Distributed management of competitive access to common resources using algorithmic game theory

REMOUS-ARIS KOUTSIAMANIS

Department of Electrical and Computer Engineering

Democritus University of Thrace

Advisor: Pavlos S. Efraimidis

A thesis submitted for the degree of

Doctor of Philosophy

Xanthi, February 2016

Copyright © 2016 Remous-Aris Koutsiamanis

Democritus University of Thrace
Department of Electrical and Computer Engineering
Building A, ECE, University Campus, 67100 Xanthi, Greece

All rights reserved. No parts of this book may be reproduced or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author.

I would like to dedicate this thesis to my parents.

Acknowledgements

I would first like to express my sincere thankfulness and gratitude to my PhD advisor Professor Pavlos Efraimidis, who has guided and supported me throughout my studies and who has helped me with ideas, advice and time to successfully complete my studies.

I would then like to thank my very helpful advisory committee members Professor Alexandros S. Karakos and Professor Paul Spirakis, for promptly attending to all necessary academic and administrative issues with respect to my PhD studies. I would also like to thank the members of my defence committee, Professor Maria Satratzemi, Professor Spyros Kontogiannis, Professor Vassilis Tsaoussidis and Professor Avi Arampatzis, for carefully evaluating my work, proposing further avenues for exploration and contributing to a stimulating and very enjoyable process. I would like to especially thank Professor Vassilis Tsaoussidis and Professor Avi Arampatzis for our insightful discussions and their guidance and support.

During my PhD studies I worked in parallel at the Xanthi branch of the ATHENA R.C., under the supervision of Dr. Christos Emmanouilidis. To a large extent, I owe the objective ability to perform my PhD to his incredible support as well as becoming a more methodical, ethical and mature researcher. Christos, thank you for everything from the bottom of my heart.

In the course of my studies I have been fortunate to collaborate with a large number of kind and inspiring people. I would like to thank Professor Dimitris Gritzalis, Dr. Yiannis Soupionis and Lazaros Tsavlidis for their constructive collaboration in preparing the joint works

described in this thesis. I would also like to especially thank Professor Vasilis Katos for providing fantastic extracurricular opportunities as well as being a model of a thoughtful and efficient academic.

I should note that I would not have pursued the academic path had it not been for the inspiration and guidance provided to me by my past advisors; Professor Themis Panayiotopoulos and Professor Michael Rovatsos thank you very much for your sincere and thoughtful encouragement and support.

Many of my long-time friends have been very supportive and I would like to thank them in no particular order and in brief here; Ioannis Paraskevopoulos for all the deep discussions, for helping me keep a positive perspective and for making me feel like I actually know more than I really do, Fotis Nalbadis for being an unending source of fun and relaxation and for making me take time off for decompression despite my protestations, Dr. Aris Belesiotis for being a great listener and a relaxed and insightful friend, Kostas Tilelis for always being brutally honest, compassionate, supportive, challenging me to improve myself and also for the sense of humour I partially rubbed off of him, Apostolis Hardalias for always being supportive, kind and grumpy.

I would also like to thank some of the good friends I made during my studies; Dr. George Drosatos for being a kind and very supportive colleague during this time, Kostantinos Pyloudis and Viky Barboka for being great friends and guinea pigs for my next steps, Dr. Michalis Savelonas and Maria Douma for so graciously entertaining my hypotheticals and for always being supportive counsellors, Kostas Zagoris for his insightful and fun approach to all discussion topics, Dr. George Hlapanis for his warm friendship and for being my mentor, and Dr. Savvas Chatzichristofis for his academic support and fun company.

There is one person I owe a large part of my accomplishments to and who has been my rock during both the happy times and the difficulties, my partner in life Aimilia. I sometimes cannot believe my luck in finding this incredible, intelligent, compassionate and fun partner and I am not sure I could have managed it all without her. She has my eternal gratitude and love.

I would also like to thank Aimilia's family for their welcome and warm embrace into their family and for their patience and support during all this time.

Lastly, but by no means least, I want to express my love, respect and indebtedness to my unconditional supporters, my parents. They are the reason I have become the person who I am and by extension this is their accomplishment as much as it is mine. I hope to be able to sometime repay them for all their efforts and sacrifices.

Abstract

The Internet is today an inextricable part of daily personal, educational and business activity, turning any problems in its operation or availability into a significant interruption of these activities and their users. The resources offered by the Internet (capacity, coverage) are continuously increasing but at the same time the users and their demands are increasing at an even larger pace. If measures for the efficient and fair management of the network resources are not taken, the Internet will cease to be able to support new users and applications with high quality.

Internet users operate in an independent manner, by creating data flows (sending and receiving network packets) which satisfy their demands. Each user prefers for his needs to be served in the best possible way but the resources of the network are shared and finite, making it often impossible to provide the best service to everyone. This leads to users competing amongst themselves for access to the network resources and its services.

Whenever the demands placed on the services by the users exceed the capacity of the services, a means of selecting which users and to which degree they will be served is required. In the case of the Internet, the resources are network capacity, the demands of the users are requests for transferring network packets and the functionality of selecting which users are served and how they are served is generally referred to as Quality of Service (QoS).

One feature of the Internet which significantly affects the possible solutions to providing QoS is its decentralized structure: there exists no central authority which is responsible for the whole operation of the network and which could centrally perform the resource allocation. Instead, resources are allocated locally at each network node to the users which access it.

In this work, we address the issue of managing competitive access to common resources through the use of algorithmic game theory. This approach is validated by the competitive, selfish and independent

nature of the users. Additionally, in the case of QoS provision for the Internet, our solutions have to be distributed in order to be applicable.

Specifically, we start by proposing the Prince mechanism for distributing network flow throughput in a (MaxMin-resembling) fair manner. We then propose an efficient data structure and algorithm set for implementing Prince on a network router queue.

We continue by providing the theoretical description and first simple experimental implementation of PacketEconomy, a network economy where each flow is modelled as a population of rational network packets, and these packets can self-regulate their access to network resources by mutually trading their positions in router queues. This theoretical model is then adapted to the OMNET++ simulator and via thorough experimentation we present the validation of the efficacy of our solution in a realistic context.

Applying the same principles of game-theoretic analysis to realistic service provision problems, we also study an Internet-based VoIP service access problem in the context of the prevention of SPIT (SPam over Internet Telephony).

Extended Abstract in Greek

Αποκεντρωμένη διαχείριση ανταγωνιστικής πρόσβασης σε κοινόχρηστους πόρους με αλγοριθμική θεωρία παιγνίων

Το Διαδίκτυο σήμερα αποτελεί αναπόσπαστο μέρος καθημερινής ιδιωτικής, εκπαιδευτικής και επιχειρηματικής δραστηριότητας, με αποτέλεσμα προβλήματα στη λειτουργία του να προκαλούν σημαντικές διαταραχές σε αυτές τις δραστηριότητες και να επηρεάζονται οι χρήστες του. Οι πόροι που διαθέτει το Διαδίκτυο (χωρητικότητα, συνδέσεις) αυξάνονται διαρκώς αλλά ταυτόχρονα, με μεγαλύτερο ρυθμό, αυξάνονται οι χρήστες του και οι απαιτήσεις των εφαρμογών τις οποίες καλείται να υποστηρίξει. Αν δεν ληφθούν μέτρα για την αποδοτική διαχείριση και την δίκαιη κατανομή των πόρων δικτύου, το Διαδίκτυο θα πάψει να έχει την δυνατότητα να υποστηρίζει νέους χρήστες και εφαρμογές και να διασφαλίζει υψηλή ποιότητα παροχής υπηρεσιών.

Οι χρήστες του Διαδικτύου λειτουργώντας ανεξάρτητα ο ένας από τον άλλο δημιουργούν ροές δεδομένων (στέλνοντας και λαμβάνοντας πακέτα δεδομένων) οι οποίες χρησιμοποιούν τους κοινόχρηστους και πεπερασμένους πόρους του Διαδικτύου. Καθώς κάθε χρήστης προτιμά την καλύτερη δυνατή εξυπηρέτηση για τις ροές του και καθώς η χωρητικότητα του δικτύου επιτρέπει συχνά μόνο ένα υποσύνολο των πακέτων να εξυπηρετηθούν, δημιουργείται ανταγωνισμός κατά την πρόσβαση στους κοινόχρηστους πόρους.

Καθώς δεν υπάρχει κάποια κεντρική αρχή που να είναι υπεύθυνη για την ανάθεση πρόσβασης στους πόρους του δικτύου, η πρόσβαση ανατίθεται με αποκεντρωμένο τρόπο σε κάθε δρομολογητή. Οι χρήστες του δικτύου μπορούν να υποβάλουν ένα αυθαίρετο πλήθος από πακέτα ανά πάσα στιγμή στο δίκτυο και ο κάθε δρομολογητής αποφασίζει πόσα και ποια θα δεχθεί και πως θα τα εξυπηρετήσει. Η έλλειψη συντονισμού μεταξύ των ανεξάρτητων ροών οδηγεί το Διαδίκτυο να εμφανίζει μια ”άναρχη” μορφή λειτουργίας και δημιουργεί προβλήματα τα οποία μπορούν να αντιμετωπιστούν με έννοιες και εργαλεία από

την αλγοριθμική θεωρία παιγνίων. Πιο συγκεκριμένα, ο στόχος είναι να βρεθούν οι προϋποθέσεις και ο τρόπος επίτευξης δίκαιης, αποδοτικής και υπολογιστικά εφικτής (tractable) αποκεντρωμένης διαχείρισης πόρων σε δίκτυα υπολογιστών.

Στα παιγνιοθεωρητικά μοντέλα αυτό που ενδιαφέρει συνήθως είναι να διερευνηθεί ποιες είναι οι καταστάσεις ισορροπίας του συστήματος. Μια από τις πιο σημαντικές κατηγορίες καταστάσεων ισορροπίας είναι οι ισορροπία Nash (Nash Equilibrium). Στην ισορροπία Nash κανένας παίκτης/χρήστης δεν έχει κίνητρο να αλλάξει τη στρατηγική του οπότε από τη στιγμή που θα επιτευχθεί αυτή η ισορροπία το σύστημα σταθεροποιείται σε αυτή την κατάσταση. Είναι συχνά επιθυμητό ένα παίγνιο να τείνει προς μια τέτοια ισορροπία αλλά έχει αποδειχθεί πως η υπολογιστική πολυπλοκότητα εύρεσης των Ισορροπιών Nash ακόμα και σε απλά μοντέλα είναι PPAD-complete [17], κάνοντάς τις ενδεχομένως δύσκολο να επιτευχθούν. Επιπλέον, η έρευνα στο χώρο στην συντριπτική της πλειοψηφία έχει ασχοληθεί με θεωρητικά μοντέλα δικτυακών παιγνίων τα οποία απέχουν σημαντικά από την δομή και την λειτουργία των πραγματικών δικτύων. Για τους παραπάνω λόγους καθίσταται σημαντική η μελέτη αυτών των μοντέλων όχι μόνο θεωρητικά αλλά και πειραματικά, ώστε να αποδειχθεί η πρακτική υλοποιησιμότητα του μοντέλου και η ρεαλιστική μελέτη των επιδόσεων και των χαρακτηριστικών του.

Η παρούσα διδακτορική έρευνα περιλαμβάνει τη διερεύνηση των τρόπων αξιοποίησης της αλγοριθμικής θεωρίας παιγνίων για την κατασκευή αποκεντρωμένων μηχανισμών διαχείρισης πρόσβασης σε πόρους δικτύου. Βασική αρχή της έρευνας είναι ότι η δημιουργία κατάλληλων αλγορίθμων διαχείρισης των πόρων, τέτοιων που να δίνουν κίνητρο στους χρήστες να ρυθμίζουν σωστά τις ροές δεδομένων τους, οδηγούν στα επιθυμητά αποτελέσματα για όλους τους χρήστες. Χωρίς τα κατάλληλα κίνητρα οι χρήστες, συμπεριφερόμενοι εγωιστικά, κάνουν κατάχρηση των κοινών πόρων και ζημιώνουν και τρίτους χρήστες. Στόχος είναι η σχεδίαση, υλοποίηση και μελέτη της συμπεριφοράς τέτοιων αλγορίθμων σε θεωρητικό και πειραματικό επίπεδο.

Με βάση τις παραπάνω επιδιώξεις, προτάθηκαν και υλοποιήθηκαν οι παρακάτω εργασίες που επιδιώκουν την ανταγωνιστική διαχείριση πόρων:

- Η πολιτική ουράς εξυπηρέτησης δρομολογητή Prince, που δίνει αντικίνητρο σε χρήστες να καταλαμβάνουν δυσανάλογο (του δίκαιου μεριδίου τους) χώρο στην πεπερασμένου μεγέθους ουρά του

δρομολογητή. Ο στόχος είναι όλοι ο χρήστες σε συνθήκες συμφόρησης να παίρνουν, αν το ζητούν, το δίκαιο μερίδιό τους. Τρεις παραλλαγές της πολιτικής προτείνονται και μελετώνται, με διαφορετικές δικτυακές επιδόσεις και υπολογιστικές απαιτήσεις.

- Ο αλγόριθμος ροών δεδομένων HL-Hitters που πραγματοποιεί την εύρεση και τον περιορισμό της πιο επιβαρυντικής ροής στην ουρά εξυπηρέτησης δρομολογητή σε σταθερό χρόνο (O(1)). Ο αλγόριθμος επιτρέπει την αποτελεσματική υλοποίηση της πιο δίκαιης παραλλαγής από τις πολιτικές ουράς Prince.

- Η θεωρητική σχεδίαση και μελέτη του συστήματος PacketEconomy, που με την εισαγωγή ενός ανταλλάξιμου είδους, εικονικών «χρημάτων», εξετάζει κατά πόσο είναι δυνατόν να οδηγηθεί το δίκτυο σε υψηλή απόδοση με υπολογιστικά απλούς και εφικτούς μηχανισμούς. Οι χρήστες δεν καλούνται να πληρώσουν με πραγματικά χρήματα για τη χρήση του δικτύου, αλλά το «χρήμα» κινεί μηχανισμούς διαχείρισης των πόρων και επιτρέπει την έκφραση των προτιμήσεων εξυπηρέτησης (Quality of Service) των ροών των χρηστών με κοινούς οικονομικούς όρους.

- Η πρακτική υλοποίηση και μελέτη του συστήματος PacketEconomy σε εξομοιωτή δικτύων, για την εξέταση της υλοποιησιμότητας και της απόδοσής του σε ρεαλιστικό περιβάλλον με ροές πολλαπλών τύπων καθώς και η παιγνιοθεωρητική συμπεριφορά του σε αυτό το περιβάλλον.

- Η παιγνιοθεωρητική ανάλυση της αποτροπής ανεπιθύμητων κλήσεων σε περιβάλλον Διαδικτυακής τηλεφωνίας (VoIP), όπου με την δημιουργία και μελέτη ενός μοντέλου χρήσης μιας τέτοιας υπηρεσίας αναζητείται με ποιους τρόπους και με ποιες ρυθμίσεις ενός φίλτρου ηχητικού CAPTCHA επιτυγχάνεται η καλύτερη δυνατή κατανομή πόρων της υπηρεσίας στους επιθυμητούς χρήστες και αποτρέπονται οι ανεπιθύμητοι.

Οι εργασίες αυτές αποδεικνύουν ότι είναι εφικτή η χρήση της αλγοριθμικής θεωρίας παιγνίων για την σχεδίαση πραγματικών μηχανισμών διαχείρισης πόρων και ότι η πρακτική υλοποίησή τους μπορεί να οδηγήσει σε συστήματα με τα επιθυμητά χαρακτηριστικά απόδοσης και δικαιοσύνης. Συνοπτικές περιγραφές των εργασιών δίνονται παρακάτω στις περιλήψεις των αντίστοιχων κεφαλαίων της διατριβής.

ΔΟΜΗ ΤΗΣ ΔΙΑΤΡΙΒΗΣ

Παρακάτω συνοψίζονται τα περιεχόμενα των κεφαλαίων της διατριβής:

Κεφάλαιο 1: Introduction

Περιγράφονται τα κίνητρα και οι στόχοι της παρούσας διατριβής. Περιγράφεται η σημερινή κατάσταση σε σχέση με την διαχείριση πόρων δικτύων, ποια προβλήματα αντιμετωπίζονται και ποια είναι τα χαρακτηριστικά των λύσεων που αναζητούνται.

Κεφάλαιο 2: Background

Παρέχεται το απαραίτητο υπόβαθρο για τη κατανόηση των βασικών εννοιών που χρησιμοποιούνται σε αυτή την διατριβή. Πιο συγκεκριμένα, πρώτα περιγράφονται έννοιες που αφορούν τα δίκτυα, όπως οι αλγόριθμοι και οι πολιτικές ουρών που χρησιμοποιούνται καθώς τα είδη και τα χαρακτηριστικά των ροών δεδομένων. Στη συνέχεια περιγράφονται οι έννοιες της αλγοριθμικής θεωρίας παιγνίων που χρησιμοποιούνται ως δομικά στοιχεία στην ανάπτυξη των προτεινόμενων λύσεων της διατριβής.

Κεφάλαιο 3: Prince: an Effective Router Mechanism for Networks with Selfish Flows

Περιγράφεται η πολιτική ουράς εξυπηρέτησης δρομολογητή Prince, που σκοπό έχει την δίκαιη κατανομή δικτυακών πόρων στις ροές δεδομένων που τους χρησιμοποιούν. Πιο συγκεκριμένα, ξεκινώντας από την αρχή ότι οι δρομολογητές δεν προστατεύονται από επιθετικές ή μη-ανταποκρίνουσες ροές, προτείνεται η ενεργή πολιτική διαχείρισης ουράς Prince. Η βασική ιδέα είναι η προστασία το δίκαιο μερίδιο των μη-επιθετικών ροών, μέσω παιγνιοθεωρητικής προσέγγισης που δίνει αντικίνητρο κακής συμπεριφοράς στην πιο επιβαρυντική ροή απορρίπτοντας πακέτα της όταν υπάρχει συμφόρηση.

Δημιουργούνται τρεις παραλλαγές της πολιτικής Prince (Prince-G/S/A), και η συμπεριφορά και οι επιδόσεις τους μετρώνται με την χρήση του προσομοιωτή ns-2. Τα αποτελέσματα δείχνουν η Prince μοιάζει σε συμπεριφορά με την πολιτική MaxMin fairness.

Κεφάλαιο 4: A Heaviest Hitters Limiting Mechanism with O(1) Time Complexity for Sliding-window Data Streams

Σε αυτό το κεφάλαιο περιγράφεται ο HL-Hitters, ένας αλγόριθμος ροών δεδομένων που επιτυγχάνει την εύρεση και τον περιορισμό της πιο επιβαρυντικής ροής στην ουρά εξυπηρέτησης δρομολογητή. Ο αλγόριθμος επιλύει το πρόβλημα με ακριβή (μη-προσεγγιστικό) τρόπο σε σταθερό χρόνο (O(1)) με υψηλή πιθανότητα και για την λειτουργία εύρεσης και για την λειτουργία ενημέρωσης. Επιπρόσθετα, δίνει πρόσβαση στο πρώτο και τελευταίο πακέτο μίας ροής σε σταθερό χρόνο. Αυτές οι ιδιότητες επιτρέπουν την αποτελεσματική υλοποίηση της βέλτιστης παραλλαγής της Prince.

Στο κεφάλαιο περιγράφεται η δομή δεδομένων και αλγόριθμοι που υλοποιούν αυτή τη λειτουργικότητα και εξηγείται πως χρησιμοποιούνται. Επιπλέον, πραγματοποιούνται μετρήσεις επιδόσεων που παράγουν ποσοτικά αποτελέσματα τα οποία επιβεβαιώνουν τα θεωρητική περιγραφή και ότι η υλοποίηση είναι αρκετά υψηλών επιδόσεων ώστε να χρησιμοποιηθεί σε πρακτικές εφαρμογές.

Κεφάλαιο 5: On Money as a Means of Coordination between Network Packets

Περιγράφεται θεωρητική σχεδίαση και μελέτη του PacketEconomy, ενός συστήματος που στοχεύει στον συντονισμό ροών δεδομένων με τη χρήση της έννοιας ενός ανταλλάξιμου είδους, εικονικών «χρημάτων». Στο σύστημα κάθε ροή δεδομένων μοντελοποιείται ως ένας πληθυσμός από ορθολογικά πακέτα δικτύου και αυτά τα πακέτα μπορούν να αυτορρυθμίσουν την πρόσβασή τους στους κοινόχρηστους δικτυακούς πόρους ανταλλάσοντας θέσεις εντός των ουρών αναμονής των δρομολογητών. Τα πακέτα δεν καλούνται να πληρώσουν με πραγματικά χρήματα για τη χρήση του δικτύου, αλλά το «χρήμα» κινεί μηχανισμούς διαχείρισης των πόρων και επιτρέπει την έκφραση των προτιμήσεων εξυπηρέτησης (Quality of Service) των ροών με κοινούς οικονομικούς όρους.

Μελετάται το μοντέλο Markov της ανταλλαγής και αποδεικνύεται ότι υπάρχουν ισορροπίες Nash όπου πραγματοποιούνται ανταλλαγές θέσεων μεταξύ των πακέτων. Επιπλέον, το βασικό υπολογιστικό βήμα του PacketEconomy είναι εκτελέσιμο σε σταθερό χρόνο (O(1)) σε παράλληλο υλικό επιτρέποντας την υλοποίηση σε σύγχρονους δρομολογητές.

Κεφάλαιο 6: Implementing PacketEconomy: Distributed Money-based QoS in OMNET++

Περιγράφεται η πρακτική υλοποίηση και μελέτη του PacketEconomy σε εξομοιωτή δικτύων. Αρχικά, γενικεύεται η μορφή της συνάρτησης ωφέλειας ώστε να καλύπτει ένα μεγαλύτερο εύρος μορφών. Στη συνέχεια, περιγράφεται πως το μοντέλο τροποποιείται ώστε να προσαρμοστεί στο ρεαλιστικό περιβάλλον δικτύου που υλοποιείται στον προσομοιωτή OMNET++ με την βιβλιοθήκη μοντέλων δικτύου INET. Η προσομοίωση πραγματοποιείται σε δίκτυο νέας γενιάς IPv6 εκμεταλλευόμενη τις δυνατότητες επέκτασης των μεταπληροφοριών (IPv6 extension headers) που μπορούν τα πακέτα να αποθηκεύσουν.

Υλοποιούνται ροές δεδομένων που χρησιμοποιούν τις δυνατότητες του PacketEconomy και με μεγάλο αριθμό προσομοιώσεων μετράται η συμπεριφορά και οι επιδόσεις τους. Επιπλέον, πραγματοποιούνται προσομοιώσεις με ροές που δεν χρησιμοποιούν το PacketEconomy για λόγους σύγκρισης.

Τα αποτελέσματα δείχνουν ότι το PacketEconomy λειτουργεί σε ρεαλιστικά δίκτυα, παρέχει την δυνατότητα συντονισμού και αυτορύθμισης στις ροές δεδομένων , παιγνιοθεωρητικά παρέχει κίνητρο στις ροές να το χρησιμοποιήσουν συμμετέχοντας και συμπεριφέρεται συγκρίσιμα με άλλες πολιτικές ουρών έχοντας πλεονεκτήματα την αυξημένη ευελιξία και τις υπολογιστικές επιδόσεις.

Κεφάλαιο 7: A Game-theoretic Analysis of Preventing Spam over Internet Telephony via Audio CAPTCHA-based Authentication

Περιγράφεται η παιγνιοθεωρητική ανάλυση της αποτροπής ανεπιθύμητων κλήσεων σε περιβάλλον Διαδικτυακής τηλεφωνίας (VoIP). Αρχικά περιγράφεται το πρόβλημα των ανεπιθύμητων κλήσεων σε περιβάλλον Διαδικτυακής τηλεφωνίας (VoIP) και των μεθόδων αντιμετώπισής του. Το ηχητικό CAPTCHA (Completely Automated Public Turing Test to Tell Computer and Humans Apart), μία από τις μεθόδους, υποχρεώνει τον καλώντα να αποδείξει ότι είναι άνθρωπος και όχι αυτοματοποιημένο λογισμικό απαντώντας ερωτήσεις. Επιπλέον, ένα ανακριβές φίλτρο επιτρέπει ή απορρίπτει την πρόσβαση σε καλώντες.

Στο κεφάλαιο πραγματοποιείται παιγνιοθεωρητική ανάλυση του προβλήματος, όπου με την δημιουργία και μελέτη ενός μοντέλου χρήσης μιας τέτοιας υπηρεσίας αναζητείται με ποιους τρόπους και με ποιες ρυθμίσεις ενός φίλτρου και ενός ηχητικού CAPTCHA επιτυγχάνεται η καλύτερη δυνατή κατανομή πόρων της υπηρεσίας στους επιθυμητούς

χρήστες και αποτρέπονται οι ανεπιθύμητοι.

Οι χρήστες μοντελοποιούνται ως εγωιστές αλλά ορθολογικοί παίκτες και χωρίζονται σε ανεπιθύμητους και επιθυμητούς. Το σύστημα υποδοχής των κλήσεων μοντελοποιείται ως ένα παίγνιο το οποίο αντιπροσωπεύει την αλληλεπίδραση των καλούντων με το τηλεφωνικό σύστημα.

Τα αποτελέσματα αυτής της προσέγγισης δείχνουν ότι οι αμυντικές τεχνικές που χρησιμοποιήθηκαν οδηγούν σε επιθυμητές ισορροπίες Nash όπου τα CAPTCHA συμβάλλουν στην ωφέλεια (utility) των επιθυμητών χρηστών. Επίσης, το μοντέλο δείχνει ότι ακόμα και αν οι ανεπιθύμητοι καλώντες γνωρίζουν τα χαρακτηριστικά των αμυντικών τεχνικών δεν μπορούν να εξάγουν οφέλη από αυτές τις γνώσεις.

Κεφάλαιο 8: Conclusions and Directions

Για την ολοκλήρωση της διατριβής το κεφάλαιο αυτό περιγράφει τις κύριες συνεισφορές της δουλειάς αυτής και παρέχει μια επισκόπηση των εν εξελίξει και μελλοντικών εργασιών. Πιο αναλυτικά, σε αυτή τη διατριβή, μελετήθηκαν προβλήματα με παιγνιοθεωρητικούς όρους και προτάθηκαν μηχανισμοί, αλγόριθμοι, δομές δεδομένων και συστήματα για την επίτευξη της ανταγωνιστικής πρόσβασης σε κοινόχρηστους πόρους. Μέσω των προτεινόμενων λύσεων παρέχεται ένας αποτελεσματικός τρόπος αντιμετώπισης αυτού του προβλήματος γεγονός που αποτελεί καινοτομία σε σχέση με τη σημερινή πρακτική.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

The Internet is today a focal point of activity in modern societies. Commerce, entertainment, government, education, human relations, and most other aspects of human endeavour are regularly mediated by the Internet, which makes its sustained operation of critical importance.

The Internet provides the infrastructure for multiple independent network traffic flows. This infrastructure and its resources are limited and shared between these flows, each of which attempts to optimize its own performance. As a result of entities sharing a limited common resource, with individual optimization targets, competition arises between these flows.

Without any central authority to regulate its operation, the available network resources of the Internet are allocated by independent routers to the flows in a decentralized manner. Internet flows may submit at any time an arbitrary amount of packets to the network and then adjust their packet rate with an appropriate flow control algorithm, like the AIMD-based algorithms for TCP-flows. The apparent lack of coordination between the independent flows leads the Internet to an "anarchic" way of operation.

In general, services which are provided to users have finite capacity, the consumption of which leads to increasing network congestion, a major issue on the Internet. Under congestion, networks struggle to allocate resources efficiently and fairly. Congestion builds up easily when some of the flows try to gain a large share of the network capacity, either by excessively increasing their sending rate or by not cutting back despite their packet losses. This situation, in which multiple selfish players can ultimately overload a shared resource even when it is obvious that it is not in anyone's long term interest, is an instance of the "Tragedy of the Commons" problem [41]. This behaviour leads to heavy congestion and threatens the stability and efficiency of the Internet.

Beyond these consequences, our greatest concern is the unfairness that arises. During congestion, misbehaving flows may retain their sending rate while well-behaved ones cut back. The result is that the misbehaving flows receive an unfair proportion of the throughput at the expense of the well-behaved flows.

Viewing the Internet as a service to its competing users, some of the high-level requirements which it has to satisfy are:

- **Fairness** The allocation of resources to the users requesting them must be fair, for a given definition of fair. Fairness may mean receiving the same resources, guaranteeing a minimum amount of resources, receiving resources proportional to a user property or distributing the resources in any other way which makes sense in the context of the service.

- **Flexibility** The provision of the resources should be flexible enough so that all useful ways of distributing the resources should be supportable.

- **Control** Users should be able to affect or even control the distribution of the resources, in a way that does not completely override other users' ability to do the same.

- **Efficiency** Any service provided will have to process large amounts of requests from users, and thus high efficiency is not just a feature but a fundamental requirement.

These requirements are fulfilled by Quality of Service (QoS) mechanisms on networks and have been the subject of intensive research.

The fact that the problem consists of independent and selfish flows which compete for Internet network resources leads to its suitability for analysis with concepts and tools from algorithmic game theory. In these terms, network flows are selfish and independent players, the router's queueing algorithm is the game mechanism, the players' service request characteristics constitute the set of possible strategies and the allocated resources is the players' utility.

Achieving fairness and efficiency in the network can be translated to achieving a desirable Nash Equilibrium (NE) in the game theoretic model. In order to accomplish this goal we turn to mechanism design, through which we construct algorithms that incentivise the flows to select strategies in such a way that the resulting resource allocation distribution possesses the desirable characteristics. The overall novelty of this work is the application of game theoretic tools to create incentives in a real network in order to implement Quality of Service for the network flow players, while at the same time employing lightweight mechanisms on the routers.

1.1 Methodology

In this work, we aim to solve problems concerning the distribution of common resources to independent actors. Due to the nature of the problems, a game-theoretic approach is broadly applicable to analyse them and to propose solutions. The methodology used in this work follows a common theme. We initially study the problem with theoretical tools and then we follow up with experimental implementations, employing the theoretical results to devise practical solutions for the problem.

After selecting the problem to be addressed we examine how it can be modelled using game theory. As a part of the game-theoretic modelling process we first determine the aspects the problem consists of and identify the fundamental ones amongst them. This process establishes the players in the game, their preferences and the way these are expressed through the players' pay-off functions, as well as the actions the players have at their disposal. We then create a game-theoretic model which captures as many of the fundamental aspects of the problem as possible. This prioritisation aims to allow our model to be a reasonably faithful representation of the original problem whilst simple enough to be studied analytically.

After the basic model is defined, we perform an initial theoretical analysis to verify that our attempt to capture the problem is sound. In parallel, we implement a simple experimental version of the model and we evaluate it in order to verify that the model is an approximate proxy for the original problem. If either the theoretical or the experimental analysis expose inconsistencies or a large divergence from the original problem, we repeat the modelling and analysis processes amending the model until it exhibits the appropriate behaviour.

Once the theoretical model is complete, we perform further theoretical and experimental analysis, in order to converge on a solution of the problem. This often requires a second and more realistic implementation which solves any remaining issues regarding efficiency, performance, generality, or flexibility. After all the results are obtained, they are analysed to extract patterns and to reveal general insights.

1.2 Synopsis of Results

During the this PhD research, the following shared resource management solutions were implemented.

1. Prince: An active queue management policy resembling MaxMin fairness for throughput by protecting the fair share of well-behaved flows.

    - Joint work. Contribution of this PhD research: Participation in the research, resulting in the creation and selection of the variants of the algorithm used. Participation in performing the experiments and the interpretation of the results.

2. HL-Hitters: A heaviest hitters limiting mechanism with O(1) time complexity for sliding-window data streams.

    - Core work of this PhD research.

3. PacketEconomy theoretical model: a network economy in which the application of the common economic tool of money allows the coordination of network packets in order to self-regulate access to network resources.

    - Joint work. Contribution of this PhD research: Participation in the research, resulting in the creation of the theoretical model. Implementation of the experiments and the analysis of the results.

4. PacketEconomy adaptation to OMNET++: Examination how quality of service (QoS) can be achieved in a real network by allowing packets to coordinate using fiat money in a market economy for router queue positions.

    - Core work of this PhD research.

5. SpitGame: A game-theoretic analysis of preventing spam over Internet Telephony via audio CAPTCHA-based authentication.

    - Joint work. Contribution of this PhD research: Participation in the research, resulting in the creation and selection of game-theoretic model of the problem. Implementation of the experiments and analysis of the results.

1.3 Overview of the Thesis

Chapter 2: Background

The main concepts which underpin the solutions proposed in this work are presented in this chapter. The aspects of the subjects of network structure, network flow type, quality of service, router queue management mechanisms as well as game-theoretic modelling that are relevant to this thesis are discussed.

Chapter 3: Prince: an Effective Router Mechanism for Networks with Selfish Flows

Starting from the premise that modern routers are not protected from aggressive and unresponsive flows, we define a new, almost stateless, active queue management scheme, called Prince. The basic idea is to protect the fair share of well-behaved flows. We adopt a game theoretic view, where incentive is given to the majority flow by dropping its packets at congestion. In order to find the majority flow, we focus on the queue of the router and detect the flow with the most packets in it. From a game-theoretic point of view, Prince manages to track and bound aggressive flows and favour socially responsible ones. Our results show that in this context Prince resembles MaxMin Fairness allocation. Finally, we also examine a streaming version of the algorithm that can be fine-tuned to any desired performance/accuracy trade-off point.

Chapter 4: A Heaviest Hitters Limiting Mechanism with O(1) Time Complexity for Sliding-window Data Streams

In this work we address the problem of identifying and limiting the heaviest hitters in a sliding-window data stream. We propose the first, to our knowledge, exact (i.e., not approximate) algorithm which achieves O(1) with high probability time complexity in both update and query operations. Additionally, it tracks the first and last item of any itemset in the window in O(1) time complexity as well as the lightest hitters with no additional computational costs. These properties allow us to efficiently implement a mechanism to limit the heaviest hitters by evicting them from or not allowing them in the window. We describe the algorithms and data structure which implement this functionality, we explain how they can be used to accomplish the goal of limiting the heaviest hitters and perform experiments to produce quantitative results to support our theoretical arguments.

Chapter 5: On Money as a Means of Coordination between Network Packets

In this work, we apply a common economic tool, namely money, to coordinate network packets. In particular, we present PacketEconomy, a network economy where each flow is modelled as a population of rational network packets, and these packets can self-regulate their access to network resources by mutually trading their positions in router queues. We consider a corresponding Markov model of trade and show that there are Nash equilibria (NE) where queue positions and money are exchanged directly between the network packets. This simple approach, interestingly, delivers significant improvements for packets and routers.

Chapter 6: Implementing PacketEconomy: Distributed Money-based QoS in OMNET++

In this work we examine how quality of service (QoS) can be achieved in a real network by allowing packets to coordinate using fiat money in a market economy for router queue positions. In this context we implement and evaluate the PacketEconomy mechanism in the discrete-event simulator OMNET++, using the standard INET library for simulating IPv6 networks and evaluate throughput, end-to-end delay and packet drop rates. Additionally, we examine whether the flows have a game theoretic incentive to participate in the market economy, while covering both TCP- and UDP-based flows in multiple different cases. The mechanism achieves QoS by allowing packets with different QoS requirements waiting to be served in router queues to mutually trade positions by exchanging money. Notably, each flow can independently and selfishly define the ask and bid prices of its packets. In this manner, packets can coordinate in order to self-regulate their packet-specific access to shared network resources. The results are promising and show that the innovative PacketEconomy mechanism provides robust, effective and fine-grained QoS while maintaining end-user control for both rate- and window-based flows.

Chapter 7: A Game-theoretic Analysis of Preventing Spam over Internet Telephony via Audio CAPTCHA-based Authentication

Spam over Internet Telephony (SPIT) is a potential source of disruption in Voice over IP (VoIP) systems. The use of anti-SPIT mechanisms, such as filters and audio CAPTCHA (Completely Automated Public Turing Test to Tell Computer and Humans Apart) can prevent unsolicited calls and lead to less unwanted traffic. In this work, we present a game-theoretic model, in which the game is played between SPIT senders and Internet telephony users. The game includes call filters and audio CAPTCHA, so as to classify incoming calls as legitimate

or malicious. We show how the resulting model can be used to decide upon the trade-offs present in this problem and help us predict the SPIT sender's behaviour. We also highlight the advantages in terms of SPIT call reduction of merely introducing CAPTCHA, and provide experimental verification of our results.

Chapter 8: Conclusions and Directions

The overall conclusions of this work are presented in this chapter. A brief comparison of the component works in this thesis is conducted and the challenges and possible future directions regarding the field are discussed.

CHAPTER 2

Background

2.1 Networking

This thesis is concerned with the problem of resource allocation on networks. The specific network topics of interest include the structure of the networks used for modelling and experimental evaluation, the types of network flows transmitted over the networks, and the router queue management policies. In the following sections brief descriptions of these topics are presented, while more details can be found in [87, 102].

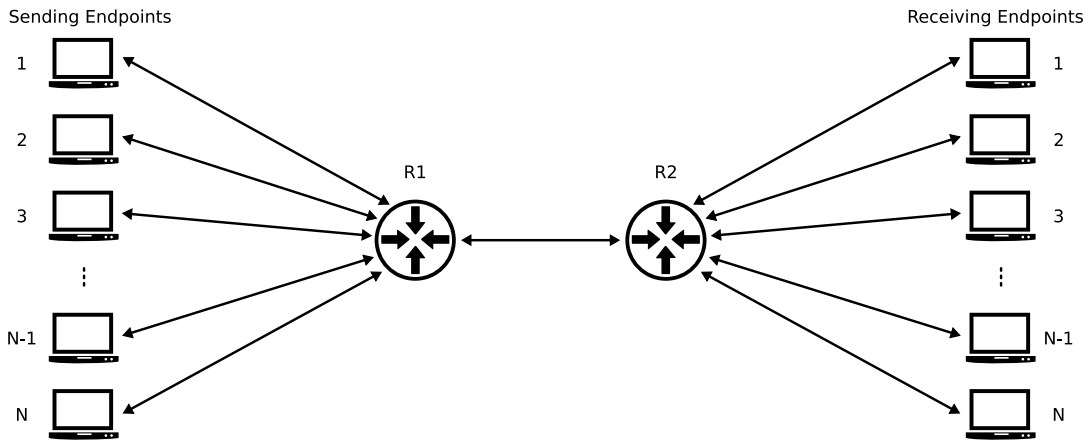


Figure 2.1: Structure of a dumbbell network with N hosts on each side ($2 * N$ total hosts) and 2 routers ($R1$ and $R2$) between them.

2.1.1 Network Structure

The network structure we commonly consider consists of a dumbbell topology, illustrated in Figure 2.1, with N hosts on each side ($2*N$ total hosts) and 2 routers ($R1$ and $R2$) between them. The hosts on the left are the sending endpoints and the hosts on the right are the receiving endpoints. Each host is connected via Ethernet with exactly one link to either $R1$ or $R2$. The connections between endpoints and routers, as well as the single connection between the two routers, are typically full duplex and have the same bandwidth and propagation delay. This structure captures the basic components of any network (multiple hosts, flows, and routers) as well as typically leads to network congestion, a important phenomenon commonly studied. At the same time, the structure is simple and regular enough for some analytic modelling to be performed while experimental results are reasonably generalisable to more complex networks.

2.1.2 Network Flow Types

The network flow types represented in this work are either window-based or rate-based. These two categories cover a wide range of flows and as a result are good proxies for real Internet traffic.

2.1.2.1 Window-based Flows

Window-based flows employ a feedback-based mechanism, the congestion window, which determines the maximum number of packets that the flow may have in-flight (i.e. being in transmission anywhere in the network). Every packet that is in-flight occupies one of the available positions in the congestion window of a window-based flow. The more a packet delays its arrival, the longer the following packet will have to wait to use the occupied window position.

Most window-based flows are implemented with the Transmission Control Protocol (TCP), which belongs to the Internet protocol suite and is the main reliable connection-oriented data transmission protocol of the Internet. TCP flows transmit their data by sending a series of packets. Assume a TCP flow that is ready to send a large volume of data as a sequence of packets. In order to send the data in a controlled manner, a first parameter w is used, called the size of the *congestion window*.

The TCP protocol dictates that the flow starts by submitting w packets to the network and then waits until one of two conditions are met: Either a packet's arrival is confirmed, normally by receiving a matching acknowledgement packet (ACK) within a certain time-frame, or the time-frame passes, whereby the packet is considered lost. As soon as the number of the in-flight packets of the flow is

less than w, the flow submits new packet(s); the result is that, at any moment in time, the flow can have at most w packets in flight. Thus, the size w of the congestion window has a strong impact on the transmission rate of a flow [42]. Consequently, the selection of an appropriate value for w is a very critical task for every flow, and this is where the AIMD (Additive Increase Multiplicative Decrease) scheme is useful.

The AIMD algorithm is the most popular procedure for a TCP flow to constantly adapt its window size to the changing network conditions. The basic principle of AIMD is that, for each successful packet delivery the flow increases its congestion window size additively by an amount proportional to a parameter $\alpha > 0$ (usually $\alpha = 1$) and for each lost packet, the flow decreases its congestion window multiplicatively by a parameter $0 \leq \beta < 1$ (usually $\beta = 1/2$). The values of the α and β parameters have a decisive role on the behaviour of the AIMD flow. A large value of α and/or β makes the flow more aggressive, whereas a small value makes it more temperate.

2.1.2.2 Rate-based Flows

Rate-based flows are simpler than window-based ones. Their operation is governed by the sending rate of the packets which is typically (almost) constant. These flows employ no feedback mechanism to control their sending rate and as a result they are also labelled unresponsive flows, since there is no way to signal to them that they should alter their sending rate depending on network conditions.

The protocol most commonly used for rate-based flows is the User Datagram Protocol (UDP), which, as TCP, belongs to the Internet protocol suite. Its main advantages in comparison to the TCP protocol is the lower overhead of the protocol headers (8 bytes for UDP versus 20 bytes for TCP) as well as being much simpler to implement due to the lack of feedback mechanism and reliable delivery guarantees.

2.1.3 Router Queue Management

Hardware-based routers fall into two large categories based on their maximum throughput: High-end routers and medium/low-end routers. High-end routers are typically employed in backbone networks and thus need to support extremely high throughput. To achieve this, they employ fixed-function dedicated and highly parallel hardware computation units (Network Processing Units - NPU) as well as specialized high-speed memory (Ternary Content Addressable Memory - TCAM). However, this comes at the cost of flexibility and customisability, as the algorithms which can be used by the router while maintaining

its high-speed processing are predetermined and implemented into hardware. Some parameters may be configurable but only to the extent predetermined by the manufacturer. Often, for the target applications these limitations may not be a problem, since backbone routers often do not have enough context in order to make flow-dependant routing choices. For example, one limitation which affects our system as well, is that it is impossible to perform packet re-ordering within the queue (the queue is strictly FIFO). If higher flexibility is desired, it is possible in many cases to use custom algorithms within these routers, however this is done at the expense of bypassing a part of the hardware-based pipeline through a software-based one. The immediate effect is that throughput drops significantly.

While these trade-offs have to do with high-end backbone routers, lower-cost middle- and low-end routers, which do not need to provide the same throughput since they are typically used near the leaves of the network, largely do away with the specialized and costly hardware implementation and use a software pipeline. As a result, it is much easier to implement custom algorithms on this class of routers.

Router queue algorithms can be classified according to their computational requirements. On one hand, there are stateless algorithms, which are lightweight and simple. Typical queue admission policies include DropTail and RED (Random Early Detection) [31]. The handicap of DropTail is its indiscriminate packet dropping mechanism, which causes unfairness. RED notifies more flows about congestion than DropTail by deploying a randomized dropping mechanism. RED also constrains the queue length between two thresholds in order to prevent overflow and high queueing delay.

On the other hand, there are stateful queueing policies, like Fair Queueing [19], which are sometimes too computationally demanding to be deployed at routers. Fair Queueing accomplishes the desired result (fairness) but at the cost of a separate queue for each flow and increased management complexity. In response, a variety of buffer management schemes were proposed that maintain a FIFO queue while trying to fairly allocate bandwidth. For example, Core-stateless Fair Queuing (CSFQ) [103] does not need to maintain state on core routers but it has to on the edge routers. Its disadvantage is that the architecture of the Internet has to be modified to allow routers to exchange messages relaying the flows' rate estimations. Other queueing policies use the history of packet drops (e.g. RED-PD [66]) or the history of the incoming packets (e.g. AFD [83]) to detect the aggressive flows. While these policies do not keep separate queues for each flow, they still require complex computations and extra buffering operations.

CHOKe [84] is based on the assumption that the queue content during congestion constitutes a sufficient statistic about the incoming traffic and provides

useful information about candidate flows for pruning. CHOKe penalises flows that overcome their fair share by deploying a probabilistic algorithm. Every incoming packet is compared with an already queued packet and if they match they are both dropped. The performance of this algorithm is good when only one misbehaving flow is traversing the router but degrades when more than one flow is aggressive.

2.2 Game Theory

A game is a mathematical model of the interaction among rational, mutually aware players. In this thesis we generally consider that players are selfish, strategic, and rational by having the objective to maximize their own pay-off. The pay-off of each player is determined by the outcome of the game, which in turn depends on the decisions (strategies) of all players. A strategy defines a set of moves or actions a player will follow in a given game.

A mixed strategy is a randomized strategy that assigns a probability to each pure strategy. The support of a mixed strategy is the set of actions to which it assigns a strictly positive probability. A strategy profile is a set of strategies that includes one and only one strategy for every player. Clearly, a strategy profile fully specifies a single execution of a game. A Nash equilibrium is a strategy profile were no player has an incentive to unilaterally deviate from their strategy.

We also refer to the concept of a weak Pareto improvement, which (in this context) is any change to the current strategy profile that makes every player at least as well off and at least one player strictly better off. For more details on the game-theoretic terms, the reader may refer to textbooks on Game Theory [80][79][81], or to a recent volume on Algorithmic Game Theory [77].

2.3 Related Work

In this work we address the fair and balanced distribution of resources (and in this case specifically network resources) to competing entities. In this field, network congestion has been described game-theoretically by Nagle [73] and the solution put forth used a market wherein the rules of the game would lead to the optimal strategy for the individual entities also being the optimal solution for the system. In a later work, Shenker [93] describes the relation between the selfish entities and the switch service mechanisms and proposes a method of guaranteeing efficient and fair operating points. Since then, the coordination of Internet entities has been modelled through various game definitions, some representative ones being [3, 85, 57] and overviews of which are presented in

[4, 77].

Certain game-theoretic approaches to congestion problems of the Internet, and especially the TCP/IP protocol suite, are discussed in [93, 3, 32, 25]. A combinatorial perspective on Internet congestion problems is given in [48]. The focus of the above works and the present work is on sharing the network resources between selfish flows.

The use of economic tools like pricing, tolls and taxes as a means to regulate the operation of networks and/or to support quality of service (QoS) functionalities in the presence of selfish flows is discussed in [78, 33, 16, 15, 65, 69]. In particular, the Paris Metro Pricing approach, using pricing to manage traffic in the Paris Metro, is adapted to computer networks in [78]. A smart market for buying priority in congested routers is presented in [65]. In [16, 15] taxes are used to influence the behaviour of selfish flows in a different network model. An important issue identified in [15] is that taxes may cause disutility to network users unless the collected taxes can be feasibly returned to the users.

In their seminal work, Kiyotaki and Wright [55] examine the emergence of money as a medium of exchange in barter economies. Subsequently, Gintis [34, 35] generalizes the Kiyotaki-Wright model by combining Markov chain theory and game theory.

CHAPTER 3

Prince: an Effective Router Mechanism for Networks with Selfish Flows

3.1 Introduction

Network congestion is a major issue on the Internet. Under congestion, networks struggle to allocate resources efficiently and fairly. Congestion builds up easily when some of the flows try to gain a large share of the network capacity, either by excessively increasing their sending rate or by not cutting back despite their packet losses. This situation, in which multiple selfish players can ultimately overload a shared resource even when it is obvious that it is not in anyone's long term interest, is an instance of the "Tragedy of the Commons" problem [41]. This behaviour leads to heavy congestion and threatens the stability and efficiency of the Internet.

Beyond these consequences, our greatest concern is the unfairness that arises. During congestion, misbehaving flows may retain their sending rate while well-behaved ones cut back. The result is that the misbehaving flows receive an unfair proportion of the bandwidth at the expense of the well-behaved flows. In this work, a game theoretic point of view is adopted. In these terms, network flows are selfish and independent players, the router's queueing algorithm is the game mechanism, the players' bandwidth requests constitute the set of possible strategies and the allocated bandwidth is the players' utility.

Achieving fairness and efficiency in the network can be translated to achieving a desirable Nash Equilibrium (NE) in the game theoretic model. In order to

accomplish this goal we turn to mechanism design. We opt not to try to control the flows but to give them incentives to act responsibly [85, 93]. We use the core elements of a network, the routers, to warn or diminish selfish flows. In a previous work [25], we analysed the Prince algorithm in an abstract network-game model and obtained interesting results. In this work, we adapt Prince to a realistic Internet-centric model. Our basic principle is to ground the packet dropping decisions on the buffer contents. In particular, at every congestion, a packet from a flow with the largest number of packets in the buffer, i.e. a majority flow, is dropped. We present three versions of Prince: Prince-G precisely implements the basic principle, Prince-S is a more vindictive instance of the basic principle and Prince-A approximates Prince-G with a data stream algorithm.

The novelty of this work is the application of game theoretic incentives in a real network in order to accomplish fairness among the players, while at the same time employing a lightweight mechanism on the routers. The mechanism is a new active queue management scheme which resembles MaxMin fairness by protecting the fair share of well-behaved flows. We do not achieve this by trying to implement a strict instance of MaxMin by continuously controlling every flow. Rather, and this is our innovation, we apply either moderate (Prince-G) or strong (Prince-S) incentives to the aggressive player who stresses the router most during congestion. This will force any rational player to back off in order to avoid further detriment to his utility. We study Prince and provide theoretical arguments and extensive experimental results. For the latter, we experimented with TCP, UDP and mixed TCP and UDP flows of varying aggressiveness and we compared Prince against other popular queueing policies such as DropTail, RED, CHOKe and MaxMin. Additionally, we propose a low complexity approximation of Prince to allow for an almost stateless router implementation.

3.2 Related Work

Nagle [73] proposed a game-theoretic view of network congestion and suggested a market solution according to which the rules of the game should be set in such a way, so that the optimal strategy for the individual user results in an optimal situation for all users. Shenker [93] correlates the selfish behaviour of the users with the design of the switch service disciplines and suggests a fair share scheme which guarantees efficient and fair operating points. Other researchers also tried to model the interaction between Internet users with various game definitions [3, 25, 85, 93] and emphasized the importance of mechanism design in this process.

Router queue algorithms can be classified according to their computational requirements. On the one hand, there are stateless algorithms, which are

lightweight and simple. For similar games to ours, it has been proven that DropTail or RED routers lead to undesirable NE when modern TCP flows (e.g. SACK) participate [3, 24]. The handicap of DropTail is its indiscriminate packet dropping mechanism, which causes unfairness. RED [31] notifies more flows about congestion than DropTail by deploying a randomized dropping mechanism. RED also constrains the queue length between two thresholds in order to prevent overflow and high queueing delay. The drawback is that RED imposes the same loss rate for all flows, therefore a flow has no incentive to be socially responsible.

On the other hand, there are stateful queueing policies, like Fair Queueing [19], which are too computationally demanding to be deployed at routers. Fair Queueing accomplishes the desired result (fairness) but at the cost of a separate queue for each flow and increased management complexity. In response, a variety of buffer management schemes were proposed that maintain a FIFO queue while trying to fairly allocate bandwidth. For example, CSFQ [103] does not need to maintain state on core routers but it has to on the edge routers. Its disadvantage is that the architecture of the Internet has to be modified to allow routers to exchange messages relaying the flows' rate estimations. Other queueing policies use the history of packet drops (e.g. RED-PD [66]) or the history of the incoming packets (e.g. AFD [83]) to detect the aggressive flows. While these policies do not keep separate queues for each flow, they still require complex computations and extra buffering operations.

CHOKe [84] is based on the assumption that the queue content during congestion constitutes a sufficient statistic about the incoming traffic and provides useful information about candidate flows for pruning. CHOKe penalizes flows that overcome their fair share by deploying a probabilistic algorithm. Every incoming packet is compared with an already queued packet and if they match they are both dropped. The performance of this algorithm is good when only one misbehaving flow is traversing the router but degrades when more than one flow is aggressive. Another approach was also based on the same queue management guidelines and a game theoretic model [32]. Despite that it also aims at the highest rate flow, it requires delicate refinement of the in-between queue thresholds. Additionally, its dropping policy does not shield the fair share when the queue usage is above the predefined high threshold.

3.3 The Prince Algorithms

As already stated, our goal was to design a game mechanism for the network which provides incentives to the flows to behave in a socially responsible manner. The design criteria we used for the mechanism should:

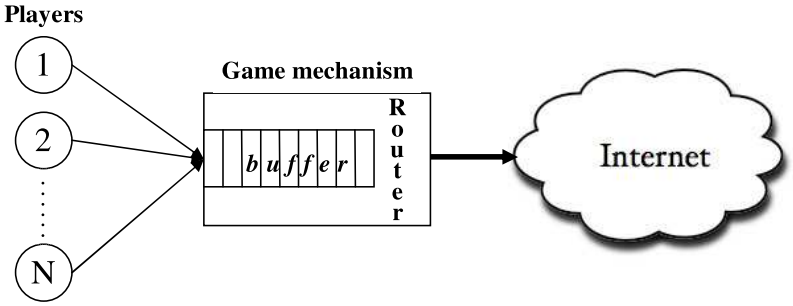


Figure 3.1: The game model

- Lead the game towards a desirable NE
- Provide a stateless and simple implementation.
- Not depend on being deployed on the whole network.

Based upon our criteria, we propose the Prince mechanism, which uses the router queue and focuses on the majority flow in it, i.e., the flow with the most packets.

Our algorithms work on a FIFO router queue and drop packets only during congestion. We define three implementations with different trade-offs:

- Prince-G (*G*entle) drops a packet from the majority flow whenever a packet drop is required.
- Prince-S (*S*evere) marks all the majority flow's packets and drops one of the marked packets whenever a packet drop is required.
- Prince-A (*A*daptive) emulates Prince-G with a data stream algorithm adapted from [49].

3.3.1 Theoretical Arguments

We will introduce a simple but concise game definition in order to specify the model under analysis. The game that represents the interaction between the flows and the Internet infrastructure (Figure 3.1) is the following:

- The n players of the game are the flows that compete for the common resource (link capacity).
- The moves available to each player are:
  - set the AIMD parameters (α,β) for TCP flows,

  - set the constant sending rate for UDP flows.

- The mechanism of the game is the router's packet dropping protocol.
- The goal of each player is to maximize their utility function (e.g. maximizing goodput).
- The solution concept of the game is the Nash Equilibrium.

A desirable NE for the above game is characterized by efficiency and fair bandwidth allocation. While fairness can be defined in multiple ways, we consider the MaxMin Fairness criterion [19] to be the most appropriate for our model. According to MaxMin, a set of rates is fair if no rate can be increased without simultaneously decreasing another, smaller, rate. MaxMin Fairness results in an equal share of the bottleneck link for each flow traversing it [51] unless a flow requests less than its fair share. In this case, the frugal flow receives the bandwidth it requested, and the remaining capacity is distributed equally to the more greedy flows.

The Prince algorithm attempts to protect the fair share of each player in the game. In essence, the Prince-G algorithm resembles the MaxMin Fairness bandwidth allocation by minimizing the majority flow's sending window and sharing the released bandwidth with the rest of the players. Every time a new packet arrives at the queue the Prince-G algorithm is triggered. If the queue is full, then a decision has to be made on which flow's packet to drop. As the following lemma shows, Prince and MaxMin both decide on a flow with the maximum number of packets.

Lemma 1 *The Prince-G policy implements MaxMin Fairness for buffer sharing.*

Proof 1 *Assume a Prince-G router with queue size C and a set of n flows. Assume that the queue is full and that a new packet has just arrived at the router. Hence, a total number of $C+1$ packets are currently at the router. Let $w_1, w_2, \dots, w_n$ be the number of packets that belong to flows $1, 2, \dots, n$, respectively. Without loss of generality we can assume that*

$$w_1 \leq w_2 \leq \dots \leq w_n \ . \tag{3.1}$$

The Prince-G policy will drop a packet from the flow n with the largest number of packets in the queue (ties are solved randomly). This way Prince-G implements the MaxMin criterion.

Lemma 2 *In both of the Prince-G and Prince-S policies, a flow that did not exceed its fair share in the queue buffer, does not lose any packet. Furthermore, in Prince-G, a flow is never forced to have a buffer share smaller than its fair share.*

Proof 2 *As in Lemma 1 assume a router with queue size C and a set of n flows. A new packet has just arrived while the queue is full. Let $w_1, w_2, \ldots, w_n$ be the number of packets that belong to flows $1, 2, \ldots, n$, respectively, and assume relation 3.1 holds.*

By combining relation 3.1 with

$$\sum_{i=1}^{n} w_i = C + 1 \,, \tag{3.2}$$

we can show by contradiction that the number of packets of flow n is $w_n > C/n$. Clearly, if $w_n \leq C/n$ then $\sum_{i=1}^{n} w_i \leq n \cdot (C/n) = C < C + 1$, a contradiction.

We conclude that in both Prince-G and Prince-S, a flow that has not exceeded its fair share cannot experience packet drops. Furthermore, since Prince-G identifies a flow with the maximum number of packets each time a packet has to be dropped, a flow is never pushed strictly below its fair share.

We consider the above lemmas to be evidence that our algorithms and especially Prince-G lead the game to desirable NE. Further evidence is provided by the experimental results in Section 3.5.

3.3.2 Algorithm Descriptions

We examine three algorithms that embody the basic principle of Prince, i.e., dropping packets from the majority flow. All three algorithms operate by dropping packets when the router experiences congestion, that is, when the router queue is full and another packet arrives for which there is no more space. The algorithms are differentiated by the way they select which packet to drop under such circumstances. We consider a router queue with C packets and n unique flows.

Prince-G

The Prince-G algorithm scans the queue and counts the packets of each flow whenever a packet needs to be dropped. Then it drops the first[1] packet in the buffer of the most frequent flow, making space for the new packet to enter the queue. If the new packet belongs to the most frequent flow in the queue, then only this packet is dropped immediately.

Complexity

Building the list of frequencies per flow can be achieved in $O(C)$ amortized time, by using a single pass over the queue and accumulating the counts in a hash-table-based dictionary (key:flowid, value:packet count). This time complexity

[1] to quickly alert the flow about congestion

can be improved to $O(1)$ worst case with high probability if one of the hashing algorithms of [20] or [7] is used. The most frequent flow can be identified within the same process. The required space is $\Theta(\min\{C, n\})$.

Prince-S

The Prince-S algorithm retains a list of marked packets which are candidates for being dropped. To create the list, we execute once what is essentially a two-pass Prince-G algorithm resulting in all of the majority flow's packets being marked (one pass to find the majority flow as in Prince-G, one pass to mark all the majority flow's packets). If the queue experiences congestion and there are no marked packets in the queue, the list is created on-demand and then the first marked packet is selected immediately for dropping. On the other hand, if the list already contains marked packets then the marking process is not executed and the next marked packet in the list is dropped.

Complexity

Building the list of frequencies per flow is achieved in the same $O(C)$ time as Prince-G. The marking of the most frequent flow's packets, w_{max} in number, can be stored in a linked list in time $O(C)$ and in space $\Theta(w_{max})$. Dropping a marked packet can be achieved in $O(1)$ time.

Prince-A

Prince-A is the window-based adaptation of the data stream algorithm of Karp et al. [49]. The data stream technique identifies the top-k heavy hitters in order to approximately spot the majority flow while being as lightweight as possible at the same time. Prince-A uses only a limited number of counters (k) which is significantly less than the queue capacity. The purpose is to implement the Prince algorithm with less computational resources.

When a new packet arrives, irrespective of congestion, the original algorithm is executed and the flow who sent the packet may or may not get a counter. More precisely, the router examines if the flow that the incoming packet belongs has already a counter. If it already has a counter then this counter is incremented by one. If it doesn't, first checks if there is an empty counter to correlate it with the current flow or else decrements all counters by one.

The adaptation consists of triggering when a packet is either served or dropped. In these cases, if the packet's flow had a counter associated with it, its value is decremented by one. This function allows fast adaptation to changing network conditions.

Complexity

The more complex implementation for storing the counts, also proposed in [49] is used, allowing for $O(1)$ worst case with high probability time complexity when a packet arrives. Space complexity in this implementation is $\Theta(1/\theta + c)$, where c is the largest frequency and $1/\theta$ is the maximum number of counters used. Both have a small upper bound: $c, 1/\theta \leq \min\{C, n\}$. When a packet is served or dropped, the time complexity is the same $O(1)$ as when one arrives.

Additionally, if one is willing to trade space complexity for time complexity, it is possible to substitute the approximate Prince-A algorithm with HL-HITTERS, an exact $O(1)$ worst case with high probability time and $O(C)$ space complexity algorithm for finding the heaviest-k hitters described in Chapter 4.

3.3.3 Effects of the Packet Size Assumption

In this work, we have focused on scenarios with packets of equal size and showed that Prince handles them very well. Indeed, the case of packets with different packet sizes is very important for network routers. In brief, the Prince-G and Prince-S algorithms can be adapted to count the total size of the packets of each flow and then drop one or more packets from the majority (in bytes) flow. For Prince-A this approach does not apply. However, we can still handle packets of various sizes by exploiting the fact that the size of IP packets does not vary more than a constant factor. Thus, for Prince-A we can consider a minimum packet size (mps) and handle any larger packet as being k minimum packets for some appropriate integer k. The data structure of HL-HITTERS can continuously monitor the majority flow in a router queue with a time complexity of $O(1)$ worst case with high probability per packet. This data structure can also be adapted to packets of variable size with the same trick as above in Prince-A.

3.4 Discussion

The Prince mechanism embodies the following fundamental game theoretic principle. At the moment of congestion, we drop packets from the player who contributes the most to the congestion. As a result, his utility diminishes if he continues to be aggressive. This is a strong incentive for a selfish but rational player to back off, when he wants to maximize his utility function, even if packet loss has a minor cost for him. At the same time, Prince ensures that well-behaved players receive appropriate service. The power of this technique lies in that we need only target the most aggressive player to motivate *all* the players to behave well. Even though all players desire the largest possible proportion of

the link capacity, no one will want to have the maximum share because of the penalty. Since it is not possible for a player to find out the shares of the other players, he will have to be careful not to request too much bandwidth in order not to become the most aggressive one. The result is that the players restrain themselves to avoid the penalty, until no congestion is present.

We should note that both Prince-G and Prince-S are motivated by the same principle, i.e. punishing the most aggressive player, but they accomplish this using different means and have slightly different results. Arguably, Prince-S is the most "vengeful" of the two. It will invariably provide the strongest incentive to moderate aggressiveness, at the expense of being less sensitive to majority flow fluctuations due to the lag between majority flow re-evaluations. It will also be more computationally lightweight, on average, than Prince-G.

The Prince algorithms presented in this work implement work-preserving queue disciplines that drop packets from the router queue only in case of overflows and, even then, the minimum possible number of packets is dropped. When there is no overflow, every flow is granted the buffer capacity it requests. During overflows, the Prince algorithms implement (Prince-G) or approximate (Prince-S, Prince-A) MaxMin fairness for queue buffer sharing. MaxMin fairness is considered, in general, one of the most effective ways to handle resource sharing for heterogeneous (and homogeneous) demands.

Under the above perspective, Prince is a queueing mechanism that can either enforce socially responsible behaviour on a misbehaving player or cooperate with a player who has the following desirable features:

- Adoption of end-to-end congestion control, that is, being responsive to packet losses by throttling down upon congestion and throttling up to discover the fair share.

- Self-optimization by taking into account the packet losses in the utility function.

It should be noted that the buffer size plays an important role in the Prince algorithm. On the one hand, using a large buffer provides us with a good approximation of the players' sending windows. The more packets the buffer contains at congestion, the better our queue snapshot captures each flow's contribution. On the other hand, a large buffer creates more queueing delay for all the flows traversing the router and extra computational cost to the router's overall job. However, in our experiments we obtained fair bandwidth allocations even with small buffer sizes.

3.5 Experiments

3.5.1 Experimental Setup

We carried out a large set of experiments on the established ns2 network simulator [1]. As a first step, we verified that Prince manages to shield the fair share of the well-behaved flows by reducing the bandwidth of the aggressive players. We also examined the efficiency of our algorithm by monitoring its achieved goodput, loss rate and fairness. Finally, we used the heuristic methodology of [3] to find symmetric NE for our game and then evaluated its efficiency.

This methodology is executed in iterations. In the first iteration, we set $\alpha^1 = 1$ for flows $1, \dots, n-1$ and search for the best response of flow n. Let $\alpha^{1,best}$ be the value α, with which n achieves the best goodput. In the next iteration, flows $1, \dots, n-1$ play with $\alpha^2 = \alpha^{1,best}$ and we search for the best α_n in this profile. If at iteration k, $\alpha^{k,best} = \alpha^k$ then this value, denoted by α_E, is the SNE of the game.

Furthermore, we defined the Normalized Fairness Index (NFI) which is the Fairness Index normalized to the MaxMin Fairness bandwidth allocation, in order to measure the distance between the bandwidth allocation of Prince and MaxMin. The NFI is given by:

$$f(x_1, \dots, x_n, y_1, \dots, y_n) = \frac{(\sum_{i=1}^{n} \frac{x_i}{y_i})^2}{n \sum_{i=1}^{n} (\frac{x_i}{y_i})^2}$$

where x_i is the goodput of the i-th flow using the under examination algorithm and y_i is the goodput of the same flow achieved with MaxMin (DRR implementation).

We selected a simple dumbbell topology with two set of parameters. The first set (Topology 1) defines a topology with a bottleneck connection of 10Mbps/10ms (Bandwidth/Delay) and source/sink connections of 10Mbps/1ms. The queue size of the congested router is set to the Bandwidth × Delay product (BWxD), which is 25 packets. The second set (Topology 2) uses a topology with bigger capacity; 100Mbps connections. For this set, the queue size is 100 packets, which is significantly less than BWxD packets (250), in order to examine the effectiveness of Prince under limited information.

The number of the players in the game was 10 for the first topology and in the range 10 ... 100 for the second one. The players were TCP, UDP or mixed TCP and UDP flows. The TCP flows could define their strategy by selecting the value for the additive increase parameter α from 1 (standard TCP value) to 20. We have chosen the TCP SACK version for the implementation of the loss recovery mechanism because it is widespread and tolerant to packet losses.

UDP flows can define their strategy by selecting their constant sending rate from the fair share value to the bottleneck's bandwidth value.

We evaluated the performance of Prince and compared it to MaxMin, DropTail, RED and CHOKe. For MaxMin and RED we used the default implementations of ns2 while for CHOKe (which was not available) we used the implementation from [111]. Each experiment starts with a 10sec period for stabilization and continues with 100sec for measurements. The flows start randomly between 0 ... 1 sec and use a constant packet size of 1Kbyte. The minimum and maximum thresholds for RED and CHOKe were set automatically, depending on the link bandwidth and delay. The ideal MaxMin Fairness policy was represented by DRR (Deficit Round Robin) [40] with the number of queues equal to the number of players. The number of counters for Prince-A was set according to the queue size and number of flows of each experiment; in the following figure legends, the number of counters used appears parenthesized.

3.5.2 Results

3.5.2.1 Synthesis of TCP Flows

This synthesis was examined with both topologies and various aggressive players. Using Topology 1 we ran experiments with nine standard TCP players and an aggressive one that changes his additive increase parameter α from 1 to 20 in a series of identical games. The results showed that the aggressive player gains at most 15% more than his fair share under Prince-G and Prince-A, and at most 25% under Prince-S (Figure 3.2). Note the inability of DropTail to restrict the aggressive flow. RED has similar performance to DropTail and is omitted from the figure for clarity.

With MaxMin or CHOKe the aggressive player has goodput below his fair share for all α values except $\alpha = 1$, but the loss rate for CHOKe is higher (over 10%) than Prince-G (max 5%) and the total goodput is lower (1150 versus 1250 packets/sec). Prince-G sets an upper bound to the goodput of each player and a lower bound which is close to the fair share. Therefore the Fairness Index of Prince-G is close to 1 regardless of the aggressiveness of the player (Figure 3.3). DropTail has similar performance to RED and is omitted from the figure for clarity.

For Topology 2 with 99 standard TCP players and an aggressive one, all variants of Prince manage to track and restrict the selfish player (Figure 3.4), having similar loss rate and goodput with MaxMin and CHOKe. A direct comparison of Prince-G to MaxMin (Figure 3.5) showed that the difference between the goodput of the standard and the aggressive player is lower under Prince-G, achieving a better Fairness Index.

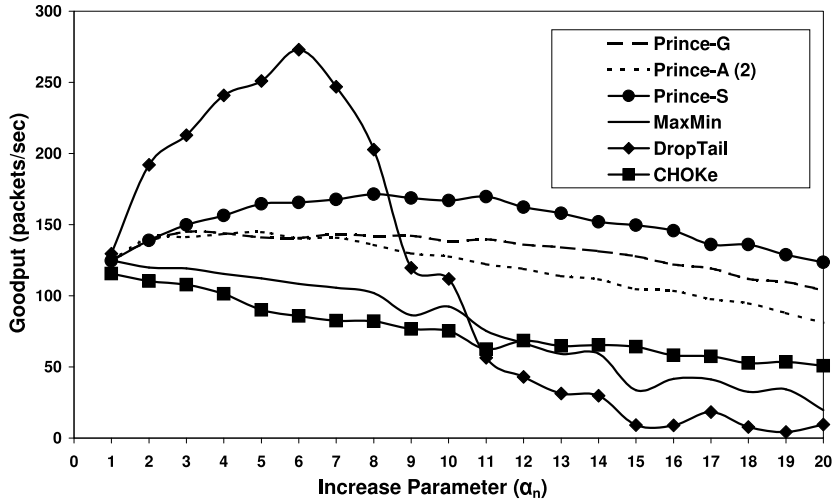


Figure 3.2: Goodput of the aggressive TCP player

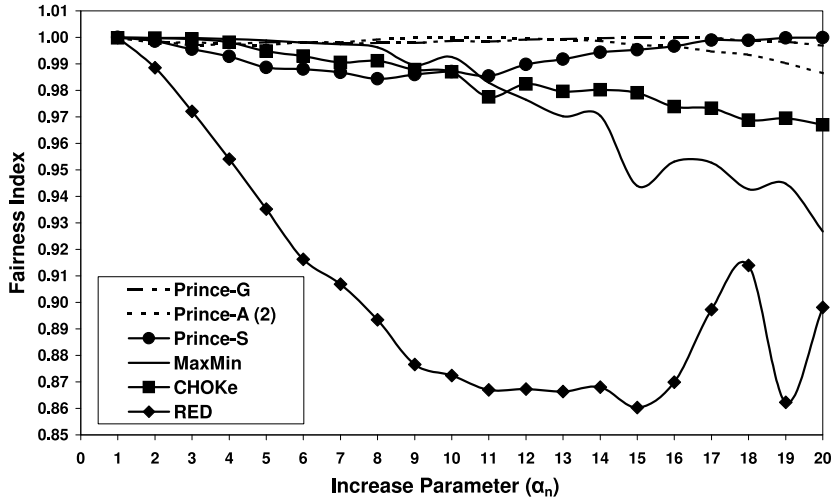


Figure 3.3: Fairness Index

Furthermore, we performed additional experiments with larger numbers of aggressive players and found that Prince-G's performance advantage increases. For the same topology with 90 standard and 10 aggressive TCP players, Prince-A achieves to moderate the aggressive players despite using only 10 counters. Prince-G and Prince-S can easily detect the aggressive flows. This is due to the fact that standard players are more rarely the majority players when many greedy players participate, so their fair share is guaranteed. Moreover, the more aggressive a player is, the easier it is for Prince to protect the standard players. On the contrary CHOKe fails when many selfish flows participate and

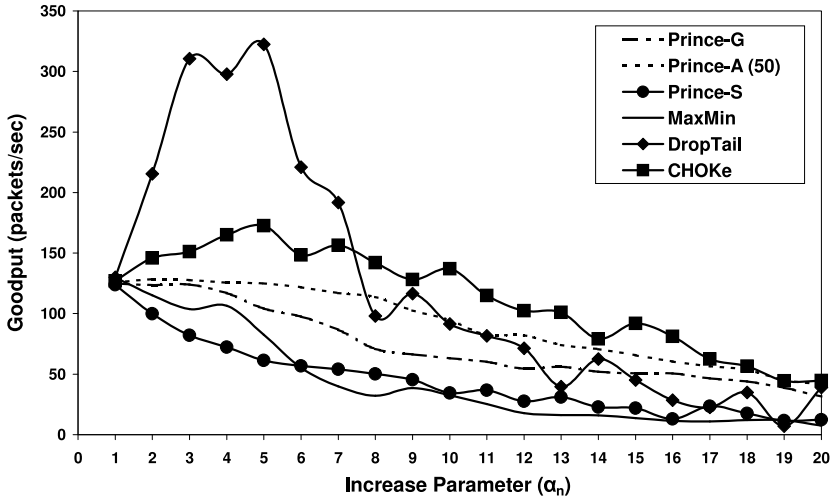


Figure 3.4: Goodput of the aggressive TCP player

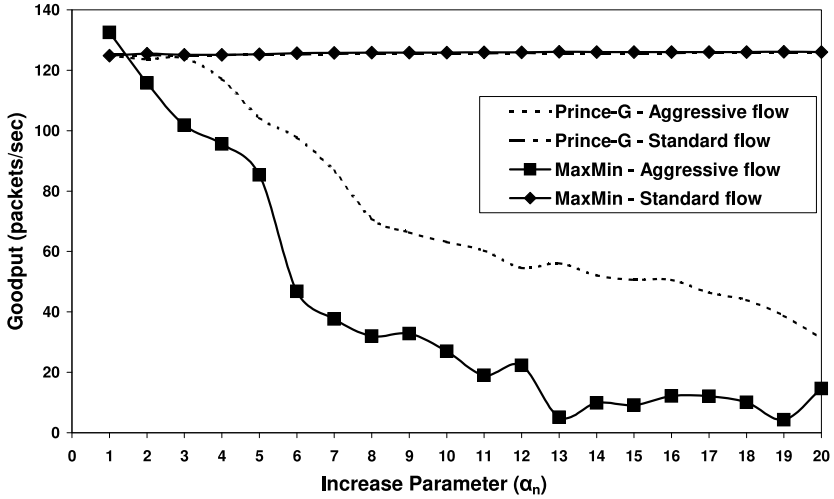


Figure 3.5: Prince-G Vs. MaxMin

the deficiency of RED and Droptail is also obvious on Figure 3.6.

On Figure 3.7 a direct comparison of Prince-G and CHOKe is depicted. Prince-G shields the fair share of the standard players no matter how aggressive the players are. As the aggressive players increase their parameter α the difference between them and the standard players becomes more pronounced and thus Prince-G can more easily safeguard the latter. CHOKe seizes the selfish players only when they choose high values for parameter α ($\alpha > 10$).

Prince-A is highly effective when many selfish TCP flows are traversing the same bottleneck. The convergence of the goodput between the standard and the

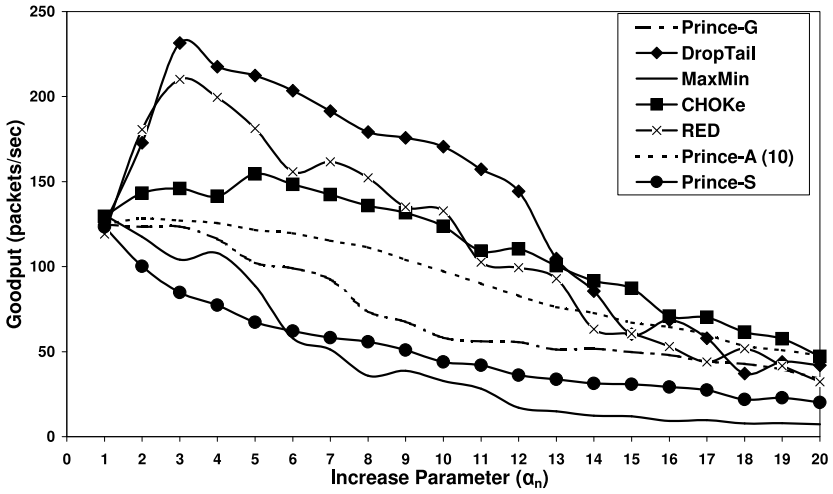


Figure 3.6: Average Goodput of the aggressive TCP players

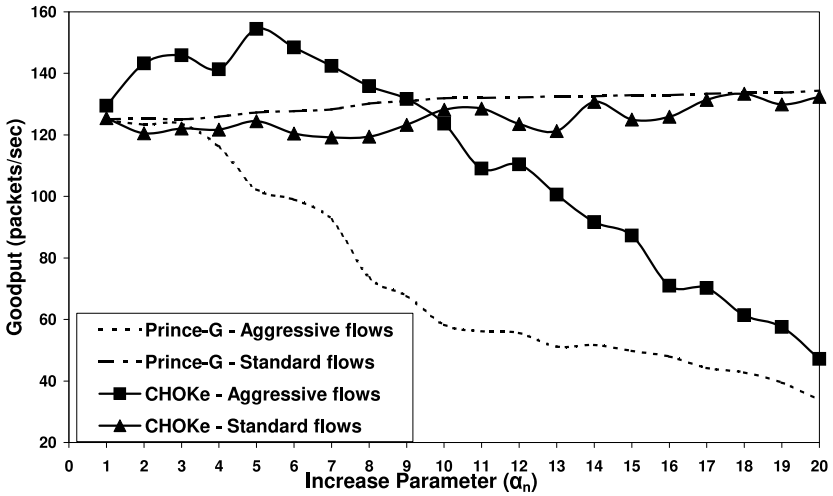


Figure 3.7: Prince-G Vs. CHOKe

aggressive flow is depicted on Figure 3.8. For $\alpha < 8$, Prince-A allocates equally the bandwidth between standard and aggressive flows, while RED encourages players to behave greedily.

3.5.2.2 Synthesis of UDP Flows

For Topology 1, we use nine UDP players with sending rate equal to their fair share (1Mbps) and one aggressive player that chooses his rate in the range 1 ... 10 Mbps for each game. It is evident that only Prince-G and MaxMin can minimize the greedy player, while DropTail and RED fail (Figure 3.9).

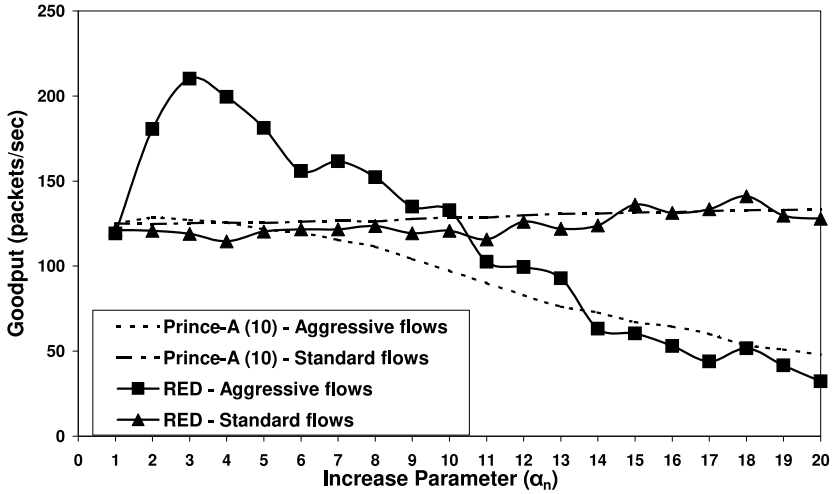


Figure 3.8: Prince-A Vs. RED

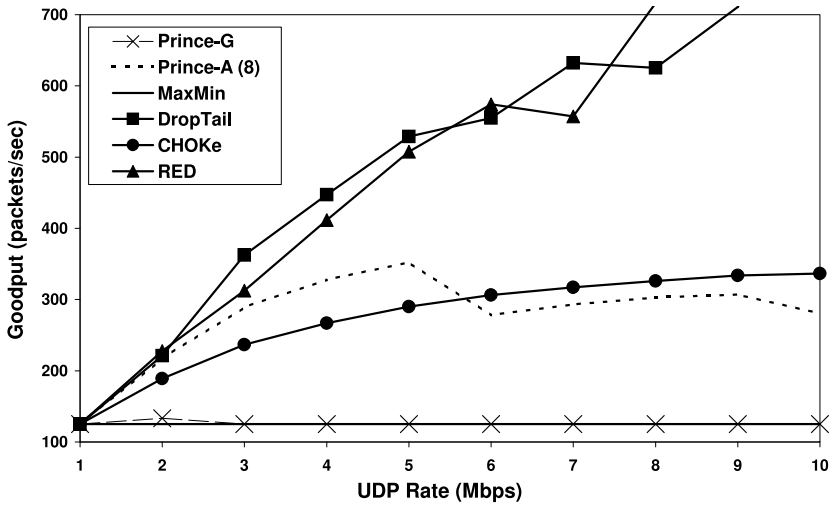


Figure 3.9: Goodput of the aggressive UDP player

Prince-S has identical performance to Prince-G and is omitted. CHOKe does not effectively minimize the selfish player, therefore standard players suffer losses. A UDP flow sending at the fair share cannot be the majority player in the buffer. Therefore, Prince-G shields its fair share and achieves a Fairness Index equal to 1 (> 0.99), just like MaxMin (Figure 3.10).

For Topology 2, we used 90 standard UDP players and 10 aggressive players that choose their rate in the range 1 ... 100 Mbps for each game. The effectiveness of Prince-G is depicted in Figure 3.11, where the fair share of the standard UDP players is shielded even better than by MaxMin. For MaxMin, the goodput of the standard players is less than the fair share because the queue capacity is less

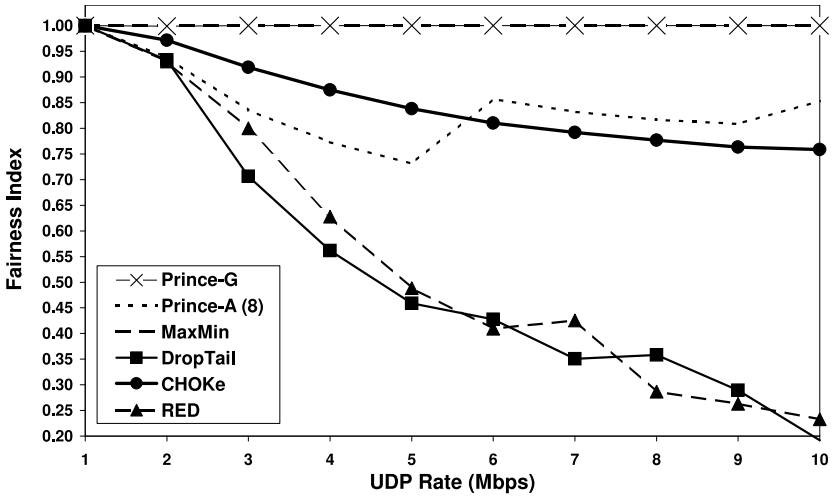


Figure 3.10: Fairness Index

than the BWxD product.

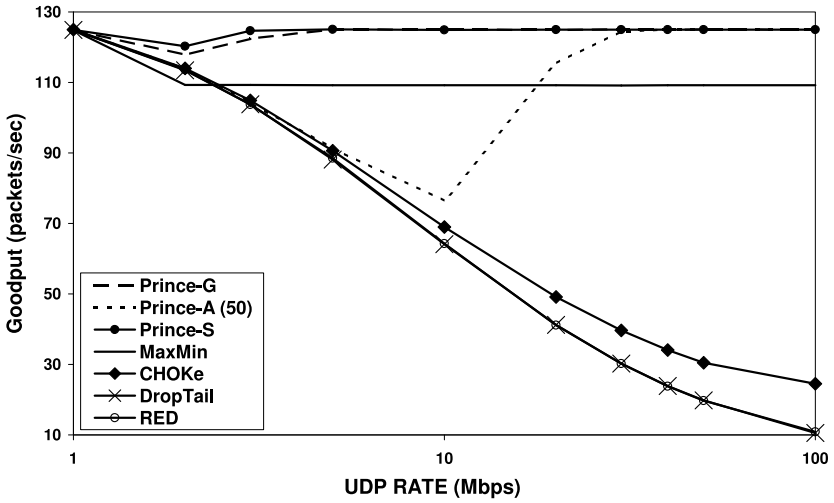


Figure 3.11: Goodput of the standard UDP player

3.5.2.3 Mixed Synthesis of TCP and UDP Flows

It is important to examine the efficiency of our queueing mechanism with diverse player sets. Therefore, in Topology 1, we use four standard TCP players ($\alpha = 1$) and four standard UDP players (1Mbps) as well as one aggressive TCP player with $\alpha = 2$ and one aggressive UDP player with a 10Mbps sending rate. In Figure 3.12, we see that Prince resembles MaxMin Fairness for the aggressive

UDP player, unlike DropTail, RED and CHOKe.

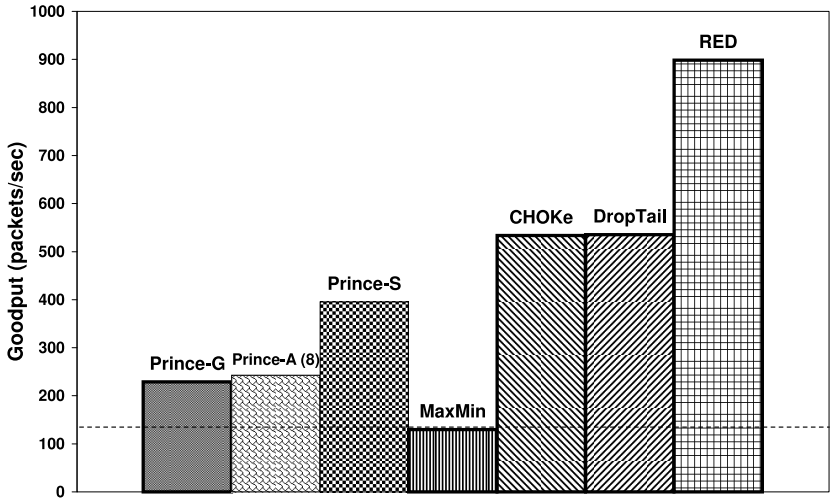


Figure 3.12: Goodput of the aggressive UDP player

Moreover, the aggressive TCP flow is also limited to the fair share (Figure 3.13). With RED and CHOKe all the TCP players are deprived of their fair share (equal to 125 packets/sec), while with DropTail the aggressive TCP player obtains 30% more than his fair share.

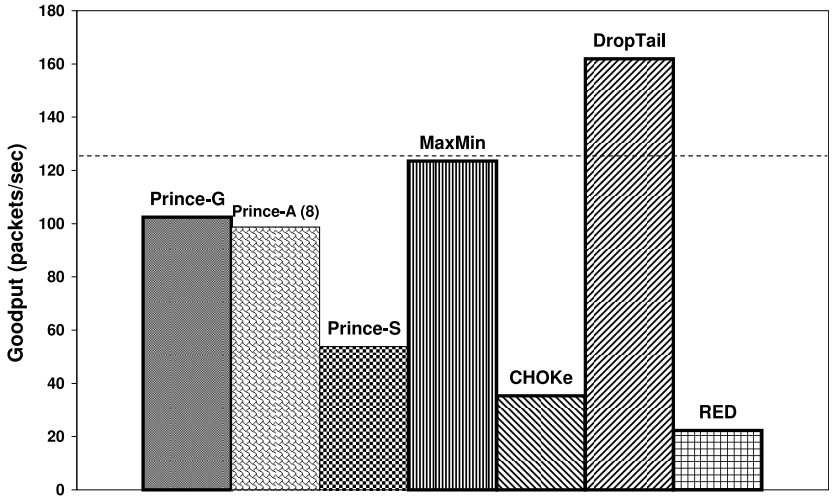


Figure 3.13: Goodput of the aggressive TCP player

The convergence of Prince-G to MaxMin is more clear by using the Normalized Fairness Index, shown in Figure 3.14. Moreover, Prince-G ensures a fair allocation of bandwidth to all the players and as a consequence achieves a high Fairness Index.

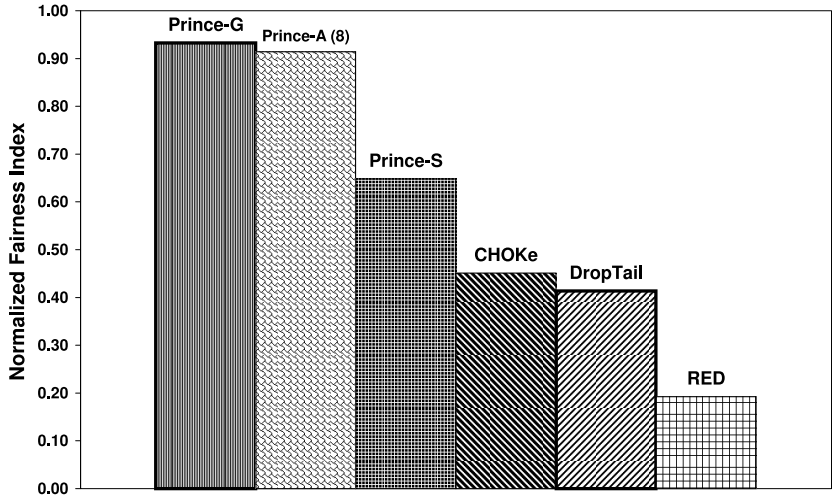


Figure 3.14: Normalized Fairness Index

3.5.2.4 NE Results

We used the aforementioned methodology to heuristically find a symmetric NE of the game, with either only TCP or only UDP flows. For the TCP game, a part of the results can be deduced directly from Figure 3.2. If the mechanism of the game is Prince-G, Prince-A or MaxMin, then the player has nothing to gain by increasing his additive increase parameter α beyond the standard TCP value. For Prince-S, CHOKe, RED and DropTail, the derived NE are less desirable due to the high loss rate and the slightly reduced goodput (Figure 3.15).

Queue Policy	α_E	goodput packets/sec	loss rate (%)
Prince-G	1	124,9	5,62
Prince-S	1	123,0	12,12
Prince-A	1	124,9	5,65
MaxMin	1	124,9	5,71
DropTail	2	122,5	8,26
RED	2	122,8	8,19
CHOKe	2	121,7	9,26

Figure 3.15: Efficiency of NE with TCP players

For the UDP game, MaxMin, Prince-G and Prince-S lead to efficient and fair NE because a player has equivalent performance for almost every available

sending rate (Figure 3.9). The other queueing policies fail to control the aggressive players, so the game results in an unfair and inefficient NE (Figure 3.16).

Queue Policy	sending rate (Mbps)	goodput packets/sec	loss rate (%)
Prince-G	1	125,0	0,0
Prince-S	1	125,0	0,0
Prince-A	5	125,0	79,51
MaxMin	1	125,0	0,0
DropTail	10	125,0	89,12
RED	10	125,0	89,89
CHOKe	10	125,0	89,99

Figure 3.16: Efficiency of NE with UDP players

3.5.2.5 Comparison

The three variants of Prince express the same game theoretic idea but do not always achieve equivalent results. Prince-G adopts a moderate treatment to limit aggressive flows, so it leads the game to a desirable NE. Prince-A can achieve similar performance to Prince-G, despite its stateless implementation, in certain problem classes. It allows us fine grained control over the complexity/performance trade-off, by selecting the desired number of counters. When the number of counters reaches the upper limit, i.e., the maximum queue size, then we obtain a streaming version of Prince-G. Prince-S features lower computational complexity than Prince-G at the expense of increased loss rate at the NE due to the aggressive penalization of the majority flow.

We ran experiments to evaluate whether Prince-S is computationally less intensive than Prince-G. At the same time we examined the severity of Prince-S, namely, how often Prince-S drops a packet from the last majority flow even though the majority flow has in the meantime changed to another flow. The following Figures (3.17, 3.18 and 3.19) show how many packets were dropped with Prince-S in relation to the aggressiveness of the greedy flow(s). In particular, the third column shows how many already marked packets were dropped and originated from the current majority flow. The fourth column shows the same, except that these packets were dropped from a flow that is no longer the majority flow. Finally, the last column shows how many times Prince-G ran on behalf of Prince-S, i.e. no marked packets existed in the queue.

For the TCP synthesis on Topology 1 (Figure 3.2) we can discern that although Prince-S restricts the aggressive player less than Prince-G, it also needs to compute the majority flow 60% less often than Prince-G. (Figure 3.17).

Increase parameter α_n	Loss rate	Dropped packets		
		Prince-S (hit on the majority flow)	Prince-S (hit on a non majority flow)	Prince-G (deployed by Prince-S due to lack of marked packets)
1	5,5%	1863	3008	3218
2	5,9%	2132	3316	3308
3	6,4%	2371	3504	3334
4	6,7%	2479	3779	3374
5	6,9%	2717	4012	3354
6	7,2%	2834	4233	3289
7	7,5%	3017	4309	3307
8	7,7%	3163	4504	3305
9	7,7%	3243	4467	3311
10	8,0%	3373	4513	3266
11	8,0%	3463	4599	3248
12	8,1%	3759	4543	3217
13	8,3%	3892	4683	3212
14	8,2%	3778	4455	3209
15	8,1%	3858	4559	3255
16	8,2%	3920	4583	3205
17	8,3%	3821	4438	3145
18	8,2%	3867	4395	3195
19	8,1%	3997	4384	3164
20	8,1%	3874	4325	3152

Figure 3.17: Prince-S with TCP synthesis

For the UDP synthesis, Prince-S has the same efficiency as Prince-G on limiting the aggressive flow. The results (Figure 3.18) for this corner case show that Prince-S periodically deploys Prince-G (from 33% to 6% in inverse proportion to the aggressiveness of the UDP flow) while the effect is the same. The column which shows the hits on a non majority flow is replaced by the Prince-G deployment percentage column, because a standard CBR flow cannot be marked as a majority flow (never exceeds its fair share). Finally, in the mixed synthesis the deployment of Prince-S succeeds in the restriction of the aggressive TCP flow but fails to diminish the fairly greedy UDP flow (Figure 3.12). However, its effectiveness is quite good considering that it executes Prince-G for only 10% of the dropped packets (Figure 3.19).

UDP rate (Mbps)	Loss rate	Dropped packets		
		Prince-S (hit on the majority flow)	Prince-G (deployed by Prince-S due to lack of marked packets)	Prince-G deployment percentage
1	0 %	0	0	0%
2	9,1%	9075	4538	33,3%
3	16,6%	20489	6830	25,0%
4	23,0%	34110	6822	16,6%
5	28,5%	47756	6828	12,5%
6	33,3%	61436	6835	10,0%
7	37,5%	73735	8201	10,0%
8	41,1%	86318	9087	9,5%
9	44,4%	102216	7205	6,6%
10	47,4%	112771	10254	8,3%

Figure 3.18: Prince-S with UDP synthesis

Loss rate	Dropped packets		
	Prince-S (hit on the majority flow)	Prince-S (hit on a non majority flow)	Prince-G (deployed by Prince-S due to lack of marked packets)
42,2%	78392	11927	7715

Figure 3.19: Prince-S with mixed synthesis

3.5.3 Multiple Flows

In the experiments we implicitly assumed that every network flow is considered to be a selfish player that seeks to optimize its utility function. One can consider all packets originating from the same IP address or the same subnet address to belong to the same selfish player. This would ensure that a user/player that can launch multiple flows concurrently (for any reason) will not be able to obtain an unfair part of the network bandwidth in total. Moreover, the impact of multiple flows per user on the fairness of the network is a general issue discussed for example in [43]. In general, it should be possible to apply any other successful approach to handle this issue (beyond the simplistic grouping of flows) to the Prince algorithms.

3.6 Conclusions

Based on our theoretical and experimental results, the following features of Prince emerge:

- It allocates bandwidth to each player close to his fair share.
- It leads to efficient NE with high goodput and low loss rates.
- It sustains its high performance even in the presence of multiple aggressive TCP or unresponsive high-rate UDP flows.
- It exhibits the positive side-effect of avoiding both the synchronization and the starvation of flows.

The previous features make us confident that Prince, besides being simple, is highly effective.

The basic game-theoretic idea of Prince, targeting and restricting the majority flow, yielded interesting results. A secondary outcome is that fair *buffer* sharing can result in fair *bandwidth* sharing.

Our future endeavours include examining hybrid variants of Prince in order to further optimize its computational performance. Additionally, we need to examine the behaviour of Prince in complex network topologies and heterogeneous router compositions.

CHAPTER 4

A Heaviest Hitters Limiting Mechanism with O(1) Time Complexity for Sliding-window Data Streams

4.1 Introduction

In this work, we aim to combine a novel algorithm for identifying the heaviest hitters in a sliding-window data stream with the ability to track the items in that sliding window in order to implement the fair rate-limiting mechanism experimentally analysed in Chapter 3. This results in a constant time algorithm which is able to fairly distribute the shared service resource to the incoming items.

The sliding-window data stream model is very similar to a traditional limited-size queue, used frequently in network routers to buffer packets while they await service. This is the motivating problem we used to implement and evaluate our algorithms and data structures. More generally, however, the problem of finding the heaviest hitters in a data stream, i.e., the problem of finding which category of items in a long succession of them are the most frequent ones, has a number of applications, some of them quite pervasive. Some applications are in financial data streams, where it is useful, for example, to know which stocks are showing the most mobility. Other applications include sensor networks (for example, helping an intrusion detection scheme [88]) and filtering sensed data, behaviour analysis on websites and trend tracking of hot topics (for example,

accurately counting the hottest queries for caching [21]).

The motivating application, as mentioned, is network traffic monitoring (and shaping) on Internet routers. Being able to tell at any moment in time which set of packets is the most frequent passing through a router (collectively referred to as a flow of packets) helps in both being able to tell what may be causing problems and subsequently resolving these problem in a "fair" manner towards those not contributing to the problem. In this work, we specifically address this issue by implementing the Prince queue policy initially described in [25] and experimentally analysed in Chapter 3. This policy has been shown to be able to successfully and fairly limit aggressive flows which send service requests, in our case packets, at a rate higher than the fair share they should request in order not to disadvantage other non-aggressive flows. To solve this problem we create a data structure and a set of associated algorithms which operate on it to solve the heaviest hitters problem on the network router queue. The basic heaviest hitters problem consists of a data stream where at each moment in time one item, which belongs to some itemset, arrives for processing. The goal is to be able to provide a list of the itemsets whose item counts are above a given θ threshold. Given the unbounded number of itemsets and length of the data stream, this cannot be achieved without unbounded memory. As a result, all of the proposed solutions for this problem have provided approximate results.

We address a variant of the basic problem in this work which stems from the observation that only a section of the whole history of the data stream may be interesting. Usually, the most recent items are considered to be more important. This is one of the most common and arguably one of the most useful of these variations: finding the heaviest (and lightest) hitters in a sliding-window data stream.

In the sliding window model, at each moment in time the maximum number of items which participate in a window over the data stream is constant. This window contains at most the Q most recent items. This scenario resembles the operation of a queue with an upper limit on its capacity. As items arrive to be processed they are inserted at the end of the queue and as items are processed they are removed from the front of the queue.

All the algorithms proposed for both the basic problem and the sliding window variation have in common the requirement that they be able to operate on-line. This entails being able to do only one pass over the data, i.e., each arriving item may be examined only once by the algorithm. This is usually called an update operation and the complexity of this operation must be constant time. Furthermore, querying for the heaviest hitters must also be as fast as possible, ideally proportional to the number k of the heaviest or lightest hitters that we request to be found.

Our algorithm supports the ability:

1. To provide *exact* results in the query operation and at the same time maintain constant time update and query operations.

2. To provide not only the heaviest but also the lightest hitters in the sliding window with the same performance and no overhead.

4.2 Related Work

This work merges the results from two separate fields to achieve our goals. The first field relates to the fair and balanced distribution of resources (and in this case specifically network router resources) to competing entities. In this field, network congestion has been described game-theoretically by Nagle [73] and the solution put forth used a market wherein the rules of the game would lead to the optimal strategy for the individual entities also being the optimal solution for the system. In a later work, Shenker [93] describes the relation between the selfish entities and the switch service mechanisms and proposes a method of guaranteeing efficient and fair operating points. Since then, the coordination of Internet entities has been modelled through various game definitions [3, 85]. We use the model proposed by [25] and experimentally analysed in Chapter 3 in order to achieve the fair and balanced distribution of resources.

The second field relates to the heaviest hitters problem and its solution in a sliding-window data stream context. This problem was first posed by Moore in 1980 and together with Boyer they presented the solution (in [10]) for finding the majority hitter in the basic version of the problem, i.e., non-window-based data streams. This problem was studied and approximate solutions were proposed much later and concurrently by [18, 49]. Since, a significant body of work has been performed on both the basic problem and on its numerous variations. A good presentation of this work can be found in [61, 71].

4.3 Proposed Abstract Data Type

In order to provide an accurate description of our algorithm and the accompanying data structure we describe here its interface. The abstract data type which we define supports the operations shown in Table 4.1. All the operations in our HL-HITTERS implementation have constant time complexity.

Table 4.1: The HL-HITTERS Abstract Data Type

Operation	Input	Output	Description
`Initialize`	–	–	Initializes the ADT
`Append`	`Item`	–	Records a new item into the counts
`Expire`	`Item`	–	Removes an item from the counts
`QueryHeaviest`	k: `Int`	`Array`[k]	Gets the heaviest-k `ItemSet`s
`QueryLightest`	k: `Int`	`Array`[k]	Gets the lightest-k `ItemSet`s
`GetOldestItem`	`ItemSet`	`Item`	Finds oldest item
`GetNewestItem`	`ItemSet`	`Item`	Finds newest item

4.3.1 Building Blocks

To implement the data structure we use common basic building blocks. More specifically, we use exactly one array of fixed size, multiple doubly linked lists and one hash table. With each of these data structures we only use the constant time operations. Thus, for example, we never iterate over the nodes of the linked list to reach a sought entry, rather we keep references to the node itself. We will proceed by describing exactly which operations will be used on each data structure and its time complexity.

4.3.1.1 Array

The array must be of size Q, the same as the size of the window, and its size remains constant during the execution of the algorithm. We only perform the operations `Get` and `Set` on the array, which execute in constant time. The elements of the array are never iterated over.

In the implementation for our experiments we used the standard vector provided by the C++ STL (Standard Template Library) `std::vector` class.

4.3.1.2 Doubly-linked List

The linked lists start out empty and as the algorithm executes nodes are added and removed. We only use the *Head* and *Tail* fields of the doubly-linked list to access the respective nodes in constant time. As far as the inserts and deletes are concerned, they are always executed with respect to a reference node and as such are constant time as well. To be more specific, `InsertBefore` and

`InsertAfter` require two arguments: the new node to insert and a reference node before or after which to insert the new node. Similarly, `Delete` requires a direct reference to the node to delete. Furthermore, the maximum number of nodes is known a priori to be Q, and thus we can eliminate the overhead of dynamic memory allocation for the nodes by using a preallocated node pool.

In the implementation for our experiments we used the a custom doubly-linked list implemented by using the Boost intrusive list [58] and a simple pool allocator to avoid all list node memory allocations and deallocations during the operation of the algorithm.

4.3.1.3 Hash-table

In the HL-HITTERS data structure the id of each itemset with at least one item in the window, is stored in a dynamic dictionary. A hash-table is used to implement the dynamic dictionary. Hashing is commonly assumed to require $O(1)$ amortized time for the operations `Get`, `Set` and `Delete` or at least for one of these operations. However, there are at least two examples of hashing schemes which achieve worst case $O(1)$ time with high probability (whp): the early work of [20] and the recent algorithm of [7]. Consequently, we can assume that an efficient, $O(1)$ hashing scheme can be used in the HL-HITTERS data structure.

There is an additional reason why we can assume $O(1)$ time for our hashing scheme. Given that our original motivation were router queues, we can assume that the maximum size of a window does not typically exceed 1000 items (packets in this case). The most common values are a few hundred items. This fact admits us the luxury to run the hashing data structure with a very low load factor. For example, even a hash table with 1 million entries would not be a significant cost for a modern router.

Consider the following naive approach with chained hashing using a uniform hashing function with n hash table entries, $m \ll n = cm$ packets, and k, the constant upper bound on the number of collisions. The probability ρ of experiencing more than k collisions in any of the n table entries is

$$\rho \leq n \cdot \binom{m}{k+1} \left(\frac{1}{n}\right)^{k+1} \leq n \left(\frac{e}{k+1}\right) \left(\frac{1}{c}\right)^{k+1}$$

For $n = 10^6$, $m = 10^3$ and $k = 10$ the first inequality gives that $\rho \leq 2.38 \times 10^{-35}$. Consider now a router which serves 10^9 packets per second (a bit unrealistic today but allows for future enhancements) and operates continuously for 20 years. This router can serve not more than $Z = 10^9 \times 60 \times 60 \times 24 \times 366 \times 20 \leq 6.34 \times 10^{17}$ packets during its lifetime. Even if we consider the case

where every one of these Z packets is unique, i.e., the router never receives two packets from the same flow and thus maximizes the potential for collisions to appear, the probability of a "bad" collision event occurring during its lifetime is $\rho * Z \leq 2.38 \times 10^{-35} \times 6.34 \times 10^{17} = 1.51 \times 10^{-17}$. This probability is thus practically negligible. Consequently, even the naive approach seems to meet the requirements for a router. In addition to this naive implementation there are many, very efficient, hashing schemes which will perform much better.

Unfortunately, however, in practice a standard cuckoo hash table occasionally experiences insertion operations that take significantly more time than the average. The question of which of the published hashing schemes offers the optimal trade-off between space redundancy and worst case bounds could be an interesting problem to investigate. However, for our purposes, any lightweight hashing scheme will be sufficient if sufficient memory is provided. Moreover, for our main motivation application, special hardware-based memory is available in many routers which can achieve de-amortized $O(1)$ performance [82].

Based on the above arguments, we plausibly assume that we can employ an efficient $O(1)$ whp hashing scheme for our data structure in a modern network router. Additionally, we believe that the arguments used for the router case can apply to other applications of window-based heaviest and lightest hitter problems. In the implementation used for the experiments of this work, we used chained hashing provided by the C++ `boost::unordered_map` class[44].

4.3.2 Data Structure

We now proceed to describe how the data structure is composed out of the basic building blocks. An overview of the layout used is presented in Figure 4.1. It should be noted that the Queue is not part of the HL-HITTERS data structure itself but is displayed in order to illustrate the pointers to the items it contains stored in the data structure.

Before proceeding with the description of the data structure further, we need to describe two types of simple record-like structures which are used:

- `CountNode`, which is the type of the list node used in the doubly-linked list. The data stored (besides the *Previous* and *Next* fields) is an integer named *Count*, the identifier of an `ItemSet` named *ItemSet* and a linked list of references to items in the queue named *QItems*.

- `CountRange`, which has two fields, named *First* and *Last*, both of which are references to a doubly linked list node of type `CountNode`. This structure is meant to store the endpoints of a sub-range of the *Counts* `DLList`. To support this, it supports two simple operations: `Insert` (a

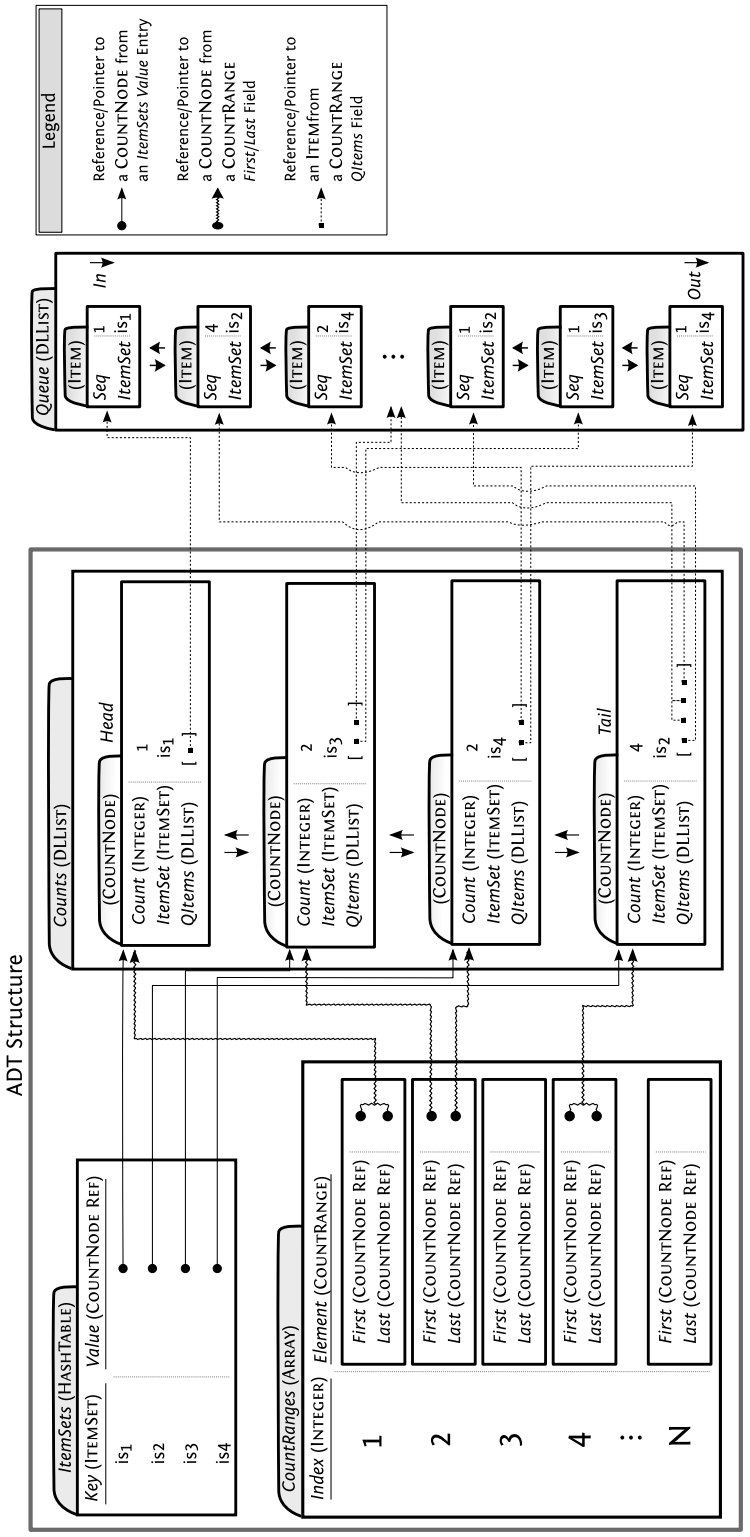


Figure 4.1: The ADT's structure.

Algorithm 1 The `Initialize` operation

```
1: procedure Initialize
2:     ItemSets ← new HashTable
3:     Counts ← new DLList
4:     Ranges ← new Array
5: end procedure
```

new node in range) and `Remove` an existing node from the range. Both are $O(1)$ operations as they manipulate only the *First* and *Last* fields and do not iterate over the nodes in the range.

4.3.2.1 Layout of the Data Structure

Itemsets that have no items in the window, i.e., a count of zero, will not have any entries in any of the data structures. Conversely, each itemset which has at least one item in the window, i.e., a count ≥ 1, will have one entry in the *ItemSets* `HashTable`. Additionally, for each itemset, there will exist one node of type `CountNode` in the *Counts* `DLList`, with a *Count* field corresponding to its exact count of items in the window and a *QItems* field containing pointers to its items in the queue. Finally, for each group of itemsets which have the same item count there will be one entry in the *Ranges* `Array`, in the position of the array which is equal to the itemset group's count.

4.3.3 Algorithms

We now present the operations which are supported by the data structure using pseudo-code and describe their operation and computational complexity in detail.

4.3.3.1 Initialization

The `Initialize` operation is shown in Algorithm 1. While its functionality is simply to initialize the *ItemSets* hash table, the *Counts* doubly linked lists and the *Ranges* array, it is useful nevertheless to illustrate that initialization is straightforward and that only memory allocations are performed. For the `DLList`, the allocation of the node pool is also performed here.

4.3.3.2 Append

In Algorithm 2 we present the `Append` operation. It receives the item which is to be appended as a parameter. The itemset of the item is looked up in the

Algorithm 2 The `Append` operation

```
 1: procedure Append(item: Item)
 2:  itemset ← item.GetItemSet()
 3:  cn ← cn′ ← null
 4:  if itemset ∈ ItemSets then
 5:   cn ← ItemSets.Get(key:itemset)
 6:   cn′ ← Ranges.Get(index:cn.Count).Last.Next
 7:   Ranges.Remove(node:cn)
 8:   Counts.Remove(node:cn)
 9:   cn.Count ← cn.Count + 1
10:   cn.QItems.Push(item)
11:   Counts.InsertBefore(before:cn′, ins:cn)
12:   Ranges.Insert(node:cn)
13:  else
14:   qi ← new DLList
15:   qi.Push(item)
16:   cn←new CountNode(ItemSet:itemset,Count:1,QItems:qi)
17:   Counts.InsertBefore(before:Counts.Head, ins:cn)
18:   Ranges.Insert(node:cn)
19:   ItemSets.Set(key:itemset, value:cn)
20:  end if
21: end procedure
```

ItemSets hash table. If it is found, then the itemset is already being counted, i.e., has other items in the window, and therefore its count must be increased by one. If not, then it is a new itemset, i.e., it has no other items in the window, and thus must be recorded with a count of one and a pointer to item in the queue has to be stored.

For the case of being already counted, only the *Counts* and the *Ranges* structures will be modified. The idea is to move the count node corresponding to the itemset to the position in the *Counts* linked list where it will be the first linked list node with the new count. In order to do this, the count node of the itemset is looked up via the `Get` operation on the hash table and a reference to it is stored in *cn*. Before removing the *cn* node from the list, the position in the linked list where it will be moved to is recorded in *cn*′, with help from the *Ranges Last* field. This will point to the immediately next linked list node after the last node with the old count. Subsequently, the count node *cn* is removed from the linked list and the corresponding *Ranges* count range entry is updated with the `Remove` operation. Finally, the *cn* node is inserted in the linked list before the *cn*′ node, the new *Ranges* count node entry is updated to include it and a pointer to the

item in the queue is pushed at the end of the *QItems* queue (in $O(1)$).

For the case of not being already counted, all of the structures will be modified. A new count node will be created to hold the count for the new itemset. Since allocating a new object on the heap may not be $O(1)$, we can take advantage of the fact that the maximum number of itemsets is Q, as explained in Section 4.3.1.2, and as such we can just take out a preallocated count node out of a preallocated pool in $O(1)$. A new `DLList` is created to store the pointers to items in the queue which belong to this itemset and is used in the new count node. This node is then inserted in the position of the *Counts* linked list indicated by the *First* field in the first count range entry of the *Ranges* array and then it is recoded in the same count range entry. Finally, the *itemset* hash table is updated by creating an entry that maps the new itemset to the count node which was created previously using the `Set` operation.

4.3.3.3 Expire

In Algorithm 3 we present the `Expire` operation. It receives the item which is to be removed as a parameter. The item's itemset is looked up in the *ItemSets* hash table via the `Get` operation and the reference to the count node in the *Counts* linked list representing it is stored in *cn*.

Since the count of the itemset will be decremented by one, we need to move the *cn* count node to the position in the *Counts* linked list where it will be the first linked list node with the new (old minus one) count. Similarly to the `Append` operation, before removing the *cn* node from the list, the position in the linked list where it will be moved to is recorded in cn', with help from the *Ranges First* field. This will point to the immediately previous linked list node after the first node with the old count. Subsequently, the count node *cn* is removed from the linked list and the corresponding *Ranges* count range entry is updated with the `Remove` operation. The first item in the count node's *QItems* queue is popped and the count node *Count* field is decremented by one. If the count has not reached zero a check is made to see whether the position to be moved is valid:

- The reference in cn' must be not null, which would indicate that the previous count range was the first in the linked list, and
- the count of the cn' referenced node must be the same as the new count of the moving node, i.e., the target count node must belong to the correct count range.

If this check succeeds, the new corresponding *Ranges* count range entry is fetched with the `Get` operation. Its *First* field is set as the new cn'' insertion position. Afterwards the moving node is inserted there. If the check fails, then there is

Algorithm 3 The `Expire` operation

```
1:  procedure Expire(item: Item)
2:      itemset ← item.GetItemSet()
3:      cn″ ← null
4:      cn ← ItemSets.Get(key:itemset)
5:      cn′ ← Ranges.Get(index:cn.Count).First.Previous
6:      Ranges.Remove(node:cn)
7:      Counts.Remove(node:cn)
8:      cn.QItems.Pop(item)
9:      cn.Count ← cn.Count - 1
10:     if cn.Count ≥ 1 then
11:         if cn′ ≠ null and cn′.Count = cn.Count then
12:             cn″ ← Ranges.Get(index:cn′.Count).First
13:             Counts.InsertBefore(before:cn″, ins:cn)
14:         else
15:             Counts.InsertAfter(after:cn′, ins:cn)
16:         end if
17:         Ranges.Insert(node:cn)
18:     else
19:         delete cn.QItems
20:         delete cn
21:         ItemSets.Delete(key:itemset)
22:     end if
23: end procedure
```

no `CountRange` entry in the *Ranges* array corresponding to the new count and the count node is inserted right where the original cn' reference pointed to.

In both cases, the moving count node will be inserted in the *Ranges* entry with the new count using the `Insert` operation.

If the new count after decrementing by one is zero, the count node is deleted. Before doing that, the count node's *QItems* `DLList` is also deleted and returned to the preallocated pool. If a preallocated pool was used it is returned to the pool in $O(1)$. Finally, the *itemset* hash table is updated by deleting the entry that maps the itemset to the count node which was previously deleted.

4.3.3.4 Query

In Algorithm 4 we present the `QueryHeaviest` and the `QueryLightest` operations simultaneously. The basic algorithm is the same; only the start of the iteration and its direction is different. In the algorithm, the left side of the

Algorithm 4 `Query Heaviest` ↔ `Lightest` operation

```
 1: function QueryHeaviest(k: Integer)
 2:     results ← new Array[k]
 3:     cn ← Counts.Tail ↔ Counts.Head
 4:     i ← 1
 5:     while i≤k and cn≠null do
 6:         results[i] ← cn.ItemSet
 7:         cn ← cn.Previous ↔ cn.Next
 8:         i ← i+1
 9:     end while
10:     return results
11: end function
```

↔ symbol corresponds to the `QueryHeaviest` operation while the right side to the `QueryLightest` operation.

The algorithm receives the threshold k as a parameter. Initially, a new *results* array of size k is created to hold the results. In some cases, there may be less than k itemsets available, therefore a number of positions at the end of the array will have null entries.

The count node reference *cn* is set to point to the last (for `QueryHeaviest`) or the first (for `QueryLightest`) node in the *Counts* linked list via its *Head* or *Tail* fields. Afterwards, an iteration is performed up to k times. In each step, the current itemset stored in the node referenced by *cn* is stored in the current (the i-th) index of the array. Finally, the result is returned.

The whole operation makes up to k iterations, at each one adding a different itemset to the result. This makes this operation have a time complexity of $O(k)$ and as such is constant time as well. The operation of the query algorithm can easily be extended without changing the computational complexity to also return the actual count of each itemset along with each itemset. In addition it is possible instead of specifying a k parameter to return all the itemsets with the highest/lowest count. To implement this, retrieve the *Tail*/*Head* count node of *Counts*, get the highest/lowest count, access the *Ranges* entry corresponding to that count and get the range of count nodes between the *First* and *Last* fields with the max/min count. This algorithm's computational complexity will depend on the number of itemsets which will be the max/min count. As it is possible to have Q itemsets each with a count of one, this algorithm will have a worst case complexity of $O(Q)$. However, in practice in many applications this will seldom be the case. Another extension would be to return the heaviest-θ/lightest-θ hitters, where θ is relative, expressed as a proportion of the window size (e.g., $\theta = 10\%$). However, here the `QueryHeaviest` and the

Algorithm 5 `Get Oldest ↔ Newest Item` operation

```
1: function GetOldestItem(itemset: ItemSet)
2:     cn ← ItemSets.Get(key:itemset)
3:     item ← cn.QItems.Front() ↔ cn.QItems.Back()
4:     return item
5: end function
```

`QueryLightest` operations will have different complexities. Since there is an upper bound on the number of itemsets which can have a frequency more than or equal to θ equal to $1/\theta$, one can just execute `QueryHeaviest` with $k = 1/\theta$ and the complexity will be as originally $O(k)$. However, no such bound exists for the `QueryLightest` case, and therefore its worst case complexity will be $O(Q)$. Finally, if one is willing to accept an $O(Q)$ worst case complexity it is possible to create cumulative versions of both the original and the relative version of the query operations, where the k or θ parameters denote the cumulative count or proportion of the window. This would return the first itemset whose counts together add up to the specified threshold.

4.3.3.5 GetItem

In Algorithm 5 we present the `GetOldestItem` and the `GetNewestItem` operations simultaneously. The basic algorithm is the same; only the retrieved end of a queue is different. In the algorithm, the left side of the ↔ symbol corresponds to the `GetOldestItem` operation while the right side to the `GetNewestItem` operation.

The algorithm receives the itemset of which the oldest or newest item in the queue is to be found. Initially, the count node corresponding to the itemset is retrieved from the *ItemSets* hash table. Subsequently, the *QItems* linked list in the count node is accessed and depending on whether the oldest or newest item in the queue is requested, the front or back item in the queue is returned.

Since no iterations are performed and since only the first or last item of the linked list *QItems* is accessed, these operations are performed in $O(1)$.

4.3.4 Space Complexity

The space complexity of the HL-HITTERS data structure can be fully derived and is exclusively dependent on the maximum window size Q. The *ItemSets* hash table contains a maximum of Q entries, the *Ranges* array has a constant size of Q entries and the *Counts* doubly linked list contains a maximum of Q count nodes. Furthermore, each node in the doubly linked list *Counts*, contains

Table 4.2: Computational Complexity

Operation	DirectCounting	HL-Hitters
`Initialize`	O(Q)	O(Q)
`Append`	O(1)	O(1)
`Expire`	O(1)	O(1)
`QueryHeaviest`	*O(Qlogk)*	*O(1)*
`QueryLightest`	*O(Qlogk)*	*O(1)*
`GetOldestItem`	O(1)	O(1)
`GetNewestItem`	O(1)	O(1)

QItems, a linked list of pointers to items in the queue. This linked list uses a pool of preallocated nodes which is shared between all the *Counts* nodes. Since there can only at most Q items in queue, the preallocated pool of *QItems* nodes also has a size of Q. It follows that the space complexity of the whole HL-HITTERS data structure is $O(Q)$.

4.4 Results

It is clear from the previous analysis that the computational complexity of the HL-HITTERS algorithms presented is overall constant time whp. However, this does not guarantee an acceptable level of performance if in practice the constant time required is too high. We have created a router-like scenario, and have performed experiments to gauge the actual performance of the proposed algorithms. We have to note that, to our knowledge, there exists no other algorithm for calculating the heaviest-k hitters exactly, which also provides close to constant time performance. Therefore, we have implemented a naive but efficient as far as possible algorithm to find the heaviest-k hitter. This algorithm, each time the heaviest hitter is requested, creates a hash-table, and records within it the counts for each itemset. As it does this, it keeps track of the running heaviest hitter. However, this algorithm has an $O(Qlogk)$ time complexity, due to the partial ($k-$largest) sort needed to find the heaviest-k hitters. Furthermore, in the experiments performed, we restricted ourselves to finding the top heaviest hitter only, i.e., $k = 1$, in order not to significantly disadvantage the direct counting algorithm. For reference, the computational complexity of the operations implemented by the direct counting and the HL-HITTERS algorithm is presented in Table 4.2.

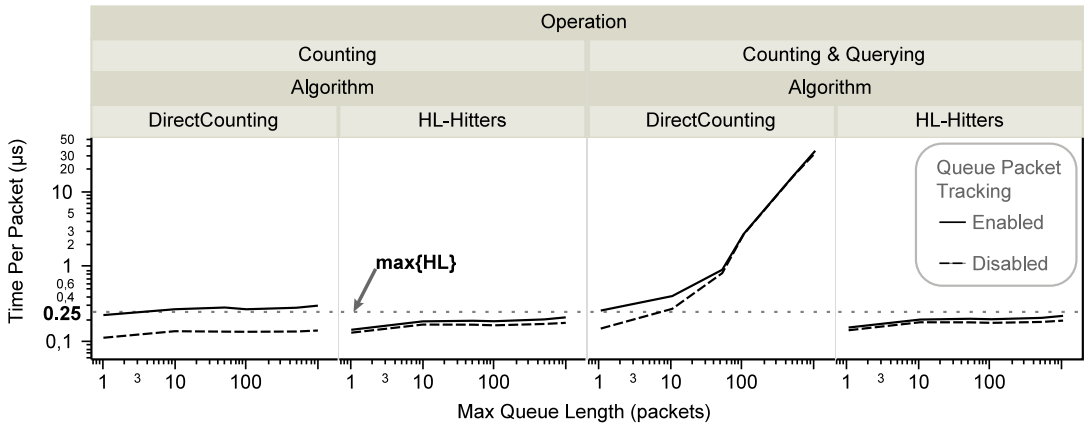


Figure 4.2: Scenario 1. Performance of HL-HITTERS vs. direct counting for different Q queue lengths and grouped based on operation performed (counting or counting+querying) and on whether the packet positions in the queue are tracked. Measured in mean processing time per packet (shown in μs). The maximum time taken by HL-HITTERS is $0.25\mu s$.

4.4.1 Experimental Scenarios

The experimental evaluation of our implementation is performed in two distinct scenarios. The first scenario is geared towards evaluating the performance of HL-HITTERS when the queue is full but experiences no dropped packets, i.e., the rate of serving packets from the end of the queue is the same as the rate of arriving packets at the beginning of the queue. Furthermore, this scenario seeks to evaluate how much impact querying to find the heaviest hitter has when it is performed every time a new packet arrives at the queue, since this is what would happen in a real application. Finally, it seeks to measure the impact of tracking the packets which belong to each flow within the queue. This ability will permit the implementation of the Prince policy in the second scenario.

The second scenario aims to measure both the performance and the efficiency of the Prince policy in contrast to a simple FIFO (DropTail) policy when the queue is full and experiences dropped packets, i.e., the rate of serving packets from the end of the queue is higher than the rate of arriving packets at the beginning of the queue. In this scenario, we use two groups of flows, normal and aggressive. The normal flows, which constitute 90% of the total number of flows never send packets at a rate higher than their fair share while the aggressive flows (10% of total flows) always exceed their fair share (within a range of different amounts). As a result, the queue is overflown and needs to drop packets. To compare performance, the Prince policy is implemented by both the naive direct counting

algorithm and HL-HITTERS. We measure the time taken to service packets as well as how fairly the policies manage to limit the aggressive flows while not disadvantaging the normal flows.

4.4.2 Experiment Setup

The implementation has been performed using C++, with standard C++ versions of the building blocks, as described in section 4.3.1. We used the G++ compiler with all the optimizations enabled ($-Ofast$) for our specific architecture. The experiments were executed on an Intel Quad Core Q9300 processor with $4GB$ of main memory, using one dedicated core for the execution of the experiments. The operating system used was Arch Linux, with the 3.0.1 version kernel. For each result point 10 identical sequential executions of the experiment were performed to remove any bias.

4.5 Discussion

A selected but representative and indicative of the worst case performance subset of the experimental results are presented here.

4.5.1 Scenario 1

The results obtained for the first scenario are summarized in Figure 4.2 where the performance of the direct counting algorithm is compared to the HL-HITTERS algorithm. When counting only, i.e., just keeping track of the count of packets of each flow in the queue the two algorithms perform similarly, whether they also track the positions of the packets in the queue or not. This performance is consistent with the theoretical O(1) complexity given in Table 4.2 for the `Append` and `Expire` operations. However, when querying to find the heaviest hitter ($k = 1$) is introduced (counting needs to be performed as well since without it querying is not possible), the results reflect the O(Q) complexity of direct counting and the $O(1)$ complexity of HL-HITTERS. It is noteworthy to examine the absolute numbers as well. The HL-HITTERS algorithm has a maximum processing time per packet of $0.25\mu s$. This means that despite using general purpose building blocks and no hardware-based content addressable memory or specialized CPUs, we can process at least 4 million packets per second using our implementation. According to [95] IP packet sizes vary between $40bytes$ and $1500bytes$, with strong polarization tendencies. Given those values, we can achieve a throughput between $1.2Gbit/sec$ and $48Gbit/sec$. We stress the fact that this performance is achievable without any specialized hardware as would

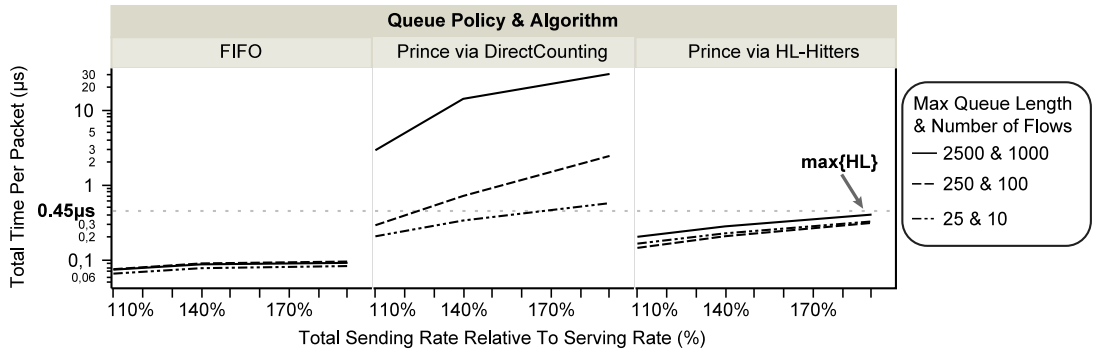


Figure 4.3: Scenario 2. Performance of simple FIFO (no packet tracking) vs. HL-HITTERS and direct counting implementing the Prince policy. Results shown for different Q queue lengths and number of flows as a function of the total sending rate of the flows vs. the serving rate of the queue. Measured in mean processing time per packet (shown in μs). The maximum time taken by HL-HITTERS is 0.45μs.

typically exist in an Internet router. Furthermore, performance profiling has shown that approximately 50% of the processing time is spent on the hash-table operations. Since these would heavily benefit from optimizations on a hardware router, we are confident that significantly higher performance is attainable under such conditions.

4.5.2 Scenario 2

The results generated from the experiments in the second scenario are displayed in Figures 4.3 and 4.4. Figure 4.3 shows the results of the comparison between the HL-HITTERS and direct counting algorithms implementing the Prince policy with packet tracking and a simple FIFO DropTail policy (with no packet tracking). The simple FIFO policy is the most performant and is not significantly affected by the increase in total sending rate. The direct counting algorithm slows down linearly with the increase in sending rate and scales badly as the queue size used increases. The loss of performance due to sending rate increase is expected since the `QueryHeaviest` operation is executed analogously more as well. However, the bad scaling in relation to the queue size leads to unusable performance for a router. Finally, the HL-HITTERS algorithm also slows down as the sending rate increases, at a much lower rate, and scales very well even when the size of the queue is increased. The absolute numbers show that the HL-HITTERS algorithm has a maximum processing time per packet of 0.45μs when implementing Prince, which as described in the previous paragraph, would accordingly lead to a

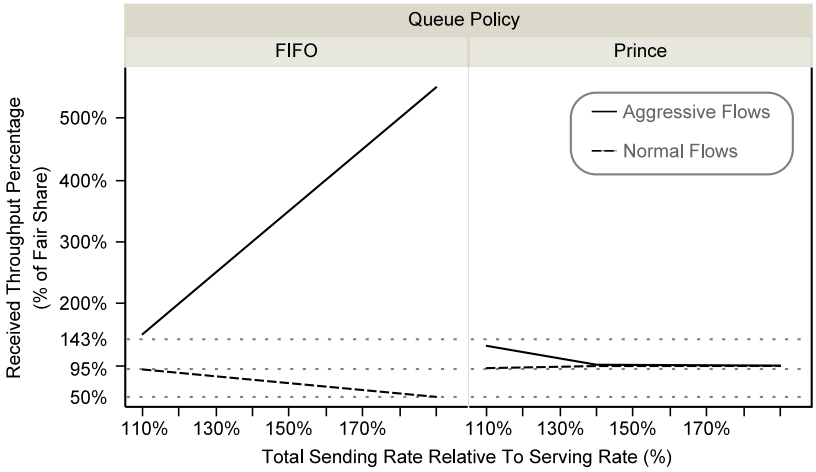


Figure 4.4: Scenario 2. Measure of policy fairness for the simple FIFO and the Prince policy. The ideal received throughput for both aggressive and normal flows is 100% of their fair share. Here the actual achieved throughput of the aggressive and normal flows is displayed as a function of the total sending rate of the flows vs. the serving rate of the queue. Measured in percent of fair share achieved. For the Prince policy the aggressive flows achieve a maximum of 143% of the fair share and the normal flows a minimum of 95% of the fair share.

throughput between $0.7 Gbit/sec$ and $26 Gbit/sec$.

Figure 4.4 shows the results of the comparison between a simple FIFO Drop-Tail policy (with no packet tracking) and the HL-HITTERS algorithm implementing the Prince policy with packet tracking. These results show that although the FIFO policy is very fast, as seen in Figure 4.3, it is not able to limit the aggressive players effectively. As the sending rate of the aggressive players increases and the total sending rate as a result increases (since the sending rate of the normal flows is constant) the aggressive players manage to obtain a much higher portion of throughput in respect to the fair share that they should get. For example, when the aggressive players send 10 times faster than the normal players the total sending rate becomes 190% of the service rate and the aggressive players get more than 500% of the fair share while the rest of the 90% of the flows, the normal flows, all receive 50% of the fair share. In contrast, using the Prince policy, the aggressive flows only manage to get 143% of the fair share and as they increase their sending rate they make themselves clearer targets for limiting and are limited even more effectively. At the same time, the lowest share of throughput the normal flows receive is 95% of the fair share.

4.6 Conclusions

Our work on the problem of the heaviest-k and lightest-k hitters in a sliding-window data stream has resulted in a data structure and an efficient set of algorithms for its operations. These in tandem allow us to achieve constant time updates and queries. Building on this feature, we implement the Prince policy, an effective rate-limiting mechanism, on a simulated router queue and show that it is possible to achieve both a highly performant and extremely fair rate-limiter on a router queue. We have also shown that the performance achieved is high enough in absolute numbers to be used in practical applications. We have attempted to maximize performance on a standard PC while at the same time have found that using a fairly standard component in hardware routers can potentially double performance.

An interesting idea would be to extend this mechanism to incorporate the size of the packets as well, not only their number. This would allow us to make decisions based on the quantity of data that an itemset is responsible for, rather than how many items it is generating. Another direction would be to use multiple HL-HITTERS structures in a queue in parallel, each monitoring a different length of history. This would allow monitoring not only the highest hitters currently in the queue but also in longer periods of time.

CHAPTER 5

On Money as a Means of Coordination between Network Packets

5.1 Introduction

It is known that a large number of independent flows is constantly competing on the Internet for network resources. Without any central authority to regulate its operation, the available network resources of the Internet are allocated by independent routers to the flows in a decentralized manner. Internet flows may submit at any time an arbitrary amount of packets to the network and then adjust their packet rate with an appropriate flow control algorithm, like the AIMD-based algorithms for TCP-flows. The apparent lack of coordination between the independent flows leads the Internet to an "anarchic" way of operation and gives rise to issues and problems that can be addressed with concepts and tools from algorithmic game theory.

Two representative works on applying game theory to network problems are [57, 85]. Certain game-theoretic approaches to congestion problems of the Internet, and especially the TCP/IP protocol suite, are discussed in [93, 3, 32, 25]. A combinatorial perspective on Internet congestion problems is given in [48]. The focus of the above works and the present work is on sharing the network resources between selfish flows. In this work, however, we propose an economy where packets belonging to selfish flows may interact directly with each other.

The use of economic tools like pricing, tolls and taxes as a means to regulate the operation of networks and/or to support quality of service (QoS) functional-

ities in the presence of selfish flows is, for example, discussed in [78, 33, 16, 15, 65, 69]. In particular, the Paris Metro Pricing approach - using pricing to manage traffic in the Paris Metro - is adapted to computer networks in [78]. A smart market for buying priority in congested routers is presented in [65]. In [16, 15] taxes are used to influence the behaviour of selfish flows in a different network model. An important issue identified in [15] is that taxes may cause disutility to network users unless the collected taxes can be feasibly returned to the users. In our economic model this issue is naturally solved; trades take place between the flows, so the money is always in the possession of the flows.

In this work, we apply a common economic tool, namely money, to coordinate network packets. This is in contrast to much of the existing literature, which aims to impose charges on Internet traffic, and to our knowledge, this is the first work to propose economic exchanges directly between packets. In particular, we present a network economy, called PacketEconomy, where ordinary network packets can trade their positions in router queues. The role of money in this approach is to facilitate the trades between the network packets. Queue positions and money are exchanged directly between the packets while the routers simply carry out the trades. We show that, in this economy, packets can self-regulate their access to network resources and obtain better services at equilibrium points.

In their seminal work, Kiyotaki and Wright [55] examine the emergence of money as a medium of exchange in barter economies. Subsequently, Gintis [34, 35] generalizes the Kiyotaki-Wright model by combining Markov chain theory and game theory. Inspired by the above works, we propose the PacketEconomy where money is used as a coordination mechanism for network packets and prove that there are Nash equilibria where trades are performed to the benefit of all the flows. In the PacketEconomy, specialization - the reason for the emergence of money as per Adam Smith ([97, Chapter 4], cited in [55]) - originates from the diverse QoS requirements of network flows. In particular, the various types of PacketEconomy flows differ in their tolerance for packet delays.
The main contributions of this work are:

- A new game-theoretic model representing network packets as populations of rational agents. In this model, a network flow is represented as a population of in-flight packets that can make bilateral trades with other packets.

- Application of bilateral trades and virtual money at a microeconomic level to support better coordination of rational network packets.

- Application of an interesting combination of ergodic Markov chains and strategic games within the context of network games.

5.2 An Economy for Packets

The PacketEconomy is comprised of a network model with selfish flows, a queue that supports packet trades, a currency and a specific economic goal. The solution concept is the Nash equilibrium (NE), i.e., a profile of the game in which no player has anything to gain by changing only his/her own strategy unilaterally.

The Network Model. We assume a one-hop network with a router R and a set of N flows, as shown in Figure 5.1. This setting is equivalent to the common dumbbell topology used for the analysis of many network scenarios, including the seminal paper of Chiu and Jain [14] on the AIMD algorithm. The router R has a FIFO DropTail queue with a maximum capacity of q packets and operates in rounds. In each round, the first packet (the packet at position 0 of the queue) is served. At the end of the round, the first packet reaches its destination. Packets that arrive at the router are added to the end of the queue.

Packet Trades. At the beginning of each round all packets in the queue are shifted one position ahead. A packet that enters the queue in this round, occupies the first free (after the shift) position at the end of the queue. After the shift, the packet that has reached position zero is served, while the other packets in the router queue are simply waiting. These idle packets can engage in trades. During each router round a fixed number b of trading periods take place. In each trading period the idle packets are matched randomly in pairs with a predefined pairing scheme. Each packet pair can perform a trade, as shown in Figure 5.2, provided the negotiation performed between them leads to

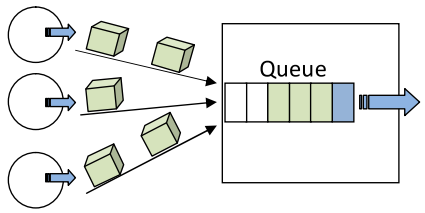


Figure 5.1: The network model with the flows, their packets, the router, and the queue.

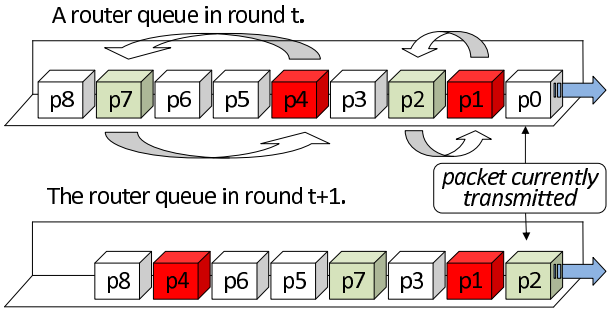


Figure 5.2: The state of a router queue in two successive rounds. In round t, two trades take place; one between the packet pair (p1,p2) and one between the pair (p4,p7).

an agreement. The way the trades take place at a microeconomic level between paired packets resembles the models of [34, 55] where agents meet in random pairs and can make trades.

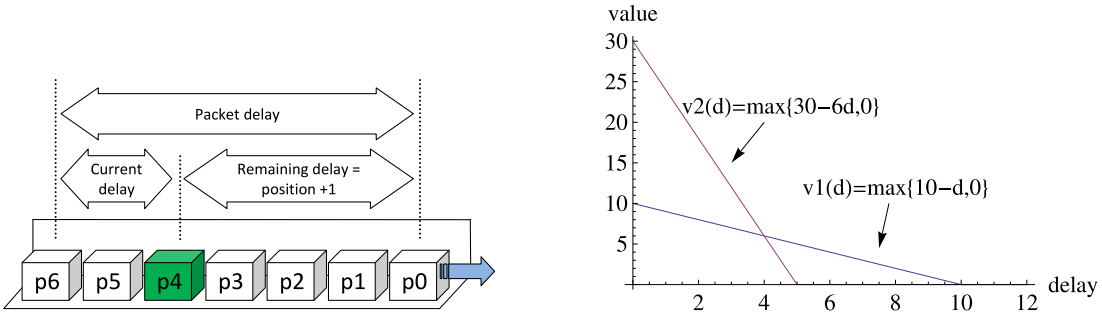
Packet Delay. The delay d_p of a packet p that starts at position k of the zero-based queue and does not make any trade is $k + 1$ rounds (Figure 5.3a). If, however, the packet engages in trades and buys a total of r_b router rounds and sells r_s router rounds, then its delay d_p, including the time to be served, becomes $d_p = k + 1 + r_s - r_b$ rounds. A packet may have an upper bound $d_{p,\max}$ on its delay; for delays larger than $d_{p,\max}$ the value of the packet becomes zero and the packet will not voluntarily accept such delays (that is, it will not sell).

Details. The router operates in rounds and can serve one packet in each one. All packets are assumed to be of the same size and no queue overflows occur. In generating the random packet pairs, the use of predefined pairing reduces the computational burden and avoids stable marriage problems. We make the plausible assumption that flows with different QoS preferences are competing for the network resources. We also make the assumption that the preferences of each flow can be expressed with a utility function for its packets. Thus, packets with different utility functions will, in general, co-exist in the router queue.

Packet Values. For each packet p there is a flow-specific decreasing function $v_p(d)$ which determines the value of p, given its delay d. The value function of each flow must be encoded onto each packet. Thus, its computational requirements should be low in order not to overload the router. A class of simple value functions are $v_p(d) = \max\{v_{\max} - c_p \cdot d, 0\}$ where c_p is the cost per unit of delay (Figure 5.3b). The value of a packet can be calculated anytime during the packet's journey via the $v_p(d)$ function.

In the PacketEconomy every packet has its compensatory price p. For prices lower than p, the packet is ready to buy better queue positions and for prices higher than p it is ready to sell its position, provided that the extra delay will not cause it to exceed its maximum delay limit.

Inventories. Every time a packet is delivered in time, wealth is created for the flow that owns the packet. Each packet p has an inventory $I_p(t)$ containing two types of indivisible goods or resources; the packet delay $d_p(t)$ and the money account $a_p(t)$. Note that delay bears negative value, whereas money represents positive value. We assume positive integer constants s_a, s_b and s_d, such that $a_p(t) \in \{-s_a, \dots, s_b\}$ and $d_p(t) \in \{0, \dots, s_d\}$. The inventory also contains the current position $pos_p(t)$ of the packet in the queue if it is waiting in the queue. When the packet reaches its destination, the contents of the inventory of the packet are used to determine its utility. This utility is then reimbursed to the flow that owns the packet and a new packet of the same flow enters the queue. An inventory is called admissible, if the delay of the packet does not exceed its maximum delay. A packet would not agree to trade an admissible



(a) Packet delay terminology for p4.

(b) Two simple packet value functions.

Figure 5.3: Delays and Packet Values.

inventory state for a non-admissible one. We assume that all packets start with an admissible inventory when they enter the queue.

Benefit and Utility. Every packet has two types of resources that bear value, the packet value and the budget of a packet. We define the notion of the packet benefit as the sum of the value of a packet plus/minus its budget. Then we use the benefit concept to define the utility function of the packet. For rate-based flows (see below), the utility of a packet is equal to its benefit. For window-based flows the utility function is the benefit rate (benefit per round).

Trades. The objective of each packet is to maximize its utility. Thus, when two packets are paired in a trading period, their inventories and their trading strategies are used to determine if they can agree on a mutually profitable trade, in which one packet offers money and the other offers negative delay. The obvious prerequisite for a trade to take place is that both packets prefer their post-trade inventories to their corresponding pre-trade inventories. For this to be possible, there must be "surplus value" from a potential trade. In this case, both packets can benefit, i.e., increase their utility, if they come to an agreement.

Flow Types and the Cost of Delay. The delay that a packet experiences has a negative impact on its utility. The value is a non-increasing function of the delay. Window-based flows employ a feedback-based mechanism, the congestion window, which determines the maximum number of packets that the flow may have in-flight. Every packet that is in-flight occupies one of the available positions in the congestion window of a window-based flow. The more a packet delays its arrival, the longer the following packet will have to wait to use the occupied window position. Therefore, the impact of packet delays for window-based flows is twofold; the decreased value of the delayed packet and the reduced packet rate. On the other hand, for rate-based flows which submit packets with some given rate, the only consequence due to packet delays is the reduced packet value.

Assume a rate-based packet p with balance α_1 and delay $d_1 < d_{p,\max} - d_\epsilon$, for some d_ϵ. When a trade changes the delay from d_1 to $d_2 = d_1 + d_\epsilon$, then this also changes the value of the packet from $v(d_1)$ to $v(d_2)$. The difference between these two values determines the compensatory price ρ for the packet.

$$\rho = v(d_1) - v(d_2) = v(d_1) - v(d_1 + d_\epsilon) = c_p d_\epsilon \ . \tag{5.1}$$

At this price, the utility of the packet remains unchanged after the trade. A packet would agree to sell for a price $\rho_s > \rho$, or to buy for $\rho_b < \rho$.

For window-based flows, however, the price estimation needs more attention. Assume a window-based packet with delay $d_1 < d_{p,\max} - d_\epsilon$ and account balance α_1. Before the trade, the utility (benefit rate) is $r_1 = (v_1 + \alpha_1)/d_1$. If the packet agrees to trade its position and to increase its delay by d_ϵ, then the utility is $r_2 = (v_2 + \alpha_2)/d_2$. Then, by setting $r_1 = r_2$ we obtain the compensatory price ρ for the trade.

$$\frac{v_1 + \alpha_1}{d_1} = \frac{v_2 + \alpha_2}{d_2} \Rightarrow \frac{V - c_p d_1 + \alpha_1}{d_1} = \frac{V - c_p(d_1 + d_\epsilon) + (\alpha_1 + \rho)}{d_1 + d_\epsilon} \Rightarrow$$

$$\rho = (V + \alpha_1)\frac{d_\epsilon}{d_1} \ . \tag{5.2}$$

The above expression for the price ensures that the utility function of the packet remains unchanged. A packet would agree to sell its position, for a price $\rho_s > \rho$, or to buy a position ($d_\epsilon < 0$) for $\rho_b < \rho$. Unless otherwise specified, the final trading price when a trade takes place will be the average of the ρ_s of the seller packet and ρ_b of the buyer packet. We illustrate the PacketEconomy approach in a representative scenario.

5.3 Equilibria with Monetary Trades

A Representative Scenario. We examine a simple scenario that produces an interesting configuration. It consists of a set of N window-based flows f_i, for $i \in \{1 \dots N\}$, each with a constant window size w_i, and $\sum_i w_i = q$. When a packet is served by the router it is immediately replaced by an identical packet submitted by the same flow. This is a simplifying but plausible assumption. In reality, when a flow packet arrives at its destination, a small size acknowledgement packet (ACK) is submitted by the receiver. When the sending flow receives the ACK it submits a new identical packet that immediately enters the queue. We assume $b = 1$ trading period per round but in general b can be any integer $b > 0$.

Failure states. For each packet, there is a small probability p_f for an extra delay of d_f rounds, where d_f is a discrete random variable in $\{1, 2, \dots, q-1\}$. These

delays correspond to potential packet failures of real flows, and occur between the service of a packet and the submission of its replacement. By convention, the delay d_f is added to the delay of the packet that has just been served. If more than one packets enter the queue at the same time (synchronized due to delays), their order in the queue is decided upon uniformly at random. A packet that does not participate in any trade and does not suffer delay due to failure will experience a total delay of q rounds.

Packet states and strategies. The state $\tau_p(t)$ of a packet p in round t is a pair $\tau_p(t) = (I_p(t), rel_p(t))$, where $I_p(t)$ is the inventory of the packet and $rel_p(t)$, which is meaningful only in failure states, is the remaining number of failure rounds for the packet. The state of all packets of the economy in round t determines the state of the whole economy $\tau(t) = \prod_{p=0}^{q-1} \tau_p(t)$. From a packet's point of view, a trade is simply an exchange of its inventory state (budget, delay and position) with a new one. Consequently, a pure strategy of a packet is a complete ordering of the possible states of its inventory. In each round, the packets that are waiting move by default one position ahead and, thus, enter a new inventory state. We assume that the packet ignores the impact of its state and strategy on the state of the packet population. In every trading period the packet assumes the same stationary state of the economy.

Definition 1 *Let $\tau(t)$ be the state of the economy in round t.*

Lemma 3 *$\tau(t)$ is an ergodic Markov chain.*

Proof 3 *Assume $b = 1$ trading period per round. In each round, the economy moves to a new state with transition probabilities that depend only on the current state and the strategies of the packets. Let σ_p be a pure strategy of each packet p of a flow and σ be a pure strategy profile of the whole economy. Then, there is a corresponding transition probability matrix P^{σ} for the economy. Let σ_m be a mixed strategy profile of the whole economy. Then the corresponding transition probability P^{σ_m} of the economy for σ_m is an appropriate convex combination of the transition matrices of the supporting pure strategies. In case of multiple trading periods per round ($b > 1$), the economy makes b state transitions per round.*

The number of potential states for a packet is finite and, consequently, the number of states for the whole economy is also finite.

Definition 2 *A zero state τ_0 is a state of the economy in which all packets have zero budget and each packet p has delay $d_p(t_0) = pos_p(t_0) + 1$, where t_0 is the current round of the router.*

Assume that in round t the packet at position 0 fails for $q - 1$ rounds, in round $t + 1$ the next packet at position 0 fails for $q - 2$ rounds etc. Then after q rounds all new

packets will simultaneously enter the queue. Each packet will have zero budget and by definition their ordering will be random. This also means that for each packet p, $d_p(t) = pos_p(t) + 1$. Thus, in round $t + q$ the economy will be in a zero state. The probability for this to happen is strictly positive and thus each zero state τ_0 is recurrent. Since the number of states of the economy is finite, the states that are attainable from zero states like τ_0 form a (finite in size) class of irreducible states. Moreover, each zero state is aperiodic, and thus each of the states of the class of attainable states is also aperiodic. It is known that any finite, irreducible, and aperiodic Markov chain is ergodic.

Lemma 4 *For each pure strategy profile σ of the economy, there is a unique stationary distribution π_σ of the economy.*

Proof 4 *For each pure strategy profile σ, the Markov chain of the economy has a finite number of states, is aperiodic and ergodic. Thus, it must have a unique stationary distribution π_σ (see for example [70, Theorem 7.7]).*

An interesting argument which can now be applied is that given the stationary distribution of the economy, each trading period becomes a finite state game.

Lemma 5 *For every idle packet, each trading period of the economy corresponds to a finite strategic game.*

Proof 5 *Let σ_m be a mixed strategy of the whole economy and P^{σ_m} the corresponding transition matrix of the Markov chain of the economy. Note that P^{σ_m} is a convex combination of the transition matrices P^σ that correspond to the pure strategies σ in the support of σ_m. Moreover, let π_{σ_m} be the stationary distribution of the Markov chain for transition matrix P^{σ_m}. We assume that the utility of each player (packet) for the profile σ_m is the expected value of the player in the stationary distribution π_{σ_m}. In this way, we obtain for each trading period a finite game where every packet of the queue is a player. The strategy of the packet is its trading strategy.*

This leads us to the following theorem, which holds under plausible assumptions.

Theorem 1 *A NE exists where packets perform trades.*

Proof 6 *Since each trade is a finite game, the classic theorem of Nash [74, 75] assures that there is at least one mixed Nash equilibrium. However, the state of the economy where no packet participates in trades is a trivial NE where no trades take place. We have to show that there at least one more NE. A nice property of the current proof technique (due to Gintis [34]) is that we can impose conditions on the equilibrium point. We can assume a restricted version of the economy, where each packet has a non-empty pure*

trading strategy set. In a sense each packet is enforced to accept at least some types of profitable trades every time it is possible.

In the restricted economy each round is again a finite game and, consequently, it has a mixed NE. This time the NE has trades assuming that packets with different utility functions exist in the population. Assume now a NE state of the restricted game in the original unrestricted economy. It can be shown that, assuming appropriate utility functions for the packets, if we relax the forced-trade restriction, then no packet has an incentive to unilaterally change its strategy. That is, there exists a NE with trades for the original PacketEconomy.

Pipelined Shuffling. A core operation of the PacketEconomy is the random pairing of the packets that takes place in each trading period to generate the trading pairs. We present a new parallel algorithm that can support the random pairing procedure in real time. The new algorithm (Algorithm 6) is a parallel, or better, a pipelined version of the random shuffling algorithm of Fisher-Yates, which is also known as Knuth shuffling [112, 56]. The Fisher-Yates shuffling technique was introduced in [30], later Durstenfeld [23] proposed a corresponding $O(n)$ algorithm, and finally Knuth [56] popularized Durstenfeld's algorithm. The random shuffling of Fisher-Yates is a simple and elegant way to generate a random shuffle with a single pass over an array of items. We call the new algorithm *Pipelined Shuffling*. Its core is a pipeline of q instances $0, 1, \ldots, q-1$ of the Fisher-Yates algorithm. At time t, instance k is at step $t + k \bmod q$ of the random shuffling algorithm.

Algorithm 6 Pipelined Shuffling

```
1: procedure SHUFFLE(int[] a)
       for i from 0 to q-2 do {
           j = random int in i ≤ j ≤ q − 1;
           exchange a[j] and a[i]}
2: end procedure
1: procedure PARALLELSHUFFLE(int[][] A)
       for i from 0 to q-1 do in parallel {
           processor i: wait for i periods;
           processor i: while (true) {Shuffle(A[i]);}
2: end procedure
```

Theorem 2 *The Pipelined Shuffling algorithm delivers a random shuffle every $O(1)$ parallel time steps on a q processors EREW PRAM.*

Proof 7 *A running instance of the Pipelined Shuffling algorithm contains q independent instances of the basic Shuffle algorithm. Each Shuffle instance is executed by one*

of the q processors. From the pseudocode of the algorithms we can conclude that each instance of the Shuffle algorithm is at a different round of its main loop. Moreover, each instance of the Shuffle algorithm has its own vector of q memory positions to store its current permutation and, thus, there is no possibility of two processors concurrently accessing the same memory position. In each round, one Shuffle instance completes its execution and delivers a random permutation of the q numbers $\{1, 2, \ldots, q\}$.

The PacketEconomy packet pairing algorithm uses the delivered random permutation to generate a random pairing in $O(1)$ parallel time on $\lceil q/2 \rceil$ processors.

Theorem 3 *A random packet pairing can be generated every* $O(1)$ *parallel time on a q processors EREW PRAM.*

Proof 8 *From Theorem 2 we know that a random permutation can delivered with Pipelined Shuffling every* $O(1)$ *parallel steps. The algorithm to generate a random pairing from it requires* $\lceil q/2 \rceil$ *parallel processors and works as follows. A separate vector of q memory positions is used to store the final pairing. Each processor i of the involved processors reads the values* x_{2i} *and* x_{2i+1} *of the positions* $2i$ *and* $2i+1$ *of the permutation, respectively, and then writes into position* $2i$ *of the pairing vector the value* x_{2i+1} *and into position* $2i+1$ *the value* x_{2i}*. If q is an odd number, then one position will not be paired. The contents of the final vector specify for each position the corresponding paired position.*

The Scheduling Problem. The underlying algorithmic problem of the PacketEconomy is a scheduling problem of network packets. From the router's point of view, this problem is a single machine scheduling problem with a max weighted total wealth objective.

Definition 3 *Max-Total-Wealth Scheduling (MTW). A set of n jobs* j_i*, for* $i = 1, \ldots, n$*. Job* j_i *has processing time* p_i*, release date* r_i*, deadline* d_i *and weight* w_i*. Let* c_i *be the completion time of job i in a schedule. The objective is to find a non-preemptive schedule that maximizes the total wealth* $W = \sum_i w_i \cdot \max(d_i - c_i, 0)$.

The release date r_i is the time when packet i enters the queue and the deadline d_i is the time when the value of the packet becomes zero. For MTW on a network router the following assumptions hold: a) The queue discipline is work-preserving, meaning a non-empty router is never left idle, b) the number of packets in the queue at any time is bounded by a constant (the maximum queue size), and c) the packet sizes may differ by at most a constant factor. In this work, we assume that all packets are of the same size.

The complexity of the MTW problem depends on the assumptions made. Without deadlines, i.e., without a limit on the delay of each packet, an optimal

schedule can be obtained by applying a greedy rule like Smith's rule [98]. In particular, the router may simply serve in each round the packet with the largest cost factor c_i.

Theorem 4 *The MTW problem without deadlines can be optimally solved in polynomial time.*

This holds even for the on-line version of the problem where the router knows only the packets in its queue; the greedy rule gives a 1-competitive algorithm.

Theorem 5 *There is* 1*-competitive algorithm for MTW without deadlines.*

However, in realistic scenarios with IP packets, there are deadlines. Common IP packets have a time-to-live (TTL) field. In TCP, a packet that is not acknowledged within the specified time-out period is considered lost. The scheduling problem for packets with deadlines can be solved off-line as a linear assignment problem (LAP), where packets are assigned to time-slots (rounds). This approach is used in [36] for a min-weighted-tardiness problem that is related to the MTW problem.

Theorem 6 *The MTW problem with deadlines can be optimally solved in polynomial time.*

However, due to the on-line nature and the finite queue size of the PacketEconomy router, the above conventional scheduling algorithms do not seem to naturally fit the MTW problem of the PacketEconomy. Especially for window-based flows, where packet transmission is a closed loop, the order in which the queued packets are served influences, if not determines, the next packet that will enter the queue. Thus, even the on-line assumption may not be appropriate. A different approach to study the scheduling problem of the PacketEconomy is to consider the (average) packet rate of the flows, as shown in the following example.

Example 1 *Assume a scenario with window-based flows and 5 economy packets and 5 business packets. There is a deadline of 40 rounds on the maximum delay of the economy packets. Moreover, all business packets have to be treated equally. The same holds for the economy packets. Consider the scenario where each economy packet will be served with a rate of 1/40 packets/round and delay of 40 rounds and the business flows share the remaining bandwidth; each business packet is served at a rate of 7/40 packets/round and delay 40/7 rounds. This is an upper bound on the rate of total wealth for the router for this scenario.*

5.4 The Effect of Trades

The NE of the representative scenario shows that, in principle, money can be used at a microeconomic level to coordinate network packets. By definition, the flows of the scenario can only benefit through the use of money; each trade is a weak Pareto improvement for the current state of the economy. In this section we further examine the effect of trades.

In the PacketEconomy, each packet can increase its utility by making trades. To show the potential of the approach, consider a packet of maximum priority that pays enough to make any trade that reduces its delay. In the analysis, we will assume that the probability of packet failures is very low, and thus ignore it. We focus on window-based flows, present an exact calculation for the average delay of this packet and then derive simpler, approximate bounds.

Lemma 6 *The average delay $E[d]$ of the packet is*

$$E[d] = \sum_{s=1}^{q} s \left(\frac{1}{q-2}\right)^{s} (s-1) \frac{(q-2)!}{(q-s-1)!} \,. \tag{5.3}$$

Proof 9 *Let $rand(L, U, s)$ be a uniformly random integer in $\{L, L+1, ..., U\}\backslash\{s\}$ and $pos(p)$ the current position of packet p. Then, the probability $Pr[d > s]$ is*

$$= \prod_{k=1}^{s} Pr[rand(1, q-1, pos(p)) \geq s-k+1] = \frac{q-s-1}{q-2} \cdot \frac{q-s}{q-2} \cdots \frac{q-2}{q-2} \Rightarrow$$

$$Pr[d > s] = \left(\frac{1}{q-2}\right)^{s} \cdot \frac{(q-2)!}{(q-s-2)!} \text{ , and}$$

$$Pr[d = s] = Pr[d \leq s] - Pr[d \leq s-1] = \left(\frac{1}{q-2}\right)^{s} (s-1) \frac{(q-2)!}{(q-s-1)!}.$$

Applying the definition of the expected value completes the proof

$$E[d] = \sum_{s=1}^{q} s \left(\frac{1}{q-2}\right)^{s} (s-1) \frac{(q-2)!}{(q-s-1)!}.$$

Lemma 7 *Let $X^d_{\min}$ be the minimum of $n > 0$ discrete uniform random variables (RV) in $[L, U]$ and $X^c_{\min}$ be the minimum of n continuous uniform RV in $[L, U]$. Then, for the average values of $X^d_{\min}$ and $X^c_{\min}$, it holds that*

$$E[X^d_{\min}] \leq E[X^c_{\min}] \leq E[X^d_{\min}] + 1 \,. \tag{5.4}$$

Proof 10 *Assume a random variable X_c that is uniformly distributed in $[L, U]$, where L and U are integers, such that $L < U$. Let X_d be a random variable that is calculated from X_c in the following way:*

$$X_d = L + i, \text{where } i \text{ is such that} : L + i \cdot A \leq X_c \leq L + (i+1) \cdot A \ ,$$

where $A = \frac{U-L}{U-L+1}$. The random variable X_d corresponds to a uniform discrete random variable in $\{L, L+1, \ldots, U\}$. The absolute difference $X_c - X_d$ is not larger than one. Consequently, the absolute difference between the minimum $X^c_{\min}$ of m draws of X_c and the corresponding minimum $X^d_{\min}$ of the m values of X_d is also not larger than one. The same bound holds for the difference between the expected values of the minimums after k draws. Thus, we obtain that

$$E[X^c_{\min}] - 1 \leq E[X^d_{\min}] \leq E[X^c_{\min}] + 1 \ . \tag{5.5}$$

Moreover, note that $k >= 1$, the average minimum of k random draws will be in the lower half of the interval $[L, U]$, that is in $[L, \frac{L+U}{2}]$. Real values in this interval are on average rounded to smaller integer values in the above rounding procedure for X_c to X_d. Thus, the average discrete minimum will not be larger than the average continuous minimum. Thus,

$$E[X^d_{\min}] \leq E[X^c_{\min}] \leq E[X^d_{\min}] + 1 \ . \tag{5.6}$$

Lemma 8 *The average delay of the packet does not exceed $\frac{-1+2b+2\sqrt{2b(q-2)}}{2b}$. For $b = 1$ the bound is $\frac{1}{2} + \sqrt{2(q-2)}$.*

Proof 11 *A packet that enters at position $q-1$ has been served when it advances at least q positions. Note that each random trading partner corresponds to a uniform random number in $[1, q-1]$. To admit a more elegant mathematical treatment we prefer the continuous distribution. Lemma 7 makes this possible.*

Assume that a packet has just entered the queue at position $q-1$. Let b be the number of trading periods per router round. Assume that the packet spends at least k rounds in the queue until it reaches position 1. During these k rounds the packet will make bk random draws and will make k single position advancements. From Lemma 9 we obtain that the average value of the minimum of the bk random draws is

$$\frac{1}{bk+1}(q-2) \ .$$

Lemma 9 *Let $X_1, X_2, \ldots, X_k$ be continuous uniform random variables in $[0, U]$ and let $X_{\min} = \min_{i=1,\ldots,k} X_i$. Then $E[X_{\min}] = \frac{1}{k+1}U$.*

Proof 12 *The probability distribution of each continuous uniform random variable X_i is $F_{X_i}(x) = \frac{x}{U}$. The probability distribution of the minimum $X_{\min}$ is*

$$F_{X_{\min}}(x) = 1 - \prod_{i=1}^{k} \left(1 - F_{X_i}(x)\right) .$$

Now, applying the definition of the expected value function completes the proof of Lemma 9.

Note that the average number of rounds and draws until it achieves its best draw is $(k+1)$ and $(bk+1)/2$, respectively. We will add one to the value of the average minimum draw, because the minimum position that can be traded is position 1. Position 0 is the one that is currently being served.

Now, assume that after the k rounds and bk draws the packet advances for h additional rounds until it reaches position 1. From position 1 it needs a final round to proceed to position 0 and be served. Thus, the total delay of the packet is $k + h + 1$, and

$$\frac{1}{bk+1}(q-2) + 1 - (k+1)/2 - h - 1 \leq 0 .$$

We solve for k and obtain that the larger of the two roots of k is

$$k = \frac{-1 - b - 2bh + \sqrt{(1 + b + 2bh)^2 + 4b(2q - 5 - 2h)}}{2b} . \tag{5.7}$$

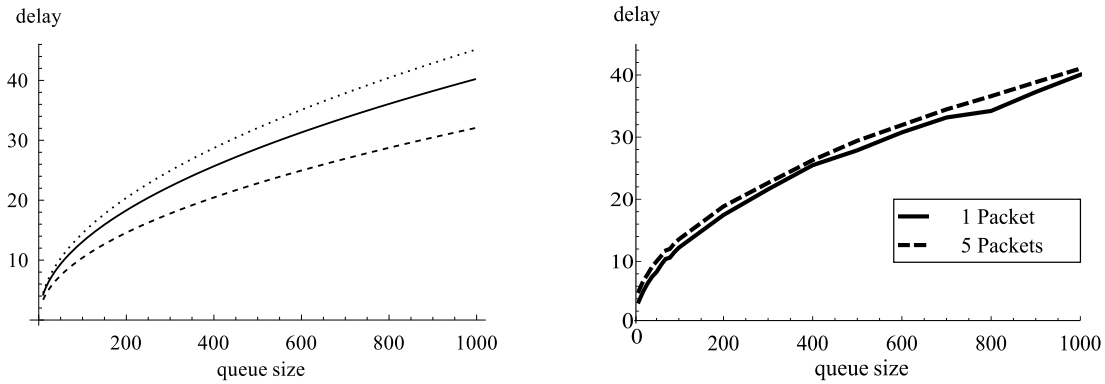
The total delay $k+h+1$ is minimized at $h = (1-b)/(2b)$. Substituting $h = (1-b)/(2b)$ in Equation 5.7 gives that the minimum value of $k + h + 1$ is

$$k + h + 1 = \frac{b - 1 + 2\sqrt{2b(q-2)}}{2b} .$$

The average delay cannot be larger then the above value. This completes the proof of Lemma 8.

The above lemma can be generalized to the case where only one packet in every $c > 0$ packets in the queue is ready to sell its position. We simply assume b/c trading periods per round. Then, the average delay of the business packet is not larger than $1 - (c/2) + \sqrt{2c(q-2)}$. Similarly, we can show:

Lemma 10 *The average delay of the packet is at least $\frac{-1+2b+\sqrt{1-8b+4bq}}{2b}$. For $b = 1$ the bound is $(1/2) + \sqrt{4q-7}$.*



(a) The exact average delay (inner line) and the lower and upper bounds on the average delay.

(b) Experimental measurement of the delay for the cases of one business packet and five business packets.

Figure 5.4: Delay of the business packet with respect to the queue size.

Proof 13 *Assume $k+1$ rounds with $b = 1$. The continuous average minimum of k rounds with b trading periods is $(q-2)/(bk+1)$. From Lemma 7 we obtain that the average discrete minimum is at least $(q-2)/(bk+1) - 1$. We will add one to this number because the minimum possible trade is position 1. In the best case the minimum is achieved with the first draw. In the remaining k rounds the packet will be (in any case) shifted by $k-1$ positions (in each round except the round when it entered the queue). This number of simple steps/shifts is subtracted from the min. Finally, a delay of one round is needed to serve the packet, when it arrives at position 0. Consequently,*

$$\left(\frac{1}{bk+1}(q-2)-1\right)+1-(k-1)-1 \leq 0\,. \tag{5.8}$$

From the above inequality and the fact that k is positive we obtain

$$k \geq \frac{-1+\sqrt{1-8b+4bq}}{2b}\,.$$

Using $b = 1$ the expression is simplified to $k \geq -(1/2)+\sqrt{4q-7}$. Thus the average delay is at least

$$k+1 \geq \frac{1}{2}+\sqrt{4q-7}\,.$$

This lemma too, can be generalized to the case where only one packet in every $c > 0$ packets in the queue is ready to sell its position. In this case the average delay of the business packet is not less than $\frac{1}{2}(2-c)+\frac{1}{2}\sqrt{c^2-8c+4qc}$.

In Figure 5.4, analytical and experimental results for the delay of the business packet are presented.

5.5 Conclusions

We presented an economy for network packets and showed the existence of NE where money circulates to the benefit of the flows. The basic computational step of the PacketEconomy can be executed in $O(1)$ parallel time on fairly simple multi-core hardware, making it appropriate for modern network router demands.

There are several other issues that have to be addressed for such a model to be of practical importance. For example, a greedy flow may submit economy packets to the network simply to collect money. A realistic economic model has to anticipate such scenarios and address them with appropriate rules. One approach could be to have the router restricting the final budget of any packet to be non-positive, or more effectively, impose router-entry costs on every packet depending on the current load.

Overall, we examined how money can be used at a microeconomic level as a coordination tool between network packets and we believe that our results show that the PacketEconomy approach defines an interesting direction of research for network games.

CHAPTER 6

Implementing PacketEconomy: Distributed Money-based QoS in OMNET++

6.1 Introduction

The Internet provides the infrastructure for multiple independent network traffic flows. This infrastructure and its resources are limited and shared between these flows, each of which attempts to optimize its own performance. As a result of entities sharing a limited common resource, with individual optimization targets, competition arises between these flows. Thus, a game-theoretic approach can be used to examine the issues that arise, an approach which has been taken in several works with some of the early ones being [57, 85], and overviews of which are presented in [4, 77]. Congestion games in particular have also been addressed, with focus on the TCP/IP protocols, in [3, 32, 25, 93]. However, in previous approaches, the interaction between flows has been very limited and mainly indirect. Each flow typically can only control the amount and timing of the data it sends. This in turn affects the shared network resources available for both this and other flows. In game theoretic terms, the current strategies available to flows can only control the size of packets and the rate of their transmission. In this work, we present an implementation of PacketEconomy, a distributed quality of service (QoS) mechanism for network packets, aiming at allowing high performance, network-wide, fine-grained, user-controlled QoS presented in Chapter 5. PacketEconomy comprises two aspects, firstly allowing flows to formulate their strategies in a more direct and clear way by using packet utility functions to express and packet budgets to "finance" their QoS requirements.

Secondly, it allows the flows to interact while waiting in router queues, providing opportunities for mutually beneficial exchanges between packets. Additionally, we implement the platform in a non-intrusive way, allowing for gradual and opt-in participation, without affecting flows which do not participate in the PacketEconomy.

In this work, we present a realistic implementation of PacketEconomy, within the OMNET++ discrete-event simulator [106, 107], using the INET network simulation library [108] and the experimental evaluation of the implementation.

6.2 Related Work

The general problem addressed by our work is that of providing QoS for network flows. Within that context, we focus on solutions which work when assuming selfish competitive flows, instead of cooperating ones, since in practice the flows are created by independent and selfish end-users. Our proposal comprises using money and packet trades as a coordination mechanism at the microeconomics level, described in detail in Chapter 5.

If in the problem addressed the game-theoretic aspects are ignored, then PacketEconomy still provides a simple, fast and hardware acceleratable solution. These characteristics are not merely an advantage of our solution, but are hard requirements imposed on any potential solution by the nature of the problem, i.e. processing of large numbers of packets with minimal overheads on routers with limited resources. Other works trying to solve the same problem include [48, 50, 51, 62, 114, 59, 60] with a good overview presented in [101], however it is important that non-decentralized, typically computationally inefficient or complex-to-implement algorithms are avoided.

Another aspect concerns the role of the service providers, which PacketEconomy remains agnostic of, such that the providers implementing the mechanism in their routers are relegated to "dumb pipes" with no strategic interests (as far as providing the service to their users is concerned) while the strategic interaction takes place between network end-users. On the other hand, some approaches [8, 2, 63] model the problem of QoS from the standpoint of efficiency or performance for network service providers, a problem certainly interesting and important, but which is not necessarily aligned with the interests of the end-users.

If QoS is to be performed, a means of dynamically modifying the end-to-end delay is needed, which affects both rate- and window-based flows. End-to-end delay is the sum of transmission delay (the time taken to transmit the data of a packet over the network links, which relates to link bandwidth), propagation delay (the time taken for the signal to propagate between link endpoints,

which relates to the physical medium and the distance of the link), processing delay (the time taken within routers to process the packet headers), and queuing delay (the time spent waiting in router queues). Transmission delay can be changed by physically changing the network and thus changing the bandwidth available, however this is not something that can be varied dynamically. Propagation delay is also a physical property and cannot be changed dynamically. Processing delay is also rather inflexible to change, since it is a required step for every packet passing through a router. The only part of end-to-end delay which remains and which can relatively easily be varied dynamically, is queuing delay. By manipulating the order in which packets are served, it is possible to increase or decrease queuing time, and as an extension, end-to-end delay. Moreover, in highly congested networks, where QoS is more needed, queuing delay represents a larger part of the total end-to-end delay. Since packets spend a significant portion of their time waiting in packet queues within routers, we claim that this would be an ideal place to perform coordination and provide QoS. A common alternative formulation considers the routing problem [15], i.e. not deciding the order of service in queues but deciding on which output queue and thus effectively which network path to take to implement QoS. This framing is also often coupled with the network service provider-centric view of the problem. In this case, taking different routes impacts player utility by experiencing different congestion levels, delays and bandwidth limitations over different network links. However, we would like to provide end-to-end, user-controlled QoS, but typically end-users are not able to control routing decisions in routers. Thus, we expect that our focus on scheduling in the router queues, maintaining the ability to affect both delay and throughput, is more amenable to a realistic implementation.

We reinforce our commitment to the importance of realism by providing a proof-of-concept implementation of our solution in OMNET++. This implementation consists of additional logic at the router and the end-points, operates on IPv6 flows and utilises an additional extension IPv6 header on each packet. We chose to make these assumptions and impose limitations on our implementation in order to maintain practicality. In contrast, other approaches result in simpler solutions, but are coupled with the disadvantages of being less realistic and harder to implement, due to the abstractions performed which fail to take into account practical concerns.

We have also taken into account the fact that changes in network infrastructure are slow, especially where non-programmable routers are used. To address this issue, we have designed the mechanism in such a way that PacketEconomy can offer advantages even if only a part of the network participates (end-users or routers). Thus, our solution allows for a piecemeal introduction, taking advantage of the new features where available, and falling back to the default

implementation where not. More specifically, non-participating end-users will experience service as if no QoS was being applied to their packets, although they do have a positive incentive to participate. Additionally, different segments of a network path may not support our solution (non-PacketEconomy routers), but even if only a subset of the segments does, then, any packet travelling within those segments will take advantage of our solution within them. This contrasts with solutions which require a total switch to the alternative mechanism.

6.3 Implementation

In this section we describe the adaptation of the theoretical model described in Chapter 5 to an implementation based on the OMNET++ discrete-event simulator.

6.3.1 Packet Utility Functions

The theoretical model described in Chapter 5 uses linear packet utility functions as examples, but in general any positive and monotonically decreasing function can be used. We have generalized the function definition to a larger class of functions to allow flows to express more complex QoS requirements and also to illustrate the generality of our overall approach. For our experiments, we have decided on a fixed form for the packet utility functions with three parameters. The utility function is defined as:

$$d_p(t) = \begin{cases} b - at^c & 0 \leq t < \sqrt[c]{\frac{b}{a}} \\ 0 & t \geq \sqrt[c]{\frac{b}{a}} \end{cases} \text{ with } b \geq 0 \text{ and } a, c > 0 \tag{6.1}$$

where, t is the time which has passed since the packet has been sent, and a, b, c are the parameters which define the starting point (at $t = 0$) for the utility as well as the rate of loss of utility as time passes. Once the utility reaches zero, at $t = t_0$, it decreases no more. The described utility function is monotonically decreasing over $t \geq 0$. We have selected this function form because it allows for easy calculation of the t_0 point, it also allows sufficient flexibility in defining the utility function, it only requires three parameters and for specific values of c it can be efficiently implemented directly in hardware. Although any number of parameters can be used, since the parameters need to be carried along with the packet, a trade-off between flexibility and network overhead needs to be

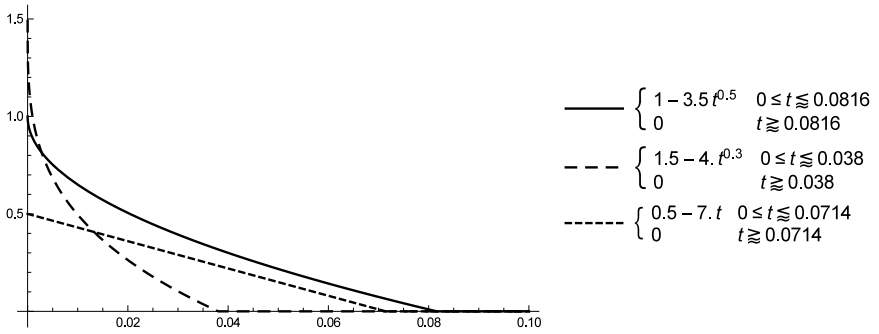


Figure 6.1: Example packet utility functions. The point where the functions meet the t axis is t_0.

made. A few examples of the packet utility functions which are possible using this function form are presented in Figure 6.1.

6.3.2 Compensation Price

When two packets perform a trade, each one has to specify its bid or ask price, depending whether it is a buyer or seller. In order to specify this price, each packet needs to calculate how much utility will be gained or lost if the trade takes place. An analysis on the optimal compensation prices or rate-based and window-based flows when linear utility functions are used is presented in Chapter 5. This work generalizes which kinds of utility functions can be used in the manner described by Equation 6.1. Consider a packet which, without participating in the trade, has delay $d_p(t_1)$ and a balance of $m_p(t_1)$, where $t_1 \leq t_0, t_2 \leq t_0$. If it participates in the trade it will receive a new estimated delivery time $d_p(t_2)$ and a balance $m_p(t_2) = m_p(t_1) + \rho$. For rate-based flows, the total benefit must be the same or higher, thus the compensation price ρ becomes:

$$\begin{aligned} d_p(t_1) + m_p(t_1) &= d_p(t_2) + m_p(t_2) \Leftrightarrow \\ b - at_1^c + m_p(t_1) &= b - at_2^c + m_p(t_1) + \rho \Leftrightarrow \\ \rho &= a(t_2^c - t_1^c) \end{aligned} \tag{6.2}$$

As described in Chapter 5, window-based flows have to wait for an acknowledgement of receipt of a packet in order to send another packet. Therefore, delaying a packet not only affects this packet's benefit but also the next one's which will also be additionally delayed. Taking this behaviour into account, for window-based flows the total benefit *rate* needs to be the same or higher, thus

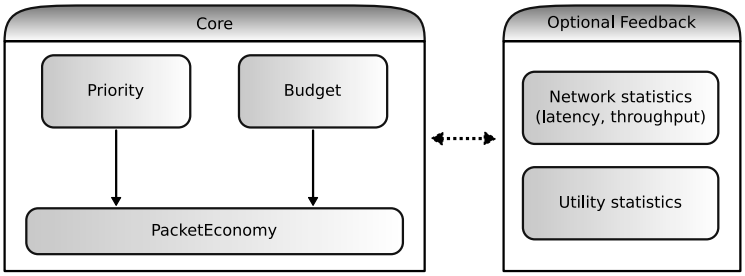


Figure 6.2: Viewed as a service, PacketEconomy requires priority and available budget as inputs. Optionally, network and utility statistics feedback can be used to deduce utility function parameters.

the compensation price is:

$$\begin{aligned} \frac{d_p(t_1) + m_p(t_1)}{t_1} &= \frac{d_p(t_2) + m_p(t_2)}{t_2} \Leftrightarrow \\ \frac{b - at_1^c + m_p(t_1)}{t_1} &= \frac{b - at_2^c + m_p(t_1) + \rho}{t_2} \Leftrightarrow \\ \rho &= \frac{b(t_2 - t_1) + a(t_1 t_2^c - t_1^c t_2) + m_p(t_1)(t_2 - t_1)}{t_1} \end{aligned} \tag{6.3}$$

It follows that $\rho < 0$ when the packet is a buyer and $\rho > 0$ when it is a seller.

6.3.3 PacketEconomy as a Service

PacketEconomy provides QoS as a service to the endpoint users. To do so, it firstly requires one or more intermediate routers which are able to perform trades. These routers do not need any additional configuration from the endpoints and they can function completely independently, while the PacketEconomy mechanism itself is also stateless. Access to an accurate time source is useful but not required. Secondly, at the endpoints, besides the necessary modules, two parameters must be given, presented in Figure 6.2. The mandatory parameters are the priority values for each flow as well as the available budget. The priority parameter can be given directly by the user or can be derived automatically from a higher level configuration. The budget needs to be either given by the user or it can be retrieved, via an appropriate network service, directly by the budget provider, usually the ISP. Optionally, PacketEconomy can use feedback from previous traffic as well as the given priority and budget in order to deduce the appropriate utility function parameters.

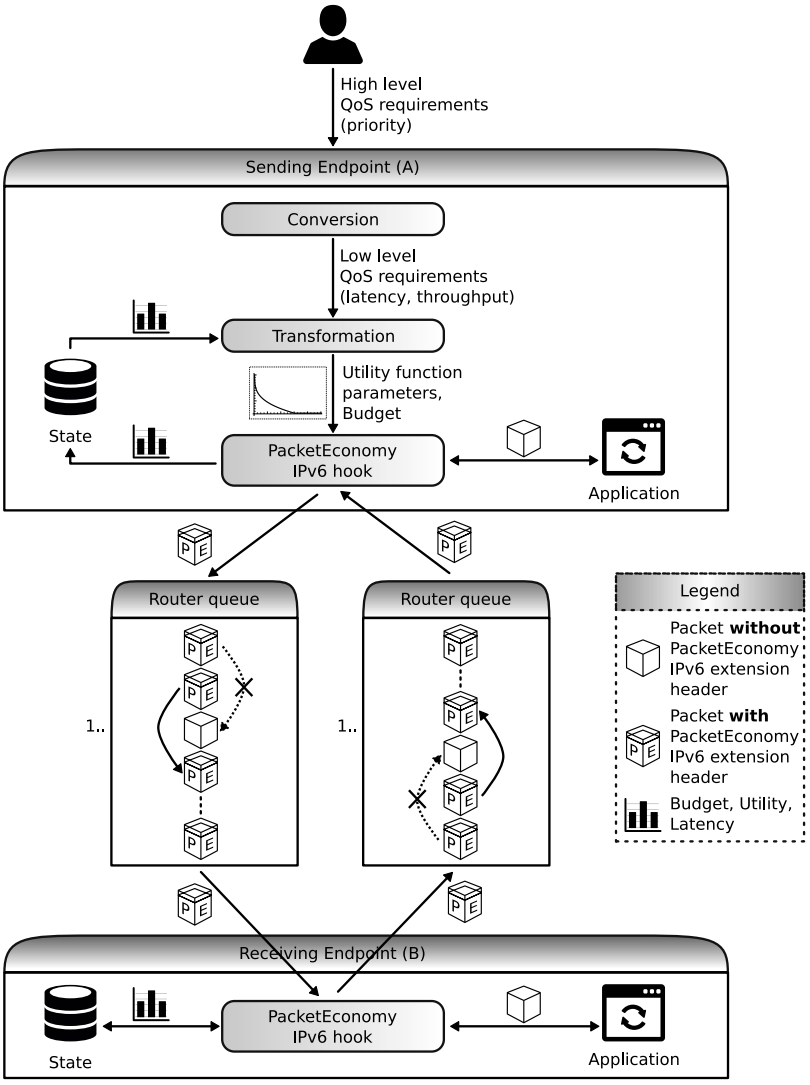


Figure 6.3: Overview of the operation of PacketEconomy. The PacketEconomy hook attaches and detaches the custom extension header at the endpoints. State is maintained to be used in deciding which utility function parameters and budget value to use. Routers perform trades statelessly, directly rejecting pairs non-PacketEconomy pairs. Feedback is sent from the receiving endpoint B to the original sending endpoint A to inform its parameter selection.

6.3.4 Operation Overview

The overall PacketEconomy operation is presented in Figure 6.3. At the sending endpoint A, a user decides upon the high-level QoS requirements for flows, which can be predefined, based on application profiles, or have otherwise provided default values, in the form of flow priorities. These are then first converted to relatively static delay and/or throughput QoS requirements. The requirements in turn are then converted to more dynamic utility function parameters (a, b, c) as well as a packet budget. The parameters and budget are used when the flow upon which QoS is applied sends an IPv6 packet. Before being sent from the sending endpoint A, a PacketEconomy hook handles the normal IPv6 packet and attaches a PacketEconomy extension header containing the utility function parameters and related PacketEconomy data. This header is then used to perform trades in any PacketEconomy-enabled routers along the path to the receiving endpoint. When pairing packets, trades are only performed if both packets in a pair are PacketEconomy-enabled, otherwise the trade is directly rejected. At the receiving point B, a PacketEconomy hook handles the packet, removes the extension header and delivers the packet as normal to the receiving flow endpoint. It also records relevant network and utility statistics. When the receiving endpoint B has to send a packet to the original sending endpoint A, it attaches a PacketEconomy extension header with feedback. This is eventually received at the original sending endpoint A, where the feedback is stripped and recorded. The next time a packet has to be sent the new feedback will be used to select appropriate utility function parameters and budget values.

6.3.4.1 Adaptivity

Each endpoint needs to track both its own, as well as the other endpoint's budget and network performance. Specifically, each endpoint needs to track the total packet benefit from received packets, which constitutes a form of return-on-investment information, and attach this information when sending packets to the other endpoint. The feedback from the opposite endpoint, along with information regarding the available budget and the flow priorities, allows each endpoint to adapt to changing network conditions. When network congestion increases, depending on the number of flows and their priorities, an endpoint may choose to spend its budget differently, taking into account both the priorities of each flow and how well spent the budget is for each flow. The user may also have a means of requesting additional budget from their provider in order to support their QoS needs.

6.3.5 Technical Details

The core of PacketEconomy has been implemented as a C++ library, independent of OMNET++, with a clean interface and implementation. We intended for the library to be used in OMNET++ initially, but we envisioned the ability to use the library with either other simulators, such as ns-2 or ns-3, or with real networking stacks, such as that of the Linux OS, hooking into it via a user-space networking hook. We also intend to provide on-line access to a more refined version of it via an open source license. For this implementation, version 4.6 of the OMNET++ simulator has been used in conjunction with version 3.2.0 of the INET networking library. The part of the PacketEconomy model which is specific to OMNET++ has been implemented in the form of two OMNET++ modules, extending pre-existing INET modules. The first one extends the standard `IPv6` stack module, in order to read, write and process the PacketEconomy IPv6 headers on incoming and outgoing packets. This module is only used on endpoint nodes. The second extended module is an alternative queue which is used in Ethernet interfaces for outgoing packets. This module implements the PacketEconomy trading on the queued packets and is used only in routers. The module `IPv6PE` extends `inet.networklayer.ipv6.IPv6` and is used within module `StandardHost6PE` which extends `inet.nodes.ipv6.StandardHost6`.

6.3.5.1 Extension Header Description

The PacketEconomy extension header contains the fields which encode the d_p packet utility function (the a, b and c parameters) as well as the available budget. Each field is represented by a 32-bit IEEE 754 floating point number and as a result the overhead involved is 128 bits. In addition, the IPv6 extension header itself requires another 16 bits, thus a total of 144 bits or 18 bytes are required. To decrease this overhead, a smaller floating point number representation may be used with some numeric accuracy compromise. In this version of the experiments no adaptivity is implemented and thus the feedback header is not used on returning packets. If it were to be implemented, two 32-bit IEEE 754 floating point number fields would be the maximum required, one for the accumulated budget and one for the estimation of the average one-way delay.

6.3.5.2 The TCP/IP Stack at Endpoints

At the endpoints the TCP/IP stack has been modified mainly at the Network/Internet layer. In particular, the Internet layer is used as the central point where the appropriate per packet processing is performed. We only intervene at the IPv6 layer where we attach / detach the IPv6 extension header containing the

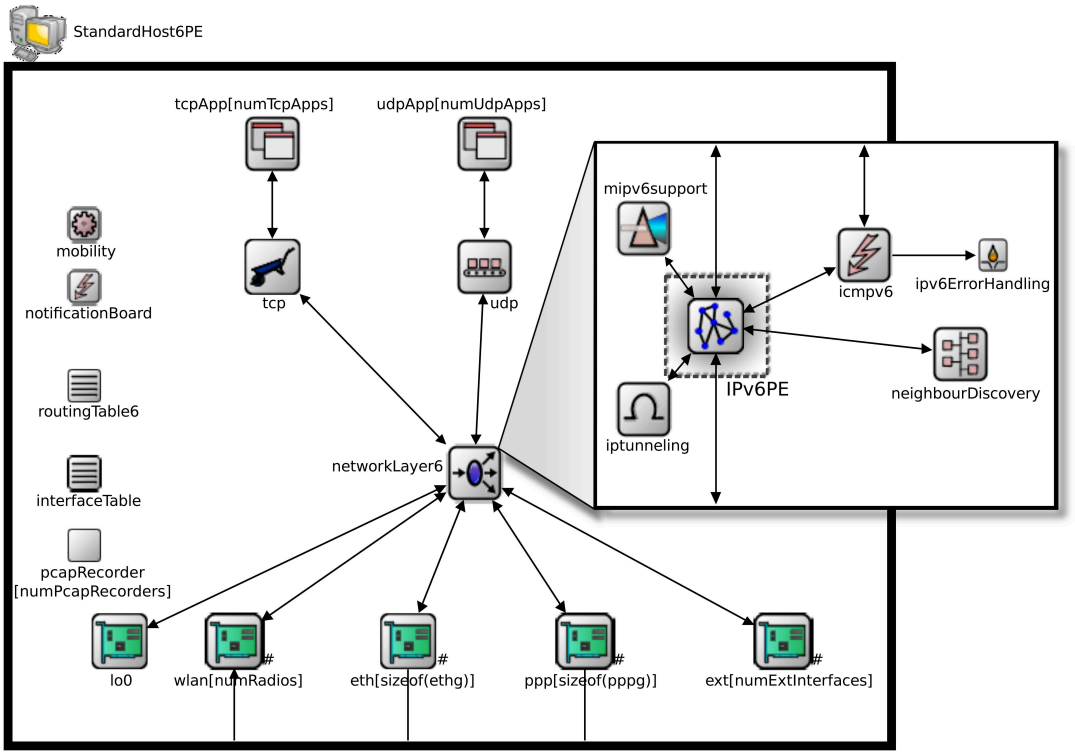


Figure 6.4: Graphical representation of OMNET++ module `IPv6PE` within `StandardHost6PE` highlighted by a dashed frame.

PacketEconomy information. Additionally, the flow priority and packet utility function parameters are defined within each application, although this does not mean that these parameters are considered to be part of the application layer in terms of the network stack; they are placed there for reasons of implementation simplicity. These definitions are used as packets pass through the Internet layer. Finally, the link layer is unmodified and unused.

6.3.5.3 The TCP/IP Stack at Routers

Within PacketEconomy-enabled routers we only modify the standard output router queues, thus avoiding any interaction with the routing functionality itself. In general, we manage the queue as normal, but allow PacketEconomy trading to be performed between IPv6 packets with the PacketEconomy extension header present. When trading is successful for a pair of packets, only their order and their PacketEconomy extension header is modified. The link layer is unmodified and unused. Also, depending on the queue admission policy used, when packets are dropped they are done so before entering the queue and thus no

money is lost from the economy during PacketEconomy trading.

Specifically, when a new packet arrives it is added to the queue, as it would normally be. Also, packets are dequeued and sent by the queue as they would normally be. All PacketEconomy processing is performed on the queue during the time-frame within which a packet is being sent. In the simulation, one trade round is performed per packet sent by the queue, however this can be changed if necessary.

The packets in the queue are defined as p_i where $i \in \{0, 1, .., |Q| - 1\}$ and $|Q|$ is the size of the queue each time a trade round is performed. The first packet (p_0) is considered to be the one being sent and thus does not participate in the trades. Thus, packets $p_{tr} = (p_1, p_2, .., p_{|Q|-1})$ participate in trades. If a packet arrives during the trade round, it will be held but will not participate in the currently executing trade round. The sequence of the packets in the queue is not used when pairing them; instead, the participating packet sequence is permuted randomly yielding p'_{tr}. It should be noted that this permutation does not affect the actual sequence of the packets in the queue, as it just a part of the pairing scheme. Afterwards, the trading pairs are created by taking sequential neighbouring and non-overlapping trading pairs: $T_i = (p'_{2i}, p'_{2i+1})$ where $i \in \{1, 2, .., \lfloor (Q-1)/2 \rfloor\}$. The trading negotiation and exchange is performed as described in Chapter 5. However, in the course of taking PacketEconomy from a theoretical model to a real network implementation some issues appeared which had to be addressed. Firstly, the random nature of the packet pairings will under normal conditions produce out-of-order packet sequences. This negatively affects flows, especially window-based flows such as TCP, which will reduce their throughput by assuming the reordering to be indicative of adverse network conditions. Thus, reordering prevention has been implemented such that trades do not result in packets of the same flow being serviced in a non-FIFO manner. This is implemented efficiently using the HL-Hitters mechanism described in Chapter 4. Secondly, in contrast to Chapter 5, trades are directly rejected when the pair of packets belong to the same flow because allowing them constitutes a waste of computational effort in the context where all the packets of the same flow used the same utility function parameters. Finally, a game-theoretic concern has been addressed by directly rejecting trades when the buyer packet is larger than the seller packet (discussed further in Section 6.6.2).

It is also implied by the definition that if the number of participating packets is odd, then one packet will not participate, chosen randomly. Afterwards, the packet trades are attempted for each pair. In the simulation these are performed sequentially, but a hardware implementation could easily implement them in parallel since each packet pair is independent from the other pairs.

One issue which may be raised is the computational cost of inspecting IPv6 extension headers, required in this implementation. If this cost is prohibitive, it

would make sense to only execute PacketEconomy on routers which are closer to the edges of the network, since the transmission speed is typically lower there and thus the packet rate needed to be served is also lower and where congestion is typically higher due to the network practices service providers typically implement, such as high contention ratio.

6.3.5.4 Time Source Considerations

An accurate time source in both endpoints and intermediate routers aids in the determination of the total delay of the packet since its original sending time. Its availability allows for measuring the total time which has passed since the packet was send from its sending endpoint. In the OMNET++ implementation, we have used the global time source provided by the simulator.

If a global time source or sufficiently well synchronized local time sources are not available, there is a fall-back option possible, which calculates the time spent at each hop incrementally. At each hop, the time spent is the sum of the processing delay, the queuing delay, the transmission delay, and the propagation delay. The first two, i.e. the processing and queuing delay, can be calculated accurately by the host or router internal clock, with no requirement of time synchronization with other hops. The transmission time can be estimated very accurately (especially for wired or optical links) by the transmitting interface given the packet's length and the interface's bandwidth. Finally, the propagation delay is not typically known a priori by interfaces, but it can be estimated when it becomes significantly large (e.g by using an ICMP ping packet). The sum of these delays can be added by each hop to the total time spent field in the PacketEconomy packet extension header.

6.4 Experimental Setup

In this section the setup for the experiments is described including which parameters are used and how they are combined. An overview of all the evaluated experimental cases is provided in Figures 6.6, 6.7, and 6.8. For the experiments we have selected a representative set of queue admission policies (DropTail and RED [31]), priority policies (PacketEconomy, Deficit Round Robin [94], and Strict Priority), flow types (TCP and UDP) and other characteristics.

6.4.1 Non-QoS Configuration

We first describe the non-QoS-specific aspects of the experiments, leaving the QoS-specific ones for the end of the section. In particular, in this first part we

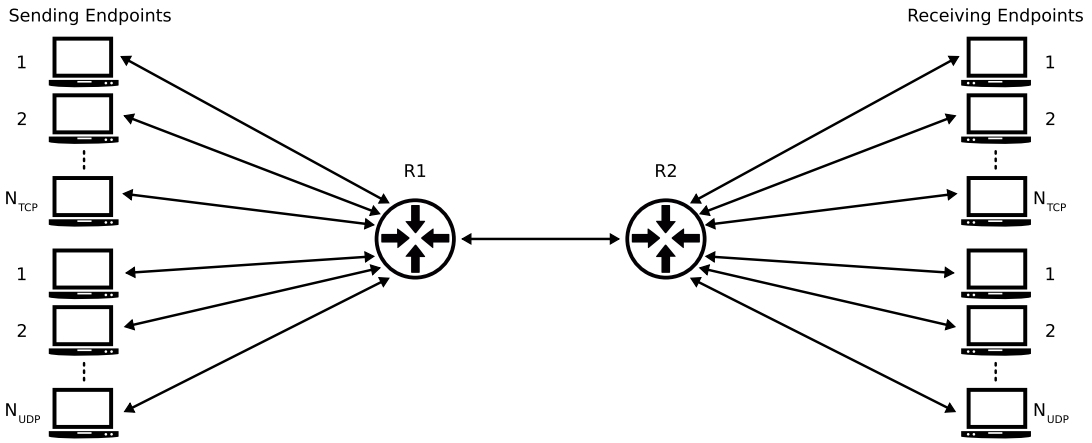


Figure 6.5: Dumbbell network topology with N_{TCP} TCP flows ($2 \times N_{TCP}$ endpoints) and N_{UDP} UDP flows ($2 \times N_{UDP}$ endpoints) for a total of $N = N_{TCP} + N_{UDP}$ flows ($2 \times N$ endpoints). All links are full duplex 10 Mbps with 50 ns propagation delay.

will not describe the PacketEconomy, the Deficit Round Robin (DRR) or the Strict Priority (SP) configurations, which are the ones that implement QoS.

6.4.1.1 All Cases

The following aspects are taken into consideration in the experimental setup, irrespective of the composition of flow types (e.g. TCP or UDP).

Layer 2 Setup

The network consists of a dumbbell topology, illustrated in Figure 6.5, with N hosts on each side ($2 * N$ total hosts) and 2 routers ($R1$ and $R2$) between them. The hosts on the left are the sending endpoints and the hosts on the right are the receiving endpoints. Each host is connected via Ethernet with exactly one link to either $R1$ or $R2$. The connections between endpoints and routers, as well as the single connection between the two routers, are full duplex 10 Mbps with 50 ns propagation delay. Each host uses either a TCP- or a UDP-based application through which it communicates with its peer.

Simulator Setup

The total simulation execution time is 160 simulated seconds with a warm-up of 110 sec (during which no statistics are recorded to allow flows to stabilize). Each experiment is executed in 5 repetitions with different random number generator

seeds. The random number generator affects the first packet send time for both UDP and TCP flows (a normal distribution $N(10s, 0.1s)$), the sending time interval for the CBR UDP flows (a normal distribution $N(30ms, 1ms)$) and when using RED on the queue Q, the admittance of incoming packets (due to the probabilistic nature of RED). This randomness is used to avoid synchronization and bias problems as much as possible.

Queue Parameters

Each possible combination of the following parameters is examined, in 5 repetitions as mentioned above.

- Router queue Q parameters:
  - Maximum Size ($|Q|$): 100 packets
  - Admission policy: DropTail, RED ($w_q = 0.002$, $min_{th} = 10$, $max_{th} = 100$, $max_p = 0.02$)

6.4.1.2 Flow Composition Cases

Additionally, we examine three flow composition cases: one in which only TCP flows are present, one in which only UDP flows are present and one in which both TCP and UDP flows coexist.

The TCP-only Flows Case

In each experiment all the flows use the same TCP congestion avoidance algorithm (Reno), with increased initial window support enabled, TCP window scaling support enabled, TCP delayed ACKs support enabled, SACK support disabled, the TCP Nagle algorithm disabled and an advertised window of 300000 bytes. In all cases the number of flows $N_{TCP} = N$ is 10 and the TCP Maximum Segment Size (MSS) is 1400 bytes.

The UDP-only Flows Case

In each experiment all the flows are of Constant Bit Rate (CBR) type and use the same send interval and payload size. The number of flows $N_{UDP} = N$ for each unique send interval and payload size combination is calculated as the number of flows required to achieve a given cumulative bandwidth requirement. Two sub-cases are created: one where the cumulative bandwidth requirements marginally pass the bottleneck router bandwidth, namely $10Mbps$, and one where the cumulative bandwidth requirements are 150% of the bottleneck router

bandwidth, i.e. 15*Mbps*. In both cases all protocol overheads are taken into account when calculating the number of flows. The former is used to assess performance at full queue capacity and the latter to assess performance under overload.

The TCP and UDP Flows Case

This case combines the TCP-only and UDP-only cases. As in the TCP-only case, the number of TCP flows N_{TCP} is constant, but half of what it was in the TCP-only case (5 instead of 10), while the rest of the TCP-only case parameters are used as before, including the two sub-cases for the MSS value. The combinations for the UDP flows include all the send interval and payload size combinations of the UDP-only case, but the two sub-cases for the cumulative bandwidth requirements are reduced to one where the bandwidth requirements are 50% of the bottleneck router bandwidth, i.e. 5*Mbps*.

6.4.2 QoS Configuration

In the following paragraphs the QoS-specific aspects of the experiments are described.

6.4.2.1 Layer 2 Setup

When PacketEconomy is enabled and therefore trading between packets is performed, it is only performed on the egress queue Q of the $R1 \rightarrow R2$ connection.

6.4.2.2 Queue Parameters

In addition to the non-QoS queue parameters, an extra parameter, the priority policy is examined as a part of the QoS configuration. The priority policy may be either PacketEconomy (with 1 trading round per served packet), Deficit Round Robin in the TCP-only cases (independent levels used = $\{N, \lceil N/2 \rceil, \lceil N/4 \rceil\}$) or Strict Priority in the UDP-only cases (independent levels used = $\{N, \lceil N/2 \rceil, \lceil N/4 \rceil\}$). In the TCP and UDP flows case, a hierarchical structure is used where QoS for TCP flows is performed via DRR and QoS for UDP flows via SP, and both policies are then merged via a secondary SP policy which gives absolute priority to UDP flows. A classifier is present before the DRR and SP queues which classifies the incoming packets. The number of independent levels refers to the number of classes used by the classifier, with a higher number meaning a finer-grained differentiation between flows at a cost of higher memory requirements.

6.4.2.3 Flow Composition Cases

In all flow composition cases the priority of the TCP and UDP flows is independent, i.e. the TCP and UDP flows have priorities that span the $[0, 1]$ range independently. This has been chosen so that the full range of priority values for both TCP and UDP flows can be examined.

6.4.2.4 Flow Priority

When assessing performance with a priority policy (PacketEconomy, DRR, SP) each flow is assigned a priority value p. In our experiments $p \in [0, 1]$, where $p = 0$ corresponds to the lowest priority and $p = 1$ to the highest priority. This value is used directly in SP as the priority and in DRR as the weight, but PacketEconomy needs the a, b, c parameters of the utility function to be defined. Thus, we have defined functions mapping the priority value p to the a (C_a, Equation 6.6), b (C_b, Equation 6.5) and c (fixed) parameters for PacketEconomy use.

This mapping function depends on an estimate of the baseline delay d_{bl}, which corresponds to the experienced delay of a packet without a priority value, i.e. is not affected by a priority policy. This d_{bl} value can be continuously updated by the network stack of each endpoint as the flow transmits and receives data, however in our experiments d_{bl} has been precalculated for each case, by executing a corresponding experiment for each case with the priority policy disabled and measuring the median delay of all the flows.

Additionally, a spread parameter tuple (s_l, s_u) is used to configure the intensity of the difference between the highest and lowest priority flows. In essence, it defines the range of values the t_0 parameter of the utility function will receive. Specifically, s_l and s_u are factors which define the highest priority flow's maximum delay t_0 and the lowest priority flow's maximum delay t_0 given a baseline delay d_{bl}. The intermediate flows' maximum delays are linearly interpolated between those two extremes.

Priorities for flows are meant to be more static, typically defined once and infrequently changed, as an expression of relative importance and QoS requirements of each flow. They may also be predefined for select application types based on common knowledge guidelines, for example, all VoIP flows should get maximum priority. The main parameter a user needs to configure is the (s_l, s_u) spread parameter, which essentially controls the amount of degradation lower priority flows will make in order to satisfy higher priority flows. Given constraints on the acceptable delay for all flows, this parameter can also be automatically controlled adaptively.

We use the Y_{t_0} function (Equation 6.4)

$$Y_{t_0}(p, d_{bl}, s_l, s_u) = d_{bl}s_l + (1-p)(d_{bl}s_u - d_{bl}s_l), \text{with } p \in [0,1], s_l, s_u \geq 0, d_{bl} > 0 \quad (6.4)$$

to perform a linear interpolation between the maximum delay $d_{bl}s_u$ corresponding to $p = 0$ and the minimum delay $d_{bl}s_l$ corresponding to $p = 1$.

We then use the C_b function (Equation 6.5)

$$C_b(p) = 1 + (10p)^2, \text{with } p \in [0,1] \quad (6.5)$$

to calculate a flow's utility function b parameter value. Other forms of C_b have also been found to work well, but this one performed consistently well in all the experimental cases.

Finally, we use the C_a function (Equation 6.6)

$$C_a(p, d_{bl}, s_l, s_u, c) = C_b(p)/(Y_{t_0}(p, d_{bl}, s_l, s_u)^c), \text{with } p \geq 0 \quad (6.6)$$

to calculate a flow's utility function a parameter value.

In our experiments we examine the performance for two spread parameter tuple cases $(s_l, s_u) \in \{(1,2), (1,4)\}$ in combination with all the previously described parameters and cases.

6.4.3 Collected Measurements

For each combination we measure the following statistics:

- Throughput per priority level.
- End-to-end delay per priority level.
- Packet drop per priority level.
- Utility (packet value d_p, balance m_p, total $d_p + m_p$).

A total of 2150 experiments have been executed, as 430 parameter combinations in 5 repetitions to assess performance. Additional experiments have been executed to determine the baseline delay for each experimental case as well as a much larger number of experiments during the development of the system. Due to space considerations, we have selected to present the above described representative subset of cases.

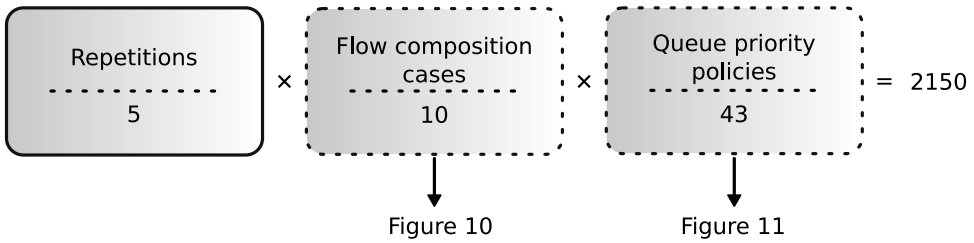


Figure 6.6: Overview of the experimental parameter combinations, producing the total number of experiments carried out. A total of 2150 combinations are examined.

6.4.4 Evaluation

We measure the performance of the PacketEconomy by comparing it to the performance of DRR and SP. The defining characteristics of the DRR policy are the number of classes and the weight of each class, while in SP the priority of each flow is the only parameter. Regarding the number of classes, we have considered cases where the number of DRR classes is identical to the number of flows, half of the number of flows and a quarter of the number of flows (rounding up when the fraction is non-integral), as described in section 6.4.2.2. For example, in a case where we have 68 UDP flows, the DRR cases evaluated are ones with 68, 34 and 17 classes.

The ideal case is considered to be the one where a DRR queue policy is used, which has one independent queue for each flow (i.e., each flow belongs to a separate class). The class of each flow coincides with the priority of the flow (linearly scaled).

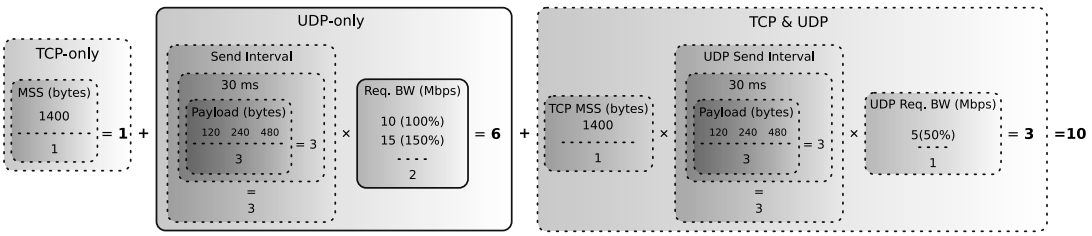


Figure 6.7: Flow composition case combinations. Three flow type cases are examined: TCP-only, UDP-only, and TCP & UDP flows. A total of 10 combinations are examined.

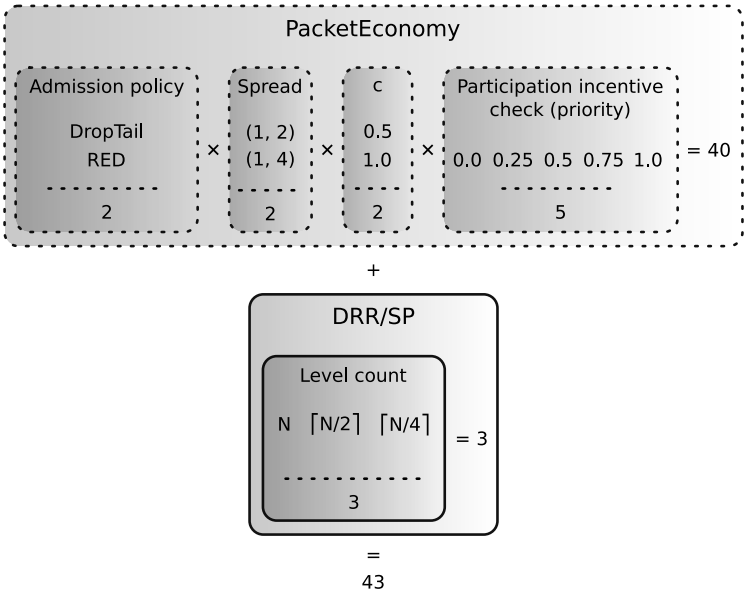
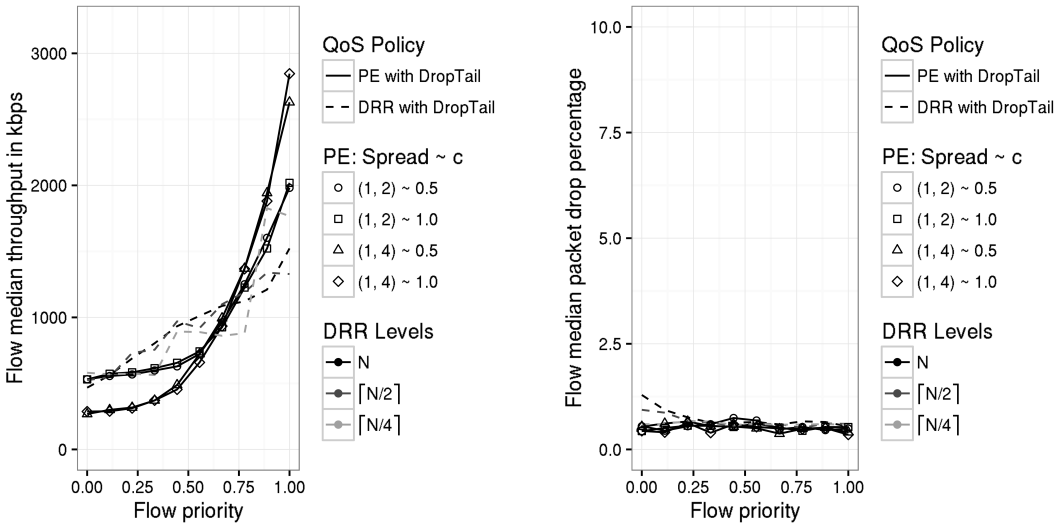


Figure 6.8: Queue priority policy combinations. PacketEconomy is investigated with different values for admission policy, spread, and c. Also, five priority levels are examined to check whether the flows have an incentive to participate in PacketEconomy. For DRR and SP, the number of levels used is examined. A total of 43 combinations are examined.

6.5 Experimental Results

In this section the results of the experiments are presented, organised on a flow composition case basis. For each flow composition case (TCP-only, UDP-only, TCP & UDP), the relevant metrics are presented. In particular, for TCP flows throughput and packet drop percentage are presented, while for UDP flows end-to-end delay and packet drop percentage are presented. Each diagram contains both the PacketEconomy results as well as the corresponding DRR (for TCP flows) or SP (for UDP flows) results, to allow for comparison between them.

Overall, the results confirm that it is possible, using appropriate utility function parameters, to control the distribution of throughput for TCP flows and delay for UDP flows meaning that PacketEconomy is effective as a QoS mechanism. The distribution of throughput and delay, is not a linear function of priority, however it is consistent in the sense that an increase (or decrease) in priority leads to an increase (or decrease) of throughput and to a decrease (or increase) in end-to-end delay.



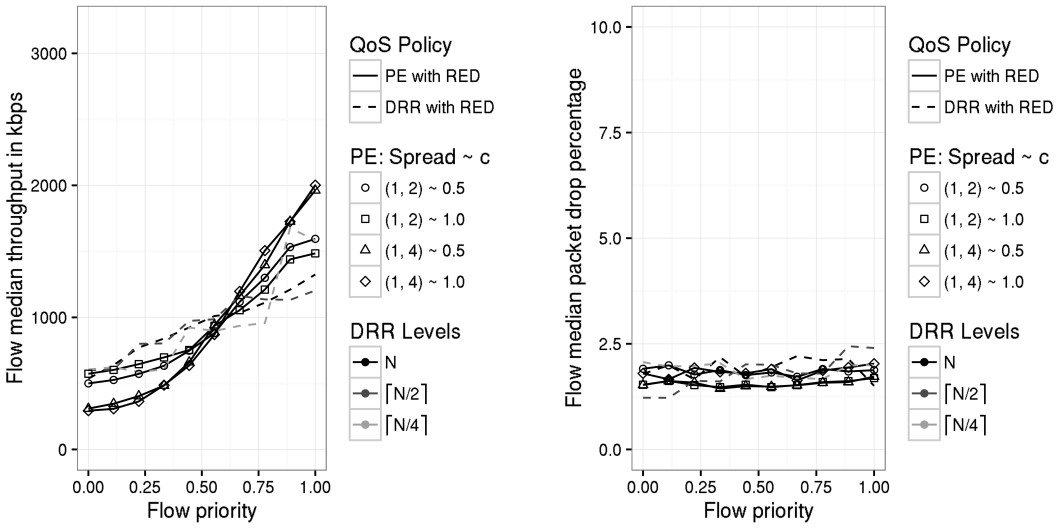
(a) Median throughput of flows in kbps. (b) Median percentage of dropped packets.

Figure 6.9: TCP-only flows case results per priority level with a DropTail bottleneck router queue. Throughput increases with priority, as expected, and two spread-c combinations distribute throughput more aggressively than the other two. Packet drop is low ($< 1\%$) and approximately the same for all priority levels.

In these experiments only one trading round was used per packet served, however multiple such rounds can be executed. The result would be a more aggressive distribution of throughput and delay (everything else being equal) and thus if a lower minimum delay or a higher maximum throughput is required without changing the utility function parameters, the trading rounds can be increased.

6.5.1 The TCP-only Flows Case

In this case we are concerned with flow throughput and packet drops. For the case where a DropTail bottleneck router queue is used, the results for throughput are presented in Figure 6.9a and for packet drop percentage in Figure 6.9b. PacketEconomy has similar performance to DRR as far as packet drop is concerned. In the case of throughput, PacketEconomy displays a non-linear, but smooth distribution, while DRR with N and $\lceil N/2 \rceil$ levels is largely linear, how-

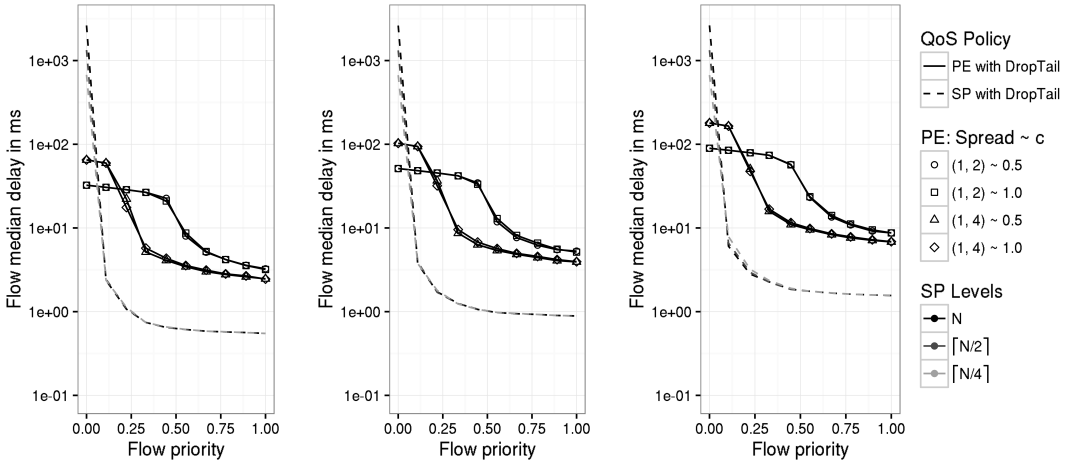


(a) Median throughput of flows in kbps. (b) Median percentage of dropped packets.

Figure 6.10: TCP-only flows case results per priority level with a RED bottleneck router queue. Throughput increases with priority, as with DropTail, but it is distributed less aggressively. Packet drop is low ($< 2\%$) but slightly higher than with DropTail and approximately the same for all priority levels.

ever DRR with $\lceil N/4 \rceil$ loses this property. We have seen from other experiments (outside the presented subset) that it is possible to select utility functions in such a way that the distribution is linear, however, this impacts the distribution of throughput and end-to-end delay in the TCP & UDP flows case. Also, we considered it useful to use the same utility function creation scheme for all flow composition cases to allow for easier and more objective comparison of performance.

Correspondingly, for the case where a RED bottleneck router queue is used, the results for throughput are presented in Figure 6.10a and for packet drop percentage in Figure 6.10b. The use of the RED admission policy makes the distribution of throughput with PacketEconomy more linear, due to limiting higher priority flows from getting a higher proportion of throughput. DRR is not affected significantly by RED and the comments on DRR's behaviour under DropTail hold for RED as well. Packet drop percentage with RED is, as expected, higher for both PacketEconomy and DRR, which perform almost identically in



(a) 120 bytes UDP payload size, 194 UDP flows.

(b) 240 bytes UDP payload size, 120 UDP flows.

(c) 480 bytes UDP payload size, 68 UDP flows.

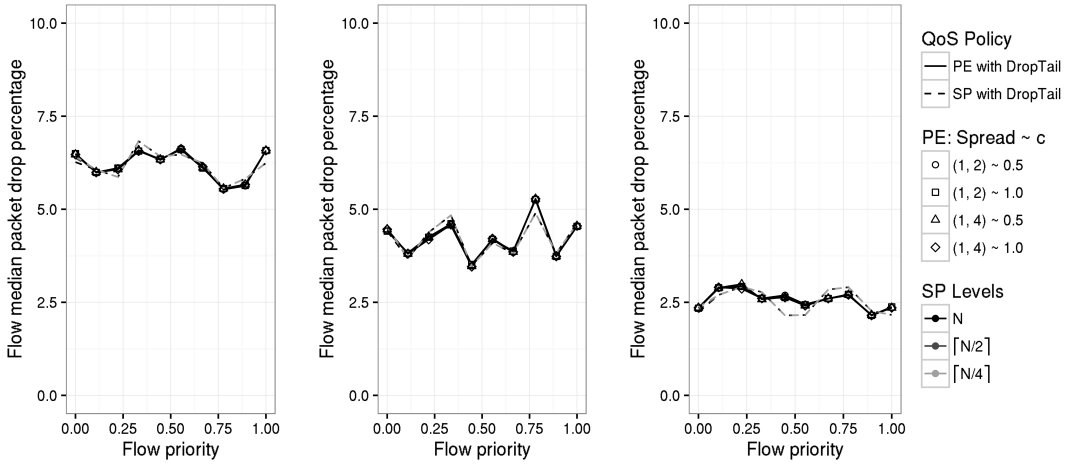
Figure 6.11: UDP-only flows case results for median end-to-end delay per priority level with a DropTail bottleneck router queue with 100% bandwidth requirements. End-to-end delay decreases with priority, as expected, and two spread-c combinations distribute delay more aggressively than the other two. Note: the y axis is logarithmic.

this respect.

6.5.2 The UDP-only Flows Case

In this case we are concerned with flow end-to-end delay and packet drops. Due to the large number of UDP flows used (68 - 194 with 100% bandwidth requirements, 102 - 290 with 150% bandwidth requirements) we only present 10 representative priority levels in the figures of this section, to preserve legibility. Both PacketEconomy and SP behave relatively smoothly with respect to priority levels and as a result, omitting some intermediate flow priority levels does not significantly impact the overall results.

For the case where a DropTail bottleneck router queue is used, the results for end-to-end delay are presented in Figure 6.11 and for packet drop percentage in Figure 6.12. PacketEconomy has similar performance to SP as far as packet drop is concerned. In the case of end-to-end delay, both PacketEconomy and



(a) 120 bytes UDP payload size, 194 UDP flows.

(b) 240 bytes UDP payload size, 120 UDP flows.

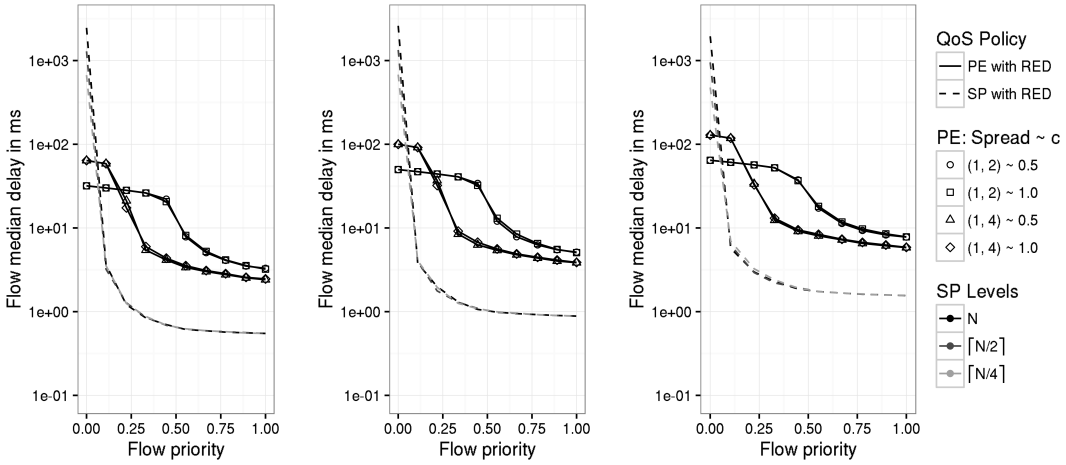
(c) 480 bytes UDP payload size, 68 UDP flows.

Figure 6.12: UDP-only flows case results for median packet drop percentage per priority level with a DropTail bottleneck router queue with 100% bandwidth requirements. Packet drop is between 7.5% and 2.5% decreasing as the number of flows decreases and as the size of the payload increases. It is approximately the same for all priority levels.

SP display a non-linear but smooth distribution. The number of SP levels does not affect performance measurably here. It can be seen that PacketEconomy distributes delay in a more equitable manner than SP, which penalizes the low priority flows disproportionately. Our solution prevents this problem by disallowing starvation of the low-priority flows through the application of the packet utility function deadline (t_0). In both cases, the number of flows and the size of the payload does not affect the basic delay distribution, although it is obvious that smaller payloads allow for a lower minimum delay.

End-to-end delay for the RED bottleneck router queue with 100% bandwidth requirements is shown in Figure 6.13 and is almost identical to the DropTail queue case shown in Figure 6.11. Packet drop percentage accordingly shown in Figure 6.14 mirrors the DropTail case shown in Figure 6.12.

End-to-end delay for both DropTail and RED bottleneck router queues with 150% bandwidth requirements is the same as in the respective cases with 100%



(a) 120 bytes UDP payload size, 194 UDP flows.

(b) 240 bytes UDP payload size, 120 UDP flows.

(c) 480 bytes UDP payload size, 68 UDP flows.

Figure 6.13: UDP-only flows case results for median end-to-end delay per priority level with a RED bottleneck router queue with 100% bandwidth requirements. End-to-end delay decreases with priority, as expected, and two spread-c combinations distribute delay more aggressively than the other two. Note: the y axis is logarithmic.

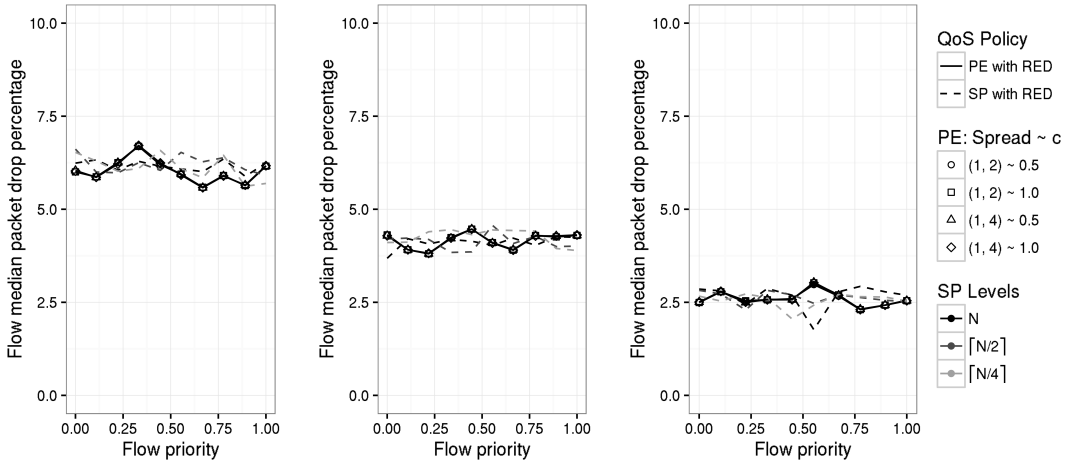
bandwidth requirements.

On the other hand, it can be seen that, as expected, when the throughput requirements of the flows are higher than the available bandwidth of the bottleneck router, packet drops increase, as shown in Figure 6.15 for a DropTail bottleneck router queue.

Packet drop percentage for the RED bottleneck router queue with 150% bandwidth requirements is almost identical to the DropTail queue case shown in Figure 6.15, albeit with slightly higher variation in packet drop due to the RED queue.

6.5.3 The TCP & UDP Flows Case

In this case we are concerned with throughput for the TCP flows, end-to-end delay for the UDP flows and packet drop percentage for both. As in the previous section, due to the large number of UDP flows used (34 - 77 with 50% bandwidth



(a) 120 bytes UDP payload size, 194 UDP flows.

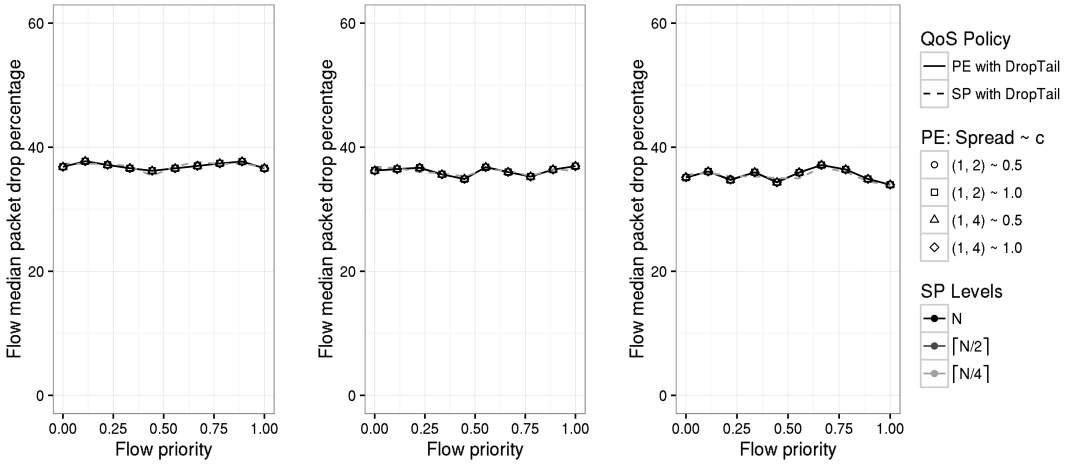
(b) 240 bytes UDP payload size, 120 UDP flows.

(c) 480 bytes UDP payload size, 68 UDP flows.

Figure 6.14: UDP-only flows case results for median packet drop percentage per priority level with a RED bottleneck router queue with 100% bandwidth requirements. Packet drop is between 7.5% and 2.5% decreasing as the number of flows decreases and as the size of the payload increases. It is approximately the same for all priority levels.

requirements) we only present 10 representative priority levels in the figures of this section to preserve legibility. For the case where a DropTail bottleneck router queue is used, the results for TCP throughput are presented in Figure 6.16 and for packet drop percentage in Figure 6.17. PacketEconomy has similar performance to DRR/SP as far as packet drop is concerned. In the case of throughput, PacketEconomy displays a non-linear but smooth distribution while DRR with N levels is largely linear, however DRR with $\lceil N/2 \rceil$ or $\lceil N/4 \rceil$ levels loses this property. We have seen from other experiments (outside the presented subset) that it is possible to select utility functions in such a way that the distribution is linear, however, this impacts the distribution of throughput and end-to-end delay in the TCP & UDP flows case. Also, we considered it useful to use the same utility function creation scheme for all flow composition cases to allow for easier and more objective comparison of performance.

The results for TCP throughput for the RED bottleneck router queue pre-



(a) 120 bytes UDP payload size, 290 UDP flows.

(b) 240 bytes UDP payload size, 180 UDP flows.

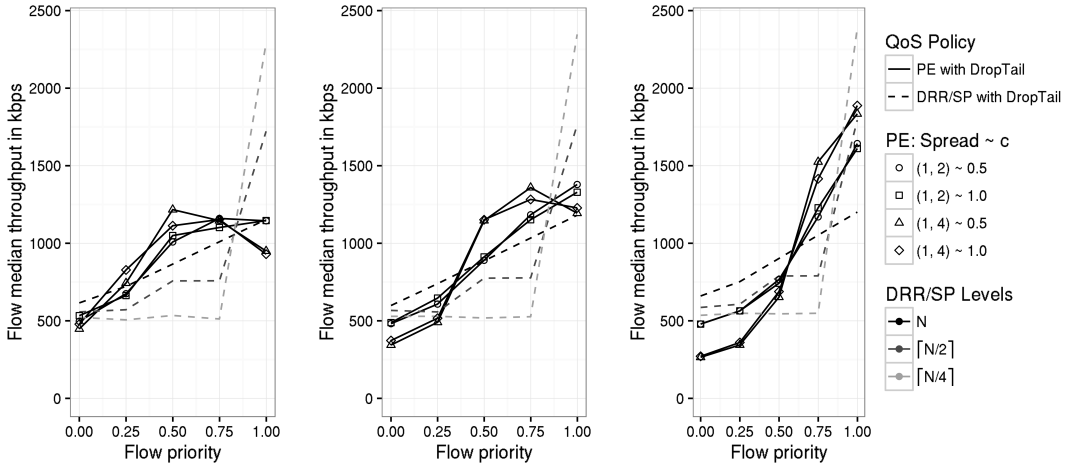
(c) 480 bytes UDP payload size, 102 UDP flows.

Figure 6.15: UDP-only flows case results for median packet drop percentage per priority level with a DropTail bottleneck router queue with 150% bandwidth requirements. Packet drop is approximately 38% for all payload sizes. It is also approximately the same for all priority levels.

sented in Figure 6.18 are more linear than the DropTail queue case shown in Figure 6.16, most probably since TCP flows keep their congestion windows smaller due to packet drops.

The packet drop percentage for the RED bottleneck router queue is similar to the DropTail queue case shown in Figure 6.17, albeit with higher variation in packet drop due to the RED queue. The DRR/SP queue also displays higher packet drop percentage, approximately the same as PacketEconomy.

Both the UDP end-to-end delay and the packet drop percentage for the RED bottleneck router queue are similar to their DropTail counterparts shown in Figures 6.19 and 6.20. However, the packet drop percentage for the RED bottleneck router queue is approximately 2.5% and does not significantly change with UDP payload sizes.



(a) 120 bytes UDP payload size, 97 UDP flows.

(b) 240 bytes UDP payload size, 60 UDP flows.

(c) 480 bytes UDP payload size, 34 UDP flows.

Figure 6.16: TCP & UDP flows case results for median TCP throughput per priority level with a DropTail bottleneck router queue. Throughput increases with priority, as expected, but two spread-*c* combinations distribute throughput less aggressively at high priority values than the other two.

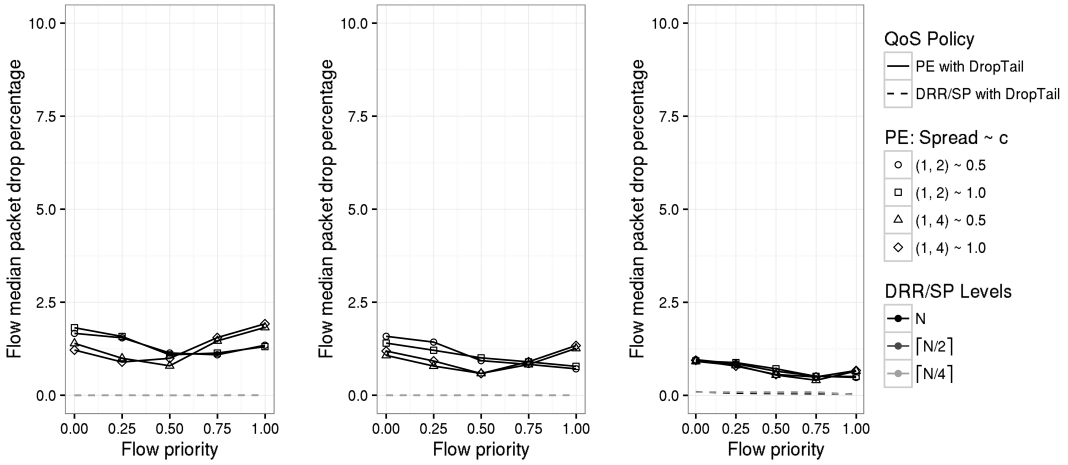
6.6 Game-theoretic Aspects

In this section the game-theoretic aspects of PacketEconomy are discussed.

6.6.1 Incentive to Participate

The first issue is whether flows have an incentive to participate in PacketEconomy, i.e. the property of individual rationality, something which has been also investigated in Chapter 5. According to that previous work, due to trades being Pareto improvements on each trading packet's benefit, we concluded that there is a NE in which all flows participate in the scheme.

The following results illustrate that individual rationality is present in the real network implementation of PacketEconomy as well, with the caveat that in practice only a subset of flows have been tested for this property.



(a) 120 bytes UDP payload size, 97 UDP flows.

(b) 240 bytes UDP payload size, 60 UDP flows.

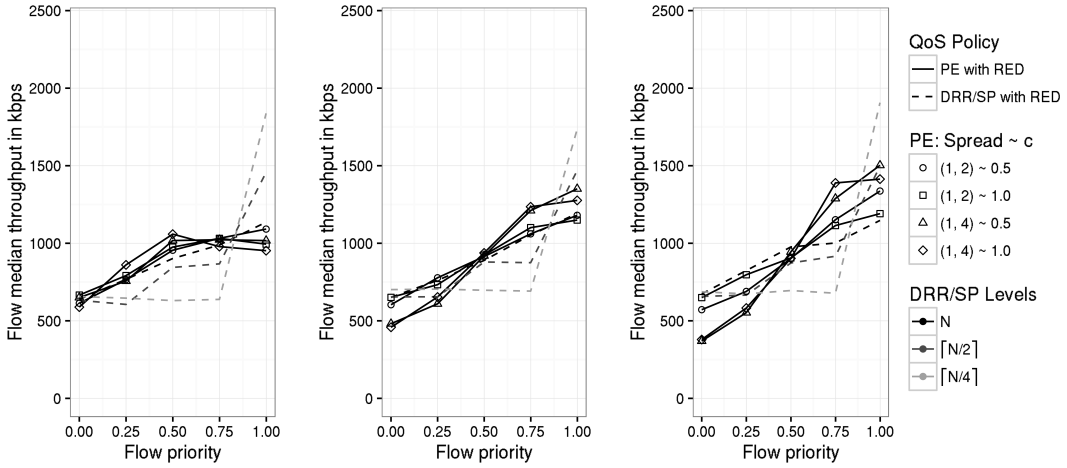
(c) 480 bytes UDP payload size, 34 UDP flows.

Figure 6.17: TCP & UDP flows case results for median TCP packet drop percentage per priority level with a DropTail bottleneck router queue. Packet drop is between 0.5% and 2%, decreasing as the number of UDP flows decreases and as the size of the UDP payload increases. It is approximately the same for all priority levels. DRR/SP packet drop percentage is very low, approximately 0.07%.

Specifically, this property requires that an individual, multiply-replicated experiment needs to be performed for each flow for which its incentive to participate needs to be established. Due to the large number of flows used, and due to the fact that utility functions are not arbitrary but vary smoothly from priority level to priority level, we have performed and we only present 5 representative priority levels in the figures of this section. Therefore, we expect that omitting some intermediate flow priority levels does not significantly impact the overall results.

6.6.1.1 The TCP-only Flows Case

The incentive to participate in the TCP-only flows case is presented in Figure 6.21 and as it can be seen, all examined flows have a strong incentive to participate in both DropTail and RED bottleneck router queue cases.



(a) 120 bytes UDP payload size, 97 UDP flows.

(b) 240 bytes UDP payload size, 60 UDP flows.

(c) 480 bytes UDP payload size, 34 UDP flows.

Figure 6.18: TCP & UDP flows case results for median TCP throughput per priority level with a RED bottleneck router queue. Throughput increases with priority, as with DropTail and the differences between spread-*c* combinations are diminished.

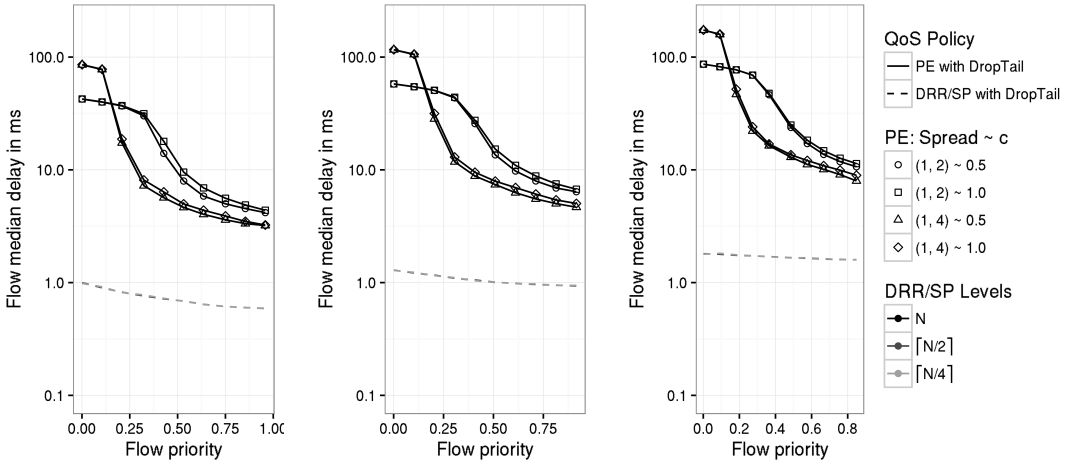
6.6.1.2 The UDP-only Flows Case

The incentive to participate in the UDP-only flows case is presented in Figure 6.22 and as with the TCP-only flows case it can be seen all examined flows have a strong incentive to participate in the DropTail bottleneck router queue case for all the different UDP payload sizes examined. The incentive to participate for the RED bottleneck router queue with 100% bandwidth requirements shown in Figure 6.23 is almost identical to the DropTail queue case shown in Figure 6.22.

The incentive to participate for both DropTail and RED bottleneck router queues with 150% bandwidth requirements is similar to the DropTail queue with 100% bandwidth requirements case shown in Figure 6.22 and is also always over 100%.

6.6.1.3 The TCP & UDP Flows Case

The incentive to participate for TCP and UDP flows in the mixed flow types case is similar to the incentive the flows have in the TCP-only and the UDP-only



(a) 120 bytes UDP payload size, 97 UDP flows.

(b) 240 bytes UDP payload size, 60 UDP flows.

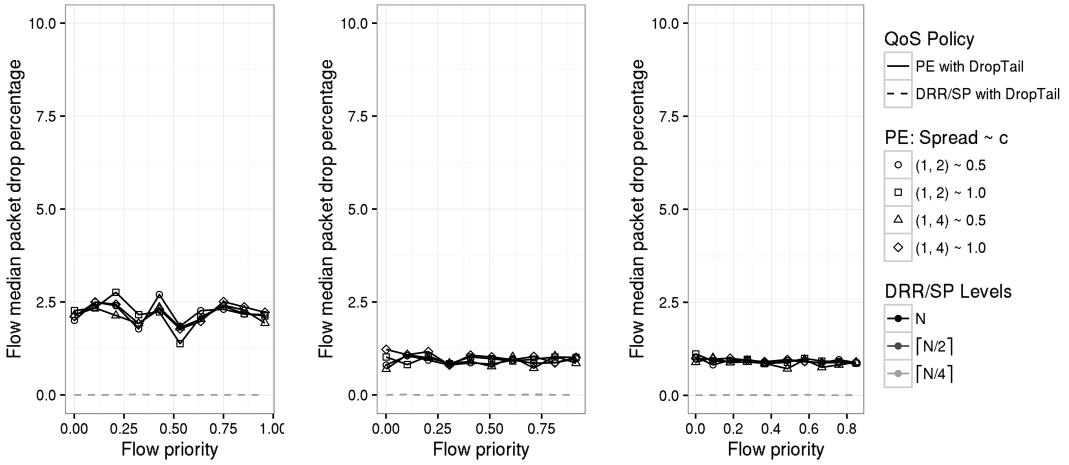
(c) 480 bytes UDP payload size, 34 UDP flows.

Figure 6.19: TCP & UDP flows case results for median UDP end-to-end delay per priority level with a DropTail bottleneck router queue. Delay decreases with priority, as expected, but two spread-c combinations distribute delay more aggressively than the other two. Note: the y axis is logarithmic.

case and is almost always over 100% providing an incentive to participate in PacketEconomy. In some rare cases, such as the one shown in Figure 6.24, the incentive is slightly lost. We have found this to happen occasionally with RED queues and higher priority TCP flows. We have concluded that this happens due to the larger error in estimation of delivery time with larger size high priority packets. We expect that due to the dynamic nature of the network, flows will sometime misestimate their delivery times and as a result affect their ask or bid prices. However, we expect these fluctuations to cancel out on average.

6.6.2 Packet Size Variability

In the previous Chapter 5, all packets were assumed to be of identical size. In this more realistic implementation, packets have different sizes in some cases and this means that a trade may affect in-between packets' queuing time. Initially, in our experiments we performed trades irrespective of the packet size



(a) 120 bytes UDP payload size, 97 UDP flows.

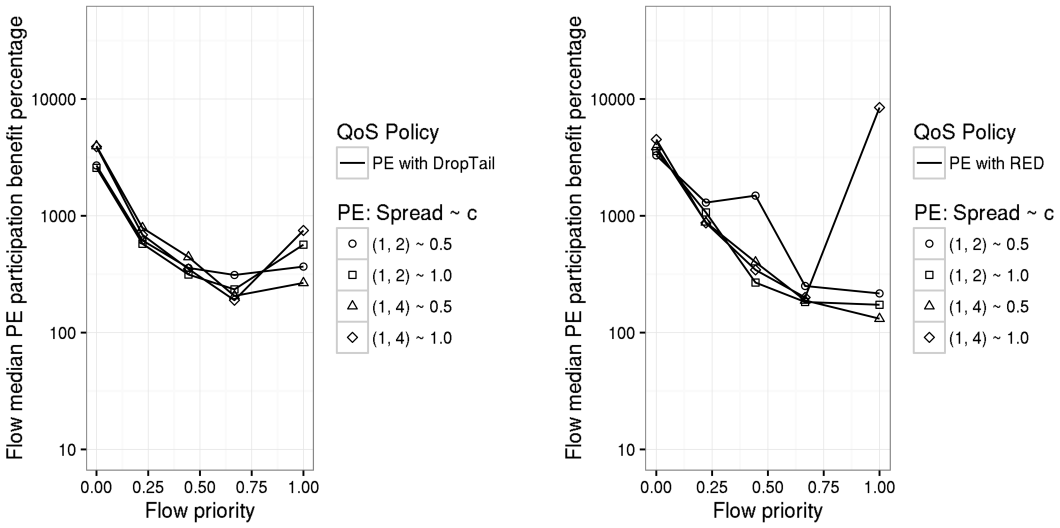
(b) 240 bytes UDP payload size, 60 UDP flows.

(c) 480 bytes UDP payload size, 34 UDP flows.

Figure 6.20: TCP & UDP flows case results for median UDP packet drop percentage per priority level with a DropTail bottleneck router queue. Packet drop is between 0.8% and 2.5%, decreasing as the number of UDP flows decreases and as the size of the UDP payload increases. It is approximately the same for all priority levels. DRR/SP packet drop percentage is very low, approximately 0.03%.

of the two trading packets. This meant that when a larger buyer packet traded positions with a smaller seller one, in-between packets were also affected, since their queuing time would increase. Conversely, with a smaller buyer packet and a larger seller one, the in-between packets would see a decrease in queuing time. The only exception was when the trading packets are adjacent. We were mainly concerned by the problem the first case introduces since in the second case the in-between packets are favoured.

There were a number of alternative approaches to this issue. The ideal one would be to incorporate the price of extra delay of each in-between packet into the ask price and then distribute the funds accordingly to all the in-between packets. However, this would significantly increase computational complexity, completely remove locality of trades and preclude easy parallelisation. Therefore, a compromise was sought in which we disallowed any trades between larger buyers and smaller sellers. Although this approach somewhat decreased



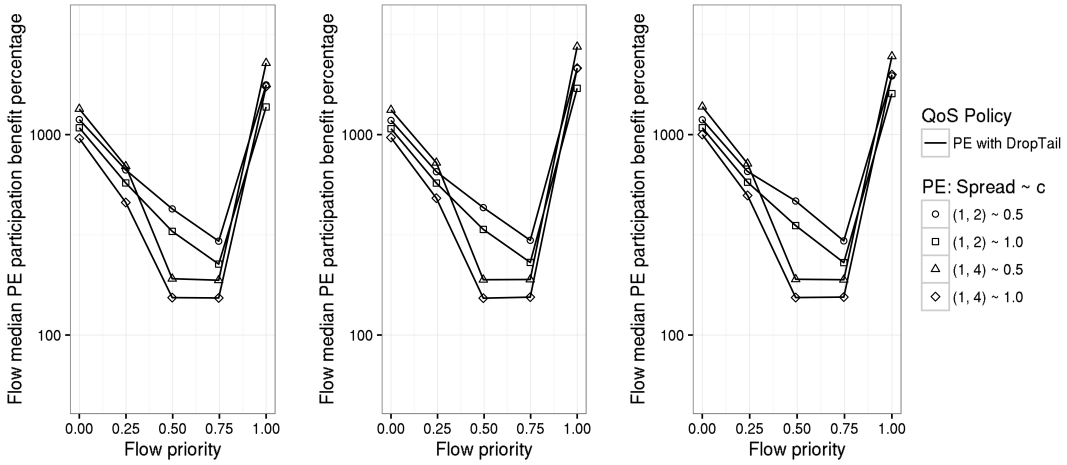
(a) DropTail bottleneck router queue.

(b) RED bottleneck router queue.

Figure 6.21: TCP-only flows case results per priority level for incentive to participate as a percentage of total benefit gained when participating versus not participating. In all cases it is over 100% and as a result there is always an incentive to participate in PacketEconomy. Note: the y axis is logarithmic.

the number of trades performed and as a result diminished the QoS effects, the results were nevertheless acceptable in both network and game-theoretic terms. Additionally, the QoS effects can be recovered by increasing the number of trading rounds, with the corresponding impact on computational complexity (a constant factor equal to the number of trading periods).

Another approach would be to create (a small number of) separate queues for different packet size ranges. This would moderate the effects of trades on in-between packets by putting an upper bound on the extra delay incurred when different size packets trade, although it would still prevent trades from always being Pareto improvements. However, we estimate that this approach would also decrease trades performed by creating more but smaller-sized packet queues, which as before, can be mitigated by increasing the number of trading rounds, with the corresponding impact on computational complexity. Additionally, it would create the problem of deciding in which manner packets from the different queues will be serviced, effectively wrapping PacketEconomy in a



(a) 120 bytes UDP payload size, 194 UDP flows.

(b) 240 bytes UDP payload size, 120 UDP flows.

(c) 480 bytes UDP payload size, 68 UDP flows.

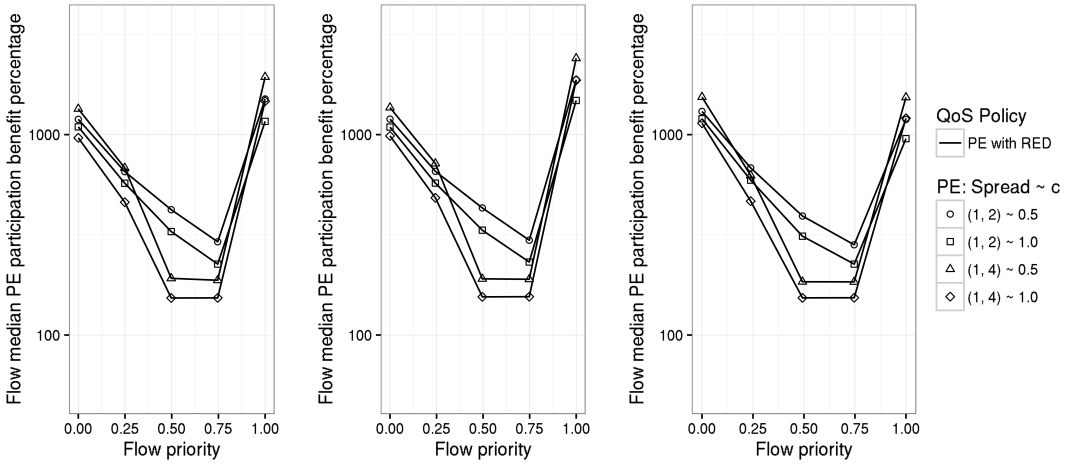
Figure 6.22: UDP-only flows case results per priority level with a DropTail bottleneck router queue with 100% bandwidth requirements. Displayed is the incentive to participate as a percentage of total benefit gained when participating versus not participating. In all cases it is over 100% and as a result there is always an incentive to participate in PacketEconomy. Note: the y axis is logarithmic.

higher-level multi-queue scheduler.

6.6.3 Truthfulness of Packet Utility Function

Another useful property of game-theoretic models is for them to provide incentive for players to report their utility function truthfully, since this defines the ask and bid prices. In mechanism design, a process is incentive-compatible if all of the participants fare best when they truthfully reveal any private information asked for by the mechanism, however there are different degrees of incentive-compatibility:

- Dominant Strategy Incentive Compatibility: truth-telling is a dominant strategy, also known as Strategyproofness.
- Incentive Compatibility (a weaker notion): truth-telling is a Bayes-Nash



(a) 120 bytes UDP payload size, 194 UDP flows.

(b) 240 bytes UDP payload size, 120 UDP flows.

(c) 480 bytes UDP payload size, 68 UDP flows.

Figure 6.23: UDP-only flows case results per priority level with a RED bottleneck router queue with 100% bandwidth requirements. Displayed is the incentive to participate as a percentage of total benefit gained when participating versus not participating. In all cases it is over 100% and as a result there is always an incentive to participate in PacketEconomy. Note: the y axis is logarithmic.

equilibrium, i.e. it is best for each participant to tell the truth, provided that others are also doing so.

In PacketEconomy the players are the flows which define the packet utility functions and the private information aimed to be truthfully revealed is the set of utility function parameters. Preliminary experiments indicate that the flows do not gain by reporting false utility functions, because changing the total utility a packet receives (the benefit) is the sum of two quantities with an inverse relation: increasing the packet value incurs higher budgetary costs and increasing the accumulated budget negatively impact packet value. In other words, the mechanism may be incentive-compatible. However, since there is a spread between ask and bid prices but no agent to minimize this spread as in stock exchanges, it is conceivable that to some limited extent flows can manipulate their utility function parameters in order to gain added benefit.

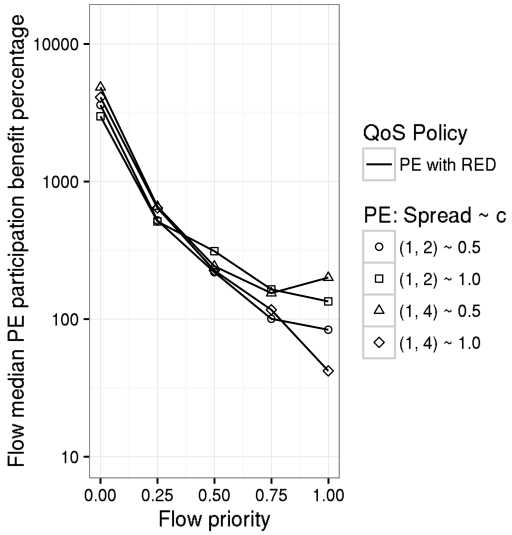


Figure 6.24: TCP and UDP flows case results for TCP flows per priority level with a RED bottleneck router queue with 240 bytes UDP payload size and 60 UDP flows. Displayed is the incentive to participate as a percentage of total benefit gained when participating versus not participating. In most cases it is over 100%, but for some high priority flows it falls below 100%. Note: the y axis is logarithmic.

Moreover, this manipulation is harder to perform for intermediate priority flows and easier for higher and lower priority ones since the ask and bid prices are determined identically. Thus intermediate flows will tend to cancel out gains from sells (or buys) with losses from buys (or sells). In any respect, further investigation is required for this issue to be fully resolved.

6.6.4 Price of Anarchy / Stability

A notion which is also interesting to investigate in this context is the Price of Anarchy (PoA) [57, 85] and its related Price of Stability (PoS) [91, 6]. These values quantify the relation between the efficiency of the outcome produced by a system in which the players behave individually, selfishly and in a decentralized manner, such as in the case of PacketEconomy, and the efficiency of the outcome produced by a centralized decision maker. Both require a means of quantifying the measure of efficiency an outcome, called a welfare function. In our case,

a natural candidate would be the sum of the benefits of the flows, called the utilitarian function. Using the welfare function, the Price of Anarchy is the ratio of the value of the welfare function for the optimization problem solution (centralized decision maker) over the worst value of the welfare function for the selfish and decentralized solutions. Correspondingly, the Price of Stability is the ratio of the value of the welfare function for the optimization problem solution (centralized decision maker) over the best value of the welfare function for the selfish and decentralized Nash Equilibrium solution.

The PoA and PoS have been investigated extensively in theoretical models, but a real network presents significant problems to overcome in calculating exact values. More specifically, although calculating the welfare function for any of the experimental case results is easily done, comparing this value to an optimal solution is harder, since deciding what this optimal solution would be is non-trivial. For example, fixed size router queues, probabilistic admission policies (e.g. RED), adaptive flows which are affected by feedback (e.g. TCP) as well as the interaction between flow types and packet sizes (e.g. UDP packets tend to interfere with TCP flow control) all make a theoretical analysis much harder. Judging from the network-centric results, we can see that both throughput and delay are being distributed in accordance to priority and there does not seem to be any significant loss of overall efficiency (e.g. the sum of flow throughputs with PacketEconomy is equal to the sum of flow throughputs using DRR, just differently distributed). We expect that it would be possible in a future work to examine a case with a centralized scheduler which, taking into account each flow's utility functions, decides which packets to deliver and in what manner. However, this would just produce an upper bound on the welfare value, since is not necessary that the solution provided is implementable in networking terms or that it produces the calculated welfare, since the above mentioned networking concerns are not taken into account.

6.6.5 Relation to Smart Market

In the seminal work of MacKie-Mason and Jeffrey [64], they propose a generalized Vickrey auction (GVA) in order to provide QoS for packets in queues. The main disadvantage of that approach is the computational complexity it induces, since a full auction needs to be performed for each packet served. While not equivalent, our approach can be seen as an approximation of the smart market mechanism, where increasing the number of trading periods improves the approximation.

6.7 Conclusions and Future Work

In this work we presented a realistic implementation of PacketEconomy, a distributed quality of service (QoS) mechanism for network packets, within the OMNET++ discrete event simulator and using the INET network simulation library. With this work we aim to provide high performance, network-wide, fine-grained, user-controlled QoS. We have presented the complexities that needed to be overcome and the required adaptations made to the theoretical PacketEconomy model for a realistic environment. We then performed extensive experimental evaluation of the implementation and presented characteristic results in comparison to the deficit round robin and strict priority QoS policies.

Possible extensions of this work comprise a larger number of scenarios to be examined, with more complex network topologies and flow compositions, as well as adaptivity being used in endpoints to auto-configure utility function parameters. We also envision PacketEconomy's applicability in alternate contexts, such as being used in DTNs (Delay-Tolerant Networks) as a QoS policy. In particular, when fast transmission is possible, the faster and more efficient PacketEconomy can be used, while when computational complexity is not at a significant premium and slower transmission is only possible, an auction-based QoS policy (such as [64]) can be used instead. An additional alternate context concerns IoT (Internet of Things) networks. Since PacketEconomy uses a notion of utility, which encodes a time-varying quantity, it may be useful for IoT networks wherein communication comes at a premium in both terms of energy and computational complexity. Being able to more accurately express the value of a packet as a function of time may allow the network to make better service decisions.

Overall, we consider our approach to be both theoretically well-founded, as well as practically applicable, a claim which is also supported by the experimental results.

CHAPTER 7

A Game-theoretic Analysis of Preventing Spam over Internet Telephony via Audio CAPTCHA-based Authentication

7.1 Introduction

The explosive growth of the Internet has introduced a wide array of new technological advances and more sophisticated end-user services. One of them is VoIP, which is a developing technology that promises a low-cost, high-quality and availability service of multimedia data transmission. Inevitably though, VoIP "inherited" not only these positive features of Internet services, but also some of their problems [22][52][53][110]. One of them is Spam over Internet Telephony (SPIT) [90][26], which is the expression of Spam in VoIP network environments. SPIT is a challenging issue that IP telephony is expected to be facing in the near future. This is the reason why a) major organizations have already started developing mechanisms to tackle SPIT [37][89], and b) the U.S. Federal Communications Commission has extended the Telephone Consumer Protection Act of 1991 to include automated calls, called robocalls [27]. Moreover, it should be stated that the U.S. Federal Trade Commission has created the "Do Not Call Registry" in order to allow users to reduce the number of telemarketing sales calls received (automated or not) [105]. The active registrations in the "Do Not Call Registry" were over 217 million on October 30th, 2012 [28].

The SPIT threat for VoIP is the analogue of spam for e-mail. However, due

to its characteristics, it may also give the opportunity to malicious users to not only send low- or zero-cost unsolicited instant messages but also to make low- or zero-cost unsolicited calls by using automated software (bots). The malicious user's main purpose could be financial, like presenting advertisements, or to extract/steal a legitimate user's personal information (phishing). A real-life example is the "Rachel" robocall enforcement case, where five companies were shut down, because they made millions of illegal pre-recorded robocalls claiming to be from "Rachel" and "Cardholder Services" while pitching credit card interest rate reduction services [29]. Although the similarity of the SPIT phenomenon to the well-established spam threat is easy to identify, this does not lead to the conclusion that the techniques handling spam are appropriate for handling SPIT as well. While applying the anti-spam techniques can be done quite easily in terms of service configuration, some characteristics of SPIT make the direct application of anti-spam techniques inefficient and ineffective. In particular, telephony and instant messaging services operate in real time while email services are based on a "store and forward" model [46][96]. Therefore, the anti-spam techniques can examine the content/body of the email in order to classify it as spam or not, but this is not possible for VoIP real-time communication services [76].

A serious obstacle when trying to prevent SPIT is identifying VoIP communications which originate from software robots ("bots") in real-time. A typical way to tackle these attacks is the use of a Reverse Turing Test, called CAPTCHA (Completely Automated Public Turing Test to Tell Computer and Humans Apart). Since visual CAPTCHA are hard to apply in VoIP systems, audio CAPTCHA appear to be appropriate for defending against SPIT calls/messages [39][100][99].

VoIP is a useful technology with significant value for legitimate users, as it enables communication and decreases costs. On the other hand, VoIP spammers can obtain significant financial revenues as the email spam paradigm has shown. Therefore, we have a situation where independent decision makers are engaged in a strategic interaction; the actions taken by SPIT senders may influence the defensive actions taken by the VoIP users and the opposite. The outcome of such scenarios is not only a matter of effective tools like audio CAPTCHA challenges, but also of how independent selfish decision makers will act and react in the presence of such tools. Such settings, where two or more independent decision makers interact, can be studied with concepts and tools from Game Theory. The equilibrium points of the respective game-theoretic model can reveal important attributes of the state(s), in which the system is expected to operate. For example, it will reveal how often the audio CAPTCHA will be used or whether the overall rate of SPIT calls decreases in the presence of audio CAPTCHA. In the presence of selfish users, there are examples where the introduction - always with good intentions - of a tool or an extra option for the users may lead to worse

overall system performance. This can happen even with the simple addition of a new tool to an existing system. For example, in [45] scenarios are identified where increasing the number of (selfish) security experts of an information network may lead to reduced overall security of the network; the Braess paradox [11] shows how adding an extra route to a traffic network may lead to worse conditions for selfish drivers.

In this work, we assume the existence of effective audio CAPTCHA challenges and discuss how the strategic interaction between SPIT senders and VoIP users can be modelled as a two-player game in the presence of such CAPTCHAs. In particular, we propose a game-theoretic model and show how the resulting model can be used to predict the behaviour that the two opponent communities will eventually adopt, how it can guide to fewer SPIT messages and how the use of CAPTCHA assists VoIP users against SPIT. As part of the legitimate user defences against SPIT we also integrated an anti-SPIT filter, which classifies each incoming call/message as legitimate, malicious or "unknown" (when it is not possible to have a confirmed answer). After the filter's incoming call classification, the user may directly accept or reject the call or request a CAPTCHA, depending on the precision and the verdict of the filter.

7.2 Related Work

As the SPIT phenomenon is practically still in its infancy, we were not able to find relevant research work focusing on the cost of spam for both the SPIT sender and the user, or on relevant game-theoretical models. Therefore, we present research work based on a close relative of SPIT, i.e., e-mail spam.

7.2.1 Cost of Unsolicited Communication

Kim Y., et al. [54] propose a method to measure the disutility experienced by e-mail users who receive spam. Their study employs conjoint analysis of stated preference data to estimate e-mail users' overall inconvenience cost attributable to spam. The results show the inconvenience-originating cost of spam to be about $0.0026 per spam message.

Kanich C., et al. [47] present a methodology for measuring the conversion rate of spam. They produced nearly half a billion spam e-mails and they identified the number that were successfully delivered, the number that passed through popular anti-spam filters, the number that elicited user visits to the advertised sites, and the number of "sales" and "infections" produced. They managed to calculate that the total revenue of a spam campaign is about $7000 and the cost to produce it is the pay-check of three "good" programmers. There-

fore the cost per message is about $0.001. Finally, a report placed the retail price of spam delivery at slightly under $80 per million [113]. This price means that each spam email costs $0.00008, but we stick to the previous paper's cost estimates, as this kind of price is an order of magnitude less than what legitimate commercial mailers charge.

7.2.2 Game-theoretic Models

Androutsopoulos I., et al. [5] present an interesting game-theoretic model for the interaction of spam and ordinary e-mail users and later extend their model in [109] to the case where the users are able to use Human Interaction Proofs (HIP). In the latter work, they focused on the scenario where the users can read messages, delete them without reading them or send HIP. They have provided an extensive theoretical analysis of a game-theoretic model for the problem of spam. As discussed earlier, there are important qualitative differences between SPIT and spam. We generalize the model proposed in [109] to a more complicated problem with more actions to account for additional situations that arise in VoIP, and apply it within a related, but substantially different, application context, namely VoIP. We also experimentally confirm the predictions of the model.

Parameswaran M. [86] suggests that the spammer can strategise to maximize the amount of spam sent by making inferences from the block-list rules. They introduce a theoretical modelling approach for the spammer's behaviour and present a comparison of this behaviour with the data that has been collected from block-list organizations. The main issue with this work is that is based on collected data, therefore its outcomes cannot be generalized. Shahroudi A.B. et al. [92] examine how VoIP service providers attempt to control the growing phenomenon of SPIT by creating a game-theoretic model of competition between providers. The model is based on the notion that two different service providers, which try to maximize their profit with different business strategies, are competing on shared resources. Each service provider can select to either detect or prevent SPIT in order to address attacks, with consequences to the overall profit of both providers. The research outcome is that the providers are going to focus on mechanisms which detect SPIT attacks, because even though they are more expensive than preventative mechanisms, it maximizes their profit.

Moreover, a discussion of game theory approaches for detection software can be found in [13]. The proposed model is able to assist firms in the configuration process of detection software and a significant outcome is that false-positive and false-negative errors in detection could affect the value of these systems significantly.

In general, even though there is work on applying game-theoretic tools to

problems of security, to the best our knowledge this is the first attempt of a game-theoretic analysis of SPIT and how to counter it with audio CAPTCHA.

7.3 Suggested Game-theoretic Model

Generalizing and building upon Androutsopoulos et al. [5][109], we define the SpitGame, a game-theoretic model with two players: the SPIT sender (Player I) and the legitimate VoIP user (Player II). The game is illustrated in Fig. 7.1. We will describe the game in detail and at the same time give short definitions of the game-theoretic terms and concepts that we encounter. For more details on the game-theoretic terms, the reader may refer to textbooks on Game Theory [80][79][81], or to a recent volume on Algorithmic Game Theory [77].

The SpitGame, as shown in Fig. 7.1, is an extensive form game with imperfect information. The game is initiated whenever a new call/message is sent towards a user. The SPIT sender (Player I) moves first and is able to interfere with the stream of incoming calls and send a new SPIT call at any point. Thus, the frequency with which SPIT senders initiate a malicious call determines the average ratio of SPIT to legitimate calls in the users' incoming streams. For example, if a SPIT sender initiates a SPIT call every four (4) legitimate calls, then the overall probability/rate of SPIT calls will be $p = 0.2$, which is presented as probability p in Fig. 7.1. Although in reality SPIT senders are not able to completely control all the incoming calls/messages, or to decide whether or not they will insert a new SPIT message/call, the assumption that the SPIT senders control the ratio between SPIT and legitimate calls is reasonable. A similar assumption has been used in the game-theoretic models for SPAM in [5][109] upon which we generalised.

The SPIT sender chooses to make the incoming call SPIT or to allow it to be a legitimate call. The VoIP user does not learn which choice the SPIT sender has made. That is, the VoIP user is imperfectly informed about the game status and for this reason we model this interaction as an extensive game with imperfect information. However, the VoIP user gets some stochastic information about the game status from the outcome of an anti-SPIT filter. After the move of the SPIT sender, the call is processed by anti-SPIT filters, which are able to flag the calls they consider SPIT. The use of filters is a common countermeasure (in some cases of Internet service providers, this is mandatorily applied to their users). We have assumed that the filter contains a deterministic first stage and a stochastic second stage. In the first stage, an accurate black/white-list, created from past calls, can accept or discard the call. The second stage is invoked if the black/white-list does not identify the caller. In this stage, the filter attempts to guess the nature of the call from the characteristics of the call (e.g. the time/date, the

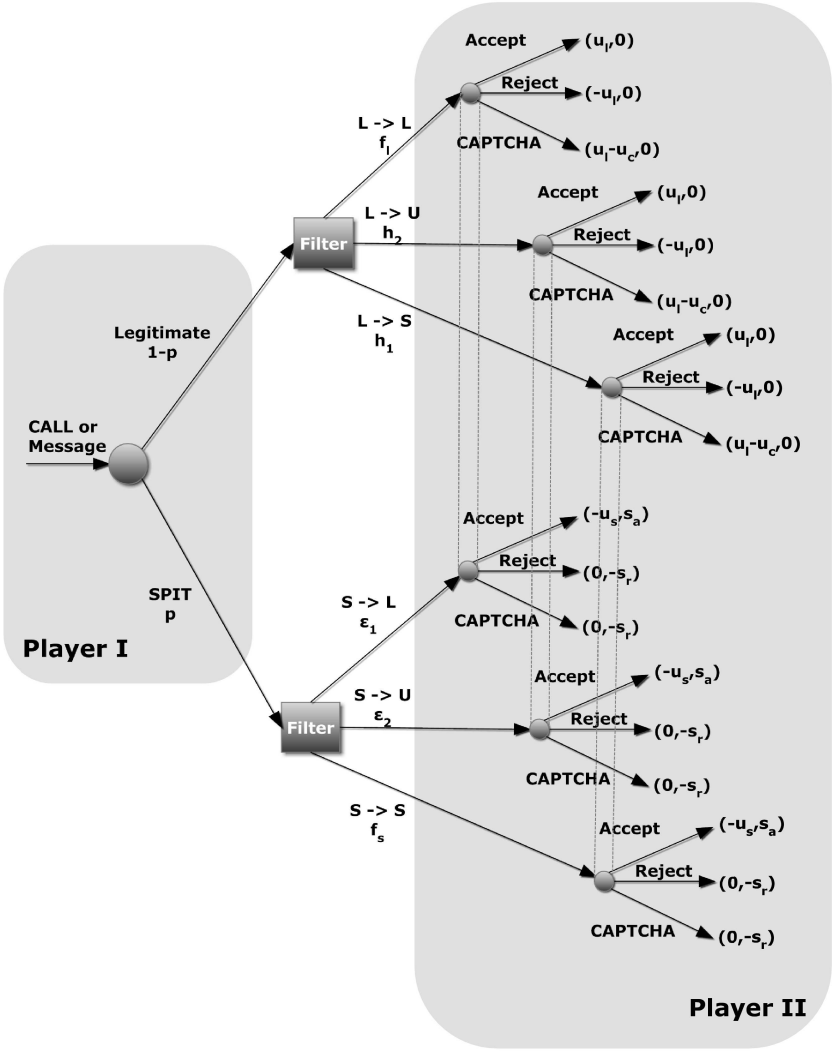


Figure 7.1: The game-theoretic model

caller domain, the user agent, etc.). In the model we describe, the filter refers only to the second stage, since the first stage does not have a game theoretic aspect.

In our model, the performance of these filters is fully described by six variables: f_l, h_2, h_1, ϵ_1, ϵ_2 and f_s. More specifically, in the case of legitimate calls, the filter will classify the calls accurately with a probability of f_l, it will consider them unknown with a probability of h_2 and it will misclassify them as SPIT calls with a probability of h_1. In the case of SPIT calls the corresponding legitimate, unknown and SPIT classification probabilities are ϵ_1, ϵ_2 and f_s. For example, consider the case when the filter misclassifies the incoming message. In Fig. 7.1 the

probability of misclassifying a SPIT call as legitimate is depicted as $S \rightarrow L$ and the probability of misclassifying a legitimate call as SPIT is depicted as $L \rightarrow S$. Moreover, the filters may not be able to come to a definite conclusion over the nature of the call. In this situation, the filter classifies the call as "*Unknown*", which is common in VoIP communication systems. Although this may be uncommon in email spam filters, since the messages can be classified based on content and header, VoIP is a real time protocol that does not grant the receiver access to the call contents prior to its acceptance/session establishment. Therefore, whenever a call arrives from an unknown number, the call may be classified as SPIT or legitimate. Since VoIP communication is synchronous, unlike email spam where email is delivered asynchronously and the marked-as-spam messages can be stored, if the call is rejected then there is no way for the user to retrieve its content/purpose. Since much less information is available than in email spam, the anti-SPIT filter should include the "*Unknown*" verdict, which is dominant when a SPIT call is received, since most SPIT calls are initiated from numbers unknown to the user.

In the context of SpitGame, after the move of the SPIT sender the filter classifies the incoming call. The action of the filter is modelled with an artificial third player; such a player is usually called chance in the game. Player chance has three moves, one for each of the possible outcomes of the filter.

The user is informed about the "move" of the filter but not the move of the SPIT sender. The user should decide his move based on the filter's prior classification. He is able to accept the call, reject it or request an audio CAPTCHA. The user is not aware of the true nature of the calls before he listen to them, so when he sees that his filter has classified a call as legitimate, he does not know whether it was misclassified or not. For example, when a user receives a legitimate filter-classified call it is impossible to distinguish in which node ($L \rightarrow L$ or $S \rightarrow L$) of the game he is. In game-theoretic terms, each of the possible outcomes of the filter defines an information set for the VoIP user. Each such set contains two nodes of the extensive-form game, because there are two nodes in the game which may lead to the particular filter decision. The VoIP user, however, is informed only about the information set and not about the particular node of the set in which the game really is.

Therefore, each user has to select a strategy consisting of what he will do with incoming calls depending only on information sets, i.e., the decisions of his filter; for example, *Accept* calls classified as *Legitimate*, *Reject* calls classified as *SPIT*, and request audio *CAPTCHA* when calls are classified as *Unknown*. Similarly, we may assume that the overall community of users adopts a strategy, whose probabilities reflect the frequencies with which it adopts actions *Accept*, *Reject*, and *CAPTCHA*. That means that the sum of the probabilities of these three actions is equal to 1 for each game node. For example, when a user

Table 7.1: Game-theoretic model utilities

	User/Player II			SPIT sender/Player I		
Message	***Accept***	***Reject***	***CAPTCHA***	***Accept***	***Reject***	***CAPTCHA***
Legitimate	u_l	$-u_l$	$u_l - u_c$	0	0	0
SPIT	$-u_s$	0	0	s_a	$-s_r$	$-s_r$

receives a new call, which is classified as *Legitimate*, then $P(Accept) + P(Reject) + P(CAPTCHA) = 1$, regardless of whether the message was misclassified or not. Likewise, this happens in the other two cases: *SPIT* and *Unknown*.

Whenever a new session is initiated, the actions which the SPIT sender and legitimate user select lead to a particular cost or utility for each player. For example, if the SPIT sender selects to initiate a *SPIT* call and the user selects to *Accept* the call, then the game ends with a utility of $s_a > 0$ for the SPIT sender and a cost of $-u_s < 0$ for user. In summary, every combination of actions of the two players leads to an outcome of the game, and this outcome determines the amount of utility for each participant, which is shown in Fig. 7.1 and Table 7.1. Notice that the utilities for the user and SPIT sender do not depend directly on the filter classification, however, the classification does affect the ratio between legitimate and SPIT calls which the user receives.

The utilities for each player are determined by five parameters:

1. u_l: This is the measure of average utility of accepting a legitimate call.

2. u_s: This is the measure of average disutility of receiving a SPIT call, taking into consideration factors such as the average cost of consumed computational resources, the time needed to answer the phone, and the average time it takes to listen to it, which means a general decrease to user productivity.

3. u_c: This is the measure of average disutility of sending a CAPTCHA puzzle, taking into consideration the annoyance of a legitimate caller, of whom it is required to solve a CAPTCHA challenge in order to reach the user. This annoyance can directly lead to profit loss if the caller is a potential customer, but also indirectly lead to social issues if the user's acquaintances are reluctant or hesitant to call him.

4. s_a: This is the measure of average utility the SPIT sender obtains from each SPIT call that is accepted, taking into consideration factors such as the percentage of users that order products after listening to the SPIT call, and the advertisement campaigns he may be paid to be part of.

5. s_r: This is the measure of average disutility to the SPIT sender of getting a SPIT rejected, taking into consideration all related costs, including the computational resources to create SPIT, and the effort to create an appropriate bot to execute SPIT attacks.

The parameters express a measure (or absolute value) of utility or disutility; as such $u_l, u_s, u_c, s_a, s_r > 0$ and when appearing in pay-offs their sign denotes whether they express utility (+) or disutility (-).

We assumed that the utility from accepting a legitimate call is exactly the opposite of the cost of rejecting it. This is justified by equating the (dis)utility of the user to the information value of the call being (rejected) accepted. Moreover, the utility of accepting the call may be the information value of the call minus the cost of the consumed computational resources for session establishment, while the cost of rejecting it may be simply the information value. This cost difference is so marginal that it was not taken into consideration.

In order to facilitate the examination and analysis of the model, we have set a few restrictions on the costs:

1. The user's disutility for sending a CAPTCHA ($-u_c$) is smaller than the user's disutility for missing a legitimate message ($-u_l$). In absolute terms, $u_l > u_c$. This means that when a user initiates a call, the process to answer a CAPTCHA for establishing the call is not cost-forbidden.

2. The user's disutility for sending a CAPTCHA ($-u_c$) is smaller than the user's disutility of accepting a SPIT call ($-u_s$). In absolute terms, $u_s > u_c$. Otherwise, the use of CAPTCHA would have no sense, since it would be better for the user to receive SPIT than request a CAPTCHA.

3. The user's disutility of accepting a SPIT call ($-u_s$) is smaller than the user's disutility for missing a legitimate message ($-u_l$). In absolute terms, $u_l > u_s$. This condition is based on the premise that receiving a SPIT call may be annoying and distracting for the callee, but missing a legitimate call is more important since it may mean loss of business opportunities, damage to a business' image and reputation or disruption of the user's social life.

4. The utility for a SPIT sender to have a SPIT call accepted (s_a) is larger than the cost of having the call rejected ($-s_r$). In absolute terms, $s_a > s_r$. Given that in practice the chance of the SPIT sender making a profit from an accepted call is very low and that the cost of making SPIT calls, due to the way VoIP works, is also very low, it can reasonably be assumed that the utility of having a call accepted needs to be high, at least higher that the disutility of making the call, in order for the SPIT sender to have an

Table 7.2: Player preferences parameters

Player	Parameter	Description	Conditions (absolute values)
User/Player II	u_l	Measure of user utility of accepting legitimate call	$u_l > u_s > u_c > 0$
	u_s	Measure of user disutility of accepting SPIT call	
	u_c	Measure of user disutility of sending CAPTCHA	
SPIT sender/Player I	s_a	Measure of SPIT sender utility of getting a SPIT call accepted	$s_a > s_r > 0$
	s_r	Measure of SPIT sender disutility of getting a SPIT call rejected	

incentive to make calls. In general, SPIT calls could be profitable even if $s_a < s_r$, if the chance of making a profit from an accepted call could be assumed to be high enough.

The above mentioned utilities for each player actions and the relevant conditions are described in Table 7.2.

7.4 Game-theoretic Analysis and Nash Equilibrium

In this section, we present a theoretical analysis of the SpitGame. The fundamental solution concept for games is the Nash equilibrium (NE), i.e., a state of the game from which no individual player has an incentive to unilaterally deviate. The Nash equilibrium is the most popular solution concept in game theory and has been used in the analysis of a vast number of scenarios with interacting decision makers coming (the scenarios) from diverse application domains including economics, biology, political science, computer science and other ([72][77][80][79]). There are numerous applications of game theory, the

Table 7.3: The filter verdicts.

	Filter verdict		
Type of call	*Legitimate*	*Unknown*	*SPIT*
SPIT call	ϵ_1	ϵ_2	f_s
Legitimate call	f_l	h_2	h_1

Nash equilibrium concept and its refinements in Computer Security. See for example the recent surveys [67][104] and the references therein.

Overall, the formulation of the Nash equilibrium has had a fundamental and pervasive impact in economics and the social sciences [72] and more recently in Computer Science [77][85]. Of course, from the development of the Nash equilibrium concept, there have also been some critiques of it. Some of the main critiques are that the Nash equilibrium concept makes misleading or ambiguous predictions in certain circumstances, that it may not capture correctly non-credible threats, that in many games there are many NE, and, more recently, that the computation of NE is intractable in the general case [17].

However, despite these critiques, the NE and its refinements are undoubtedly the most successful solution concept in game theory, widely used in theoretical and practical applications of game theory. Moreover, most critiques do not seem to apply to the NE of the SpitGame. Firstly, the SpitGame exhibits a unique NE (except for some boundary cases) as is shown in Theorem 2. Consequently, there is no ambiguity in the prediction of the state of the game. Moreover, the NE of the SpitGame is computable in polynomial time via a closed form equation (see Table 7.8) and thus, neither the critique concerning the intractability of general NE applies in this case. As discussed later in this section, the NE of the SpitGame is also Subgame Perfect, which removes the non-credible threat issue of some NE. Finally, the NE solution of the SpitGame does not seem to belong to the cases where the NE leads to counter-intuitive solutions, like for example in the case of the Traveller's Dilemma [9].

There are adaptations and refinements of the NE concept for different game settings and purposes. A variation of the NE for extensive-form games is the Subgame Perfect Equilibrium (SPE), which is more appropriate for games with perfect information. In the SpitGame, when Player II has to decide his action without seeing the action of Player I, that is, Player II is imperfectly informed about the game status. However, Player II has access to the outcome of the filter, which provides stochastic information about the action of Player I. The filter verdicts are shown in Table 7.3. Each of the filter verdicts defines an information set for Player II, who has to decide his action based on the information set. A

natural approach for analysing such a model is to use the concept of behavioural strategies ([72][13], and in particular [109][5]), in which players can randomize independently at each information set. In particular, Player II of the SpitGame will have an independent mixed strategy for each of his information sets. A well known fact in game theory, Kuhn's Theorem, states that in extensive-form games with perfect recall, behavioural and mixed strategies are equivalent. The solution concept that we will use to solve the SpitGame is the Nash equilibrium of the corresponding extensive form game, and we will base our analysis on the behavioural strategies of the players.

The interaction between legitimate VoIP users and SPIT senders is a continuous challenge for both parties. Each player, call receiver or SPIT sender, will have to make his choices repeatedly. Moreover, a legitimate caller might be required to solve audio CAPTCHAs when he calls a VoIP user for the first time. Such overheads may devalue the VoIP service in the eyes of legitimate callers. One may argue that a repeated game could be used to model this interaction. Even though one cannot (and should not) exclude such or other possible formulations of the SpitGame problem, we believe that the current formulation as a one-shot game is well suited for the problem. Each time there is an interaction between two entities, the interaction will be unique, or at least we are only interested in the unique interactions. The subsequent interactions between the same entities can be trivially solved by the outcome of the first game. Then, the legitimate player would know if the call is SPIT or not. The cost incurred to the legitimate callers for solving audio CAPTCHAS is assumed to be captured by the disutility u_c. Note that legitimate callers are not directly modelled in the current SpitGame model. Alternatively, one may consider other game-theoretic formulations of the same problem, for example as a repeated game and/or a game with strategic legitimate callers being part of the model. We leave such possibilities for future work.

We will start the analysis of the SpitGame with the following straightforward observation that Player I will never use a pure strategy at any Nash equilibrium.

Theorem 1 *The SpitGame has no Nash equilibrium where Player I plays a pure strategy.*

Proof 14 *We will use a proof by contradiction. Assume that Player I chooses a pure strategy, for example SPIT. Then the optimal response for Player II would be the pure strategy Reject. Then however, Player I would be motivated to change his strategy, i.e., there is no NE if Player I plays SPIT. If, on the other hand, Player I chooses the pure strategy Legitimate then Player II can respond with Accept, which makes the move of Player I suboptimal, i.e., again no NE.*

Table 7.4: The strategy of Player I at a NE

Action of Player I	Probability
SPIT call	p
Legitimate call	1-p

Table 7.5: The strategy of Player II at a NE

		Action of Player II		
	Information Set (Filter verdict)	*Accept*	*CAPTCHA*	*Reject*
1	*Legitimate* call	p_1	q_1	$r_1 = 1 - p_1 - q_1$
2	*Unknown*	p_2	q_2	$r_2 = 1 - p_2 - q_2$
3	*SPIT* call	p_3	q_3	$r_3 = 1 - p_3 - q_3$

Assume a NE of the SpitGame. Let $(p, 1-p)$ be the strategy of Player I at the NE (Table 7.4) and let (p_i, q_i, r_i) be the strategy of Player II at information set i, for $i = 1, 2, 3$. Thus, at the NE, the strategy of Player I is to submit SPIT calls with rate p, i.e., the probability that a new incoming call will be SPIT is p. From the proof of Theorem 1 we know that at any NE

$$0 < p < 1 \, . \tag{7.1}$$

Player II has three information sets, one for each of the outcomes of the filter, presented in Table 7.3.

Since Player II does not know which action Player I has made and the outcome of the filter is stochastic, Player II can base his decision only on conditional probabilities. Assume that a new call has arrived and that the corresponding filter verdict is *SPIT*. Player II is informed that the information set is *SPIT* and has to choose a strategy based on this information only. Let P_{LS} be the conditional probability that the incoming call is *Legitimate* given that the filter has classified it as *SPIT*. Using standard probability theory gives

$$P_{LS} = Prob[L/S] = \frac{(1-p)\, h_1}{(1-p)\, h_1 \; + \; p\, f_s} \, . \tag{7.2}$$

Similarly, we define and calculate the conditional probabilities for all possible cases.

$$
\begin{aligned}
P_{LS} &= \frac{(1-p)\,h_1}{(1-p)\,h_1 \;+\; p\,f_s}, & P_{SS} &= \frac{p\,f_s}{(1-p)\,h_1 \;+\; p\,f_s},\\
P_{LU} &= \frac{(1-p)\,h_2}{(1-p)\,h_2 \;+\; p\,\epsilon_2}, & P_{SU} &= \frac{p\,\epsilon_2}{(1-p)\,h_2 \;+\; p\,\epsilon_2},\\
P_{LL} &= \frac{(1-p)\,f_l}{(1-p)\,f_l \;+\; p\,\epsilon_1}, & P_{SL} &= \frac{p\,\epsilon_1}{(1-p)\,f_l \;+\; p\,\epsilon_1}.
\end{aligned}
\tag{7.3}
$$

Using the above conditional probabilities of Equation 7.3 and the SpitGame model as it is depicted in Fig. 7.1, the average utility of Player I for each of his pure strategies can be calculated. Firstly, note that the average utility for the pure strategy of Player I *Legitimate*, i.e., Player I does nothing, is

$$U_{1L} = 0\,. \tag{7.4}$$

When Player I submits a SPIT call, then his average utility can be calculated as follows. Given the strategy p of Player I, let $V_L(p)$, $V_U(p)$, and $V_S(p)$ be the probabilities that the filter verdict is *Legitimate*, *Unknown*, and *SPIT* respectively. Also, given the strategy of Player II, let $U_{1SL}(p_1, q_1)$, $U_{1SU}(p_2, q_2)$, and $U_{1SS}(p_3, q_3)$ be the average utility of action *SPIT* of Player I in information set *Legitimate*, *Unknown*, and *SPIT* respectively. Then the average utility of action *SPIT* of Player I is

$$U_{1S} = V_L(p)\; U_{1SL}(p_1, q_1) + V_U(p)\; U_{1SU}(p_2, q_2) + V_S(p)\; U_{1SS}(p_3, q_3), \tag{7.5}$$

where

$$
\begin{aligned}
V_L(p) &= (1-p)f_l + p\epsilon_1,\\
V_U(p) &= (1-p)h_2 + p\epsilon_2, \text{ and}\\
V_S(p) &= (1-p)h_1 + pf_s.
\end{aligned}
\tag{7.6}
$$

After expanding the terms in Equation 7.5 and doing some algebraic manipulation we obtain that

$$U_{1S} = -s_r + \epsilon_1(s_a + s_r)p_1 + \epsilon_2(s_a + s_r)p_2 + f_s(s_a + s_r)p_3\,. \tag{7.7}$$

From Theorem 1 we know that Player I uses a mixed strategy at any NE. Thus, both actions of Player I are played with strictly positive probability at any NE; in other words, both actions of Player I belong to the support of his strategy at any NE. A well known requirement for all actions that belong to the support of a NE strategy, is that each of them must achieve the same average utility. Otherwise, the user would exclude the strategies with lower average utility from his NE strategy. We know from Equation 7.4 that U_{1L}, i.e., the (average) utility of action

Legitimate for Player I, is zero. Thus, the average utility of action *SPIT* of Player I must also be

$$U_{1S} = 0\ . \tag{7.8}$$

Combining the above equation with Equation 7.7 gives the following Lemma.

Lemma 1 *At any NE of the SpitGame*

$$\epsilon_1 p_1 + \epsilon_2 p_2 + f_s p_3 = \frac{s_r}{s_a + s_r}\ . \tag{7.9}$$

We now focus on the utility of Player II. Using again the conditional probabilities of Equation 7.3 and the SpitGame model (Fig. 7.1), the average utility of Player II for each of his pure strategies at each of his information sets can be calculated. For example, in information set *Legitimate*, the average utility for Player II for action *Accept* of an incoming call is

$$U_{2LA} = P_{LL}u_l + P_{SL}(-u_s) \ = \frac{f_l(1-p)u_l - \epsilon_1 p u_s}{f_l(1-p) + \epsilon_1 p}. \tag{7.10}$$

Similarly, we can calculate the expected utilities U_{2LC} and U_{2LR} for actions *CAPTCHA* and *Reject*. In the same way, we calculate U_{2UA}, U_{2UC}, and U_{2UR} for the information set *Unknown*, and U_{2SA}, U_{2SC}, and U_{2SR} for the information set *SPIT*.

$$\begin{array}{lll} U_{2LA} = \frac{f_l(1-p)u_l - \epsilon_1 p u_s}{f_l(1-p)+\epsilon_1 p}, & U_{2LC} = \frac{f_l(1-p)(u_l-u_c)}{f_l(1-p)+\epsilon_1 p}, & U_{2LR} = \frac{-f_l(1-p)u_l}{f_l(1-p)+\epsilon_1 p}, \\ U_{2UA} = \frac{h_2(1-p)u_l - \epsilon_2 p u_s}{h_2(1-p)+\epsilon_2 p}, & U_{2UC} = \frac{h_2(1-p)(u_l-u_c)}{h_2(1-p)+\epsilon_2 p}, & U_{2UR} = \frac{-h_2(1-p)u_l}{h_2(1-p)+\epsilon_2 p}, \\ U_{2SA} = \frac{h_1(1-p)u_l - f_s p u_s}{h_1(1-p)+f_s p}, & U_{2SC} = \frac{h_1(1-p)(u_l-u_c)}{h_1(1-p)+f_s p}, & U_{2SR} = \frac{-h_1(1-p)u_l}{h_1(1-p)+f_s p}. \end{array} \tag{7.11}$$

Now, using the notation of Tables 7.4 and 7.5 for the player strategies, and the average utility for each of the pure strategies of Player II (Equation 7.11) the average utility of Player II for each information set can be calculated. For example, information set *Legitimate*, the average utility of Player II is

$$U_{2L} = p_1 U_{2LA} + q_1 U_{2LC} + r_1 U_{2LR}\ . \tag{7.12}$$

Similarly, for information sets *Unknown* and *SPIT* the average utility of Player II is

$$U_{2U} = p_2 U_{2UA} + q_2 U_{2UC} + r_2 U_{2UR} \tag{7.13}$$

and

$$U_{2S} = p_3 U_{2SA} + q_3 U_{2SC} + r_3 U_{2SR}\ , \tag{7.14}$$

Table 7.6: The coefficients for Equation 7.15

i	A_i	B_i	C_i	D_i
1	$2f_l u_l(1-p) - \epsilon_1 u_s p$	$f_l(2u_l - u_c)(1-p)$	$-f_l u_l(1-p)$	$f_l(1-p) + \epsilon_1 p$
2	$2h_2 u_l(1-p) - \epsilon_2 u_s p$	$h_2(2u_l - u_c)(1-p)$	$-h_2 u_l(1-p)$	$h_2(1-p) + \epsilon_2 p$
3	$2h_1 u_l(1-p) - f_s u_s p$	$h_1(2u_l - u_c)(1-p)$	$-h_1 u_l(1-p)$	$h_1(1-p) + f_s p$

Table 7.7: Boundary values of p

Equation	Condition	Equation	Condition
$A_1 = 0,$	if $p = \frac{2f_l u_l}{2f_l u_l + \epsilon_1 u_s} = c_1$	$A_1 = B_1,$	if $p = \frac{f_l u_c}{f_l u_c + \epsilon_1 u_s} = d_1$
$A_2 = 0,$	if $p = \frac{2h_2 u_l}{2h_2 u_l + \epsilon_2 u_s} = c_2$	$A_2 = B_2,$	if $p = \frac{h_2 u_c}{h_2 u_c + \epsilon_2 u_s} = d_2$
$A_3 = 0,$	if $p = \frac{2h_1 u_l}{2h_1 u_l + f_s u_s} = c_3$	$A_3 = B_3,$	if $p = \frac{h_1 u_c}{h_1 u_c + f_s u_s} = d_3$

respectively. Expanding Equations 7.12, 7.13, and 7.14, with the expressions of Equation 7.11 gives a closed expression for the average utility of Player II at each information set $i = 1, 2, 3$. After some algebraic manipulation, and exploiting the symmetry in the expressions for the three information sets, we obtain that the average utility of Player II in each information set is

$$\frac{A_i p_i + B_i q_i + C_i}{D_i}, \quad \text{for } i = 1, 2, 3. \tag{7.15}$$

where the coefficients A_i, B_i, C_i and D_i are as defined in Table 7.6. Note that the coefficients D_i correspond to the probabilities of each information set, as they are defined in Equation 7.6. The coefficients A_i, B_i, C_i and D_i are functions of the strategy p of Player I and other variables. We focus on p and identify the boundary values c_i and d_i for $i = 1, 2, 3$, presented in Table 7.7.

7.4.1 The Nash Equilibrium

We are now ready to determine the NE of the SpitGame. Our analysis will be valid for a wide range of parameter values. The main assumption we make is that

$$\epsilon_1 < \epsilon_2 \, . \tag{7.16}$$

This is a reasonable assumption which also holds for the empirical parameter values we use in the experiments (Table 7.3). A further plausible assumption is

that the probability that the filter verdict is correct is larger than the probability that the verdict is completely wrong. More precisely,

$$h_1 < f_l \text{ , and} \tag{7.17}$$

$$\epsilon_1 < f_s \text{ .} \tag{7.18}$$

Additionally, we assume that

$$h_2 < f_l \text{ .} \tag{7.19}$$

The final assumption, which is also a plausible one, states that

$$u_c < 2\ u_l \text{ ,} \tag{7.20}$$

that is, the cost for Player I to submit an audio CAPTCHA is less than twice the utility of accepting a legitimate call. Note that a cost u_c larger than 2 u_l would make the use of audio CAPTCHAs pointless. The cost of applying an audio CAPTCHA should actually be much lower than 2 u_l.

At any NE equilibrium, the strategy of Player II, i.e., the values of p_i and q_i, must be such that the values of U_{2L}, U_{2U} and U_{2S} are maximized, for the the given strategy p of Player I. An immediate consequence is that if the some coefficients A_i or B_i are strictly negative then the corresponding p_i or q_i will have to be null at the NE.

For each i, we will compare the coefficients of each pair of p_i and q_i. We will also compare the coefficients of all p_i with each other. In Table 7.7 the boundary values of p to satisfy specific equations on the coefficients A_i and B_i are given. For $p = c_1$, the coefficient of p_1 in Equation 7.15 for $i = 1$ becomes $A_1 = 0$. Note, that if $p > c_1$ then $A_1 < 0$, and if $p < c_1$, then $A_1 > 0$. Similarly, if $p = d_1$ then $A_1 = B_1$, if $p > d_1$, then $A_1 < B_1$, and if $p < d_1$, then $A_1 > B_1$. Similar statements hold for coefficients A_2, B_2, A_3 and B_3.

Some observations about the relations between the boundary values of p are in order. Using Equations 7.16, 7.17, 7.18, and 7.19 we obtain that

$$c_1 > c_2 \text{ and } c_1 > c_3 \text{ .} \tag{7.21}$$

Similarly, we obtain

$$d_1 > d_2 \text{ and } d_1 > d_3 \text{ .} \tag{7.22}$$

Using Equation 7.20 we immediately obtain that

$$d_i < c_i, \text{ for } i = 1, 2, 3. \tag{7.23}$$

and

$$B_i > 0, \text{ for } i = 1, 2, 3. \tag{7.24}$$

We can also make some observations about the strategy of Player II. An immediate consequence of Equation 7.9 is that

$$p_1 + p_2 + p_3 > 0 \,. \tag{7.25}$$

Thus, at least one of the A_i must be ≥ 0. This, in turn, implies that $p \leq \min\{c1, c2, c3\} = c_1$. Moreover, from Equation 7.24 we know that all coefficients B_i are strictly positive. This implies that

$$p_i + q_i = 1 \text{ for } i = 1, 2, 3. \tag{7.26}$$

In other words, the action *Reject* is not used by Player II at any NE. A careful look at the SpitGame in Fig. 7.1 reveals that action *Reject* of Player II is weakly dominated by his action *CAPTCHA*. This means, that the utility of action *Reject* is less than or equal and in some cases strictly less than the utility of action *CAPTCHA*. However, this observation alone would not be sufficient to exclude action *Reject* from NE strategies. There are well known examples of games having NE where players use also weakly dominated strategies.

Finally, let σ be

$$\sigma = s_r/(s_a + s_r) \,. \tag{7.27}$$

7.4.1.1 Case Analysis

We are now ready to obtain the NE of the SpitGame.

Case 1: $\epsilon_1 \geq \sigma$

Let us first consider the case $\epsilon_1 > \sigma$. From Equation 7.9 we obtain that $p_1 < 1$. Thus in Equation 7.12 we have $p_1 > 0$ and $q_1 > 0$. Recall, that values of p_1 and q_1 at a NE have to maximize the utility U_{2L}. The only way the expression U_{2L} is maximized for p_1 and q_1 both strictly positive is if $A_1 = B_1 \geq 0$. To have $A_1 = B_1$, it must hold that $p = d_1$. Moreover, the corresponding value of A_1 and B_1 for $p = d_1$ is strictly positive from Equation 7.24. Thus, the SpitGame has a single NE equilibrium at $p = d_1$. Moreover, for $p = d_1$, we have $A_2 < B_2$ and $A_3 < B_3$. Consequently, $p_2 = p_3 = 0$, and thus $q_2 = q_3 = 1$. Using Equation 7.7 we get $p_1 = \frac{\sigma}{\epsilon_1}$.

We will now obtain the same results for the case $\epsilon_1 = \sigma$. First we will show that $p_1 = 1$. Assume, $p_1 < 1$. Then $q_1 > 0 \Rightarrow A_1 = B_1$ and, thus $p = d_1$. Moreover, for $p = d_1$, we have $A_2 < B_2$ and $A_3 < B_3$. Consequently, $p_2 = p_3 = 0$. At the same time, using $p_1 < 1$ in Equation 7.9 gives $p_2 + p_3 > 0$, a contradiction with the previous result. Thus, in this case $p_1 = 1$. From $p_1 = 1$, we obtain $p_2 = p_3 = 0$, $q_1 = 0$, and $q_2 = q_3 = 1$.

Thus, for the case of $\epsilon_1 \geq \sigma$, the SpitGame has the following unique NE

p	p_1	q_1	p_2	q_2	p_3	q_3
d_1	$\frac{\sigma}{\epsilon_1}$	$1-p_1$	0	1	0	1

Note that we do not show the values of the r_i for the SpitGame, since their value will always be zero, as discussed earlier.

Case 2: $\epsilon_1 < \sigma$
We have to further distinguish three sub-cases based on the relation of the ratios ϵ_2/h_2 and f_s/h_1.

Case 2.1: $\epsilon_2/h_2 < f_s/h_1$
The inequality $\epsilon_2/h_2 < f_s/h_1$ implies that

$$c_2 > c_3 \text{ and } d_2 > d_3 \,. \tag{7.28}$$

Case 2.1.1: $\epsilon_1 < \sigma < \epsilon_1 + \epsilon_2$
In this case, if p_1 would be $p_1 < 1$, then (as in the case $\epsilon_1 > \sigma$) we would have $p_2 = p_3 = 0$. However, then Equation 7.9 would be infeasible. Thus,

$$p_1 = 1 \,,\; q_1 = 0 \,. \tag{7.29}$$

If $\epsilon_1 = \sigma$, then from Equations 7.29 and 7.9, we again conclude that $p_2 = p_3 = 0$. If $\epsilon_1 > \sigma$, then for the same reason it must hold $p_2 + p_3 > 0$, that is, at least one of p_2 and p_3 must be strictly positive (because else Equation 7.9 would be infeasible).

If $A_2 > B_2 \Rightarrow p_2 = 1$. This, however, makes Equation 7.9 on p_1, p_2 and p_3, infeasible. The case $A_2 < B_2$ is also not feasible, because then we would have $p_2 = 0$ and $q_2 = 1$, which would again make Equation 7.9 infeasible. Consequently, it must hold $A_2 = B_2$ and consequently $p = d_2$.

Thus, the NE for Case 2.1.1 is

p	p_1	q_1	p_2	q_2	p_3	q_3
d_2	1	0	$\frac{\sigma-\epsilon_1}{\epsilon_2}$	$1-p_2$	0	1

Case 2.1.2: $\epsilon_1 + \epsilon_2 \leq \sigma$.
Assume that $A_3 > B_3$. Then, $A_3 > B_3 \Rightarrow p < d_3 \Rightarrow p < d_2 \Rightarrow A_2 > B_2 \Rightarrow p_2 = p_3 = 1$. In this case the strategy of Player II would be always *Accept*, which is not a NE strategy (Player I would simply respond always with *SPIT*). Thus A_3 cannot be smaller than B_3. The case $A_3 < B_3$ is also not possible, because it would imply $p_3 = 0$, which in turn would make Equation 7.9 infeasible. From the above arguments, we conclude that

$$A_3 = B_3 \,. \tag{7.30}$$

Thus, in this case, $A_1 > B_1$, $A_2 > B_2$ and $A_3 = B_3$ and consequently

$$p = d_3 \,. \tag{7.31}$$

The overall NE is

p	p_1	q_1	p_2	q_2	p_3	q_3
d_3	1	0	1	0	$\frac{\sigma-\epsilon_1-\epsilon_2}{f_s}$	$1-p_3$

Case 2.2: $\epsilon_2/h_2 > f_s/h_1$.
The inequality $\epsilon_2/h_2 > f_s/h_1$ implies that

$$c_2 < c_3 \text{ and } d_2 < d_3 \,. \tag{7.32}$$

A simple adaptation of the analysis of the cases 2.1.1 and 2.1.2 gives the following results for cases 2.2.1 and 2.2.2, respectively.

Case 2.2.1: $\epsilon_1 < \sigma < \epsilon_1 + \epsilon_2$
In this case, $p = d_3$ and the overall NE is

p	p_1	q_1	p_2	q_2	p_3	q_3
d_3	1	0	0	1	$\frac{\sigma-\epsilon_1}{f_s}$	$1-p_3$

Case 2.2.2: $\epsilon_1 + \epsilon_2 \leq \sigma$.
In this case, $p = d_2$ and the overall NE is

p	p_1	q_1	p_2	q_2	p_3	q_3
d_2	1	0	$\frac{\sigma-\epsilon_1-f_s}{\epsilon_2}$	$1-p_2$	1	0

Case 2.3: $\epsilon_2/h_2 = f_s/h_1$.
In this case,

$$d_2 = d_3 \text{ and } c_2 = c_3 \,. \tag{7.33}$$

The case $p_1 < 1$ can easily be excluded, because it would imply $p_2 = p_3 = 0$, making Equation 7.9 infeasible. Thus, we conclude that $p_1 = 1$. From Equation 7.9 we obtain that $p_2 + p_3 > 0$. Any pair of values p_2 and p_3 satisfying $\epsilon_2 p_2 + f_s p_3 = \sigma - \epsilon_1$ gives a NE. In this case the SpitGame has the following continuous range of NE

p	p_1	q_1	p_2	q_2	p_3	q_3
d_2	1	0	p_2	$1-p_2$	$\frac{\sigma-\epsilon_1-\epsilon_2 p_2}{f_s}$	$1-p_3$

where $d_2 = d_3$ and the range of values for p_2 is

$$\max\{0, \frac{\sigma-\epsilon_1-f_s}{\epsilon_2}\} \leq p_2 \leq \frac{\sigma-\epsilon_1-f_s \max\{0, \frac{\sigma-\epsilon_1-\epsilon_2}{f_s}\}}{\epsilon_2} \,. \tag{7.34}$$

From the above case analysis of the SpitGame we conclude that:

Theorem 2 *The SpitGame has a unique NE equilibrium for the assumptions made earlier except for the Case 2.3. The closed forms of the NE for each case are summarized in Table 7.8.*

7.4.2 The NE without Audio CAPTCHAs

We examine now the NE of the SpitGame if users did not have the option to use audio CAPTCHAs. We can assume that the action CAPTCHA is removed from the game or equivalently that $u_c > 2\,u_l$. If $u_c > 2\,u_l$, then all coefficients B_i would be negative

$$B_i < 0 \text{ for } i = 1, 2, 3. \tag{7.35}$$

and consequently the probability of submitting an audio CAPTCHA would be $q_i = 0$, for all infomation sets. We will call the model without audio CAPTCHAs SpitGame′.

From Equations 7.25 and 7.21 we obtain that in the SpitGame′ the strategy of Player I satisfies $p \leq \max\{c_1, c_2, c_3\} = c_1$.

Case 1: $\epsilon_1 \geq \sigma$

Let us first consider the case $\epsilon_1 > \sigma$. From Equation 7.9 we obtain that $p_1 < 1$. Thus in Equation 7.12 we have $p_1 > 0$ and $q_1 > 0$. Recall, that the values of p_1 and q_1 at any NE have to maximize the utility U_{2L}. Given that $B_1 < 0$, the only way the expression U_{2L} is maximized for p_1 and q_1 both strictly positive is if $A_1 = 0$. This requires that $p = c_1$.

Since $p = c_1$ implies $A_2 < 0$ and $A_3 < 0$, we obtain that $p_2 = p_3 = 0$ (and thus $r_2 = r_3 = 1$). Using this in Equation 7.9 we obtain that $p_1 = \frac{\sigma}{\epsilon_1}$.

Thus, the SpitGame′ has the following unique NE

p	p_1	r_1	p_2	r_2	p_3	r_3
c_1	$\frac{\sigma}{\epsilon_1}$	$1-p_1$	0	1	0	1

There is an evident analogy with the corresponding NE of the original SpitGame. The strategy of Player I is c_1 instead of d_1, while the strategy of Player II is the same if we swap the values of q_i and r_i. In Section 7.4.3 we will show that the probability of SPIT calls c_1 is $c_1 > d_1$, for $u_c < 2u_l$. That is, the rate of SPIT calls in the SpitGame′ is increased in comparison with the corresponding case of the SpitGame. We will also compare the corresponding utilities of Player II in both models.

Working in the same way it is straightforward to adapt the rest of the analysis of the original SpitGame to the SpitGame′. The results are presented below.

Case 2: $\epsilon_1 < \sigma$

Case 2.1: $\epsilon_2/h_2 < f_s/h_1$

Case 2.1.1: $\epsilon_1 < \sigma < \epsilon_1 + \epsilon_2$

p	p_1	r_1	p_2	r_2	p_3	r_3
c_2	1	0	$\frac{\sigma-\epsilon_1}{\epsilon_2}$	$1-p_2$	0	1

Case 2.1.2: $\epsilon_1 + \epsilon_2 \leq \sigma$.

p	p_1	r_1	p_2	r_2	p_3	r_3
c_3	1	0	1	0	$\frac{\sigma-\epsilon_1-\epsilon_2}{f_s}$	$1-p_3$

Case 2.2: $\epsilon_2/h_2 > f_s/h_1$.

p	p_1	r_1	p_2	r_2	p_3	r_3
c_3	1	0	0	1	$\frac{\sigma-\epsilon_1}{f_s}$	$1-p_3$

Case 2.2.2: $\epsilon_1 < \sigma < \epsilon_1 + \epsilon_2$.

p	p_1	r_1	p_2	r_2	p_3	r_3
c_2	1	0	$\frac{\sigma-\epsilon_1-f_s}{\epsilon_2}$	$1-p_2$	1	0

Case 2.3: $\epsilon_2/h_2 = f_s/h_1$.

p	p_1	r_1	p_2	r_2	p_3	r_3
c_2	1	0	p_2	$1-p_2$	$\frac{\sigma-\epsilon_1-\epsilon_2 p_2}{f_s}$	$1-p_3$

where $c_2 = c_3$ and the range of values for p_2 is

$$\max\{0, \frac{\sigma - \epsilon_1 - f_s}{\epsilon_2}\} \leq p_2 \leq \frac{\sigma - \epsilon_1 - f_s \max\{0, \frac{\sigma-\epsilon_1-\epsilon_2}{f_s}\}}{\epsilon_2}. \tag{7.36}$$

The closed forms of the NE for all cases of the SpitGame and the SpitGame′ are summarized in Table 7.8.

7.4.3 The Benefit of Supporting Audio CAPTCHAs

We can now compare the NE of the SpitGame and the SpitGame′ in order to assess the effect of audio CAPTCHAs on the properties of the corresponding NE. We are interested in the rate of SPIT calls at the NE and the corresponding utility of Player II, the VoIP user.

Note that the strategy of Player I is always some value d_i, for $i \in \{1,2,3\}$ in the SpitGame, and c_i for the same index value of i in the corresponding SpitGame′. Using Equation 7.20 it is straightforward to show that $c_i > d_i$, for any $i \in \{1,2,3\}$, which implies a reduced rate of SPIT calls at the NE of the SpitGame. For example the ratio c_1/d_1 is

$$c_1/d_1 = \frac{2u_l(f_l u_c + \epsilon_1 u_s)}{u_c(2f_l u_l + \epsilon_1 u_s)} > 1. \tag{7.37}$$

Similarly, the ratios c_2/d_2 and c_3/d_3 can also be shown to be larger than 1.

Theorem 3 *At NE, the rate of SPIT calls is strictly less when users have the option to submit audio CAPTCHA's.*

The utility of Player II at any NE of the SpitGame is larger than in the NE of the corresponding SpitGame. For example, the difference of the utility of Player II in Case 1 of the SpitGame minus the utility of the NE of the corresponding NE in the SpitGame′ is

$$U_2 - U_2' = \frac{\epsilon_1 h_2 (2u_l - u_c) u_s (f_l u_l + \epsilon_1 u_s)}{(f_l u_c + \epsilon_1 u_s)(2 f_l u_l + \epsilon_1 u_s)} > 0. \tag{7.38}$$

Note, that we use the difference for the utilities instead of the ratio, because Player II may have a negative utility in the SpitGame′. For Case 2.1.1 the difference is

$$U_2 - U_2' = \frac{\epsilon_2 h_2 (2u_l - u_c) u_s (h_2 u_l + \epsilon_2 u_s)}{(h_2 u_c + \epsilon_2 u_s)(2 h_2 u_l + \epsilon_2 u_s)} > 0. \tag{7.39}$$

In the same way, the difference of the utilities of Player II at NE in the SpitGame and the SpitGame′ can be shown to be positive for the remaining cases of the game.

Theorem 4 *At NE, the utility of Player II is larger in the SpitGame than in the corresponding SpitGame′.*

7.5 Experimental Study

For the experimental analysis we produce the theoretically predicted Nash Equilibria properties independently from the theoretical analysis. We have selected realistic values for the filter's ability to discern legitimate calls from SPIT calls based on the analysis performed in Sections 3 and 4. The experimental analysis was performed for three filter specification cases, shown in Table 7.9, and for each case we examined the NE of both SpitGame and SpitGame′.

The first filter specification case represents the most realistic case: the filter has significant difficulties in identifying SPIT calls resulting in a large percentage of SPIT calls being classified as *Unknown*, but can classify legitimate calls with relatively high accuracy. The second filter specification represents the conditions in a large organisation which receives calls from a large pool of people. As a result, it tends to classify both SPIT and legitimate calls as *Unknown*. The third filter specification represents a smaller organisation with a much smaller pool of frequent callers. Therefore, it tends to identify SPIT and legitimate calls much more accurately than in the previous two cases.

Table 7.8: The NE of SpitGame and SpitGame′ (without CAPTCHAs). The ranges of values for p_2 in case 2.3 of SpitGame and 2.3 of SpitGame′ are given in Equations 7.34 and 7.36, respectively.

		Player I	Player II (Information Sets)								
			Legitimate			*Unknown*			*SPIT*		
SpitGame Case		p $1-p$	p_1	q_1	r_1	p_2	q_2	r_2	p_3	q_3	r_3
1	$\epsilon_1 \geq \sigma$	$d_1(1-d_1)$	$\frac{\sigma}{\epsilon_1}$	$1-\frac{\sigma}{\epsilon_1}$	0	0	1	0	0	1	0
2	$\epsilon_1 < \sigma$										
2.1	$\epsilon_2/h_2 < f_s/h_1$										
2.1.1	$\epsilon_1+\epsilon_2 > \sigma$	$d_2(1-d_2)$	1	0	0	$\frac{\sigma-\epsilon_1}{\epsilon_2}$	$1-p_2$	0	0	1	0
2.1.2	$\epsilon_1+\epsilon_2 \leq \sigma$	$d_3(1-d_3)$	1	0	0	1	0	0	$\frac{\sigma-\epsilon_1-\epsilon_2}{f_s}$	$1-p_3$	0
2.2	$\epsilon_2/h_2 > f_s/h_1$										
2.2.1	$\epsilon_1+\epsilon_2 > \sigma$	$d_3(1-d_3)$	1	0	0	0	1	0	$\frac{\sigma-\epsilon_1}{f_s}$	$1-p_3$	0
2.2.2	$\epsilon_1+\epsilon_2 \leq \sigma$	$d_2(1-d_2)$	1	0	0	$\frac{\sigma-\epsilon_1-f_s}{\epsilon_2}$	$1-p_2$	0	1	0	0
2.3	$\epsilon_2/h_2 = f_s/h_1$	$d_2(1-d_2)$	1	0	0	p_2	$1-p_2$	0	$\frac{\sigma-\epsilon_1-\epsilon_2 p_2}{f_s}$	$1-p_3$	0
SpitGame′ Case		p $1-p$	p_1	q_1	r_1	p_2	q_2	r_2	p_3	q_3	r_3
1	$\epsilon_1 \geq \sigma$	$c_1(1-c_1)$	$\frac{\sigma}{\epsilon_1}$	0	$1-\frac{\sigma}{\epsilon_1}$	0	0	1	0	0	1
2	$\epsilon_1 < \sigma$										
2.1	$\epsilon_2/h_2 < f_s/h_1$										
2.1.1	$\epsilon_1+\epsilon_2 > \sigma$	$c_2(1-c_2)$	1	0	0	$\frac{\sigma-\epsilon_1}{\epsilon_2}$	0	$1-p_2$	0	0	1
2.1.2	$\epsilon_1+\epsilon_2 \leq \sigma$	$c_3(1-c_3)$	1	0	0	1	0	0	$\frac{\sigma-\epsilon_1-\epsilon_2}{f_s}$	0	$1-p_3$
2.2	$\epsilon_2/h_2 > f_s/h_1$										
2.2.1	$\epsilon_1+\epsilon_2 > \sigma$	$c_3(1-c_3)$	1	0	0	0	1	0	$\frac{\sigma-\epsilon_1}{f_s}$	0	$1-p_3$
2.2.2	$\epsilon_1+\epsilon_2 \leq \sigma$	$c_2(1-c_2)$	1	0	0	$\frac{\sigma-\epsilon_1-f_s}{\epsilon_2}$	0	$1-p_2$	1	0	0
2.3	$\epsilon_2/h_2 = f_s/h_1$	$c_2(1-c_2)$	1	0	0	p_2	0	$1-p_2$	$\frac{\sigma-\epsilon_1-\epsilon_2 p_2}{f_s}$	0	$1-p_3$

Table 7.9: The experimental filter verdicts.

		Filter verdict		
Filter Specification	Type of call	*Legitimate*	*Unknown*	*SPIT*
1	*SPIT*	0.1	0.6	0.3
	Legitimate	0.7	0.25	0.05
2	*SPIT*	0.1	0.6	0.3
	Legitimate	0.3	0.6	0.1
3	*SPIT*	0.05	0.25	0.7
	Legitimate	0.7	0.25	0.05

In order to reduce the original problem from a 5D parameter space into an equivalent 3D exploration space we take advantage of the conditions on the

parameters shown in Table 7.2 to set $u_l = 100$ and $s_a = 100$. In order to further reduce the number of problem instances to solve, we take integral values for u_s, u_c and s_r. The restrictions convert the original 5D parameter space into a 3D exploration space shown in Table 7.10. Additionally, we performed adaptive exploration of the games for values of s_r near the boundary conditions for each case.

We automatically computed the Nash equilibria of these games using the *gambit-lcp* program supplied with Gambit [68] and fitted the resulting data to functions independently from the theoretical analysis.

7.5.1 Experimental Results & Discussion

The first result is that the Nash equilibria are unique, i.e., for each set of distinct values of u_c, u_s and s_r, the game produces exactly one Nash equilibrium. This has also significantly simplified our results and their analysis. It also means that there are no other equilibriums, with potentially worse outcomes for the user, for the game. As a result, the user's selection of strategies, given the SPIT sender's pay-offs always leads to exactly one equilibrium state. We have also verified empirically the validity of Theorem 1, by finding that all the NE, in all game instances, are mixed.

Another interesting result is the percentage of legitimate calls that the SPIT sender decides on (or conversely, the percentage of SPIT calls, as they are complementary) in the NE as a function of u_c and u_s. In the filter specification cases 1 and 2 there are two s_r value groups ($1 \leq s_r \leq 11.\overline{1}$ and $11.\overline{1} < s_r \leq 99$), while filter specification case 3 has three s_r value groups ($1 \leq s_r \leq 5.263$, $5.263 < s_r < 42.86$ and $42.86 < s_r \leq 99$). These results are shown in Fig. 7.2.

These s_r value groupings are correspond to the two base cases (1 and 2) illustrated in Table 7.8 for both SpitGame and SpitGame′. As an example, for the first filter specification and for case 1 in Table 7.8:

$$\begin{array}{lcl} \epsilon_1 & \geq & \sigma \Leftrightarrow \\ 0.1 & \geq & s_r/(s_a + s_r) \Leftrightarrow \\ 0.1 & \geq & s_r/(100 + s_r) \Leftrightarrow \\ 11.\overline{1} & \geq & s_r \end{array}$$

Case 2 is the complement of case 1, so $11.\overline{1} < s_r$.

Table 7.10: Solution exploration space

$\mathbf{u_s}$		$\mathbf{u_c}$		$\mathbf{s_r}$		**# Instances**
$2 \ldots 99$	$\times$	$1 \ldots u_s - 1$	$\times$	$1 \ldots 99$	$=$	~ 500000

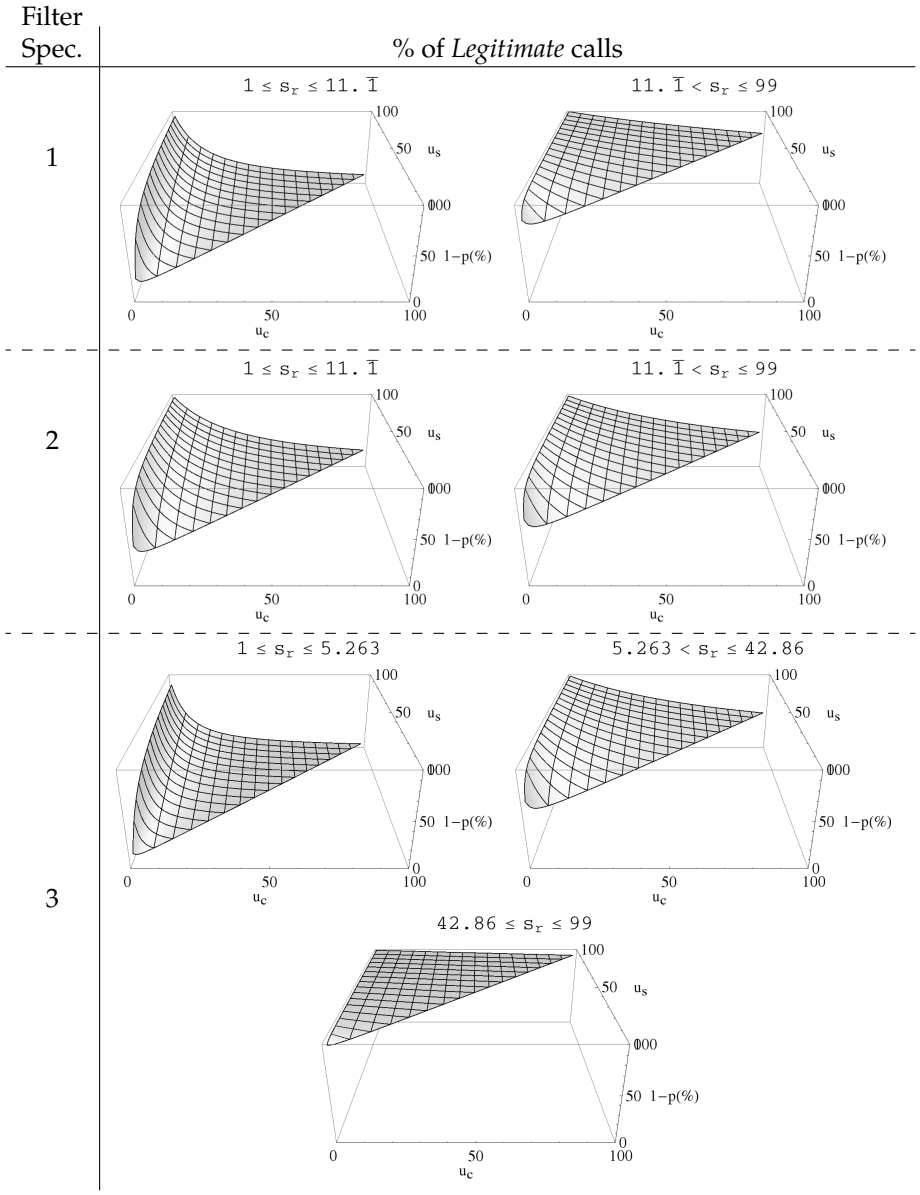


Figure 7.2: % of legitimate calls ($(1-p)*100$) (function of u_c and u_s for the s_r value groups)

In these Nash Equilibria and within each of the value groups for s_r the percentage of legitimate calls as a function of u_c, u_s is identical. Realistically, the actual s_r value for SPIT senders will be relatively low and probably $\leq 11.\overline{1}$ (i.e., the cost of attempting a SPIT call is $\leq 11.\overline{1}$% of the value gained if the SPIT call goes through) given the resources needed to make SPIT calls. The difference between the value groups pertains to the rate with which the percentage of

Filter Spec.	Fitted functions for % of legitimate calls	Absolute Fitting Error
1	$L_1(u_c, u_s) = \frac{u_s}{u_s + \alpha u_c}, \alpha = \begin{cases} 7, & 1 \le s_r \le 11.\overline{1} \quad (1) \\ 0.41\overline{6}, & 11.\overline{1} < s_r \le 99 \quad (2.1.1) \end{cases}$	$\le 5.2 \times 10^{-11}$
2	$L_2(u_c, u_s) = \frac{u_s}{u_s + \alpha u_c}, \alpha = \begin{cases} 3, & 1 \le s_r \le 11.\overline{1} \quad (1) \\ 1, & 11.\overline{1} < s_r \le 99 \quad (2.1.1) \end{cases}$	$\le 5.01 \times 10^{-11}$
3	$L_3(u_c, u_s) = \frac{u_s}{u_s + \alpha u_c}, \alpha = \begin{cases} 14, & 1 \le s_r \le 5.263 \quad (1) \\ 1, & 5.263 < s_r < 42.86 \quad (2.1.1) \\ 0.0714, & 42.86 \le s_r \le 99 \quad (2.1.2) \end{cases}$	$\le 5.14 \times 10^{-11}$

Table 7.11: Fitted functions for % of legitimate calls ($(1-p) * 100$) (function of u_c and u_s for the s_r value groups)

the legitimate calls decreases in relation to u_s and u_c. The figure illustrates that as the cost of deploying the CAPTCHA mechanism increases, the number of SPIT calls also increases (legitimate percentage decreases). At the same time, as the disutility of accepting SPIT calls (u_s) increases for the user, the number of legitimate calls increases, purportedly due to the increase in CAPTCHA use providing a strong disincentive to the SPIT sender.

The percentage $[0.0 - 1.0]$ of legitimate calls as a function of u_c and u_s has been fitted to the functions L_1, L_2, L_3 as shown in Table 7.11, which are identical to the ones produced in the theoretical analysis. For each of the fitted function cases, we have parenthesised its corresponding case in the NE Table 7.8.

7.5.2 Comparison of SpitGame and SpitGame′

In order to examine whether the use of the CAPTCHA challenge provides benefits to the users, we created a game model where all the *CAPTCHA* challenge actions have been removed (SpitGame′) and only *Accept* and *Reject* actions are present. Using the same value ranges for u_l, u_s, s_r, s_a and disregarding u_c (since there are no CAPTCHA challenges present) we performed the same experiments at the same granularity as before. Our findings from comparing the model without CAPTCHA (SpitGame′) to the model with CAPTCHA (SpitGame) are summarized in Table 7.12.

The use of CAPTCHA leads in to notable improvements to the percentage of legitimate calls since in no case does the percentage of legitimate calls drop. The improvement in percentage of legitimate calls is shown in Fig. 7.3. It is notable that for the filter specifications 1 and 2 when $s_r \le 11.\overline{1}$ and for the filter

Table 7.12: Major findings from comparison of models with (SpitGame) and without CAPTCHA (SpitGame′) in NE

Filter Spec.	Property	Model	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**
			$\mathbf{1 \leq s_r \leq 11.\overline{1}}$		$\mathbf{11.\overline{1} < s_r \leq 99}$			
1	Legit.	SpitGame	12.60%	93.45%	70.79%	99.58%		
	Calls	SpitGame′	0.14%	6.60%	2.34%	54.30%		
	User	SpitGame	0.13	92.52	46.85	99.24		
	Utility	SpitGame′	-6.60	-0.14	0.74	17.19		
			$\mathbf{1 \leq s_r \leq 11.\overline{1}}$		$\mathbf{11.\overline{1} < s_r \leq 99}$			
2	Legit.	SpitGame	25.19%	97.06%	50.25%	99.00%		
	Calls	SpitGame′	0.33%	14.16%	0.99%	33.11%		
	User	SpitGame	0.50	0.96	10.86	98.2		
	Utility	SpitGame′	-14.16	-0.33	-19.87	-0.59		
			$\mathbf{1 \leq s_r \leq 5.263}$		$\mathbf{5.263 < s_r < 42.86}$		$\mathbf{42.86 \leq s_r \leq 99}$	
3	Legit.	SpitGame	6.73%	87.61%	50.25%	99.00%	93.40%	99.93%
	Calls	SpitGame′	0.07%	3.42%	0.99%	33.11%	12.28%	87.39%
	User	SpitGame	0.13	86.73	33.02	98.65	86.86	99.86
	Utility	SpitGame′	-3.42	-0.07	0.30	9.93	10.53	74.91

specification 3 when $s_r \leq 5.263$, the CAPTCHA-less model performs so badly that the measure of improvement is almost identical to the performance of the model with CAPTCHA.

Further discoveries include the fact that for the first filter specification, for all values of s_r, when the filter identifies a call as *SPIT*, only the *CAPTCHA* action is used (never *Reject* or *Accept*). Also, even when the call is identified as either *Legitimate* or *Unknown*, the *Reject* action is never used. Furthermore, when the filter identifies a call as *Legitimate* and $s_r > 11.\overline{1}$ the user always selects *Accept*. Finally, when the filter identifies a call as *Unknown* and $s_r \leq 11.\overline{1}$ the user never selects *Accept*. These discoveries, summarized in Table 7.13, mean that for the more realistic values of s_r ($\leq 11.\overline{1}$) the *Accept* action can be removed without impact when the filter identifies a call as *Unknown* or *SPIT*.

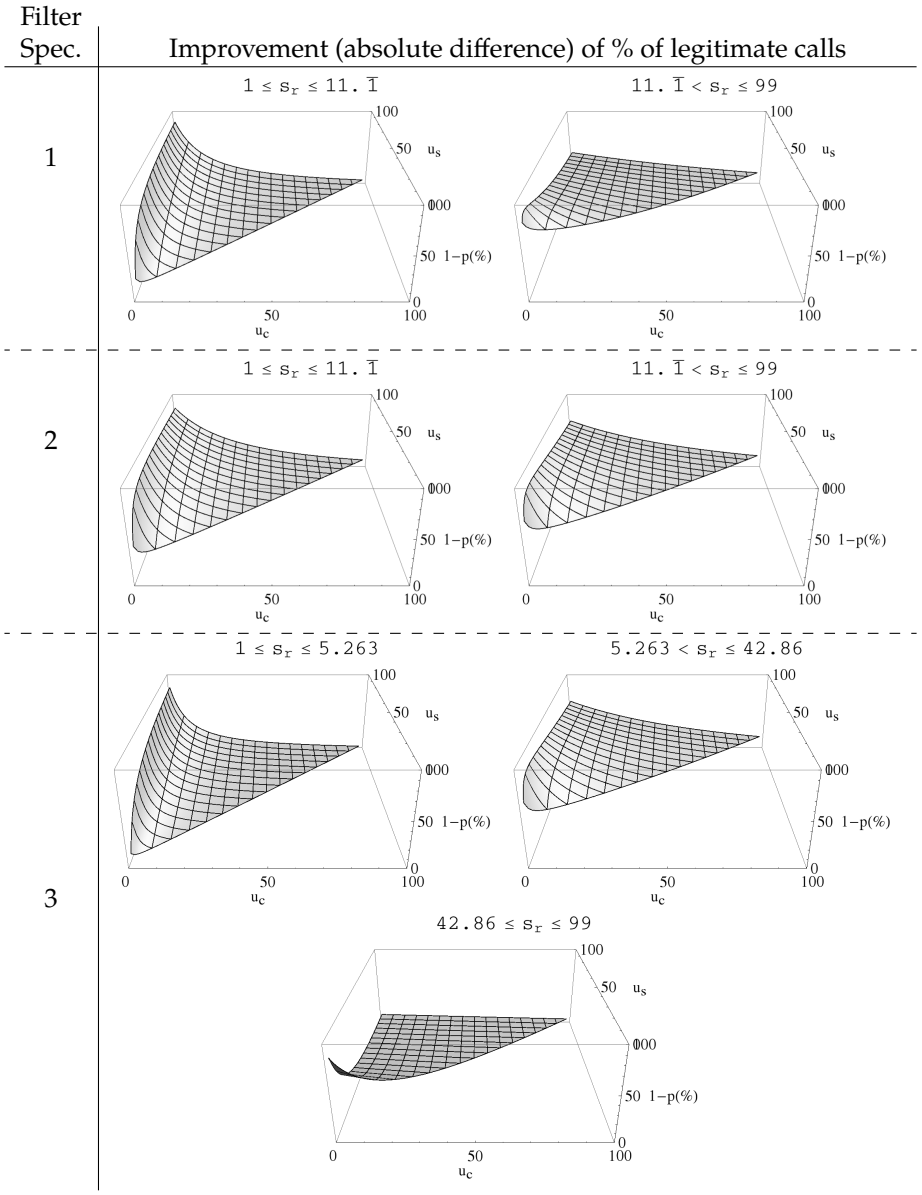


Figure 7.3: Improvement (absolute difference) of % of legitimate calls with CAPTCHA (SpitGame) vs. without CAPTCHA (SpitGame′)

7.6 Conclusions and future work

Spam over Internet Telephony is a significant threat for VoIP communications, which may become a serious problem just like ordinary spam is for email. In this work, we focused on the strategic interaction between SPIT senders and

Table 7.13: Summary of actions used based on filter call identification and the value s_r in the first filter specification case

Call identified as	**Actions Used**	
	$\mathbf{1 \leq s_r \leq 11.\bar{1}}$	$\mathbf{11.\bar{1} < s_r \leq 99}$
Legitimate	*Accept*, *CAPTCHA*	*Accept*
Unknown	*CAPTCHA*	*Accept*, *CAPTCHA*
SPIT	*CAPTCHA*	*CAPTCHA*

legitimate VoIP users. We assumed the existence of incoming call filters and effective audio CAPTCHAs and armed the VoIP users with the option to accept an incoming call, to reject it or to request an audio CAPTCHA based on a filter's verdict.

The main contribution of our work is the derivation of game-theoretic model that captures the interaction of independent, selfish SPIT senders and VoIP users. Through theoretical arguments and a comprehensive experimental analysis we studied the properties of the proposed game and identified its Nash equilibria.

The outcomes of our approach show that the use of the above mentioned defensive mechanisms lead to desirable Nash equilibria, where audio CAPTCHAs contribute to the utility of the legitimate users. Moreover, if the user and SPIT sender pay-offs are known, then the game always leads to exactly one equilibrium state with predictable characteristics.

In is noteworthy that in our model we allow for the attacker (SPIT user) to already know the performance characteristics of our filter. As a result, we are not vulnerable to attacks which would uncover the filter's characteristics. In addition, at NE, all players, hence SPIT senders too, have full knowledge of the strategies of their opponents, but still cannot achieve a better outcome. This means that in our approach we are not attempting to secure through obscurity.

The game-theoretic model of this work can be extended in several aspects to capture more properties of the real problem. An interesting topic for further research could be to refine the audio CAPTCHAs, for example, with additional parameters to model the solvability of the audio CAPTCHA. We have assumed here that the audio CAPTCHA are always solvable by a legitimate user and never solvable by a SPIT sender (automated SPIT application). New research works [12][100] have proven that about 10% of the humans are unable to solve them and that the success rate of the bots is about 5%. This new parameter

would cover these edge cases.

Building upon the theoretical arguments and the experimental results presented here, we plan to work on performing a complete theoretical analysis of the SpitGame [38]. As part of this analysis, we plan to investigate how different filter parameters influence the Nash equilibria and lead the VoIP users and the SPIT sender to adjust their behaviour. This will aid in further informing the decisions on trade-offs when implementing real CAPTCHA-based anti-SPIT systems.

CHAPTER 8

Conclusions and Directions

Efficient resource allocation on the Internet is critical due to the continuously growing demands of the users. In order to ensure sustained operation under the demands of independent and selfish users, Quality of Service mechanisms are used. The design and analysis of QoS mechanisms is a suitable field for the application of game theory, due to the nature of the problem the mechanisms pose to solve.

In this work we proposed several solutions to the QoS mechanism problem utilising game-theoretic modelling and analysis while the core aim was to investigate if (fiat) money can be used as a tool for coordination between selfish packets. Specifically, we initially designed and experimentally evaluated three variants of an active queue mechanism which allocates throughput in a distribution resembling MaxMin fairness to both responsive and unresponsive flows. We then implemented a data structure and associated algorithms which allow the efficient implementation of the best-performing variant of the Prince queue mechanism.

Aiming to implement a more general QoS mechanism, we designed and evaluated PacketEconomy, a network economy utilising money to facilitate router queue position exchanges between waiting packets in order to self-regulate access to the common resource. The analysis of this mechanism proves the existence of Nash equilibria where packets participate in the market and a simple network performance evaluation provides evidence that the QoS functionality is implemented.

In order to more rigorously verify the basic idea put forth in PacketEconomy, we examine it in the context of the OMNET++ discrete event simulator and using the INET network simulation library, adapting the theoretical model to the requirements of the realistic network environment. The thorough experi-

ments cover multiple cases and show that both the game-theoretic incentive to participate is preserved and also that the QoS functionality is flexible and consistent.

Building on our experience with game-theoretic analysis, we also model and analyse the problem of access to a VoIP service in the context of preventing Spam over Internet Telephony using CAPTCHA challenges. The game-theoretic results show both analytically and experimentally that spam can be prevented through the introduction of a CAPTCHA challenge given reasonable assumptions about its performance.

	Purpose		Managed aspect				
Work	Fairness	General QoS	Throughput	Delay	Access	Experimental evaluation	Game-theoretic analysis
Prince	●		●			●	●
HL-Hitters	●		●			●	
PacketEconomy model		●	●	●		●	●
PacketEconomy implementation		●	●	●		●	●
SpitGame	●				●	●	●

Table 8.1: Summary of mechanisms for the management of competitive access to common resources.

In Table 8.1 the main characteristics of the works implemented are summarised. As illustrated, in Prince the aspect of (MaxMin-resembling) fairness was addressed, which aims to provide similar levels of resources to all users as far as throughput is concerned. The work in HL-Hitters had the same purpose and was oriented towards increasing efficiency and optimization.

In PacketEconomy the aim was generalised to arbitrary QoS requirements for both throughput and latency, allowing the allocation of resources to follow uneven distributions. In the PacketEconomy implementation extensive work was performed to adapt the theoretical model to a realistic network and to experimentally investigate its behaviour in order to verify the theoretical claims presented. Finally, in the SpitGame work, the goal of fairness to legitimate VoIP users was addressed by attempting to limit the unnecessary burden placed on them by the presence of malicious users.

Overall, we consider the game-theoretic approach we took to address the problems in this thesis to have produced both theoretically well-founded as well as practically applicable results, a claim which is also supported by the experimental results.

Our future endeavours include further investigation into the use of money as a means of coordination between selfish flows as well as examining hybrid variants of the proposed algorithms in order to further optimize computational performance. Additionally, it would be useful to examine their behaviour in complex network topologies and heterogeneous router and flow compositions. An interesting idea would also be to extend the mechanisms to additionally take the size of the packets into account.

References

[1] NS-2 (Network simulator 2). http://www.isi.edu/nsnam/ns/. 24

[2] D. Acemoglu and A. Ozdaglar. Flow control, routing, and performance from service provider viewpoint. *LIDS report*, 2004. 74

[3] A. Akella, S. Seshan, R. Karp, S. Shenker, and C. Papadimitriou. Selfish behavior and stability of the Internet: a game-theoretic analysis of TCP. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 117–130, New York, NY, USA, 2002. ACM Press. 13, 14, 16, 17, 24, 39, 57, 73

[4] E. Altman, T. Boulogne, R. El-Azouzi, T. Jiménez, and L. Wynter. A survey on networking games in telecommunications. *Computers & Operations Research*, **33**[2]:286–311, 2006. 14, 73

[5] I. Androutsopoulos, E. F. Magirou, and D. K. Vassilakis. A game theoretic model of spam e-mailing. In S. University, editor, *Proc. of the 2nd Conference on Email and Anti-Spam*. USA, 2005. 114, 115, 122

[6] E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 295–304. IEEE, 2004. 107

[7] Y. Arbitman, M. Naor, and G. Segev. De-amortized Cuckoo Hashing: Provable Worst-Case Performance and Experimental Results. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part I*, ICALP '09, pages 107–118, Berlin, Heidelberg, 2009. Springer-Verlag. 21, 41

[8] T. Basar and R. Srikant. Revenue-maximizing pricing and capacity expansion in a many-users regime. In *INFOCOM 2002. Twenty-First Annual*

Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, **1**, pages 294–301. IEEE, 2002. 74

[9] K. Basu. The Traveler's Dilemma: Paradoxes of Rationality in Game Theory. *American Economic Review*, **84**[2]:391–395, may 1994. 121

[10] R. S. Boyer and J. S. Moore. A fast majority vote algorithm. Technical Report ICSCA-CMP-32, Institute for Computer Science, University of Texas, 1981. 39

[11] D. Braess. Über ein Paradoxon aus der Verkehrsplanung. *Unternehmensforschung*, **12**[1]:258–268, 1968. 113

[12] E. Bursztein, S. Bethard, C. Fabry, J. C. Mitchell, and D. Jurafsky. How good are humans at solving CAPTCHA? A large scale evaluation. In *Proc. of the 2010 IEEE Symposium on Security and Privacy*, pages 399–413. USA, 2010. 140

[13] H. Cavusoglu and S. Raghunathan. Configuration of detection software: A comparison of decision and game theory approaches. *Decision Analysis*, **1**[3]:131–148, 2004. 114, 122

[14] D.-M. Chiu and R. Jain. Analysis of the Increase / Decrease Algorithms for Congestion Avoidance in Computer Networks. *Comp.Netw.ISDN*, **17**[1]:1–14, jun 1989. 59

[15] R. Cole, Y. Dodis, and T. Roughgarden. How much can taxes help selfish routing? In *EC '03: Proceedings of the 4th ACM Conference on Electronic Commerce*, pages 98–107, New York, NY, USA, 2003. ACM. 14, 58, 75

[16] R. Cole, Y. Dodis, and T. Roughgarden. Pricing network edges for heterogeneous selfish users. In *STOC '03: Proceedings of the thirty-fifth annual ACM Symposium on Theory of Computing*, pages 521–530, New York, NY, USA, 2003. ACM. 14, 58

[17] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The complexity of computing a Nash Equilibrium. *Commun. ACM*, **52**[2]:89–97, feb 2009. xii, 121

[18] E. D. Demaine, A. Lopez-Ortiz, and J. I. Munro. Frequency estimation of Internet packet streams with limited space. *Algorithms-ESA 2002*, pages 11–20, 2002. 39

[19] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *ACM SIGCOMM Computer Communication Review*, **19**, pages 1–12. ACM, ACM New York, NY, USA, 1989. 12, 17, 19

[20] M. Dietzfelbinger and F. M. auf der Heide. A new universal class of hash functions and dynamic hashing in real time. In P. M., editor, *ICALP*, **443** of *LNCS*, pages 6–19. Springer Berlin / Heidelberg, 1990. 21, 41

[21] D. Dominguez-Sal, M. Perez-Casany, and J. L. Larriba-Pey. Cooperative cache analysis for distributed search engines. *International Journal of Information Technology, Communications and Convergence*, **1**[1]:41–65, 2010. 38

[22] S. Dritsas, V. Dritsou, B. Tsoumas, P. Constantopoulos, and D. Gritzalis. OntoSPIT: SPIT Management through Ontologies. *Computer Communications*, **32**[1]:203–212, 2009. 111

[23] R. Durstenfeld. Algorithm 235: Random permutation. *Communications of the ACM*, **7**[7]:420—-, jul 1964. 65

[24] D. Dutta, A. Goel, and J. Heidemann. Oblivious AQM and Nash Equilibria. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, **1**, pages 106–113. IEEE, 2003. 17

[25] P. S. Efraimidis, L. Tsavlidis, and G. B. Mertzios. Window-games between TCP flows. *Theoretical Computer Science*, **411**[31-33]:2798–2817, 2010. 14, 16, 38, 39, 57, 73

[26] S. El Sawda and P. Urien. SIP security attacks and solutions: A state-of-the-art review. In *Proc. of the IEEE International Conference on Information and Communication Technologies*, pages 3187–3191, apr 2006. 111

[27] Federal Communications Commission. FCC Strengthens Consumer Protections Against Telemarketing Robocalls, feb 2012. https://www.fcc.gov/document/fcc-strengthens-consumer-protections-against-telemarketing-robocalls-0. 111

[28] Federal Trade Commission. The National Do Not Call Registry: Data Book for Fiscal Year 2012, oct 2012. https://www.ftc.gov/reports/national-do-not-call-registry-data-book-fiscal-year-2012. 111

[29] Federal Trade Commission. FTC Settles "Rachel" Robocall Enforcement Case, jul 2013. http://www.ftc.gov/opa/2013/07/aplus.shtm. 112

[30] R. A. Fisher and F. Yates. *Statistical tables for biological, agricultural and medical research*. London: Oliver & Boyd, 3rd editio edition, 1948. 65

[31] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, **1**[4]:397–413, aug 1993. 12, 17, 84

[32] X. Gao, K. Jain, and L. J. Schulman. Fair and efficient router congestion control. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1050–1059, Philadelphia, PA, USA, 2004. 14, 17, 57, 73

[33] R. J. Gibbens and F. P. Kelly. Resource pricing and the evolution of congestion control. *Automatica*, **35**:1969–1985, 1999. 14, 58

[34] H. Gintis. A Markov Model of Production, Trade, and Money: Theory and Artificial Life Simulation. *Comput. Math. Organ. Theory*, **3**[1]:19–41, 1997. 14, 58, 60, 64

[35] H. Gintis. *Game Theory Evolving: A Problem-Centered Introduction to Modeling Strategic Interaction*. Princeton University Press, 2000. 14, 58

[36] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan. Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. *Annals of Discrete Mathematics*, **5**:287–326, 1979. 67

[37] D. Graham-Rowe. A sentinel to screen phone calls technology. *MIT Technology Review*, 2006. 111

[38] D. Gritzalis, P. Katsaros, S. Basagiannis, and Y. Soupionis. Formal analysis for robust anti-SPIT protection using model-checking. *International Journal of Information Security*, **11**[2]:121–135, 2012. 141

[39] D. Gritzalis, Y. Soupionis, V. Katos, I. Psaroudakis, P. Katsaros, and A. Mentis. The Sphinx enigma in critical VoIP infrastructures: Human or botnet? In I. Press, editor, *4th International Conference on Information, Intelligence, Systems and Applications*, pages 1–6. IEEE, 2013. 112

[40] E. L. Hahne. Round-Robin Scheduling for Max-Min Fairness in Data Networks. *IEEE Journal of Selected Areas in Communications*, **9**[7]:1024–1039, 1991. 25

[41] G. Hardin. The tragedy of the commons. *Science*, **162**[3859]:1243–1248, 1968. 1, 15

[42] V. Jacobson. Congestion avoidance and control. In *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, pages 314–329, New York, NY, USA, 1988. ACM. 11

[43] A. Jacquet, B. Briscoe, and T. Moncaster. Policing freedom to use the Internet resource pool. In *Proceedings of the 2008 ACM CoNEXT Conference*, page 71. ACM, 2008. 35

[44] James Daniel. Boost::Unordered, nov 2011. http://www.boost.org/doc/html/unordered.html. 42

[45] B. Johnson, J. Grossklags, N. Christin, and J. Chuang. Are security experts useful? Bayesian Nash Equilibria for network security games with limited information. In *Proceedings of the 15th European conference on Research in computer security*, ESORICS'10, pages 588–606, Berlin, Heidelberg, sep 2010. Springer-Verlag. 113

[46] A. B. Johnston. *SIP: Understanding the Session Initiation Protocol*. Artech House, 2nd edition, 2004. 112

[47] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage. Spamalytics: An empirical analysis of spam marketing conversion. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 3–14. ACM, oct 2008. 113

[48] R. Karp, E. Koutsoupias, C. Papadimitriou, and S. Shenker. Optimization problems in congestion control. In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 66, Washington, DC, USA, 2000. IEEE Computer Society. 14, 57, 74

[49] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.*, **28**[1]:51–55, 2003. 18, 21, 22, 39

[50] F. Kelly. Charging and rate control for elastic traffic. *European transactions on Telecommunications*, **8**[1]:33–37, 1997. 74

[51] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research society*, pages 237–252, 1998. 19, 74

[52] A. D. Keromytis. Voice-over-IP Security: Research and Practice. *IEEE Security and Privacy*, **8**[2]:76–78, 2010. 111

[53] A. D. Keromytis. A Comprehensive Survey of Voice over IP Security Research. *IEEE Communications Surveys & Tutorials*, **14**[2]:514–537, 2012. 111

[54] Y. Kim, Y. Park, J.-D. Lee, and J. Lee. Using stated-preference data to measure the inconvenience cost of spam among Korean e-mail users. *Applied Economics Letters*, **13**[12]:795–800, 2006. 113

[55] N. Kiyotaki and R. Wright. On Money as a Medium of Exchange. *Journal of Political Economy*, **97**[4]:927–954, aug 1989. 14, 58, 60

[56] D. E. Knuth. *The Art of Computer Programming*, **2 : Seminu**. Addison-Wesley Publishing Company, second edition, 1981. 65

[57] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *STACS '99: Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science*, pages 404–413, 1999. 13, 57, 73, 107

[58] O. Krzikalla and I. Gaztanaga. Boost::Intrusive, nov 2011. `http://www.boost.org/doc/html/intrusive.html`. 41

[59] S. Kunniyur and R. Srikant. A time scale decomposition approach to adaptive ECN marking. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, **3**, pages 1330–1339. IEEE, 2001. 74

[60] S. Li, W. Sun, and C. Hua. Optimal resource allocation for heterogeneous traffic in multipath networks. *International Journal of Communication Systems*, **29**[1]:84–98, 2016. 74

[61] H. Liu, Y. Lin, and J. Han. Methods for mining frequent items in data streams: an overview. *Knowledge and Information Systems*, **26**:1–30, 2011. 39

[62] S. H. Low and D. E. Lapsley. Optimization flow control—I: basic algorithm and convergence. *IEEE/ACM Transactions on Networking (TON)*, **7**[6]:861–874, 1999. 74

[63] K. Ma, Q. Han, C. Chen, and X. Guan. Bandwidth allocation for cooperative relay networks based on Nash bargaining solution. *International Journal of Communication Systems*, **25**[8]:1044–1058, 2012. 74

[64] J. K. MacKie-Mason. A Smart Market for Resource Reservation in a Multiple Quality of Service Information Network. *SSRN Working Paper Series*, 2007. 108, 109

[65] J. K. MacKie-Mason and H. R. Varian. Pricing the Internet. In *Public Access to the Internet*, pages 269–314. Prentice Hall, 1993. 14, 58

[66] R. Mahajan, S. Floyd, and D. Wetherall. Controlling high-bandwidth flows at the congested router. In *Network Protocols, 2001. Ninth International Conference on*, pages 192–201. IEEE, 2001. 12, 17

[67] M. H. Manshaei, Q. Zhu, T. Alpcan, T. Bacşar, and J.-P. Hubaux. Game theory meets network security and privacy. *ACM Comput. Surv.*, **45**[3]:25:1—–25:39, jul 2013. 121

[68] R. D. McKelvey, A. M. McLennan, and T. L. Turocy. Gambit: Software tools for game theory. `http://www.gambit-project.org`, Version 0.2010.09.01. 135

[69] L. W. McKnight and J. P. Bailey, editors. *Internet Economics*. MIT Press, Cambridge, MA, USA, 1997. 14, 58

[70] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005. 64

[71] S. M. Muthukrishnan. Data streams: algorithms and applications. *Found. Trends Theor. Comput. Sci.*, **1**:117–236, aug 2005. 39

[72] R. B. Myerson. *Game Theory: Analysis of Conflict*. Harvard University Press, Cambridge, MA, 1991. 120, 121, 122

[73] J. Nagle. On Packet Switches with Infinite Storage. *Communications, IEEE Transactions on*, **35**[4]:435–438, 1987. 13, 16, 39

[74] J. F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the USA*, **36**[1]:48–49, jan 1950. 64

[75] J. F. Nash. Non-cooperative Games. *Annals of Mathematics*, **54**[2]:286–295, 1951. 64

[76] S. Niccolini, S. Tartarelli, M. Stiemerling, and S. Srivastava. SIP Extensions for SPIT Identification. Internet draft, Network Working Group, aug 2007. 112

[77] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic Game Theory*, **1**. Cambridge University Press, New York, NY, USA, 2007. 13, 14, 73, 115, 120, 121

[78] A. Odlyzko. Paris metro pricing for the Internet. In *EC '99: Proceedings of the 1st ACM Conference on Electronic Commerce*, pages 140–147, New York, NY, USA, 1999. ACM. 14, 58

[79] M. Osborne. *An Introduction to Game Theory*. Oxford University Press, 2003. 13, 115, 120

[80] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. The MIT Press, 1994. 13, 115, 120

[81] G. Owen. *Game Theory*. Academic Press, NY, 1982. 13, 115

[82] K. Pagiamtzis and A. Sheikholeslami. Content-addressable memory (CAM) circuits and architectures: A tutorial and survey. *IEEE JSSC*, **41**[3]:712–727, 2006. 42

[83] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker. Approximate fairness through differential dropping. *ACM SIGCOMM Computer Communication Review*, **33**[2]:23–39, 2003. 12, 17

[84] R. Pan, B. Prabhakar, and K. Psounis. CHOKE, A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation. In *INFOCOM*, pages 942–951, 2000. 12, 17

[85] C. Papadimitriou. Algorithms, games, and the Internet. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, STOC '01, pages 749–753, New York, NY, USA, 2001. ACM. 13, 16, 39, 57, 73, 107, 121

[86] M. Parameswaran, H. Rui, and S. Sayin. A game theoretic model and empirical analysis of spammer strategies. In *Proc. of the Collaboration, Electronic Messaging, Anti-Abuse and Spam Conference*, 2010. 114

[87] L. L. Peterson and B. S. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufmann, fourth edition, 2007. 9

[88] Y. Ponomarchuk and D.-W. Seo. Intrusion Detection based on Traffic Analysis and Fuzzy Inference System in Wireless Sensor Networks. *Journal of Convergence*, **1**[1]:35–42, dec 2010. 37

[89] J. Quittek, S. Niccolini, S. Tartarelli, M. Stiemerling, M. Brunner, and T. Ewald. Detecting SPIT calls by checking human communication patterns. In *Proc. of the IEEE International Conference on Communications*, pages 1979–1984. IEEE, 2007. 111

[90] J. Rosenberg and C. Jennings. The Session Initiation Protocol (SIP) and Spam. Technical report, Network Working Group, jan 2008. 111

[91] A. S. Schulz and N. S. Moses. On the performance of user equilibria in traffic networks. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 86–87. Society for Industrial and Applied Mathematics, 2003. 107

[92] A. B. Shahroudi, R. H. Khosravi, H. R. Mashhadi, and M. Ghorbanian. Full Survey on SPIT and prediction of how VoIP providers compete in presence of SPITters using Game-Theory. In A. Dhabi, editor, *Proc. of the 2011 IEEE International Conference on Computer Applications and Industrial Electronics (ICCAIE)*, pages 402–406. IEEE, 2011. 114

[93] S. J. Shenker. Making greed work in networks: a game-theoretic analysis of switch service disciplines. *IEEE/ACM Transactions on Networking*, **3**[6]:819–831, dec 1995. 13, 14, 16, 39, 57, 73

[94] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round-robin. *Networking, IEEE/ACM Transactions on*, **4**[3]:375–385, 1996. 84

[95] R. Sinha, C. Papadopoulos, and J. Heidemann. Internet Packet Size Distributions: Some Observations. Technical Report ISI-TR-2007-643, USC/Information Sciences Institute, may 2007. 52

[96] H. Sinnreich and A. B. Johnston. *Internet Communications Using SIP: Delivering VoIP and Multimedia Services with Session*. Wiley, 2nd edition, 2006. 112

[97] A. Smith. *An Inquiry into the Nature and Causes of the Wealth of Nations*. Project gu edition, 1776. 58

[98] W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, **3**:59–66, 1956. 67

[99] Y. Soupionis, S. Dritsas, and D. Gritzalis. An adaptive policy-based approach to SPIT management. In *Proc. of the 13th European Symposium on Research in Computer Security*, pages 446–460. Springer, 2008. 112

[100] Y. Soupionis and D. Gritzalis. Audio CAPTCHA: Existing solutions assessment and a new implementation for VoIP telephony. *Computers & Security*, **29**[5]:603–618, 2010. 112, 140

[101] R. Srikant. *The mathematics of Internet congestion control*. Springer Science & Business Media, 2012. 74

[102] W. R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1994. 9

[103] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: achieving approximately fair bandwidth allocations in high speed networks. *SIGCOMM Comput. Commun. Rev.*, **28**[4]:118–130, 1998. 12, 17

[104] M. Tambe, M. Jain, J. A. Pita, and A. X. Jiang. Game theory for security: Key algorithmic principles, deployed systems, lessons learned. In *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, pages 1822–1829, 2012. 121

[105] USA Congress. Do-Not-Call Implementation Act, mar 2003. 111

[106] A. Varga. OMNeT++. In *Modeling and Tools for Network Simulation*, pages 35–59. Springer, 2010. 74

[107] A. Varga. The OMNET++ discrete event simulator, 2016. http://omnetpp.org. 74

[108] A. Varga and Others. The INET Framework, 2016. http://inet.omnetpp.org. 74

[109] D. K. Vassilakis, I. Androutsopoulos, and E. F. Magirou. A game-theoretic investigation of the effect of human interactive proofs on spam e-mail. In *Proc. of the Fourth Conference on Email and Anti-Spam (CEAS 2007)*, Mountain View, CA, July 2007, 2007. 114, 115, 122

[110] T. J. Walsh and D. R. Kuhn. Challenges in securing voice over IP. *IEEE Security and Privacy*, **3**[3]:44–49, 2005. 111

[111] T. Wang. Choke source code for ns2, nov 2011. http://cs-people.bu.edu/wtwang/paper. 25

[112] Wikipedia. Fisher-Yates shuffle, 2011. 65

[113] T. Wilson. Competition May Be Driving Surge in Botnets, Spam, aug 2008. http://www.darkreading.com/competition-may-be-driving-surge-in-botnets-spam/d/d-id/1129224. 114

[114] H. Yaïche, R. R. Mazumdar, and C. Rosenberg. A game theoretic framework for bandwidth allocation and pricing in broadband networks. *IEEE/ACM Transactions on Networking (TON)*, **8**[5]:667–678, 2000. 74