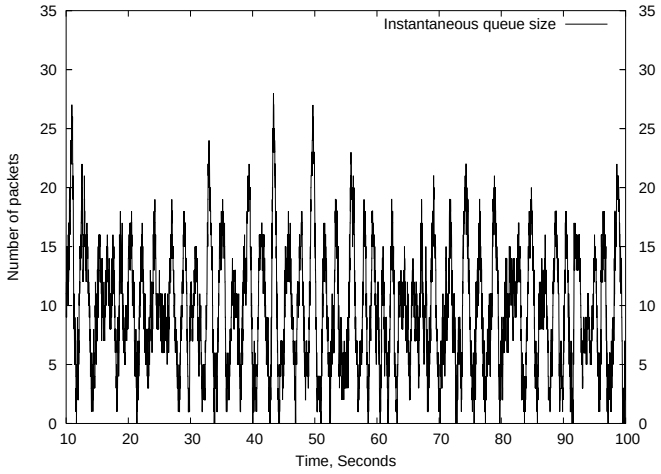


Figure 4.3: PSAND for a constant traffic for 103 sources (same RTT)

Concerning the comparison of PSAND with the first group schemes, the behavior of the queue size oscillations of PSAND outperforms the one for PI and BLUE, except that BLUE and PI utilize more the link resources since PSAND experiences more frequent occurences of an empty queue. Among the schemes of the second group, PSAND shows an advantageous or competitive performance. For instance, as compared to REM, PSAND shows a smaller amplitude of the queue size oscillations, smaller queue length and less frequent occurences of an empty queue. Moreover, the PSAND's queue length oscillates more frequently around the target queue size (10 packets).

If we compare PSAND with AVQ*, AVQ* presents an advantage over PSAND as it experiences a very small queue length much lower than the target queue size and a very small amplitude of the queue size oscillations. Nevertheless, AVQ* exhibits more frequent occurences of an empty queue as compared to PSAND.

The schemes that shows a competitive performance as compared to PSAND is LRED and LRED*. As compared to PSAND, LRED* shows in Figure 4.2(f) a more desirable behavior of the queue size oscillations by reducing the amplitude of the oscillations and making the queue size oscillates very closely to the target queue size. However, it gives more frequent occurrences of an empty queue (see Figure 4.2(e) and Figure 4.3 on pages 127 and 128). Another weakness of LRED as compared to PSAND is that LRED behaves efficiently only in presence of a high traffic load. Figure 4.4 shows that in presence of a light traffic load, LRED gives poor performances as compared to PSAND. Whatever the optimal value of β chosen, LRED does not perform better for small traffic load. We have used $\beta = 0.151$ for the simulation of Figure 4.4.

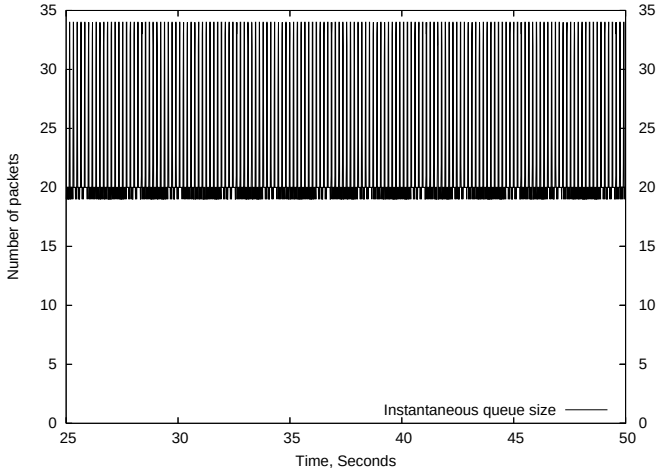


Figure 4.4: LRED queue size evolution for 3 flows (same RTT)

As for PSAND, it works efficiently whatever the traffic load even without an "optimal" configuration of its parameters $coef$ and γ.

Since LRED does not perform efficiently in presence of low traffic load, LRED could not behave better than PSAND (see Figure 2.6(d) page 82), in case of a change of the

congestion level. The ability to adapt quickly to a congestion level is an important characteristic since realistic network traffics are dynamic and do not always present a large traffic load.

From the observation of all these figures (Figures 4.2 to 4.3), we can conclude that PSAND shows a desirable behavior of the queue size oscillations as compared to the other schemes. Even competitive schemes like LRED can show a weakness as compared to PSAND. We will support this qualitative analysis with the statistical analysis that will be presented in the next section.

4.6 Quantitative analysis

In this section, we perform a statistical analysis reported in Figures 4.5 to 4.9. Every point of the curves in these figures is an average of 100 independent simulations, each one run for 100 seconds. We first consider in Section 4.6.1 statistical measures by using the topology $\text{Topo}_{\text{same}}$. In Section 4.6.2, we use the topology $\text{Topo}_{\text{diff}}$.

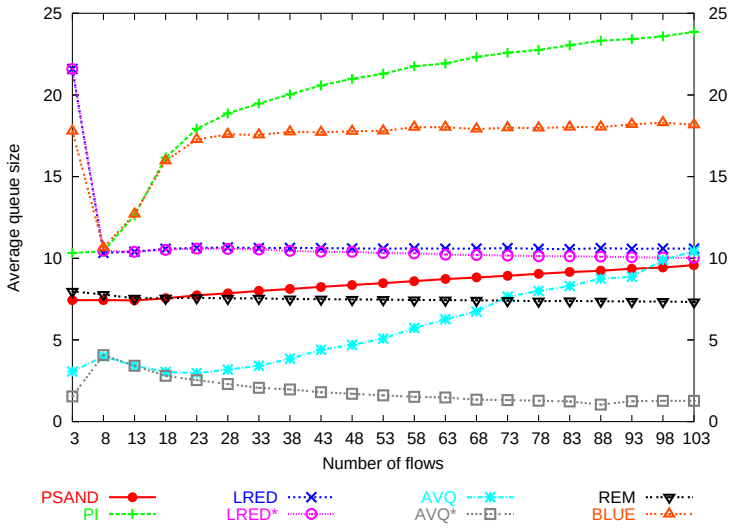
4.6.1 Similar round trip times

Figures 4.5 to 4.6 present the results of the measurements in a case where all TCP flows experience similar roud trip times.

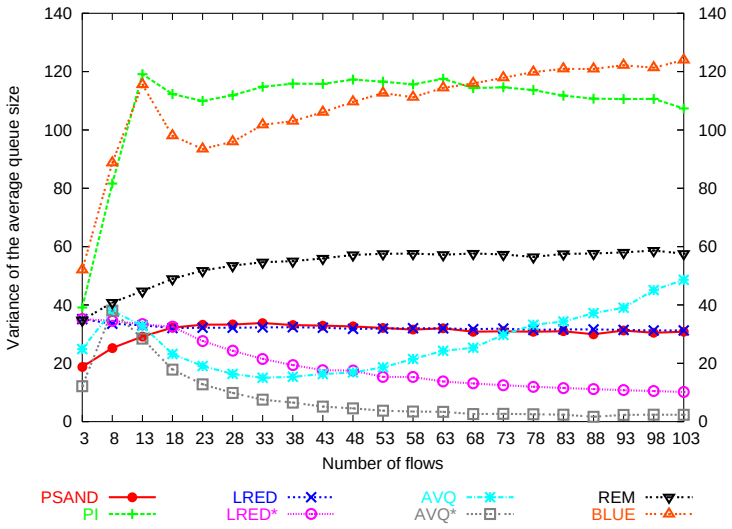
These results confirm the observations of the qualitative analysis. These figures illustrate that PSAND outperforms BLUE and PI for all the metrics we have considered. As compared to the other schemes, PSAND does not always show the best performance for all the metrics considered. Even if PSAND exhibits a less advantageous or competitive performance for a certain metric, it appears more interesting for other metrics. To clarify these observations, let us compare PSAND with each of these schemes, *i.e.* REM, AVQ, AVQ*, LRED and LRED*.

4.6.1.1 PSAND versus REM

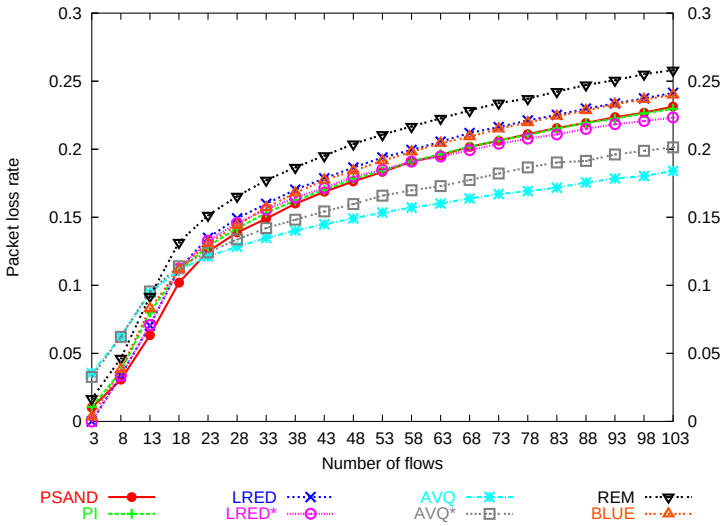
REM is close to PSAND in its design spirit. Like PSAND, REM uses the queue length mismatch, which is similar to *prox_rate* for PSAND but replaces the current average queue size by the previous average queue size. However, their minimal differences leads to significant performance differences. In particular, REM underutilizes the network resources. Indeed, Figure 4.2(b) shows that the queue size is often empty as compared to PSAND and (Figure 4.6(b)) confirms this result. Another performance difference between REM and PSAND is that REM outperforms PSAND if we consider the average queue size. That means that REM gives a fixed average queueing delay much lower than the specified target queueing delay, whatever the traffic load, whereas PSAND increases slowly the queueing delay as the traffic load increases. Our objective being to obtain a target delay, we consider that this advantage alone has little weight. Nevertheless, REM loses its advantages as compared to PSAND when considering the queue size variance and the packet loss rate (see Figure 4.5(b) and 4.5(c)).



(a) Average queue size



(b) Variance of the average queue size



(c) Packet loss rate

Figure 4.5: Evolution of the average queue size, the variance of the average queue size and the packet loss rate for different active queue management schemes (same RTT)

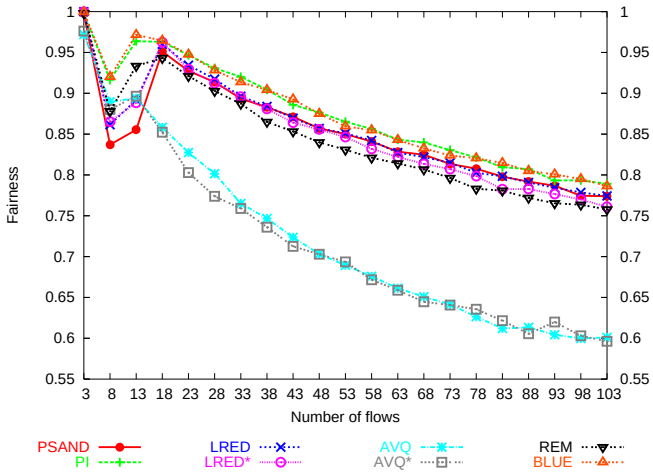
In addition, for other metrics such as the link utilization rate (Figure 4.6(b)), the fairness (Figure 4.6(a)) among TCP flows, PSAND shows better performances. The number of lost consecutive packets is larger for REM than for PSAND.

4.6.1.2 PSAND versus AVQ

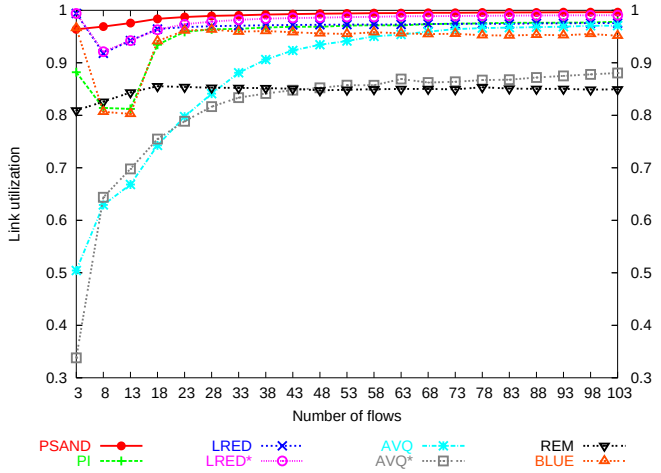
Let us now compare PSAND with the AVQ and AVQ* schemes. Unlike PSAND and REM, AVQ requires as an input the network scenario parameters such as the number of active flows. This difference in design leads to a significant performance difference between PSAND and AVQ. Indeed, AVQ* gives the lowest queueing delay (Figure 4.5(a)) as compared to all the presented schemes. This is achieved while maintaining the lowest delay jitter (Figure 4.5(b)) and the lowest packet loss rate (Figure 4.5(c)). AVQ* improves the performance of AVQ for higher traffic load concerning the queueing delay and the delay jitter with the sacrifice of the packet loss rate. Except for the performance of AVQ concerning the delay jitter in presence of a high traffic load, AVQ and AVQ* outperform PSAND for these three main metrics. If we recall all the objectives that we have first fixed for PSAND, our goal was to minimize the queue size variance while maintaining a low queueing delay close to its target value without the sacrifice of the packet loss rate. In that respect, AVQ and AVQ* are amazing since they are able in addition to reduce the packet loss rate. However, AVQ pays the price of this significant performance improvement as compared to PSAND and the other schemes if we consider other metrics. Indeed, AVQ and AVQ* experience the poorest fairness performance (Figure 4.6(a)), one of the lowest link utilization rate (Figure 4.6(b)) and the highest probability of having an empty queue (Figure 4.6(c)). Note that if we choose a configuration for $coef$ and γ that maintains the queue often empty, PSAND is also able to maintain a very low queueing delay and minimize the queue size variance with a very small additionnal packet loss rate. But in order to avoid the link underutilization and increase the link utilization rate and make PSAND works whatever the number of flows, we have used a configuration that sets the parameters $coef$ and γ to 1.75 and 1.5 respectively and that shows a slight increase of the queueing delay and the queue size variance as compared to AVQ. We can observe in Figure 4.5(b) that AVQ and AVQ* experience a larger queue size variance than PSAND and AVQ shows more queue size variations for larger traffic loads.

4.6.1.3 PSAND versus LRED

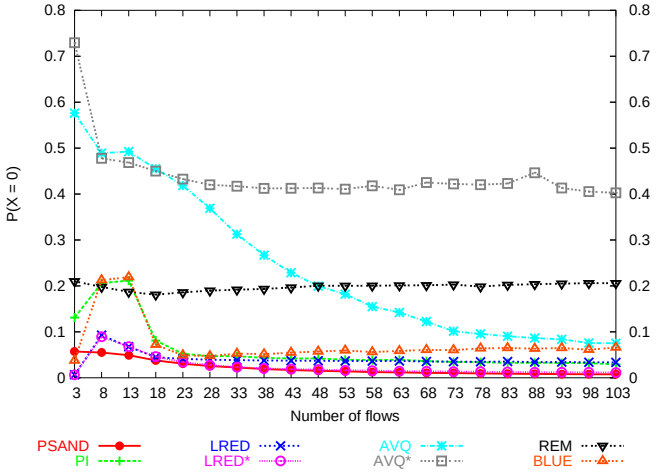
Let us now compare PSAND with LRED which presents similarity to AVQ in the sense that it requires input network parameters. LRED and LRED* are the mechanisms that show more or less similar and competitive performances as PSAND. They give a constant average queue size very close to its target value whatever the traffic load. The average queue size increases slowly with the traffic load for PSAND. Nevertheless, PSAND offers a smaller queueing delay as LRED and LRED*. LRED gives similar results as PSAND concerning the queue size variance but PSAND gives lower variance for small number of flows. LRED* improves the queue size variance as compared to LRED and PSAND with



(a) Fairness

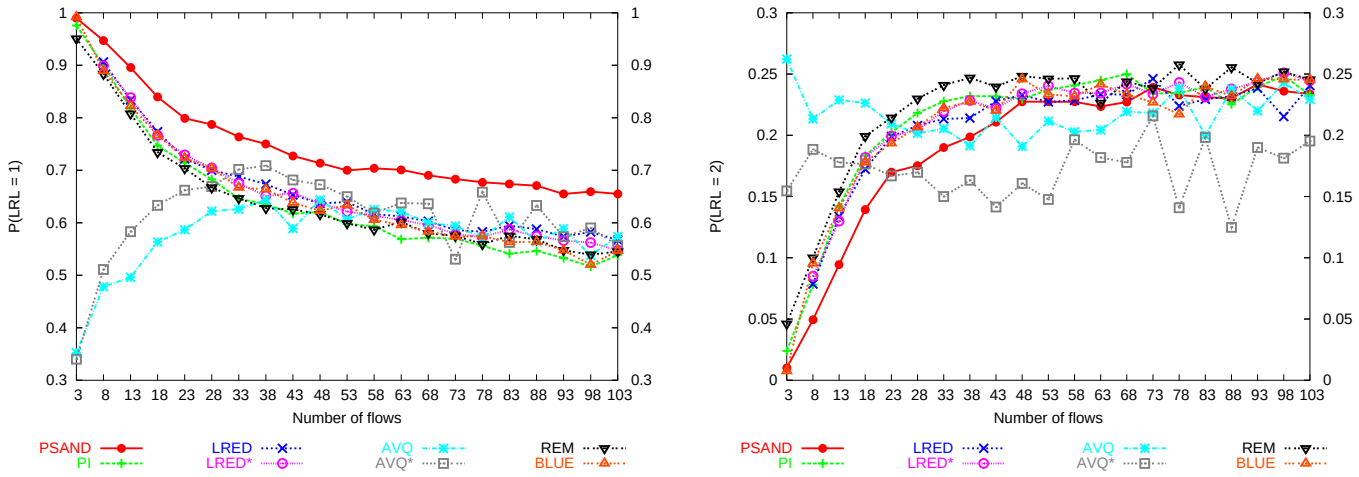


(b) Link utilization



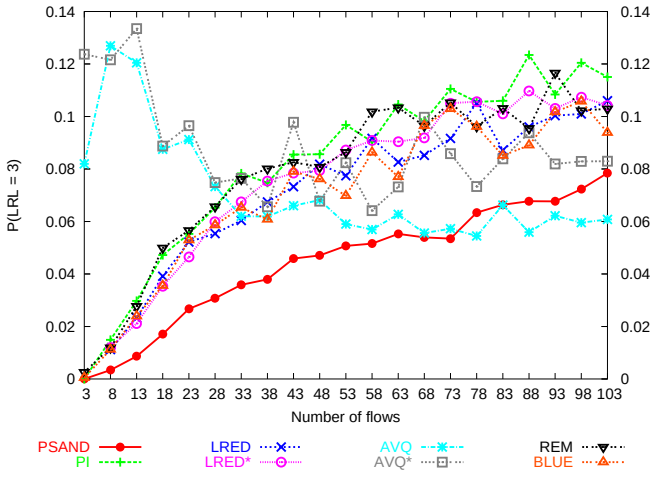
(c) Empty queue

Figure 4.6: Fairness, link utilization and empty queue probability for different active queue management schemes (same RTT)



(a) Per-flow loss run length of size 1

(b) Per-flow loss run length of size 2



(c) Per-flow loss run length of size 3

Figure 4.7: Per-flow consecutive loss for different active queue management schemes (same RTT)

a negligeable decrease of the packet loss rate. Even though LRED shows similar results as PSAND in Figure 4.6(a), PSAND shows a slightly better fairness than LRED for the TCP flows. PSAND shows also an increase of the link utilization rate in particular for small number of flows. Mechanisms like AVQ and LRED need to reduce the TCP window size in order to optimize their performance and thus underutilize the link resources. One of the weaknesses of LRED as compared to PSAND is its poor performance for small traffic load. Another weakness of LRED and LRED* is that they have a larger number of consecutive packet losses. Figures 4.7 shows that all other mechanisms reject packets in the same way

whereas PSAND has a different packet loss behavior since it offers a more isolated loss process. Indeed, and Figures 4.7(a) shows a higher probability for PSAND of having a loss run length (LRL) of size one. There is less probability of lossing consecutive packets with PSAND than with the other schemes. The probability to have a loss run length of size greater than 2 is lower for PSAND. This result is observed for per-flow loss run length metrics.

4.6.1.4 Synthesis

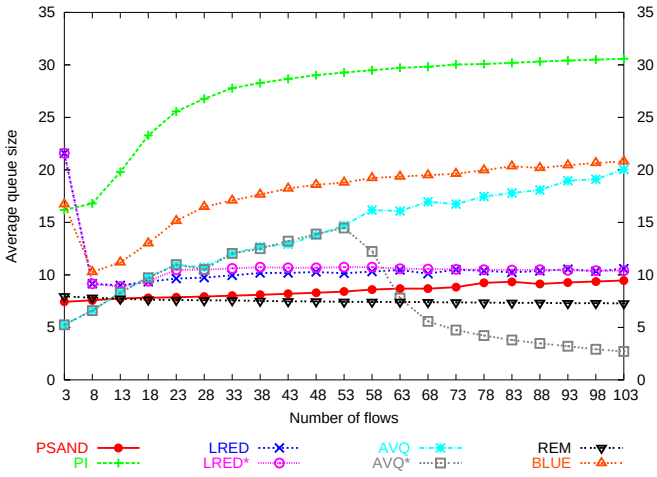
All these quantitative analysis illustrate the performance of PSAND where the TCP flows experience similar round trip times. This situation is favorable for schemes like AVQ, LRED and PI that estimate and use the network parameters such as the RTT as an input to their models. Even in this situation, PSAND showed overall desirable performances. It offers a good trade-off between the performance of all the metrics we presented. Unlike the other schemes, it does not improve significantly one metric and loses a lot its performance for other metrics. It rather tries to perform better for all metrics without searching for the best performance for every metric considered. In addition, this performance comparison has also showed that because of its good trade-off, PSAND did not loose too much in performance. It often presents a performance not far from the best performances obtained for every metric. Table 4.2 confirms these observations. The table gives a subjective evaluation of the results presented in Figures 4.5(a), 4.5(b), 4.5(c), 4.6(a) and 4.6(b). The symbol "+" represents good performance whereas the symbol "-" represents poor performance.

Table 4.2: Table of comparison of different AQM (same RTT): AQS = Average Queue Size, VQS = Variance of Queue Size, PL = Packet Loss Rate, LU = Link Utilization rate, F = Fairness

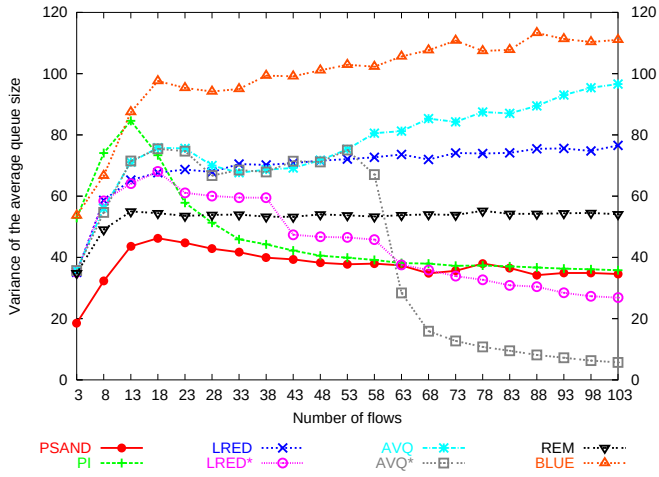
	AQS	VQS	PLR	LU	F
BLUE	---	---	+	++	+++
PI	----	--	++	++	+++
REM	+++	-	-	--	++
AVQ	++	+	++++	+	---
AVQ*	++++	++++	+++	---	---
LRED	+	++	+	+++	++
LRED*	+	+++	++	+++	++
PSAND	++	++	++	++++	++

4.6.2 Different round trip times

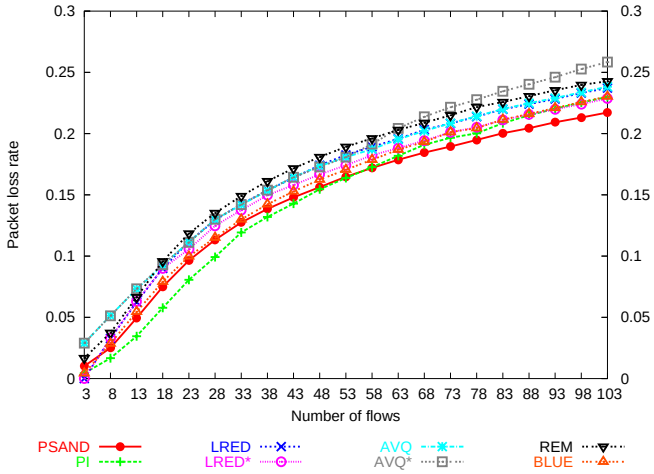
Figures 4.8 to 4.9 illustrate the results obtained by using the topology $\text{Topo}_{\text{diff}}$ where the propagation delays of different links are varied. As expected, the results observed for



(a) Average queue size



(b) Variance of the average queue size



(c) Packet loss rate

Figure 4.8: Evolution of the average queue size, the variance of the average queue size and the packet loss rate for different active queue management schemes (different RTT)

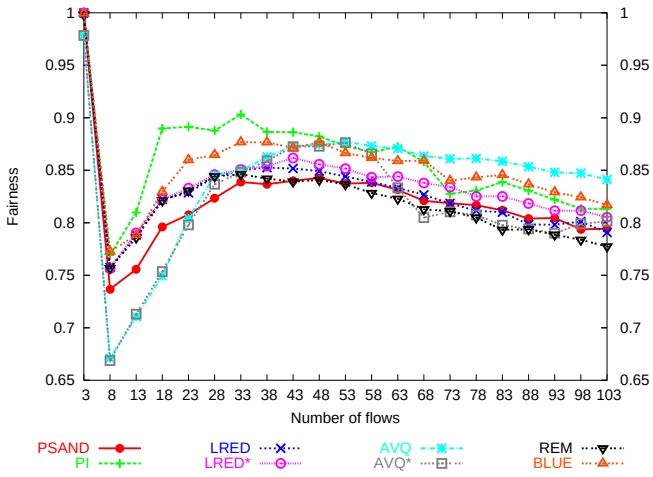
schemes like BLUE, REM and PSAND do not change significantly but on the other hand the performance of network parameter-dependent schemes like PI, AVQ and LRED shows a significant deterioration as compared to the case of $\text{Topo}_{\text{same}}$ described in Section 4.6.1. Hence, since the analysis of the comparison of PSAND with BLUE and REM that we have made in the previous section still holds for this case, we will only compare PSAND with PI, AVQ and LRED. In the previous section, we have seen overall good performances for PSAND as compared to BLUE and REM, which is still the case in this section. On the other hand, as shown by Figures 4.8 and 4.9, PI, AVQ and LRED experience performance deterioration as compared to PSAND in this case where the round trip times are different.

We first compare PSAND with the PI scheme for which the highest queueing delay is observed like in the previous section (Figure 4.8(a)). PI shows a competitive performance towards PSAND for the queue size variance except for small number of flows (Figure 4.8(b)). PI shows small variance for higher traffic load because it often maintains full queues. The PI scheme also experiences the lowest packet loss rate for small number of flows but losses its "podium" for PSAND for large number of flows. Concerning the link utilization rate, PI defeats PSAND by giving the highest rate (Figure 4.9(b)) and the lowest probability of having an empty queue (Figure 4.9(c)). However, PI cannot be considered globally better than PSAND because of its excessive queueing delay and as it does not show better performance as compared to PSAND for the other metrics we considered.

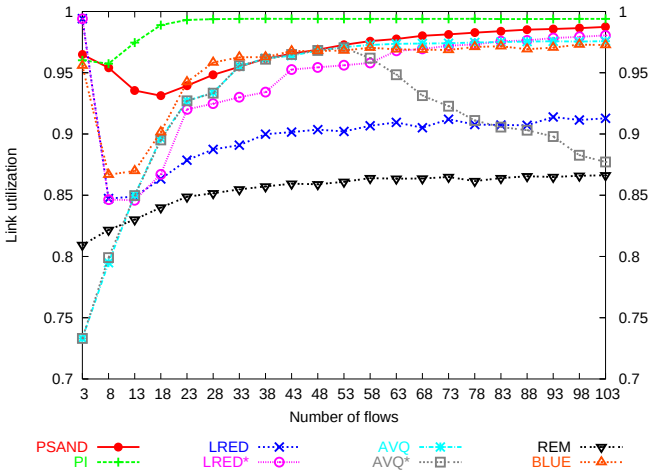
Let us now compare PSAND with AVQ and AVQ*. For the topology $\text{Topo}_{\text{same}}$, AVQ gives the best performance in terms of queueing delay, queue size variance and packet loss rate (Figure 4.5). AVQ looses its best performance when we use the topology $\text{Topo}_{\text{diff}}$ as it increases excessively the queueing delay and the queue size variance. Even AVQ* increases the queueing delay and the queue size variance. Hence PSAND outperforms AVQ and AVQ* for these two metrics (Figures 4.8(a) and 4.8(b)). It is only for larger traffic load that AVQ* decreases the values of these two metrics and gives best results. Nevertheless, this improvement is obtained at the expense of the packet loss rate (Figure 4.8) and the link utilization rate (Figure 4.9(b)) for large traffic load. Note that, in the case of the topology $\text{Topo}_{\text{same}}$, AVQ and AVQ* showed the lowest packet loss rate, which is not the case for this topology. In this case, PSAND gives a lower packet loss rate as AVQ and AVQ*.

In the previous section, AVQ and AVQ* showed the worst performance concerning the fairness metric. In the case of different round trip times, AVQ is as fair as the other schemes. There is no significant performance difference between all the schemes concerning the fairness metric. For all the metrics, PSAND shows better performances than AVQ and AVQ*.

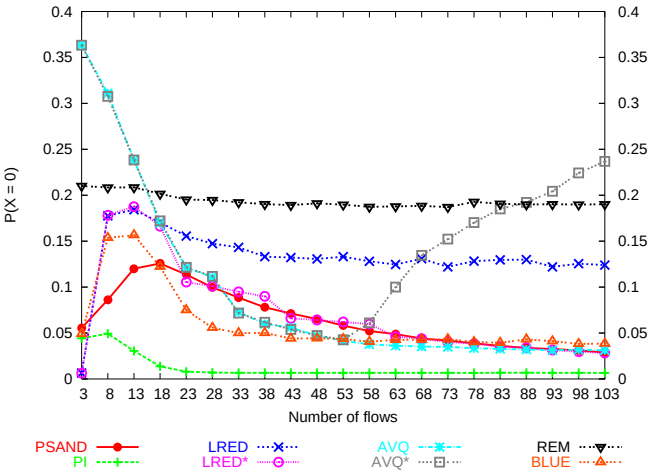
If we consider the comparison of PSAND with LRED, the competitive performance of LRED observed in the previous section only holds for the queuing delay metric in the case of different round trip times. Indeed, LRED shows in Figure 4.8(b) the third largest queue size variance after BLUE and AVQ. Even LRED* exhibits a high variance for small number of flows. It only reduces slightly its variance for a very high traffic load as compared to PSAND. In addition, both LRED and LRED* increase slightly the packet loss rate and



(a) Fairness



(b) Link utilization



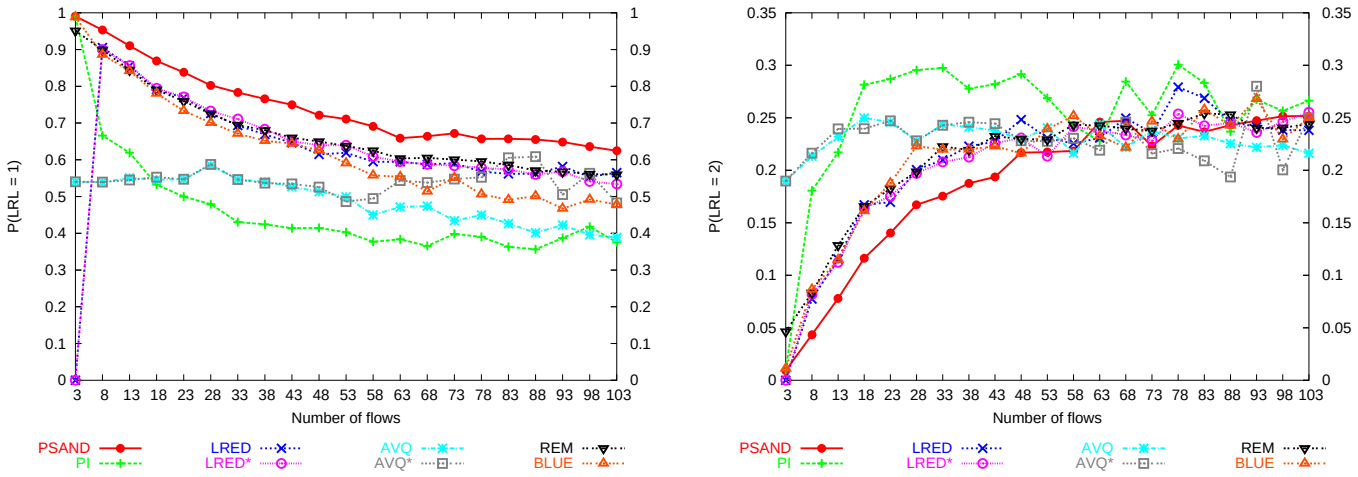
(c) Empty queue

Figure 4.9: Fairness, link utilization and empty queue probability for different active queue management schemes (different RTT)

decrease the link utilization rate as compared to PSAND.

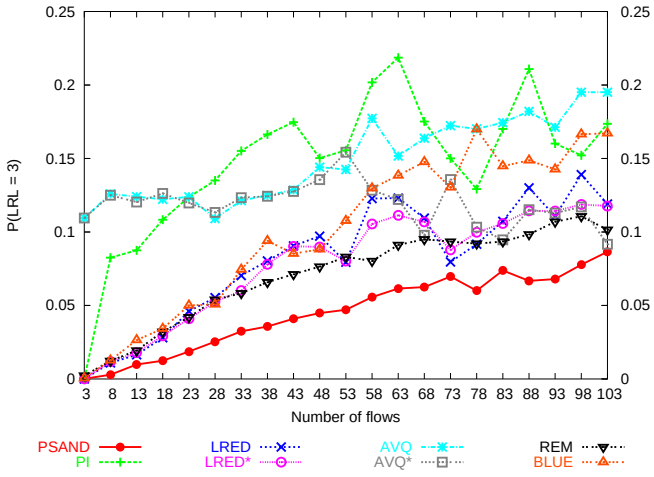
Moreover, like in the previous section, Figure 4.10 also shows that PSAND outperforms all the presented schemes by giving the highest probability to have isolated packet losses and the lowest probability to have more than 2 consecutive packets lost. AVQ and PI show in the contrary a higher probability to have a loss run length of size greater than 2.

Finally, in the case of TCP flows with different round trip times, as showed by all these above comparisons and Table 4.3, PSAND shows best performances for all the considered metrics.



(a) Per-flow loss run length of size 1

(b) Per-flow loss run length of size 2



(c) Per-flow loss run length of size 3

Figure 4.10: Per-flow consecutive loss for different active queue management schemes (different RTT)

4.7 Conclusion

We have compared the performance of our PSAND scheme with the performance of BLUE, REM, PI, AVQ and LRED schemes by using eight metrics presented in Section 4.4. The simulation results are based on two topologies:

- The topology $\text{Topo}_{\text{same}}$ where all TCP flows experience the same round trip time.
- The topology $\text{Topo}_{\text{diff}}$ where all TCP flows experience different round trip times.

Table 4.3: Table of comparison of different AQM ($\neq$ RTTs): AQS = Average Queue Size, VQS = Variance of Queue Size, PL = Packet Loss Rate, LU = Link Utilization rate, F = Fairness

	AQS	VQS	PLR	LU	F
BLUE	--	----	++	++	+++
PI	----	+++	+++	++++	+++
REM	++++	+	-	----	+
AVQ	-	--	+	++	+++
AVQ*	+	++	-	+	+
LRED	++	-	+	--	++
LRED*	++	+++	++	++	++
PSAND	+++	++++	+++	+++	++

These results showed that for the $\text{Topo}_{\text{same}}$, PSAND offers a desirable overall performances with a good tradeoff between the performance of all the metrics we considered. The other schemes can outperform PSAND for a certain metric but loose their advantage for other metrics whereas PSAND gives a robust performance not far from the best performances obtained for every metric. Note that it is in a network configuration with similar RTT, that is in a configuration which is favorable for network parameter-dependent schemes (like PI, AVQ and LRED), that PSAND offers overall good performances than PI, AVQ and LRED. For a network configuration with different RTT ($\text{Topo}_{\text{diff}}$), PSAND shows best performances for each metric considered.

For both configurations, we observed that as compared to the other schemes, the packet loss behavior of PSAND showed a smaller number of consecutive packet drops, that is a more isolated packet loss process. This behavior could be interesting in presence of error correction mechanisms such as FEC (Forward Error Correction). In Chapter 5, we will study the effect of the packet loss behavior of the original RED scheme on the FEC mechanism.

Finally, even though PSAND offers excellent performances, we do not claim that it is the best or the optimal solution for solving RED's parameters configuration problem. This chapter only showed that PSAND is one of the methods that can be used to obtain competitive and even better performances by retaining the RED basic spirit and design, and allowing minimal changes to the RED algorithm without the requirement of the knowledge of network parameters.

Part IV

FEC under RED Queue Management Scheme

Chapter 5

The interaction of forward error correction and active queue management

Contents

5.1 Introduction

The Internet traffic suffers from heavy losses due to network congestion caused by the limited capacity of queue in the routers. There exist two end-to-end error control techniques to repair these losses. The first technique called ARQ (Automatic Repeat reQuest) consists in retransmitting dropped packets upon the destination's request. The second technique, called FEC (Forward Error Correction) consists in sending redundant packets to the destination, allowing it to repair losses without requiring packet retransmission. Because of retransmissions, ARQ is not appropriate for audio and/or video applications requiring strong time constraints. This is why FEC is increasingly used in such applications, typically on top of the UDP transport protocol. Several drawbacks are attached to the use of FEC. First, FEC cannot recover all lost packets. In addition, the transmission of redundant packets increases the overall network load. Finally, the effectiveness of FEC is known to depend on the way packet drops are distributed in the data stream. FEC is more efficient when packets losses are independent, and much less when they occur in groups [25].

In conjunction to end-to-end error control techniques, there exist queue management schemes operating inside routers that control network congestion. We have described in the Introduction, the Drop Tail (DT) scheme traditionally used in the current Internet, which consists in discarding arriving packets when the buffer of the router overflows. We have also seen *RED* an *active* queue management scheme that has been proposed recently as an alternative [16], aimed at eliminating deficiencies of Drop Tail. The *RED* scheme basically discard packets earlier so that incipient stages of congestion can be detected.

The aim of this chapter is to study the interaction of FEC with RED (RED/FEC) and to compare the obtained results with those obtained from the combination of FEC with Drop Tail (DT/FEC). This study has never been conducted to our knowledge. Indeed, FEC has always been studied in presence of the Drop Tail queue management. We believe that as compared to Drop Tail, RED may give performance improvement for the UDP sources implementing FEC since it spreads randomly packet drops reducing consecutive losses for a given flow, thereby making losses "more independent". This property makes RED compatible *a priori* with the use of FEC at the packet level.It may therefore be interesting to add a "moderate" amount of redundancy for the FEC flow in presence of RED queue management in order to decrease the packet loss rate for the UDP source as compared to Drop Tail. Of course, this must be done without penalizing the TCP flows. Indeed, in case of an increase of redundancy, *i.e.* an increase of the network load, the TCP sources respond by decreasing their throughput whereas the UDP flow does not.

In Section 5.2, we briefly present the principles of the FEC coding scheme and of queue management algorithms. In Section 5.3, we describe the topology of the system and the performance metrics considered in this chapter. The performance measures obtained by simulation are detailed and analyzed in Section 5.4. Section 5.5 presents a simple model with which we explain the tradeoff responsible for the fact that sometimes RED/FEC is more efficient, sometimes DT/FEC. Section 5.6 presents conclusions of this chapter.

5.2 Forward error correction and queue management

We first recall some properties of codes used for Forward Error Correction. Given k data packets bearing the relevant information, the encoder (based for instance on a Reed-Solomon Erasure code [77, 91]) generates h redundant packets useful for the recovery of the lost data packets. The concatenation of the k data packets and the h redundancy packets is called a *FEC block* of size $n = k + h$. If these k data packets are transmitted without loss to the destination, it is not useful to recover the possibly lost redundant packets as they do not contain relevant data. It is only in case of loss of data packets that packet recovery is required. If the total number of lost (data and redundant) packets is at most h, the decoder at the destination can retrieve successfully all lost packets. As a result all the relevant information is saved. Otherwise, if the total loss exceeds h packets, it is impossible to recover the lost packets (refer to Figure 5.1).

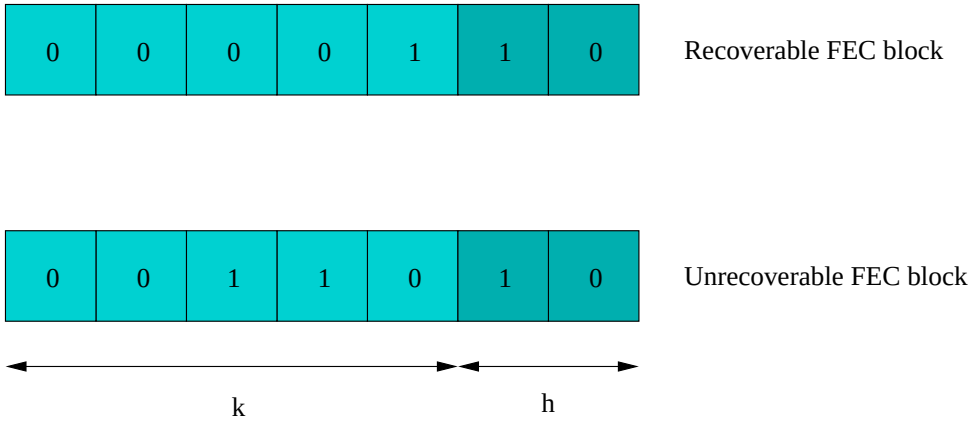


Figure 5.1: FEC block of size $n = k + h$ (1 stands for a lost packet, 0 otherwise)

The queue management schemes studied in this chapter are Drop Tail, the principle of which is straightforward, and RED. With RED, the router maintains an estimate of the average queue length, using an exponential moving average. Based on this value, it accepts or rejects incoming packets with a certain probability. The rejection probability function is a parameter of the mechanism. We have used in the following experiments the default values for RED parameters (see [43, 40] for details).

5.3 Experimental setup

We describe in this section the experimental setup we have used for this performance study. We have made extensive simulations using the `ns-2` simulator [94]. We begin with the network elements used in the simulation, and proceed with the definition of the performance metrics we have collected.

5.3.1 Network topology

The network setup is depicted in Figure 5.2. The traffic generated by nodes S_0 to S_N is multiplexed on a 10 $Mbps$ bottleneck link between nodes R_1 and R_2 with a propagation delay of $30ms$. The bottleneck link is provided with a Drop Tail or a RED queue of limited capacity of 35 packets. The other links located between nodes $S_0, \ldots, S_N$ and node R_1 have a capacity of $100Mbps$. These links have different propagation delays uniformly distributed from $20ms$ to $100ms$.

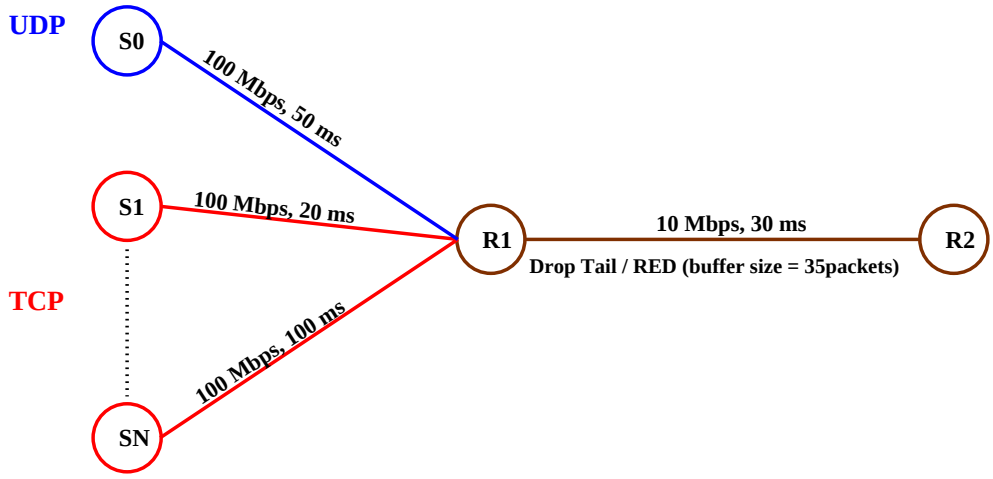


Figure 5.2: Topology of the system

A Poisson distribution is used for the generation of the foreground UDP traffic at node S_0. Node S_1 to S_N generate background long-lived FTP traffics using TCP/Sack1 agents. The offered load of the UDP traffic in the absence of redundancy is set to $\rho_1 = 500kb/s$. Without redundancy the load generated by the UDP traffic represents 5% of the bandwidth of the bottleneck link. For the generation of redundant packets, we increase the throughput ρ_1 of the UDP/FEC flow by a factor of $1 + h/k$, in order to take into account the addition of redundancy while keeping constant the rate of information generated by the source. The resulting new load ρ_{FEC} is equal to:

$$\rho_{FEC} = \rho_1 \left(1 + \frac{h}{k}\right) . \tag{5.1}$$

Under `ns` this increase of the load is obtained by decreasing the average idle time between packets. This parameter is computed as follows :

$$\text{average idle\_time\_} = \frac{\text{UDP packet size}}{500 \times (1 + h/k)} ,$$

where the packet size for the UDP traffic is set to 573 bytes following the recommendation of [19]. TCP packet sizes are set to 1200 bytes and the maximum TCP window size is set to 20 packets.

Every simulation is run for 100 seconds and statistics are collected every $10ms$ from the queue located between node R_1 and R_2. The results presented below are averages over 50 independent simulations.

5.3.2 Performance metrics

We considered the aggregate traffic performance as well as the individual flow performance. The metrics used for the aggregate traffic are :

- The *average queueing delay* inferred by the average of the instantaneous queue size. It is interesting to minimize queueing delay for applications with strong real-time constraints.

- The *delay jitter* corresponding to the variance of the instantaneous queue size. Minimizing delay jitter is useful for applications such as audio and video.

- The *aggregate throughput*, that is the amount of bytes of information correctly transmitted to the destination per unit time, regardless of the flow identity.

The metrics used for a specific flow (that is for the FEC flow) are :

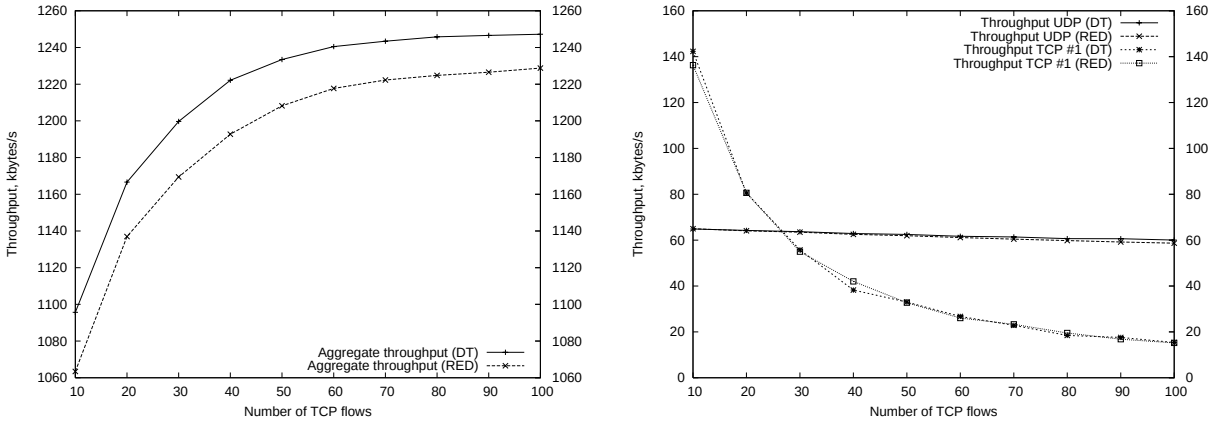
- The *packet loss rate before correction* ($PLRBC$) that is the ratio of the average number of lost packets in a FEC block before correction to the size of the FEC block.

- The *packet loss rate after correction* (PLR) that is the ratio of the average number of lost packets in a FEC block after correction to the size of the FEC block.

- The *loss run length* [92] that is the number of packets of a particular flow that are lost consecutively. This is a random variable which gives an insight into the packet loss process. Studying this metric allows to investigate which queue management is more efficient when used with FEC.

5.4 Performance measures

This section presents the results of our experiments. We first look at the influence of the cross traffic, represented by the number of TCP flows, on the efficiency of RED/FEC and DT/FEC (Section 5.4.1). We check in Section 5.4.1.3 that the theoretical performance advantages provided by RED are preserved in our experiments. Measurements on the loss run length are presented in Section 5.4.1.4. Next, we study in Section 5.4.2 the influence of the amount of redundancy on the performance.



(a) Aggregate throughput

(b) Per-flow throughput

Figure 5.3: Throughput as a function of the cross traffic for $k = 16$ and $h = 1$

5.4.1 Influence of the number of TCP flows

5.4.1.1 Throughput

As shown by Figure 5.3(a), Drop Tail offers a higher throughput than RED (a difference of about $20kbytes/s$ for 100 TCP sources) whatever the number of TCP flows. The increase of the offered load due to the addition of redundancy ($h \leq 4$) for the UDP flow does not affect significantly the overall throughput.

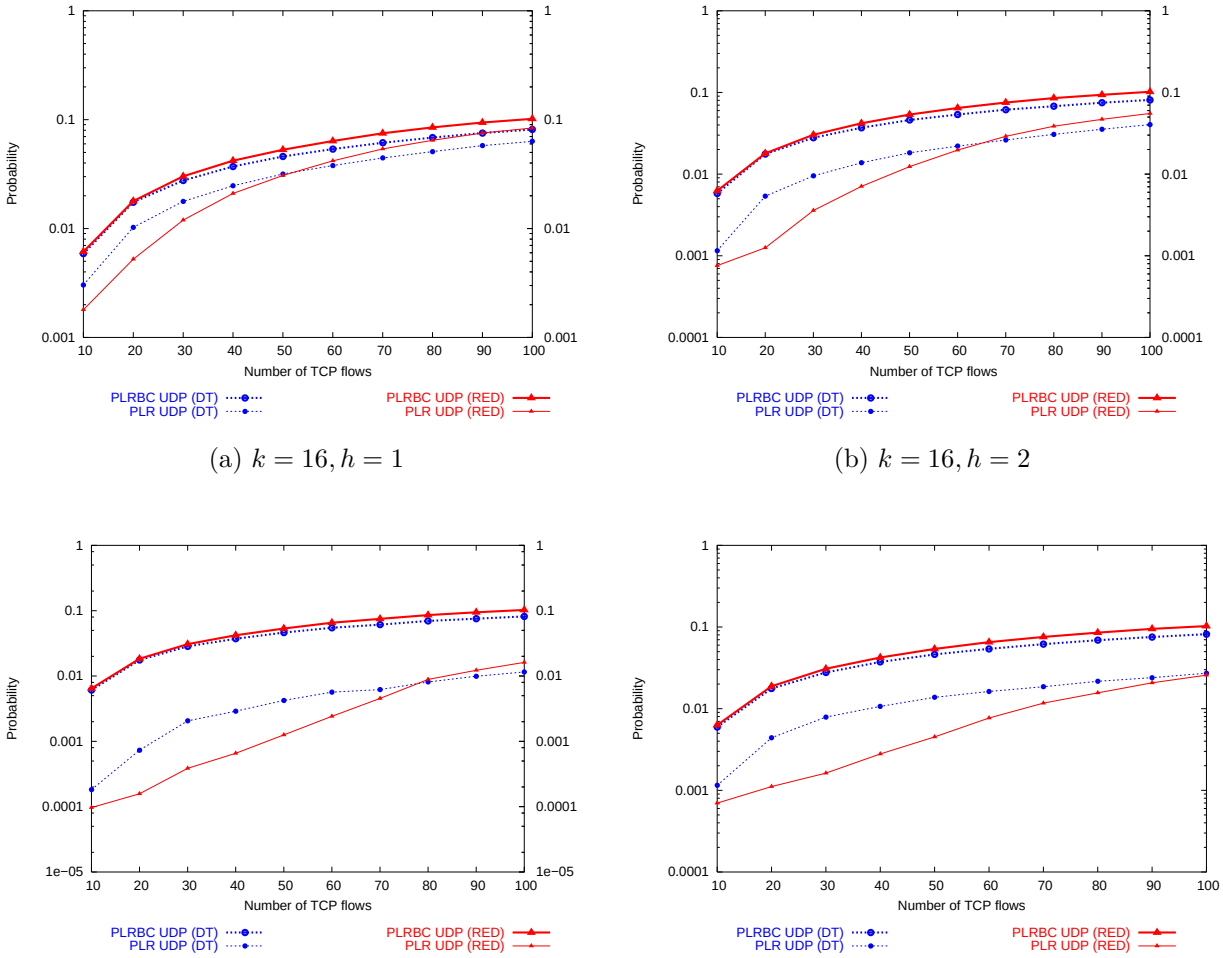
Figure 5.3(b) also shows that whatever the type of buffer management used, the UDP flow remains almost insensitive to the increase of the number of TCP flows since we observe a negligible decrease of the throughput. On the other hand, the TCP flows are constrained to reduce their throughput (from $140kbytes/s$ each for 10 TCP flows to $20kbytes/s$ each for 100 TCP flows).

5.4.1.2 Packet loss rate for the UDP source

Figure 5.4 shows the evolution of the packet loss rate before correction (PLRBC) and after correction (PLR) for the UDP source as a function of the number of TCP flows and the number of redundancy h. Tables 5.1 and 5.2 present a 99% confidence interval for the simulation results presented by Figure 5.4.

As observed in [13, 20] and as shown also in Figure 5.4, when the amount of redundancy is increased, the PLR of DT/FEC decreases for this network configuration, *i.e.* for a configuration where the number of sources implementing FEC is smaller than the number of sources not implementing FEC. This observation is maintained for RED for this network configuration.

As expected and as shown by [15], we observe that the PLRBC for RED is greater than the PLRBC for Drop Tail since RED starts dropping packets earlier without reaching the



(a) $k = 16, h = 1$

(b) $k = 16, h = 2$

(c) $k = 16, h = 4$

(d) $k = 8, h = 2$

Figure 5.4: Packet loss rate before (PLRBC) and after (PLR) correction by FEC

buffer capacity of the queue. Nevertheless, after the correction of lost packets, RED/FEC out-performs DT/FEC and gives a lower PLR for a small number of TCP flows. For a larger number of flows, the situation is reversed: RED yields a worst performance.

Indeed, for one packet of redundancy ($h = 1$) and $k = 16$ data packets, *i.e.* for an addition of 6% of load, Drop Tail can divide the PLRBC by about 1.9 for 10 TCP flows. RED does better by dividing the loss rate by about 3.4. In the case of a 25% load increase, for $h = 4$ and $k = 16$, Drop Tail can divide the PLRBC by about 33.4 for 10 TCP flows. In this case, the correction rate of RED is more significant. RED is able to divide the PLRBC by 66.5 for 10 TCP flows and by 79.6 for 30 TCP flows.

Moreover, we observe that the curve of the PLR of RED stands under the curve of Drop Tail for a number of flows less than a certain threshold: 45 flows for $h = 1$ (Figure 5.4(a)), 65 flows for $h = 2$ (Figure 5.4(b)), and 80 flows for $h = 4$ (Figure 5.4(c)). In Figure 5.4(d),

k	h	Flows	$PLRBC_{DT}$	$PLRBC_{RED}$	PLR_{DT}	PLR_{RED}
16	1	10	$.00587 \pm 5.0\,10^{-4}$	$.00613 \pm 4.0\,10^{-4}$	$.00303 \pm 4.9\,10^{-4}$	$.00179 \pm 3.1\,10^{-4}$
		20	$.01748 \pm 7.3\,10^{-4}$	$.01786 \pm 6.7\,10^{-4}$	$.01026 \pm 7.5\,10^{-4}$	$.00523 \pm 5.1\,10^{-4}$
		30	$.02761 \pm 8.6\,10^{-4}$	$.03020 \pm 1.0\,10^{-3}$	$.01776 \pm 9.5\,10^{-4}$	$.01195 \pm 9.4\,10^{-4}$
		40	$.03720 \pm 1.0\,10^{-3}$	$.04204 \pm 1.3\,10^{-3}$	$.02471 \pm 1.0\,10^{-3}$	$.02101 \pm 1.3\,10^{-3}$
		50	$.04593 \pm 1.2\,10^{-3}$	$.05291 \pm 1.3\,10^{-3}$	$.03167 \pm 1.4\,10^{-3}$	$.03074 \pm 1.5\,10^{-3}$
		60	$.05365 \pm 1.2\,10^{-3}$	$.06373 \pm 1.4\,10^{-3}$	$.03788 \pm 1.4\,10^{-3}$	$.04184 \pm 1.7\,10^{-3}$
		70	$.06131 \pm 1.1\,10^{-3}$	$.07505 \pm 1.6\,10^{-3}$	$.04446 \pm 1.2\,10^{-3}$	$.05391 \pm 1.8\,10^{-3}$
		80	$.06850 \pm 1.0\,10^{-3}$	$.08499 \pm 1.4\,10^{-3}$	$.05086 \pm 1.1\,10^{-3}$	$.06476 \pm 1.6\,10^{-3}$
		90	$.07545 \pm 1.4\,10^{-3}$	$.09406 \pm 1.4\,10^{-3}$	$.05771 \pm 1.5\,10^{-3}$	$.07535 \pm 1.7\,10^{-3}$
		100	$.08125 \pm 1.3\,10^{-3}$	$.01018 \pm 1.6\,10^{-3}$	$.06321 \pm 1.7\,10^{-3}$	$.08393 \pm 2.0\,10^{-3}$

Table 5.1: 99% confidence interval for $k = 16$ and $h = 1$

k	h	Flows	$PLRBC_{DT}$	$PLRBC_{RED}$	PLR_{DT}	PLR_{RED}
16	4	10	$.00608 \pm 3.7\,10^{-4}$	$.00645 \pm 3.8\,10^{-4}$	$.00018 \pm 1.9\,10^{-4}$	$.00009 \pm 9.3\,10^{-5}$
		20	$.01757 \pm 7.1\,10^{-4}$	$.01834 \pm 6.3\,10^{-4}$	$.00073 \pm 2.6\,10^{-4}$	$.00015 \pm 1.2\,10^{-4}$
		30	$.02847 \pm 8.2\,10^{-4}$	$.03070 \pm 8.4\,10^{-4}$	$.00205 \pm 5.2\,10^{-4}$	$.00038 \pm 2.4\,10^{-4}$
		40	$.03710 \pm 1.0\,10^{-3}$	$.04200 \pm 8.4\,10^{-4}$	$.00289 \pm 6.2\,10^{-4}$	$.00065 \pm 2.4\,10^{-4}$
		50	$.04622 \pm 1.0\,10^{-3}$	$.05327 \pm 1.0\,10^{-3}$	$.00422 \pm 6.7\,10^{-4}$	$.00125 \pm 3.8\,10^{-4}$
		60	$.05473 \pm 1.0\,10^{-3}$	$.06535 \pm 1.1\,10^{-3}$	$.00566 \pm 8.2\,10^{-4}$	$.00241 \pm 5.2\,10^{-4}$
		70	$.06103 \pm 1.5\,10^{-3}$	$.07484 \pm 1.4\,10^{-3}$	$.00618 \pm 9.0\,10^{-4}$	$.00453 \pm 7.4\,10^{-4}$
		80	$.06959 \pm 1.5\,10^{-3}$	$.08560 \pm 1.4\,10^{-3}$	$.00810 \pm 1.1\,10^{-3}$	$.00885 \pm 1.1\,10^{-3}$
		90	$.07543 \pm 1.6\,10^{-3}$	$.09430 \pm 1.6\,10^{-3}$	$.00990 \pm 1.2\,10^{-3}$	$.01222 \pm 1.0\,10^{-3}$
		100	$.08177 \pm 1.5\,10^{-3}$	$.10267 \pm 1.7\,10^{-3}$	$.01153 \pm 1.1\,10^{-3}$	$.01618 \pm 1.6\,10^{-3}$

Table 5.2: 99% confidence interval for $k = 16$ and $h = 4$

obtained with $k = 8$ and $h = 2$, the threshold is about 100 TCP flows. Above these thresholds, FEC exhibits better performances with Drop Tail than with RED.

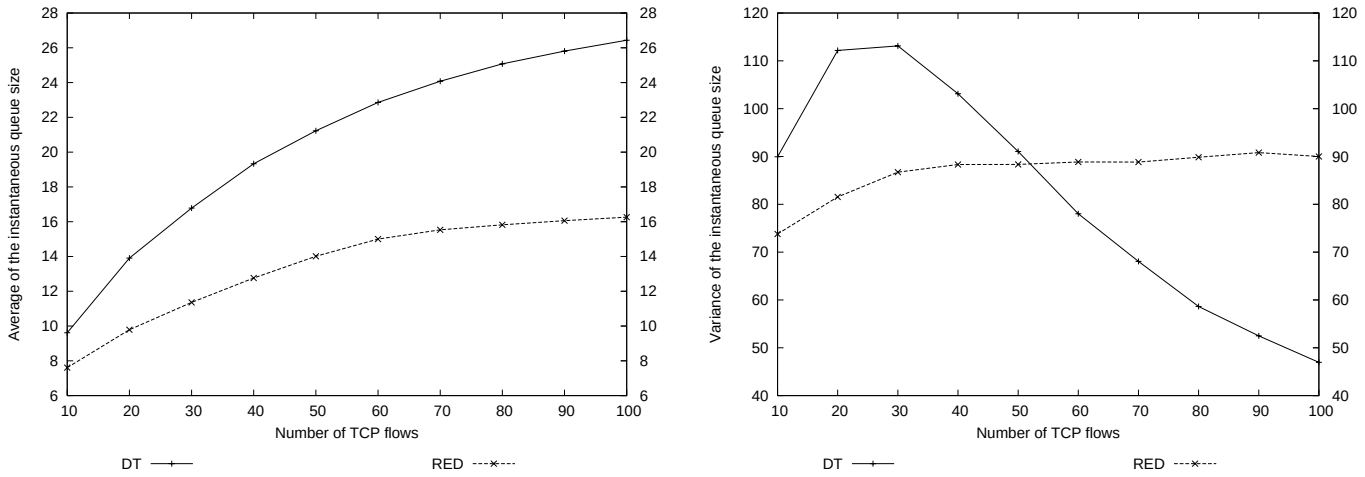
These results show that the number of TCP flows under which RED experiences an improvement on the PLR as compared to Drop Tail depends on the amount of redundancy. This number of flows increases as the number of redundancy packets increases. But the increase becomes slower as h grows.

We have therefore shown that even if RED increases the UDP packet loss rate (which is in accordance with previous studies [15, 75]), it becomes possible to reduce its PLR by using FEC and to obtain a PLR less than the PLR for Drop Tail under certain conditions (precisely for a certain number of TCP flows and a certain amount of redundancy).

5.4.1.3 Queuing delay and delay jitter

The advantage of RED over Drop Tail concerning the queuing delay is maintained with the addition of redundancy. The addition of redundancy does not influence this metric.

Figure 5.5(a) shows that whatever the number of flows, RED improves the queuing delay as compared to Drop Tail. On the other hand, the improvement of RED concerning the delay jitter represented by Figure 5.5(b) depends on the number of TCP flows. Indeed, for small number of flows the queue size variance of RED is lower than the one of Drop Tail. However, for large number of flows as the queue is always almost full we observe for Drop Tail a lower variance. Note that for few number of flows, FEC in conjunction with RED manages to reduce the PLR while exhibiting a lower queuing delay and delay jitter.



(a) Average queue size

(b) Queue size variance

Figure 5.5: Average and variance of the queue size for $k = 16$ and $h = 1$

5.4.1.4 Loss run length

We have displayed in Figure 5.6 the distribution of the loss run length, for both queue management mechanisms and in function of the cross traffic. The different curves show the probabilities of losing consecutively 1, 2 or 3 packets. For this experiment, $k = 16$ and $h = 1$.

This figure confirms that the distributions are different under RED and DT. Under RED, the probability to have a loss run length equal to 1 packet is much larger than for Drop Tail (about 90% against 60%, respectively). This holds whatever the number of flows, since the distribution does not appear to depend much on the cross traffic.

This last observation shows that the situation is not as simple as initially thought. Our starting assumption was: if RED shows a larger probability to have a loss run length of size one, then FEC will perform better with RED than with Drop Tail concerning the capacity of repairing lost packets. This turns out not to be valid for a large cross traffic, although the loss run length of RED is small throughout the range of experiments. We develop in Section 5.5 a model which shows that there is actually an efficiency tradeoff between the *size of bursts* of lost packets, and their *frequency*.

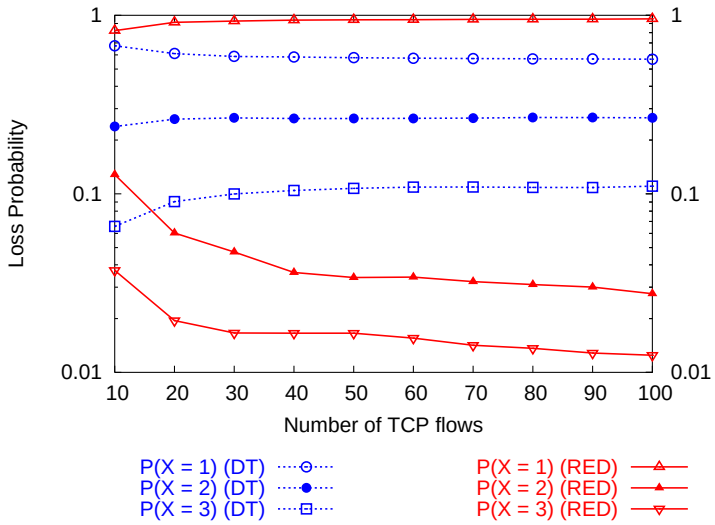


Figure 5.6: Distribution of the loss run length for $k = 16$ and $h = 1$

5.4.2 Influence of redundancy and FEC block size

In this section, we explore the effect of the FEC block size on the performance of RED/FEC or DT/FEC. To begin with, we consider a scenario where the number of data and redundancy packets are varied proportionally so as to maintain the UDP offered load constant. In the second scenario, the number of data packets is changed while keeping constant the number of redundancy, and the UDP offered load varies.

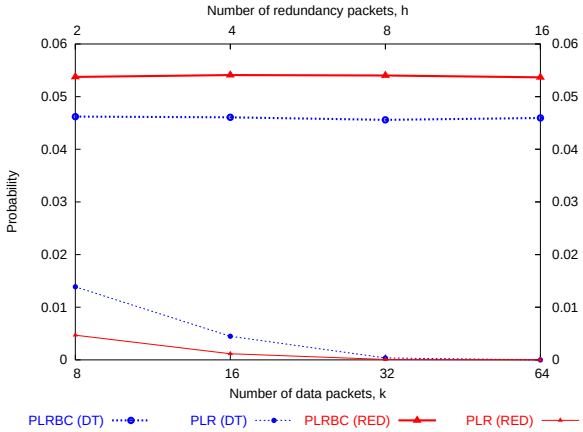
As it turns out, it is interesting to use large FEC block sizes in order to decrease the PLR. However, in practice the block size is limited due to the following constraints: with a larger block, it takes a larger delay for the reception of all packets belonging to the same block by the destination, and it takes more time for the coding (the generation of all redundancy packets) and the decoding of the FEC block (*i.e.* for the correction of all lost packets).

5.4.2.1 Fixed UDP load

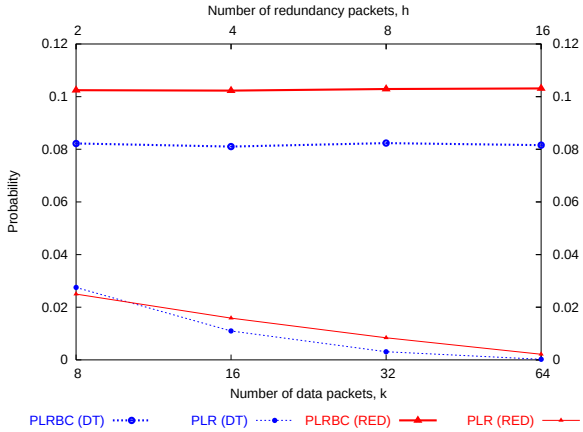
In this section, we varied the size of the FEC block while maintaining the rate $\frac{h}{k}$ constant in order to obtain the same load increase for the UDP flow and therefore maintain the same overall network load. This way we can directly observe the influence of the FEC block size without the interference of the network load.

For instance, for Figures 5.7(a) and 5.7(b), we add 2 packets of redundancy for 8 data packets and proportionally 16 packets of redundancy for 64 data packets in order to obtain a load increase of 25% with respect to the situation without redundancy. For Figures 5.7(c) and 5.7(d), we add 1 packets of redundancy for 10 data packets and proportionally 8 packets of redundancy for 80 data packets in order to obtain a load increase of 10%.

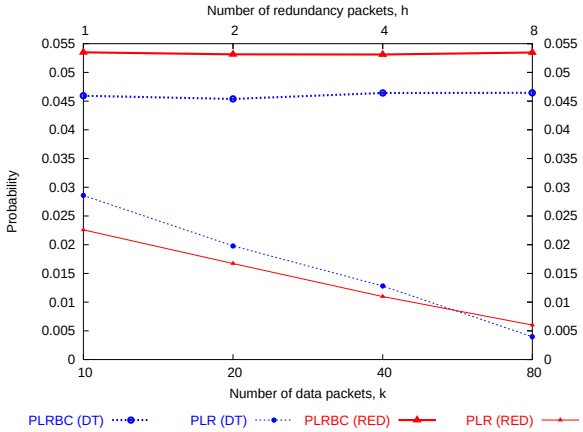
The load of the system being constant, the PLRBC is fixed for both RED and Drop Tail as illustrated by Figure 5.7. However, the PLR decreases when k (and therefore h)



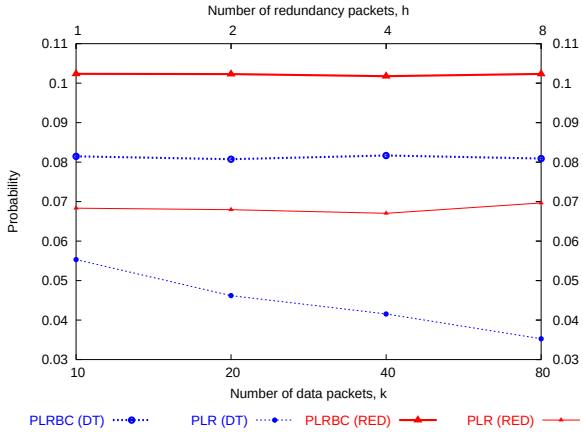
(a) Block size variation with constant rate $\frac{h}{k} = 0.25$ and with 50 TCP sources



(b) Block size variation with constant rate $\frac{h}{k} = 0.25$ and with 100 TCP sources



(c) Block size variation with constant rate $\frac{h}{k} = 0.1$ and with 50 TCP sources



(d) Block size variation with constant rate $\frac{h}{k} = 0.1$ and with 100 TCP sources

Figure 5.7: Influence of the block size

increases for both queue management schemes. This means that for this configuration of the system, it is interesting to increase the FEC block size.

We also notice in Figure 5.7 that RED/FEC appears to be more advantageous than DT/FEC concerning the PLR under certain conditions depending on the number of TCP flows, the FEC block size and the amount of redundancy.

This is the case in Figure 5.7(a), even though the PLRBC of RED/FEC is higher than the one obtained for DT/FEC. Indeed, RED queue management is able to give a slight performance improvement: for instance, for $k = 8$ and $h = 2$, and 50 TCP flows, RED reduces the PLR by almost a factor of 2.9 as compared to Drop Tail. Moreover, the pair of values ($k = 8$, $h = 2$) shows a higher difference between the PLR of RED and the

PLR of Drop Tail than the pair ($k = 16$, $h = 4$). Indeed, if we compare the intersection point between the curves of the PLR of RED and Drop Tail depicted by Figures 5.4(c) and 5.4(d), we note that the configuration with the pair of values ($k = 8$, $h = 2$) gives a better performance than the configuration with the pair ($k = 16$, $h = 4$): this intersection point is obtained for 100 TCP flows for $k = 8$, $h = 2$ instead of 80 flows for $k = 16$, $h = 4$. Hence, in this case, RED appears to be more advantageous than Drop Tail if small sizes of FEC blocks are used. Nevertheless, if the block size is increased then this performance improvement is diminished and both queue management schemes show the same performance by repairing almost all lost packets.

On the other hand, if we increase the number of TCP flows from 50 to 100 (Figure 5.7(b)), Drop Tail is more advantageous than RED. However the PLR difference of both queue management schemes is negligible (about 0.5% for $k = 16$ and $h = 4$) as compared to the difference of their PLRBC (2.1%). This shows that RED has managed to approach Drop Tail performances by repairing an important amount of lost packets. Indeed RED shows a higher correction rate since the absolute difference of the PLRBC and PLR is about 8.6% for RED whereas it is equal to 7% for Drop Tail.

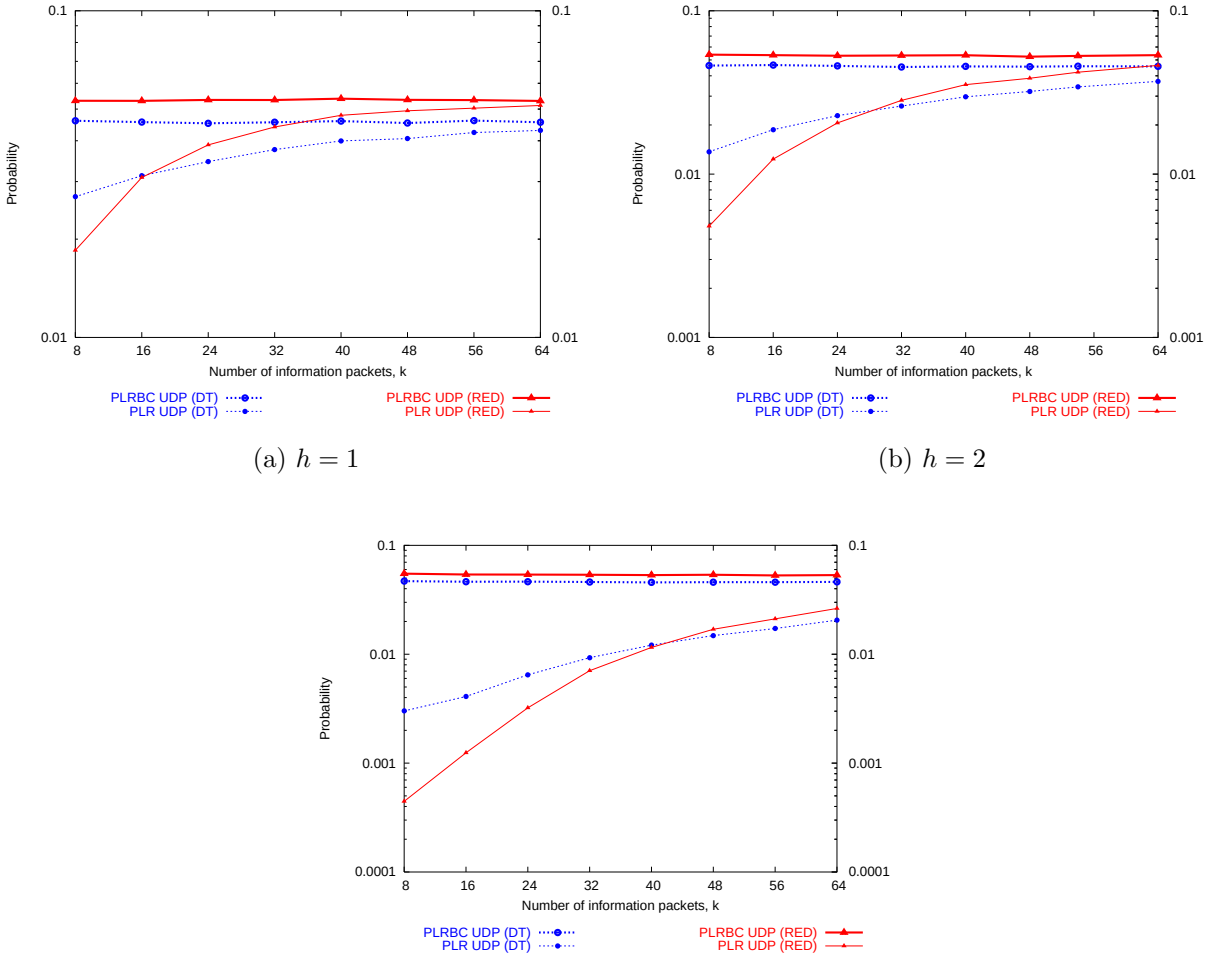
Unlike the case presented in Figure 5.7(b), the difference between both PLR is more significant when we reduce the UDP load increase from 25% to 10% for the same number of TCP flows (100 flows). This is illustrated by Figure 5.7(d). For example, for $k = 20$ and $h = 2$, the absolute difference between the PLR for RED and Drop Tail is about 2.1%. Note that for this case, Drop Tail is far more advantageous than RED. This result can be explained by a higher rejection rate of RED combined with the presence of low redundancy in FEC blocks. In addition, when the FEC blocks are larger, their vulnerability is increased, which leads to worst performances.

But even if the amount of redundancy is not large (10%), Figure 5.7(c) shows that if the number of TCP flows is not important (50 flows), then RED can be advantageous for a certain size of FEC block (approximatively for a block size smaller than a block containing $k = 40$ packets of information and $h = 4$ packets of redundancy).

To summarize, all these results suggest that in case of a constant UDP offered load, the larger the block size, the better the correction rate is for both RED and Drop Tail. It is more interesting to use FEC with RED instead of Drop Tail in case of small number of flows, moderate size of FEC block and a relatively important amount of redundancy. Indeed, our model developed in Section 5.5 confirmed that RED is more efficient if the redundancy ratio h/k is sufficiently large. If the cross traffic is large, RED/FEC loses its advantage over DT/FEC whatever the FEC block size and the amount of redundancy.

5.4.2.2 Variable UDP load increase

We now vary the size of the FEC block by increasing the number of data packets k and by keeping the number of redundancy packets h constant. Therefore, the FEC offered load ρ_{FEC} decreases when the number of data packets k increases (see Equation (5.1)). As expected, the offered load generated by the UDP flow decreases leading to a slight decrease of the PLRBC as illustrated by Figures 5.8 and 5.9. However, unlike the case presented in



(a) $h = 1$

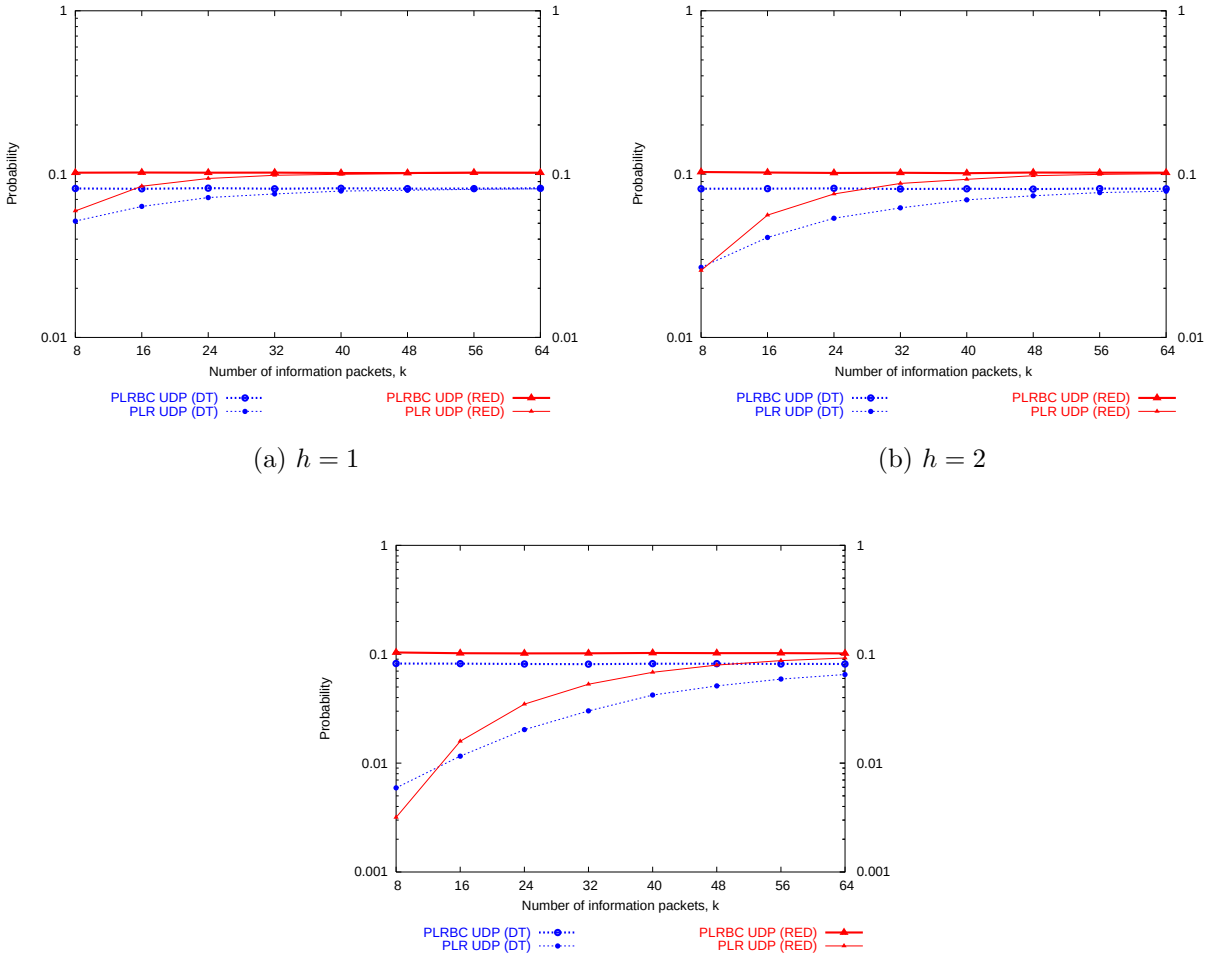
(b) $h = 2$

(c) $h = 4$

Figure 5.8: Packet loss rate before (PLRBC) and after (PLR) correction by FEC for 50 TCP flows

Section 5.4.2.1, the PLR increases with k because as h remains constant and k increases, the FEC block becomes more vulnerable, *i.e.* the probability to repair all lost packets belonging to the same FEC block becomes lower.

Figure 5.8 also shows that by increasing the FEC block size, RED is globally unable to offer an improved performance as compared to Drop Tail. However, for small sizes of FEC block that is for small number of data packets and fixed number of redundancy packets, RED does out-perform Drop Tail. In addition, we have noticed as in Figure 5.4 that there is an intersection point between the curves of the PLR of RED and the PLR of Drop Tail. This point moves when the number of redundancy packets increases (these results are also confirmed by the model developed in Section 5.5). For instance, for $h = 1$ the intersection point is obtained for $k = 16$ which represents a tolerable UDP load increase of about 6.25%.



(a) $h = 1$

(b) $h = 2$

(c) $h = 4$

Figure 5.9: Packet loss rate before (PLRBC) and after (PLR) correction by FEC for 100 TCP flows

On the other hand, for $h = 4$, the intersection point is obtained for approximatively $k = 40$ which represents a higher UDP load increase of about 10%.

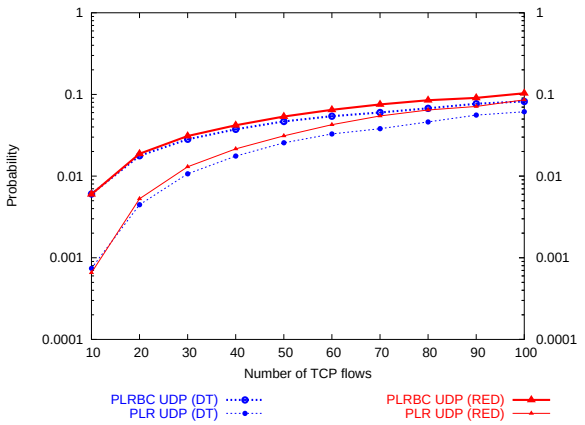
Finally, these results showed that when the FEC block size increases, and the UDP offered load decreases, the PLR for RED and Drop Tail increases since the FEC blocks become more and more vulnerable. The results also show that RED reduces the PLR as compared to Drop Tail for small number of data packets belonging to a FEC block. The number of data packets below which RED out-performs Drop Tail can be increased when the number of redundancy packets is increased. Indeed, since RED drops packets earlier and increases the loss rate, as compared to DT, it needs a sufficient amount of redundancy in order to recover from the excess of losses. As said above, this may not be feasible since the addition of a large amount of redundancy packets increases the UDP source load and

therefore may penalize the TCP sources.

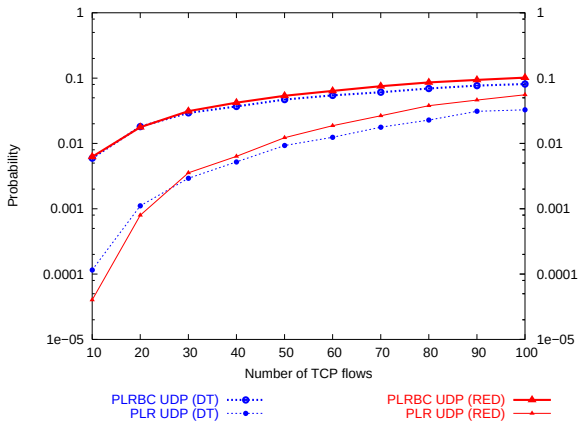
5.4.3 Influence of the number of FEC flows

In the previous section, we investigate the influence of RED buffer management on FEC scheme when only one source implemented FEC. Several works [13, 20] showed that FEC is not efficient when a large number of sources implement FEC. However this observation has been derived only in presence of Drop Tail queue management. In this section, we investigate the bahavior of RED queue management when several sources (8 UDP sources in Figure 5.10) implement FEC.

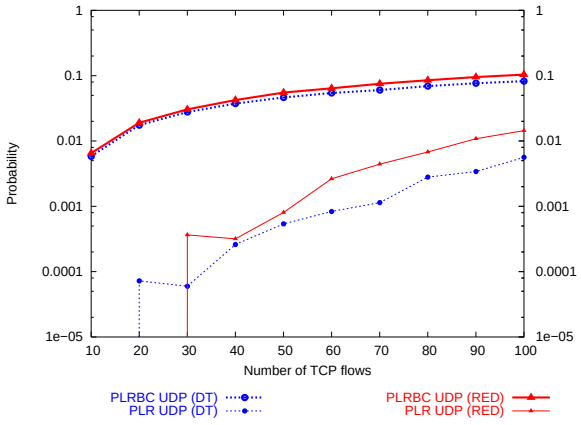
The result of these experiments are presented in Figure 5.10. If we compare these results



(a) $k = 16, h = 1$



(b) $k = 16, h = 2$



(c) $k = 16, h = 4$

Figure 5.10: Packet loss rate before (PLRBC) and after (PLR) correction by FEC in presence of 8 UDP sources

with the results of the experiment using 1 FEC source in Figure 5.4, we can observe that for $k = 16$ and $h = 1$ (Figures 5.4(a) and 5.10(a)), the advantage of RED over Drop Tail is maintained only for a very small number of flows less than 15 for 8 UDP FEC sources as compared to a number of flows less than 50 for one UDP source.

If the amount of redundancy is increased for $k = 16$ and $h = 2$ in Figures 5.4(b) and 5.10(b), the advantage of RED over Drop Tail is maintained for a number of flows less than 30 in case of 8 UDP sources whereas for one UDP source it is maintained for a number of flows less than 70.

Note that the number of TCP flows under which RED is more advantageous than Drop Tail increases highly with h for one FEC source as compared to 8 UDP flows. Indeed, for one UDP source, this number increases from 70 to 80 whereas for 8 UDP sources, it increases from 30 to 40.

In conclusion, when the number of sources implementing FEC increases, RED losses more its advantage over Drop Tail. That is the number of flows under which RED shows a higher correction rate of packet loss as compared to Drop Tail decreases.

5.5 A model for FEC and its application

We develop in this section a simple model which is able to explain most of the phenomena observed above. For this purpose, we number the packets generated by the FEC source and focus on lost packets since this model concentrates on the sequence of packets and among them, the lost packets. The sequence numbers $m_1, m_2, \ldots$ of the lost packets can be viewed as the instants of an "arrival" process.

Definition 1 (A block of losses) *Call a block of losses a set of packets lost consecutively.*

If the size of blocks of losses is small compared with the time between two of these blocks, then by ignoring the time interval (gap) that exits between packets, this discrete loss process with a geometric distribution is approximated to a continuous loss process with an exponential distribution. A *batch* arrival process, where several packets are lost simultaneously is therefore derived. The sequence numbers $m_1, m_2, \ldots$ of the lost packets belonging to the same block of losses can now be viewed as one instant of an "arrival" process. This approximation is illustrated by Figure 5.11.

Accordingly, consider the process where grouped losses occur according to a Poisson process of rate λ. Assume that the size of the m-th group of losses is a random variable $A_m > 0$. The sequence $\{A_m\}_m$ is assumed to be i.i.d.(Identically and Independently Distributed), and we shall use the notation A for the generic random variable. Hence, we have 2 processes, the first process concerns the inter arrival time of blocks of losses and the second one concerns the sizes of the blocks of losses. The resulting process is a compound Poisson process.

We can therefore compute the probability that exactly m blocks of losses occur in the time interval $[0, T]$ by $(\lambda T)^m e^{-\lambda T}/(m!)$.

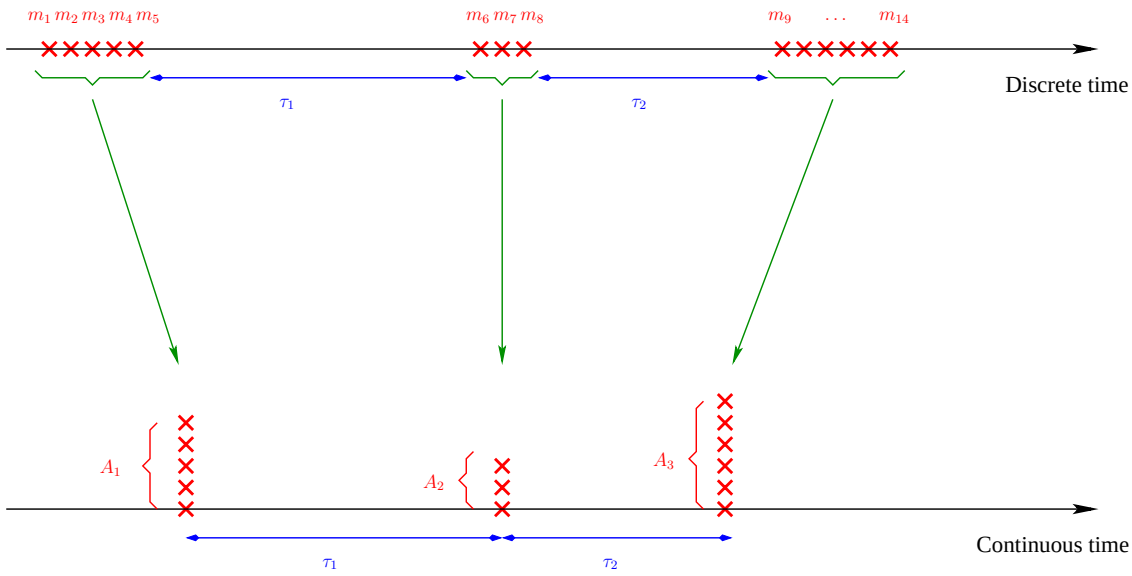


Figure 5.11: Approximating the discrete loss process to a continuous loss process

Let N_T be the distribution of the total number of lost packets in the interval $[0, T]$. We vary the number m of blocks of losses ($m \in [1, \infty]$) and compute the probability to loose h packets in the interval $[0, T]$ as :

$$P(N_T = h) \;=\; \sum_{m=0}^{\infty} \frac{(\lambda T)^m}{m!} e^{-\lambda T} P(A_1 + \ldots + A_m = h) \; . \tag{5.2}$$

The expected number of losses per unit time, which is to be interpreted as a loss rate of packets, is $p = \lambda E(A)$.

Consider now two situations where the distribution of the batch size differs, but where the average loss rate p is the same. Quantities referring to situation i will be superscripted with "(i)". Let $m^{(i)} = E(A^{(i)})$. Since the loss rate is constant, $\lambda^{(i)} = p/m^{(i)}$. Therefore, according to (5.2), we have for each h:

$$P(N_T^{(i)} \leq h) \;=\; \sum_{m=0}^{\infty} \frac{(pT)^m}{(m^{(i)})^m m!} e^{-pT/m^{(i)}} P(A_1^{(i)} + \ldots + A_m^{(i)} \leq h) \; . \tag{5.3}$$

This probability can be seen as a function of $x = pT$, the average number of lost packets in the interval $[0, T]$.

Coming back to the focus of this chapter, we consider that the first situation is RED/FEC and that the second is DT/FEC. We choose a simple distribution for the batch size: $A = 1$ with probability β and $A = 2$ with probability $1-\beta$. The frequency of blocks of losses of size 1 is therefore β. According to the measurements of Section 5.4.1.4 (see Figure 5.6), the situations of RED and DT correspond approximately to values $\beta^{(1)} = 9/10$, and $\beta^{(2)} = 6/10$, respectively.

Assume a certain fixed T and some integer h, interpreted as the length of some FEC block, and the quantity of redundancy it contains, respectively. We are interested in the difference

$$\Delta_h(x) \;=\; P(N_T^{(1)} \leq h) \;-\; P(N_T^{(2)} \leq h)\ ,$$

for $x = pT$, which measures the difference between the probabilities of repairing this FEC block in both situations, when the packet loss rate is p. We have displayed in Figure 5.12 the values of $\Delta_h(x)$ for the first values of h.

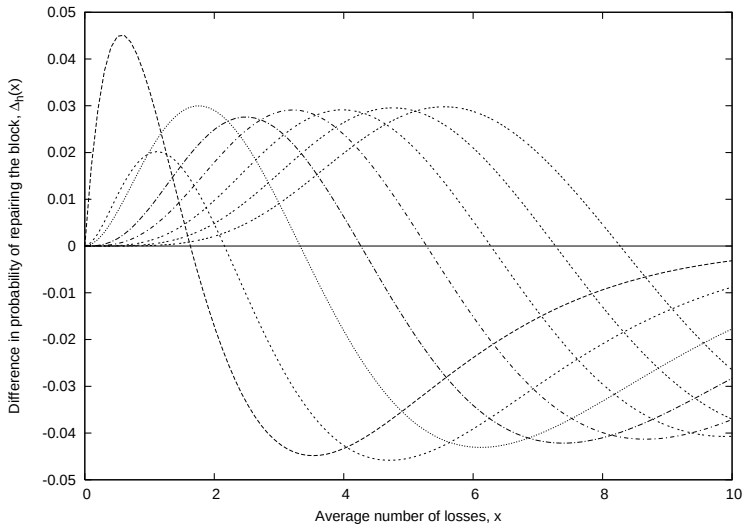


Figure 5.12: Differences $\Delta_h(x)$ for $h = 1 \ldots 8$ (from left to right)

We see that for each h there exists a threshold value x_h such that $\Delta_h(x) \geq 0$ when and only when $x \leq x_h$. This difference is therefore positive if $x = pT$ is small enough (RED/FEC has a better performance), negative if x is large (DT/FEC has a better performance).

The explanation for the shape of the functions Δ_h is found analyzing Equation (5.3). We briefly report the results here. When $x = pT$ is small, (5.3) is dominated by polynomial terms in x, with coefficients $P(A_1^{(i)} + \ldots + A_m^{(i)} \leq h)$. Because $A^{(1)}$ is stochastically larger than $A^{(2)}$, this implies that $P(N_T^{(1)} \leq h)$ is larger. Here, the fact that blocks are smaller is important. On the other hand, if x is large, (5.3) is dominated by the term $e^{-x/m^{(i)}}$. Since $m^{(1)}$ is smaller, $P(N_T^{(1)} \leq h)$ decreases faster and gets smaller than $P(N_T^{(2)} \leq h)$. Here, the fact that blocks of losses are less frequent is more important.

Computing numerically these threshold values further reveals that when h is large enough, $x_h \simeq h + C$ where C is some positive constant value. Using this empirical finding,

and since the size of the block T is $k + h$, we have: RED/FEC is better if

$$p(k+h) \leq h + C \iff k \leq \frac{1-p}{p} h + \frac{C}{p}$$
$$\iff \frac{h}{k} \geq \frac{p}{1-p} - \frac{C}{1-p}\frac{1}{k} .$$

The model allows therefore to predict that:

- for a given quantity of redundancy h, RED/FEC is more efficient only for a block size k small enough;
- RED/FEC is more efficient only if the redundancy ratio h/k is large enough;
- the larger the loss rate p, the smaller the block size k of RED/FEC can be.

All these predictions are confirmed by the experiments reported in Section 5.4.2. We have seen in Figure 5.8 that, h being fixed, RED/FEC performs better for block sizes under a threshold which indeed increases with h. In Figure 5.7 we find situations (b) and (d) where h/k is too small to compensate the loss rate, and (a) and (c) where RED/FEC as the advantage for block sizes small enough. Observe however that in the experiments, the loss rate (or PLRBC) p is not the same for both situations. Moreover, the loss run length distribution we have measured is not exactly that of the model. We believe however that this first simple model reproduces at least qualitatively the principal features of the problem.

5.6 Conclusions

In this chapter, we have studied the effect of a forward error correction (FEC) code on queue management schemes like Drop Tail and RED. We have analyzed the situation where a small UDP traffic using FEC shares a link with a number of TCP sources. It should be noted that to the best of our knowledge, no study has been conducted so far concerning FEC combined with a RED-like active queue management scheme.

It has been shown in literature that RED losses are spread as compared to Drop Tail, *i.e.* RED has a higher probability of having a loss run length of size one. For this reason, one can assume that FEC would be more efficient combined with RED than with Drop Tail. Indeed, the results have shown that even though RED experiences more losses before FEC correction as compared to Drop Tail, RED can be more advantageous concerning the losses after correction. But our results have also shown that RED/FEC does not always perform better than DT/FEC. This turns out to depend on certain parameters, in particular on the number of TCP flows that constitute the background traffic, the FEC block size and the amount of redundancy in a FEC block.

Using the packet loss rate after correction by FEC (PLR) metric, we showed that RED/FEC is more efficient than DT/FEC for small number of TCP flows since a larger

number of TCP flows increases the burstiness of the flows and also the network load, which in turn increases the loss rate for the UDP source implementing FEC. We also observed that if the loss rate before correction (PLRBC) is too high, RED/FEC gives worst performance as DT/FEC since it is unable to repair sufficiently many lost packets whatever the FEC block size and the amount of redundancy packets. The results also show that the number of TCP flows under which RED is advantageous increases with the amount of redundancy added in a FEC block. In addition, we checked that in the situation where RED is advantageous, the performance gains of RED such as queueing delay and delay jitter are maintained. Moreover, our results show that for a fixed amount of redundancy packets in a FEC block, the advantage (respectively the disadvantage) of RED/FEC over DT/FEC is more (respectively less) important for small FEC block sizes than for large FEC block sizes.

In the case where RED/FEC is more advantageous than DT/FEC, the advantage of RED over Drop Tail decreases when the FEC block size becomes too large up to a point where DT/FEC becomes advantageous. This point is reached more or less quickly depending on the relative amount of redundancy contained in a FEC block. In the case where RED/FEC gives worst performance as compared to Drop Tail, the performance difference between RED/FEC and DT/FEC increases with the FEC block size. The performance of RED/FEC degrades with the relative amount of redundancy contained in a FEC block.

All these results suggest that if the UDP flow implementing FEC has the knowledge of its PLRBC and if the traffic generated by this flow crosses a RED gateway, then it is preferable to conform to the following guidelines in order to obtain good performances of RED/FEC. For this purpose, we should be situated in the case where the PLRBC is not too high, that is in a case where the number of TCP flows is small. We can then increase reasonably the relative amount of redundancy in the FEC block without increasing the network load and penalizing TCP flows. In addition, increasing reasonably the FEC block size without increasing the delay of the reception of the block and its the time for coding and decoding improves further the performance of RED/FEC. We noticed that the amount of redundancy and the FEC block size above which RED loses its advantages is large. Since it is not practical to choose such large values, it is therefore preferable to opt for RED/FEC rather than for DT/FEC when the PLRBC is low.

From the point of view of the queue manager, our results suggest the possibility to swap between RED and Drop Tail depending on the scenario parameters. For instance, based on the estimated number of TCP flows or the observed PLRBC, and knowing that FEC is implemented in the UDP flows, the queue manager can make its decision. If the PLRBC is high, it is more advantageous to use Drop Tail. Otherwise, when the PLRBC is low, then RED scheme can be used by following the guidelines described just above.

Our next step will be to refine the model of Section 5.5 in order to investigate further the phenomenon we have observed, in which the correction capacity of FEC is worst when coupled with RED, despite the fact that losses are almost always isolated. First, it is necessary to prove the properties on the threshold values x_h which we have only empirically described. Next, we will investigate whether these properties hold with more general batch size distributions, and can be exploited to obtain decision rules concerning the use of FEC.

Conclusion

Enhancement proposals to queue management schemes are fundamental so that the current TCP/IP algorithm supports the spectacular growth of the number of users and behaves efficiently in presence of the new emerging services required by different applications. That is why the IETF recommended the use of active queue management schemes which still constitutes an active area of research in the Internet. The well-known and most studied active queue management recommended for deployment in the Internet by the IETF is RED. For all these reasons, the RED active queue management scheme is the focus of this thesis.

On the first hand, since the main drawback of RED is the dependence of its performance towards its parameters, this thesis has proposed enhancements to RED's algorithm also presented in [3, 1] with the aim of alleviating this problem and improving the QoS for TCP-based connections. The choice of a TCP-based traffic is motivated as follows. A large amount of the Internet traffic is still based on TCP. In addition, there is tendency to classify traffics to support service differentiation in the next generation Internet and use the appropriate protocol for the class of service. Hence, the study of a uniquely TCP-based flow can be useful. But the main reason to study a TCP based traffic is that RED is originally designed to work with TCP through its interaction with the mechanism of adaptation of the TCP window.

On the second hand, this thesis has studied the RED scheme in presence of the FEC (Forward Error Correction) mechanism with the perspective that if RED is deployed in the Internet, it would be interesting to study its behavior with other alternative protocols to TCP like FEC. To our knowledge, this study has never been conducted earlier.

More specifically, the enhancement of RED we have proposed is based on an adaptive approach of RED (ARED) since we believe that the basic insight for a solution that alleviates RED problems lies in the original adaptive RED proposed in [34] and [41]. Indeed, this approach is a serious proposal for deployment in the current Internet because it adapts dynamically RED parameters without relying on hypotheses on the type of traffic whereas the other approach rely on values of network variables set *a priori* such as the round trip time, the number of connections and the link bandwidth. However, as these variables are not known from RED nodes in practice, it is still difficult to retrieve or infer accurate informations about these variables from local observations.

This thesis focused on the maximum drop probability Max_p parameter since selection of this parameter affects significantly the performance of RED. References [34, 41] had

adapted the value of this parameter by using fixed adjustment factors that do not reflect the effective change rate of the traffic load. This thesis has shown that, using more elaborate dynamic and target-oriented adjustments which are a function of the distance to the performance objective and allowing modifications to the other RED parameters, offers performance improvements on the original ARED. The statistical and qualitative analysis of this new proposal named PSAND showed that as compared to the original adaptive RED:

- PSAND improves the stability of the queue size by reducing the variance of the queue length. PSAND responds more quickly to a sudden change in congestion and shows a better adaptation to the change by bringing back the queue size to a target value.

- PSAND achieves a more predictable and lower average queue size.

PSAND achieves the goal of this thesis which is to minimize the queue size variance with the constraints to deliver a specified average queue size without substantially increasing the loss rate. PSAND did not increase the loss rate and even in some cases (for large number of flows), it decreases it. Moreover, PSAND improves the link utilization rate: it reduces the probability that the queue becomes empty and keeps the queue size away from buffer overflow and buffer underflow.

All these performance improvements are obtained whatever the number of flows showing a more robust performance as the original adaptive RED.

This thesis investigated also the selection of queue size threshold parameters Min_{th} and Max_{th} for the proposed adaptive scheme of Max_p (PSAND). On the first hand, investigations dealing with fixed values of Min_{th} and Max_{th} showed that:

- Small values of Min_{th} give overall performance improvements in queue size variance, queueing delay, loss rate and link utilization rate.

- Sufficiently larger values of Max_{th} around $2\hat{K}_T$ ($\hat{K}_T$ being the target queue size) gives also overall good performances. Max_{th} should not be chosen too large to avoid queueing delays.

These results suggested to use a configuration with Min_{th} set to 0 and Max_{th} set to $2\hat{K}_T$.

On the second hand, this thesis described several adaptation methods of Min_{th} and Max_{th} and investigated if these adaptations can further improve the PSAND performances. The results showed that the previous fixed configuration gives better performance without the need to adapt Min_{th} and Max_{th}.

This thesis has proposed an enhancement to ARED scheme. It also compared this proposal with other well-known or recent schemes such as PI, LRED, AVQ, REM and Blue. The comparison shows that:

- With less complexity, PSAND gives an overall desirable tradeoff and good performances such as small delay and delay jitter, and more robustness. Indeed, other mechanisms can perform better than PSAND for a certain metric but lose their performance for another metric. For a network configuration with variable RTT, PSAND showed overall best performances except for the fairness metric. Moreover, unlike all these schemes, PSAND showed good performances not only for large number of flows but also for small number of flows.

This thesis did not claim that PSAND is the best or the optimal proposition for solving RED problems but rather shows that it is possible to obtain competitive and even better performances by retaining the RED basic spirit and design, and allowing minimal changes to the algorithm. Schemes like PI, REM, AVQ, Blue and LRED have abandoned the basic design of RED, without a substantial or for certain schemes with no improvement as compared to PSAND. Moreover, note that PSAND did not require the values of network parameters set a priori.

The results of this thesis showed also that PSAND can further be improved. We are currently working on a way to choose dynamically some of its parameters, in particular selecting a value of $coef$ and γ that gives best performance. However, we have observed that these improvements depend on a network variable which is the number of active flows. These results show that we have tried to go further to enhance the effectiveness of RED but pushing even further will show the requirement of network variables as an input. Therefore, a future work that exploits a model requiring these variables as an input will improve PSAND.

Another contribution of this thesis is the study of the interaction of FEC with RED. For this study, an UDP flow implementing FEC is multiplexed with multiple TCP flows in a single bottleneck link where a Drop Tail or RED queue management scheme is used. Our intuition was that FEC combined with RED would perform better than FEC combined with Drop Tail since the packet loss behavior of RED showed a smaller number of consecutive packets drops. Indeed, the results showed that this intuition is not always confirmed but rather depends on different parameters like the number of TCP active flows, the amount of redundancy and the FEC block size. Depending on the value of these parameters, sometimes RED, sometimes Drop Tail gives better performance. An analytical model that confirms these results obtained by simulation is also presented.

There exists a myriad of propositions for queue management schemes. We do not claim that RED is better than Drop Tail even though there is no doubt that RED presents in certain situations an obvious performance improvement as compared to Drop Tail. We do not claim either that a typical variant of RED is better than the others but rather believe that the choice of a queue management scheme should be based on the service

requirement of the application and the end-to-end protocol used. Indeed, there is no queue management scheme that offers overall good performances under different network circumstances. Every queue management scheme has its own advantages and shortcomings. In the next generation Internet where service differentiation is expected and where traffic should be classified according to the service requirement, different queue management schemes could be associated with different class of services. Every class of service will be assigned to a queue management that fits better its requirements.

Bibliography

[1] T. Alemu and A. Jean-Marie. Dynamic configuration of RED parameters. In *Proc. GLOBECOM'04*, 2004. Mentioned on page(s) xxvi, 97, 163

[2] T. Alemu and A. Jean-Marie. Étude de la configuration dynamique des paramètres de red. *revue TSI (Technique et Science Informatique)*, 2005. Mentioned on page(s) xxvi

[3] T. Alemu and Alain Jean-Marie. Dynamic configuration of RED parameters. Technical Report 03-042, LIRMM, University of Montpellier II, December 2003. `http://www.lirmm.fr/~tigist/papers/RR01-LIRMM.pdf`. Mentioned on page(s) xxvi, 97, 163

[4] M. Allman, V. Paxson, and W. Stevens. *TCP Congestion Control*, April 1999. RFC 2581. Mentioned on page(s) 11, 12

[5] F. Anjum and L. Tassiulas. Fair bandwidth sharing among adaptive and non-adaptive flows in the Internet. In *Proc. IEEE INFOCOM'99*, New York, NY, March 1999. Mentioned on page(s) 56, 126

[6] S. Athuraliya. A note on parameter values of REM with Reno-like algorithms, March 2002. `http://netlab.caltech.edu/pub/rem.htm`. Mentioned on page(s) 34

[7] S. Athuraliya, D. Lapsley, and S.H. Low. Random early marking for Internet congestion control. In *Proc. GLOBECOM'99*, 1999. Mentioned on page(s) 33, 122, 125

[8] S. Athuraliya, V.H. Li, S.H. Low, and Q. Yin. REM: active queue management. In *IEEE Network Magazine*, volume 15, May/June 2001. Mentioned on page(s) xxi, 33, 34, 35, 122, 125

[9] S. Athuraliya and S. Low. Optimization flow control, II: implementation. Technical report, Melbourne University, May 2000. `http://www.ee.mu.oz.au/staff/slow/research`. Mentioned on page(s) 34, 35

[10] J. Aweya, M. Ouellette, D.Y. Montuno, and A. Chapman. A control theoretic approach to active queue management. *Computer Networks*, 36, 2001. Mentioned on page(s) 68

[11] J. Aweya, M. Ouellette, D.Y. Montuno, and A. Chapman. Enhancing TCP performance with a load-adaptive RED mechanism. *International Journal of Network Management*, 11(1), 2001. Mentioned on page(s) 68

[12] J. Aweya, M. Ouellette, D.Y. Montuno, and A. Chapman. An optimization-oriented view of random early detection. *Computer Communications*, 2001. Mentioned on page(s) 68

[13] E. W. Biersack. A simulation study of forward error correction in ATM networks. *Computer Communication Review*, 22(1):36–47, January 1992. Mentioned on page(s) 148, 157

[14] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7), July 1970. Mentioned on page(s) 53

[15] T. Bonald and M. May. Drop behavior of RED for bursty and smooth traffic. In *Proc. IEEE/IFIP IWQoS'99*, London, June 1999. Mentioned on page(s) 148, 150

[16] B. Braden, D. Clark, 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 in the Internet*, April 1998. RFC 2309. Mentioned on page(s) xx, 17, 26, 144

[17] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. *Resource Reservation protocol (RSVP)*, September 1997. RFC 2205. Mentioned on page(s) 5

[18] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: new techniques for congestion detection and avoidance. In *Proc. SIGCOMM'94*, pages 24–35, August 1994. Mentioned on page(s) 12

[19] C. Brandauer, G. Iannaccone, C. Diot, T. Ziegler, S. Fdida, and M. May. Comparison of tail drop and active queue management performance for bulk-data and web-like Internet traffic. In *ISCC 2001*, Hammamet, Tunisia, July 2001. Mentioned on page(s) 146

[20] Y. Calas and A. Jean-Marie. On the efficiency of forward error correction at the packet level. Technical Report 03-003, LIRMM, University of Montpellier II, March 2003. Mentioned on page(s) 148, 157

[21] N. Cardwell, S. Savage, and T. Anderson. Modeling tcp latency. In *INFOCOMM 2000*, 2000. Mentioned on page(s) 13

[22] V. Cerf. *A partial specification of an international transmission protocol*, 1973. Mentioned on page(s) 8

[23] V. Cerf and J. Postel. *Specification of Internetwork Transmission Control Program - TCP Version 3*. USC/Information Sciences Institute, January 1978. IEN-21. Mentioned on page(s) 3, 8

[24] M. Christiansen, K. Jeffay, D. Ott, and F. Smith. Tuning RED for web traffic. In *Proc. of ACM SIGCOMM'00*, Stockholm, Sweden, August 2000. Mentioned on page(s) xxi

[25] I. Cidon, A. Khamisy, and M. Sidi. Analysis of packet loss processes in high-speed networks. *IEEE Transactions on Information Theory*, 39(1):98–108, January 1993. Mentioned on page(s) 144

[26] D. Clark. Explicit allocation of best effort packet delivery service. Technical report, MIT Laboratory for Computer Science, 1997. Mentioned on page(s) 57

[27] D. Clark and W. Fang. Explicit allocation of best-effort packet delivery service. *IEEE/ACM Transactions on Networking*, 6(4):362–373, August 1998. Mentioned on page(s) 57

[28] Jitendra Padhye's collection of TCP modeling. `http://www.acir.org/padhye/tcp-model.html`. Mentioned on page(s) 13

[29] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *Journal of Internetworking Research and Experience*, pages 3–26, October 1990. Mentioned on page(s) 19

[30] Differentiated service at the IETF DiffServ. Diffserv working group sites, 2004. `http://www.ietf.org/html.charters/diffserv-charter.html`. Mentioned on page(s) 4, 6

[31] W.-C. Feng, D. Kandlur, and D. Saha. Adaptive packet marking for maintaining end to end throughtput in a differentiated service Internet. *IEEE/ACM Transactions on Networking*, pages 685–697, October 1999. Mentioned on page(s) 98

[32] W.-C. Feng, D. Kandlur, D. Saha, and K. Shin. Techniques for eliminating packet loss in congested TCP/IP networks. Technical Report CSE-TR-349-97, University of Michigan, November 1997. Mentioned on page(s) xxi, 68

[33] W.-C. Feng, D. Kandlur, D. Saha, and K. Shin. Blue: a new class of active queue management algorithms. Technical Report UM CSE-TR-387-99, University of Michigan, 1999. Mentioned on page(s) 30, 53, 74, 122, 125

[34] W.-C. Feng, D. Kandlur, D. Saha, and K. Shin. A self-configuring RED gateway. In *Proc. INFOCOM'99*, volume 3, pages 1320–1328, New York, NY, March 1999. Mentioned on page(s) xxi, xxii, xxiii, xxvi, 37, 68, 69, 70, 71, 72, 74, 75, 77, 78, 95, 97, 98, 109, 112, 119, 121, 122, 163

[35] W.-C. Feng, D. Kandlur, D. Saha, and K. Shin. Stochastic fair Blue: a queue management algorithm for enforcing fairness. In *Proc. INFOCOM'01*, pages 1520–1529, Anchorage, Alaska, April 2001. Mentioned on page(s) 52, 53, 126

[36] W.-C. Feng, A. Kapadia, and S. Thulasidasan. GREEN: proactive queue management over a best-effort network. In *Proc. IEEE GLOBECOM'02*, Taipei, Taiwan, November 2002. Mentioned on page(s) 39, 41, 124

[37] V. Firoiu and M. Borden. A study of active queue management for congestion control. In *Proc. INFOCOM'00*, pages 1435–1444, Tel Aviv, Israel, March 2000. Mentioned on page(s) xxi

[38] S. Floyd. Connections with multiple congested gateways in packet-switched networks part 1: one-way traffic. *ACM Computer Communication Review*, 21(5):30–47, October 1991. Mentioned on page(s) 126

[39] S. Floyd. TCP and explicit congestion notification. *ACM Computer Communication Review*, 24:10–23, October 1994. Mentioned on page(s) 17

[40] S. Floyd. Discussions on setting RED parameters, November 1997. http://www.aciri.org/floyd/red.html. Mentioned on page(s) 100, 145

[41] S. Floyd, R. Gummadi, and S. Shenker. Adaptive RED: an algorithm for increasing the robustness of RED's active queue management. Available at `http://www.icir.org/~floyd`, August 2001. Mentioned on page(s) xxi, xxii, xxiii, xxvi, 38, 39, 68, 69, 70, 71, 72, 75, 77, 78, 97, 98, 109, 112, 119, 121, 122, 163

[42] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proc. SIGCOMM'00*, Stockholm, Sweden, August 2000. Mentioned on page(s) 126

[43] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993. Mentioned on page(s) xx, 26, 76, 145

[44] S. Floyd and V. Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4), August 1995. Mentioned on page(s) 21

[45] P. Francis, S. Jamin, C. Jin, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: a global Internet host distance estimation service. *IEEE/ACM Transactions on Networking*, October 2001. Mentioned on page(s) 40

[46] TCP friendly web site. `http://www.psc.edu/networking/tcp_friendly.html`. Mentioned on page(s) 13

[47] Y. Gao and J.C. Hou. A state feedback control approach to stabilizing queues for ECN-enabled TCP flows. In *Proc. INFOCOM'03*, volume 3, pages 2301–2311, San Francisco, CA, April 2003. Mentioned on page(s) 27, 68

[48] S. Golestani. A self-clocked fair queueing scheme for broadband applications. In *Proc. INFOCOM'94*, pages 636–646, Toronto, CA, June 1994. Mentioned on page(s) 21

[49] M. Hassan and R. Jain. *High Performance TCP/IP Networking: Concepts, Issues, and Solutions*. Prentice-Hall, 2004. Mentioned on page(s) 5, 13, 27, 94, 126

[50] C. Hollot, V. Misra, D. Towsley, and W. Gong. Analysis and design of controllers for AQM routers supporting TCP flows. *IEEE Transactions on Automatic Control*, 47(6):945–959, June 2002. Mentioned on page(s) 45, 46, 47, 122

[51] C.V. Hollot, V. Misra, D. Towsley, and W. Gong. A control theoretic analysis of RED. In *Proc. INFOCOM'01*, volume 3, pages 1510–1519, Anchorage, Alaska, April 2001. Mentioned on page(s) xxi, xxiii, 68

[52] C.V. Hollot, V. Misra, D. Towsley, and W. Gong. On designing improved controllers for AQM routers supporting TCP flows. In *Proc. INFOCOM'01*, volume 3, pages 1726–1734, Anchorage, Alaska, April 2001. Mentioned on page(s) xxi, 13, 45, 46, 47, 122

[53] T. Hutschenreuther and A. Schill. Content based discarding in IP router. In *Proc. IC3N'00*, pages 122–126, Las Vegas, Nevada, October 2000. Mentioned on page(s) 55

[54] Integrated Service at the IETF IntServ. Integrated services working group, 2004. `http://www.ietf.org/html.charters/intserv-charter.html`. Mentioned on page(s) 4, 5

[55] V. Jacobson. Congestion avoidance and control. In *Proc. SIGCOMM'88*, pages 314–329, Stanford, CA, August 1988. Mentioned on page(s) 9

[56] V. Jacobson, K. Nichols, and K. Poduri. RED in a different light. Technical report, Cisco system, September 1999. Mentioned on page(s) 75

[57] R. Jain. *The art of computer systems performance analysis*. John Wiley and Sons, 1991. Mentioned on page(s) 126

[58] R. Jain, D.M. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for ressource allocation in shared systems. Technical Report DEC TR-301, MA: Digital Equipment Corporation, 1984. Mentioned on page(s) 126

[59] Juniper Networks. *Supporting differentiated service classes: queue scheduling disciplines*, December 2001. White Paper. Mentioned on page(s) 5, 20

[60] A. Kapadia, W.-C. Feng, and R. Campbell. GREEN: a TCP equation-based approach to active queue management. Technical report, University of Illinois, February 2004. Mentioned on page(s) 39, 41, 55, 124

[61] P. Karn and C. Partridge. Round trip time estimation. In *Proc. SIGCOMM'87*, August 1987. Mentioned on page(s) 9

[62] K. Kilkki. *Differentiated services for the Internet*. MacMillan Technical Publishing, 1999. Mentioned on page(s) 4, 6

[63] W.-J. Kim and B.G. Lee. The FB-RED algorithm for TCP over ATM. In *Proc. GLOBECOM'98*, pages 551–555, Sydney, Australia, November 1998. Mentioned on page(s) 55, 126

[64] C.E. Koksal, H. Kassab, and H. Balakrishnan. An analysis of short-term fairness in wireless media access protocols. In *Proc. SIGMETRICS'00*, June 2000. Mentioned on page(s) 126

[65] V.P. Kumar, T.V. Lakshman, and D. Stiliadis. Beyond best effort: router architectures for tomororow's Internet. *IEEE Communications Magazine*, 1998. Mentioned on page(s) 4, 5

[66] S. Kunniyur and R. Srikant. Analysis and design of an adaptive virtual queue (AVQ) algorithm for active queue management. In *Proc. SIGCOMM'01*, pages 123–134, San Diego, CA, August 2001. Mentioned on page(s) xxi, 42, 44, 68, 122

[67] S. Kunniyur and R. Srikant. A time scale decomposition approach to adaptive ECN marking. In *Proc. INFOCOM'01*, Anchorage, Alaska, April 2001. Mentioned on page(s) 42, 44, 122

[68] S. Kunniyur and R. Srikant. An adaptive virtual queue (AVQ) algorithm for active queue management. *IEEE Transactions on Networking*, pages 286–299, April 2004. Mentioned on page(s) 42, 44, 122

[69] T.V. Lakshman, A. Neidhardt, and T.J. Ott. The drop from front strategy in TCP over ATM and its interworking with other control features. In *Proc INFOCOM'96*, pages 1242–1250, San Francisco, CA, April 1996. Mentioned on page(s) 16

[70] D. Lin and R. Morris. Dynamics of random early detection. In *Proc. ACM SIGCOMM'97*, volume 27, Cannes, Setpember 1997. Mentioned on page(s) 51, 56, 126

[71] S. Low, L. Peterson, and L. Wang. Understanding tcp vegas: A duality model. In *SIGMETRICS 2001*, 2001. Mentioned on page(s) 12, 13

[72] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. *TCP Selective Acknowledgement Options*, October 1996. RFC 2018. Mentioned on page(s) 12, 13

[73] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communication Review*, 27(3), July 1997. Mentioned on page(s) 13, 40

[74] M. May. *Evaluation quantitative des nouveaux mécanismes de gestion de qualité de service dans l'Internet*. PhD thesis, Université de Nice Sophia Antipolis, October 1999. Mentioned on page(s) 68, 98

[75] M. May, T. Bonald, and J. Bolot. Analytic evaluation of RED performance. In *Proc. INFOCOM'00*, pages 1415–1424, Tel Aviv, Israel, March 2000. Mentioned on page(s) xx, xxi, 26, 68, 150

[76] M. May, C. Diot, B. Lyles, and J. Bolot. Influence of active queue management parameters on aggregate traffic performance, April 2000. Research Report. Mentioned on page(s) 98

[77] A.J. McAuley. Reliable broadband communications using a burst erasure correcting code. In *Proc. ACM SIGCOMM'90*, pages 297–306, Philadelphia, PA, USA, September 1990. Mentioned on page(s) 145

[78] V. Misra, W. Gong, and D. Towsley. Fluid-based analysis of a network of AQM routers supporting TCP flows with an applicaton to RED. In *Proc. SIGCOMM'00*, Stockholm, Sweden, August 2000. Mentioned on page(s) 68

[79] M. Mitzenmacher and R. Rajaraman. Towards more complete models of tcp latency and throughput. *Journal of Supercomputing*, 2001. Mentioned on page(s) 13

[80] J.-L. Mélin. *Qualité de service sur IP*. Eyrolles, 2001. Mentioned on page(s) 5

[81] T.S.E. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *Proc. IEEE INFOCOM'02*, New York, NY, June 2002. Mentioned on page(s) 41

[82] T.J. Ott, T.V. Lakshman, and L.H. Wong. SRED: Stabilized RED. In *Proc. IEEE INFOCOM'99*, pages 1346–1355, New York, NY, March 1999. Mentioned on page(s) 31, 68, 124

[83] J. Padhye, V. Firoiu, D. Townsley, and J. Kurose. Modeling TCP throughput: a simple model and its empirical validation. In *Proc. ACM SIGCOMM'98*, 1998. Mentioned on page(s) 13

[84] Sally Floyd's page on TCP modeling. `http://www.acir.org/floyd/tcp_small.html`. Mentioned on page(s) 13

[85] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control - the single node case. In *Proc. INFOCOM'92*, 1992. Mentioned on page(s) 20

[86] M. Parris, K. Jeffay, and F. Smith. Lightweight active router queue management for multimedia networking. In *Proc. SPIE*, pages 162–174, San Jose, CA, January 1999. Mentioned on page(s) 55, 126

[87] J. Postel. *User Datagram Protocol*, August 1980. RFC 768. Mentioned on page(s) 13

[88] J. Postel. *Transmission Control Protocol*, Septembre 1981. RFC 793. Mentioned on page(s) 8, 9

[89] K. Ramakrishnan and S. Floyd. *A proposal to add explicit congestion notification (ECN) to IP*, January 1999. RFC 2481. Mentioned on page(s) 17

[90] K. Ramakrishnan, S. Floyd, and D. Black. *The addition of explicit congestion notification (ECN) to IP*, Septembre 2001. RFC 3168. Mentioned on page(s) 17

[91] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *Computer Communication Review*, 27(2):24–36, April 1997. Mentioned on page(s) 145

[92] H. Sanneck and G. Carle. A framework model for packet loss metrics based on runlengths. In *Proc. Multimedia Computing and Networking Conference (MMCN)*, pages 177–187, San Jose, CA, January 2000. Mentioned on page(s) 126, 147

[93] S. Sharma and Y. Viniotis. Convergence of a dynamic policy for buffer management in shared buffer ATM switches. *Performance Evaluation*, 36-37:249–266, August 1999. Mentioned on page(s) 16

[94] NS simulator homepage, 2003. `http://www.isi.edu/nsnam/ns`. Mentioned on page(s) xxiii, 69, 122, 145

[95] W. Richard Stevens. *TCP/IP Illustrated*, volume 1. Addison-Wesley, 1994. Mentioned on page(s) 5

[96] A. Tanenbaum. *Computer Networks*. Prentice Hall, Inc, 1996. Mentioned on page(s) 5

[97] C. Wang, B. Li, Y. Hou, K. Sohraby, and Y. Lin. LRED: a robust queue management scheme based on packet loss ratio. In *Proc. INFOCOM'04*, March 2004. Mentioned on page(s) 48, 68, 122

[98] Frank Kelly's web page. `http://www.statslab.cam.ac.uk/~frank/int`. Mentioned on page(s) 13

[99] Mark Allman's web page on TCP/IP research papers. `http://tcpsat.lerc.nasa.gov/tcpsat/papers.html`. Mentioned on page(s) 13

[100] Cisco web pages. `http://www.cisco.com/warp/public/732/netflow/qos_ds.html`. Mentioned on page(s) 5

[101] Y.R. Yang, M.S. Kim, and S.S. Lam. Transient behaviors of TCP-friendly congestion control protocols. In *Proc. INFOCOM'01*, Anchorage, Alaska, April 2001. Mentioned on page(s) 126

[102] B. Zheng and M. Atiquzzaman. DSRED: an active queue management scheme for next generation networks. In *Proc. LCN'00*, pages 242–251, Tampa, Florida, November 2000. Mentioned on page(s) 35, 52

[103] B. Zheng and M. Atiquzzaman. DSRED: improving performance of active queue management over heterogeneous networks. In *Proc. ICC'01*, June 2001. Mentioned on page(s) 35

[104] T. Ziegler, C. Brandauer, and S. Fdida. A quantitative model for parameter setting of RED with TCP traffic. In *Proc. IWQoS*, 2001. Mentioned on page(s) 68

[105] T. Ziegler, S. Fdida, C. Brandauer, and B. Hechenleitner. Stability of RED with two-way TCP traffic. In *Proc. IEEE ICCN'00*, October 2000. http://www.newmedia.at/tziegler/papers.html. Mentioned on page(s) xxi, 68, 75

Index

R

S

T

U

Résumé

Cette thèse s'intéresse à améliorer la qualité de service (QdS) offerte par des mécanismes de gestion de file d'attente tels que RED (Random Early Detection). Nous étudions une approche adaptative de RED appelée ARED. Notre but est de trouver un extension simple de ARED afin d'améliorer la prédictibilité des mesures de performance comme le délai, le temps de gigue sans pour autant sacrifier le taux de perte. Pour cela, nous proposons un nouveau mécanisme nommé PSAND qui configure les paramètres de RED et l'évaluons à l'aide de simulations. Nos résultats de la comparaison de PSAND avec ARED et d'autres schémas bien connus ou récemment proposés ont montré que avec une complexité moindre, PSAND donne un compromis désiré et de bonnes performances telles qu'un faible délai d'attente dans la file, une faible gigue et une plus grande stabilité en présence d'un fort trafic. Une autre contribution de cette thèse est l'étude de l'interaction de FEC (Forward Error Correction) avec des schémas comme Drop Tail et RED. Les résultats ont montré que selon la valeur des paramètres comme le nombre de flux TCP actifs, le taux de redondance et de la taille des blocs FEC parfois RED ou parfois Drop Tail donne de meilleures performances. Un modèle analytique ayant confirmé les résultats obtenus par simulations a également été présenté.

mots-clefs: QdS, RED, mécanisme de gestion active de file d'attente, FEC.

Abstract

This thesis focuses on improving the quality of service (QoS) offered by active queue management schemes like RED (Random Early Detection). We followed an adaptive approach of RED namely ARED (Adaptive RED). Our goal is to find a simple extension to ARED in order to improve the predictability of performance measures like queueing delay and delay jitter without sacrificing the packet loss rate. To achieve this goal, we propose a new algorithm called PSAND that tunes RED parameters according to the traffic load. We evaluate its performance by extensive simulations. Our results show that as compared to ARED and others well-known, most studied and very recently proposed schemes, PSAND gives an overall desirable tradeoff and good performances such as small delay and delay jitter, and more robustness with less complexity.

Another contribution of this thesis is the study of the interaction of forward error correction (FEC) code with queue management schemes like Drop Tail (DT) and RED. We show, through simulations, that sometimes FEC combined with RED, sometimes FEC combined with DT may be more efficient depending on several parameters like the number of TCP flows that constitute the background traffic, the FEC block size and the amount of redundancy in a FEC block. We conclude generally that using FEC is more efficient with RED than with DT when the loss rate is small, and a relatively important amount of redundancy and at most a moderate FEC block size is used. We complement these observations with a simple model, which is able to capture the tradeoff between the locality and the frequency of losses.

keywords: QoS, RED, Active Queue Management, FEC.