Received: 5 February 2016 | Revised: 5 September 2016 | Accepted: 10 November 2016

DOI: 10.1002/dac.3252

WILEY

RESEARCH ARTICLE

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

Remous-Aris Koutsiamanis | **Pavlos S. Efraimidis**

Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, 67100, Greece

Correspondence
Remous-Aris Koutsiamanis, Department of Electrical and Computer Engineering, Democritus University of Thrace, 67100 Xanthi, Greece.
Email: akoutsia@ee.duth.gr

Summary

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 Internet Protocol version 6 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 Transmission Control Protocol– and User Datagram Protocol–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 to be able 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.

KEYWORDS

game theory, networks, OMNET++, quality of service, simulation

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. The problem, thus, is challenging since the aim is to allocate the limited resources in a manner both efficient (to minimize waste) and tailored to the individual flow requirements (to improve user experience). 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 that has been taken in several works with some of the early ones being,[1,2] and overviews of which are presented in the works of Altman et al and Nisan et al.[3,4] Congestion games in particular have also been addressed, with focus on the Transmission Control Protocol (TCP)/Internet Protocol (IP), in several works.[5–8] 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. PacketEconomy comprises 2 aspects. Firstly, it allows the 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 nonintrusive way, allowing for gradual and opt-in participation, without affecting flows that do not participate in the PacketEconomy. This approach is possible because recently, router hardware has become more amenable to customization and programmability as evidenced by the adoption of software-defined networking, especially in newer networking technologies,[9,10] as well as by work to make routers able to execute custom and complex schedulers, for example, using Domino.[11]

In this work, we present a realistic implementation of PacketEconomy, within the OMNET++ discrete-event simulator,[12,13] using the INET network simulation library[14] and the experimental evaluation of the implementation. In Section 2 we present the related work and in Section 3 an overview of the theoretical model underpinning this implementation, described in more detail in the work of Efraimidis and Koutsiamanis.[15] In Section 4 we present the details of the implementation in OMNET++, used in the experiments. Afterwards, we describe the experimental setup in Section 5, and we present and discuss the experimental results in Section 6. We then describe the game-theoretic aspects of PacketEconomy in Section 7. Finally, we conclude with overall remarks in Section 8.

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 that 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 the work of Efraimidis and Koutsiamanis.[15]

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 a number of studies[16–22] with a good overview presented in the work of Srikant[23]; however, it is important that nondecentralized, typically computationally inefficient, and 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[24–26] 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 end points, 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 that remains and that 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,[27] 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 Internet Protocol version 6 (IPv6) flows, and uses an additional extension IPv6 header on each packet. We chose to make these assumptions and impose limitations on our implementation to be able 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, because of the abstractions performed that fail to take into account practical concerns.

We have also taken into account the fact that changes in network infrastructure are slow, especially where nonprogrammable 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, nonparticipating 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 traveling within those segments will take advantage of our solution within them. This contrasts with solutions that require a total switch to the alternative mechanism.

3 | THE PACKETECONOMY MODEL

This section provides an overview of the theoretical PacketEconomy model. A more detailed description of PacketEconomy can be found in the work of Efraimidis and Koutsiamanis.[15] PacketEconomy comprises 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, a profile of the game-theoretic model in which no player has anything to gain by changing only his or her own strategy unilaterally. In particular, the solution has the individual rationality property, i.e., the players have an incentive to participate in PacketEconomy. This means that players have nothing to lose and potentially something to gain by participating.

3.1 | The network model

We assume a network with router R and a set of N flows, as shown in Figure 1. The router R has a queue Q 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, and at the end of the round, it reaches its destination. Packets that arrive at the router are added to the end of the queue.

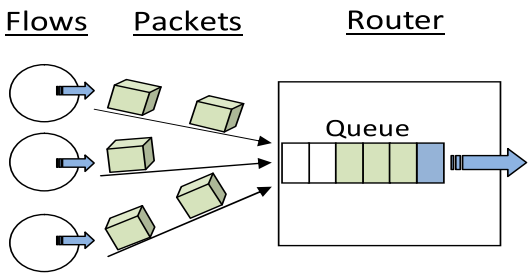


FIGURE 1 The network model illustrating the flows, their packets, the router, and the queue

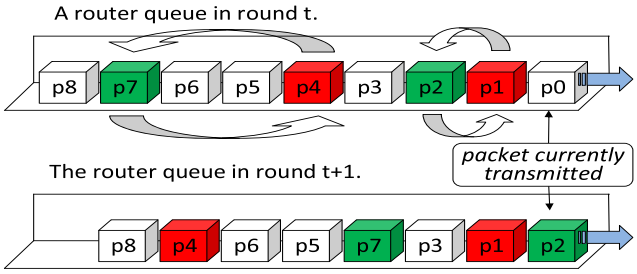


FIGURE 2 The state of a router queue in 2 successive rounds. In round t, 2 trades take place: one between the packet pair (p1,p2) and one between the pair (p4,p7)

3.2 | Packet trades

At the beginning of each round, position 0 of the queue has been freed, and thus, 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 0 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 P 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 2, provided that the negotiation performed between them leads to an agreement. The way the trades take place at a microeconomic level is that agents meet in random pairs and can make trades.

3.3 | Packet delay

The delay d of a packet p that starts at position k of the 0-based queue and does not make any trades is $k + 1$ rounds (Figure 3). If, however, the packet engages in trades and buys a total of r_b router queue positions and sells r_s positions, then its delay d_p, including the time to be served, becomes $d_p = k + 1 + r_s - r_b$ positions. 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 (Section 3.4) becomes 0, and the packet will not voluntarily accept such delays (i.e., it will not sell).

3.4 | Packet values

For each packet p, there is a decreasing function d_p that determines the value of p, given its delay d. The value function of each flow must be encoded onto each packet, and its value can be calculated anytime during the packet's journey via the $d_p(d)$ function. When calculating the value of a packet, the estimated service time in the current queue is used (see Section 4.5.4 for more details).

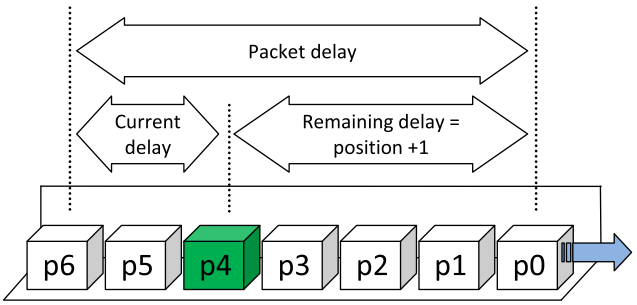


FIGURE 3 Components of packet delays in router queues

3.5 | Inventories and values

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 2 types of indivisible goods or resources: the packet delay $d_p(t)$ and the money account $m_p(t)$. Note that delay bears negative value, whereas money represents positive value.

We refer to the value of the packet delay as the *packet value* and to the value of the money account as the *packet budget*. Furthermore, we define the notion of the *packet benefit* as the sum of the value of a packet and its budget. Then we use the benefit concept to define the utility function of the packet (described in Section 4.1).

The inventory also contains the current position $pos_p(t)$ of the packet in the queue, if it is currently waiting in one. When the packet reaches its destination, the contents of the inventory of the packet are used to determine its utility.

3.6 | Trades

The objective of each packet is to maximize its utility. Thus, when 2 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. This value is the difference in benefit between the 2 packets if they perform the trade, and it results from the different utility functions of the packets and the different time frames in their flight time that the trade will affect. The "surplus value" is easily computed and involves no negotiation between the packets; each packet provides its trade price, and if the bid price is higher than the ask price, the trade takes place. In that case, both packets can benefit, i.e., increase their utility, if they come to an agreement. We assume a simple price agreement procedure whereby the final trading price will be the mean of the ask and bid prices. We also stress that both the bid and ask prices for each packet are determined via the same utility function, i.e., each packet has *one* utility function that is used both when selling and when buying positions.

4 | IMPLEMENTATION

In this section we describe the implementation of the PacketEconomy model on the basis of the OMNET++ discrete-event simulator.

4.1 | Packet utility functions

The theoretical model described in the work of Efraimidis and Koutsiamanis[15] 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 the following form for the packet utility functions with 3 parameters. The utility function is defined as

$$d_p(t) = \begin{cases} b - at^c & 0 \leqslant t < \sqrt[c]{\frac{b}{a}} \\ 0 & t \geqslant \sqrt[c]{\frac{b}{a}} \end{cases} \text{ with } b \geqslant 0 \text{ and } a, c > 0 \tag{1}$$

where t is the time that has passed since the packet has been sent and a, b, and c are the parameters that 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 0, at $t = t_0$, it decreases no more. The described utility function is monotonically decreasing over $t \geqslant 0$. 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 3 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 made. A few examples of the packet utility functions that are possible using this function form are presented in Figure 4.

4.2 | Compensation price

When 2 packets perform a trade, each one has to specify its bid or ask price, depending whether it is a buyer or seller. To be able 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 the work of Efraimidis and Koutsiamanis.[15] This work generalizes which kinds of utility functions can be used in the manner described by Equation (1). Consider a packet that, without participating in the trade, has delay $d_p(t_1)$ and a balance of $m_p(t_1)$, where $t_1 \leqslant t_0$ and $t_2 \leqslant t_0$. 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

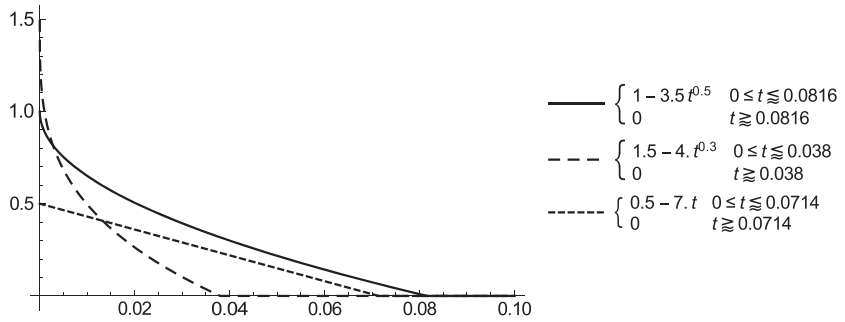


FIGURE 4 Example packet utility functions. The point where the functions meet the t axis is t_0

$$\begin{aligned} d_p(t_1) + m_p(t_1) = d_p(t_2) + m_p(t_2) \quad &\Longleftrightarrow \\ b - at_1^c + m_p(t_1) = b - at_2^c + m_p(t_1) + \rho \quad &\Longleftrightarrow \\ \rho = a\left(t_2^c - t_1^c\right). \end{aligned} \tag{2}$$

As described in the work of Efraimidis and Koutsiamanis,[15] window-based flows have to wait for an acknowledgment of receipt of a packet to be able to send another packet. Therefore, delaying a packet not only affects this packet's benefit but also the next one's that will also be additionally delayed. Taking this behavior into account, for window-based flows the total benefit *rate* needs to be the same or higher; thus, the compensation price is

$$\begin{aligned} \frac{d_p(t_1) + m_p(t_1)}{t_1} = \frac{d_p(t_2) + m_p(t_2)}{t_2} \quad &\Longleftrightarrow \\ \frac{b - at_1^c + m_p(t_1)}{t_1} = \frac{b - at_2^c + m_p(t_1) + \rho}{t_2} \quad &\Longleftrightarrow \\ \rho = \frac{b(t_2 - t_1) + a\left(t_1 t_2^c - t_1^c t_2\right) + m_p(t_1)(t_2 - t_1)}{t_1} \end{aligned} \tag{3}$$

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

4.3 | PacketEconomy as a service

PacketEconomy provides QoS as a service to the end point users. To do so, it firstly requires one or more intermediate routers that are able to perform trades. These routers do not need any additional configuration from the end points, 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 end points, besides the necessary modules, 2 parameters must be given, presented in Figure 5. 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 Internet Service Provider. Optionally, PacketEconomy can use feedback from previous traffic as well as the given priority and budget to be able to deduce the appropriate utility function parameters.

4.4 | Operation overview

The overall PacketEconomy operation is presented in Figure 6. At the sending end point A, a user decides upon the high-level QoS requirements for flows, which can be predefined, on the basis of 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 end point 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 end point. 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 end point. It also records relevant network and utility statistics. When the receiving end point B has to send a packet to the original sending end point A, it attaches a PacketEconomy extension header with feedback. This is eventually received at the original sending end point 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.

4.4.1 | Adaptivity

Each end point needs to track both its own, as well as the other, end point's budget and network performance. Specifically, each end point needs to track the total packet benefit from received packets, which constitutes a form of return-on-investment information, and attach this information

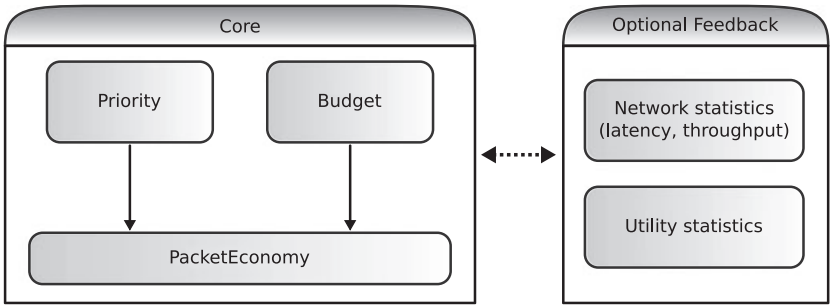


FIGURE 5 Viewed as a service, PacketEconomy requires priority and available budget as inputs. Optionally, network and utility statistics feedback can be used to deduce utility function parameters

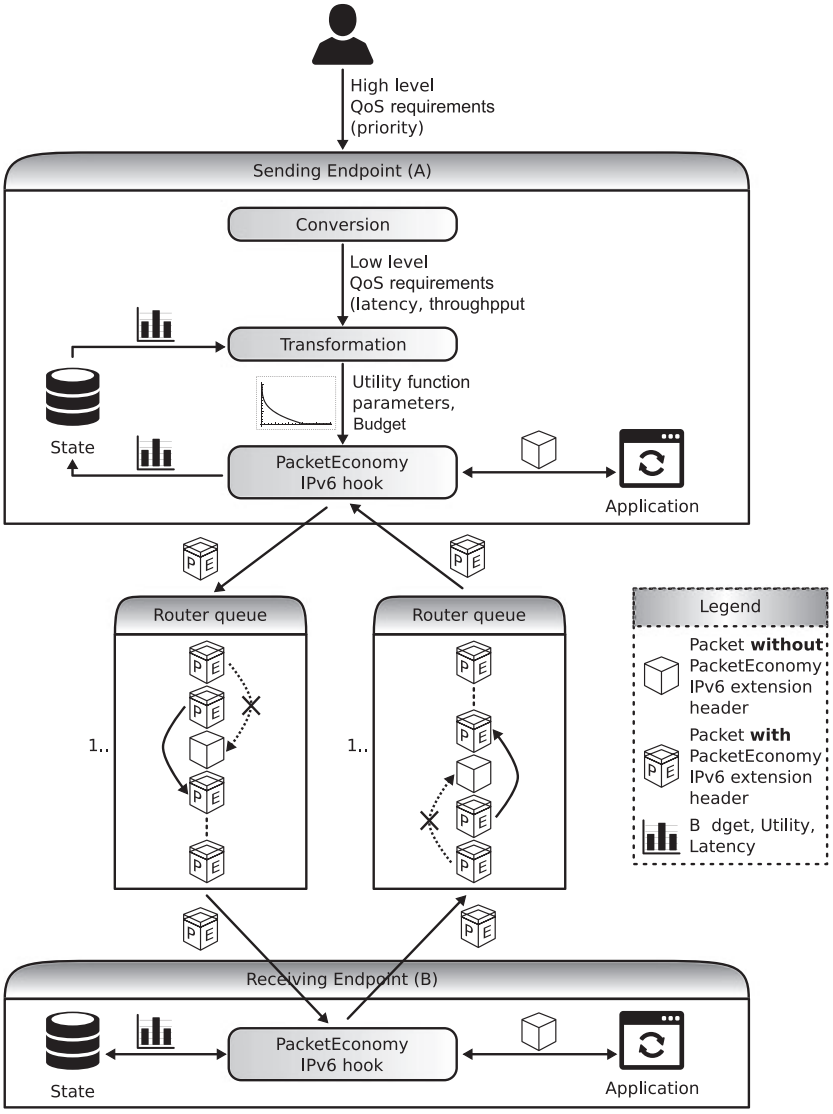


FIGURE 6 Overview of the operation of PacketEconomy. The PacketEconomy hook attaches and detaches the custom extension header at the end points. State is maintained to be used in deciding which utility function parameters and budget value to use. Routers perform trades statelessly, ignoring non-PacketEconomy pairs. Feedback is sent from the receiving end point B to the original sending end point A to inform its parameter selection

when sending packets to the other end point. The feedback from the opposite end point, along with information regarding the available budget and the flow priorities, allows each end point to adapt to changing network conditions. When network congestion increases, depending on the number of flows and their priorities, an end point 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 to be able to support their QoS needs.

4.5 | 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 online 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 that is specific to OMNET++ has been implemented in the form of 2 OMNET++ modules, extending preexisting INET modules. The first one extends the standard `IPv6` stack module, to be able to read, write, and process the PacketEconomy IPv6 headers on incoming and outgoing packets. This module is only used on end point nodes. The second extended module is an alternative queue that 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` that extends `inet.nodes.ipv6.StandardHost6`.

4.5.1 | Extension header description

The PacketEconomy extension header contains the fields that 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.

4.5.2 | The TCP/IP stack at end points

At the end points 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 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 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.

4.5.3 | 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 are 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 $i \in \{0, 1, \dots, |Q| - 1\}$ 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 $p_{tr} = (p_1, p_2, \dots, p_{|Q|-1})$ 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 neighboring and non-overlapping trading pairs: $T_i = (p'_{2i}, p'_{2i+1})$, where $i \in \{1, 2, \dots, \lfloor (Q-1)/2 \rfloor\}$. The trading negotiation and exchange is performed as described in the work of Efraimidis and Koutsiamanis.[15] However, in the course of taking PacketEconomy from a theoretical model to a real network implementation, some issues appeared that 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-First-In-First-Out (FIFO) manner. This is implemented efficiently using the HL-Hitters mechanism described in the work of Koutsiamanis

and Efraimidis.[28] Secondly, in contrast to another study,[15] 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 7.2).

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 that 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 that 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 because of the network practices service providers typically implement, such as high contention ratio.

4.5.4 | Time source considerations

PacketEconomy uses time measurements to be able to decide the compensation prices of packets during trading sessions within routers and when packets are received at end points. This is indicated by the utility function d_p in Equation (1) being a function of time (t).

An accurate time source in both end points 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 that has passed since the packet was sent from its sending end point. In the OMNET++ implementation, we have used the global time source provided by the simulator.

In real networks a global time source is not possible, but network nodes are often synchronized to within a few milliseconds using protocols such as the Network Time Protocol. Under such conditions, PacketEconomy would be able to operate since the time to reach the end point is used, which is typically much larger.

If even this level of clock synchronization is 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 2, 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 (eg, by using an Internet Control Message Protocol 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.

5 | EXPERIMENTAL SETUP

In this section the setup for the experiments is described including which parameters are used and how they are combined. For the experiments we have selected a representative set of queue admission policies (DropTail and Random Early Detection [RED][29]), priority policies (PacketEconomy and Deficit Round Robin [DRR],[30] and Strict Priority [SP]), flow types (TCP and User Datagram Protocol [UDP]), and other characteristics. We now present the used policies in brief. The DropTail admission policy, also known as First Come First Serve with a maximum size, admits packets in the order they arrive and drops packets if the queue occupancy reaches the maximum size.

The RED admission policy works by maintaining the current average queue occupancy. It uses a minimum and a maximum occupancy threshold. When the queue occupancy is within these thresholds, an arriving packet is dropped with increasing probability. If the maximum threshold is exceeded, then the packet is always dropped.

The DRR scheduling policy is an efficient (O(1)) approximation of Weighted Fair Queuing that in turn is a generalized version of fair queueing allowing for different shares of throughput to be given to different flows. It operates by using different queues for each separate flow class and serves packets for each class' queue in a round robin manner. Weights are used to allow some classes to send more or less packets than other classes.

The SP scheduling policy also classifies packets into classes like DRR but always serves the packet of the highest priority class available.

5.1 | Non-QoS configuration

We first describe the non–QoS-specific aspects of the experiments, leaving the QoS-specific ones, such as PacketEconomy, the DRR, and the SP configurations, for the end of the section.

5.1.1 | All cases

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

Layer 2 setup: The network consists of a dumbbell topology 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 end points, and the hosts on the right are the receiving end points. Each host is connected via Ethernet with exactly one link to either $R1$ or $R2$. The connections between end points and routers, as well as the single connection between the 2 routers, are full duplex 10 Mbps with 50 nanoseconds propagation delay and 0% packet error rate. 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 seconds (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 Constant Bit Rate 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:
  a. Maximum Size ($|Q|$): 100 packets
  b. Admission policy: DropTail, RED ($w_q = 0.002$, $min_{th} = 10$, $max_{th} = 100$, $max_p = 0.02$).

5.1.2 | Flow composition cases

Additionally, we examine 3 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 the number of flows $N_{TCP} = N$ is 10, and all the flows use the same TCP congestion avoidance algorithm (Reno), with increased initial window support enabled, TCP window scaling support enabled, TCP delayed acknowledgements (ACKs) support enabled, selective acknowledgements (SACK) support disabled, the TCP Nagle algorithm disabled, and an advertized window of 300 000 bytes.

Since TCP flows are typically more complex than UDP flows, we have examined performance in more detail especially for the TCP-only flows case. Thus, each possible combination of the following parameters is examined, in 5 repetitions as mentioned above:

- TCP Maximum Segment Size (MSS): 1400, 512, 200 bytes
- Packet error rate (packet loss due to transmission, set on the channel between the 2 routers [$R1$ and $R2$]): 0%, 1%, and 2%
- Additional delay (set on the channel between the 2 routers [$R1$ and $R2$], used to modify the round trip time of the flows): 50 nanoseconds and 1, 20, 30 milliseconds.

The UDP-only flows case: In each experiment all the flows are of Constant Bit Rate 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 subcases are created: one where the cumulative bandwidth requirements marginally pass the bottleneck router bandwidth, namely, 10 Mbps, 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 2 subcases 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 2 subcases for the cumulative bandwidth requirements are reduced to one where the bandwidth requirements are 50% of the bottleneck router bandwidth, i.e., 5 Mbps.

5.2 | QoS configuration

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

5.2.1 | 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.

5.2.2 | 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 PacketEconomy (with 1 trading round per served packet), DRR in the TCP-only cases (independent levels used $= \{N, \lceil N/2 \rceil, \lceil N/4 \rceil\}$), or SP 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 that gives absolute priority to UDP flows. A classifier is present before the DRR and SP queues that 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.

5.2.3 | 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.

5.2.4 | Flow priority

When assessing performance with a priority policy (PacketEconomy, DRR, and 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, and 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)), b (C_b, Equation (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., it is not affected by a priority policy. This d_{bl} value can be continuously updated by the network stack of each end point 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 that 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 2 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 on the basis of common knowledge guidelines, for example, all VoIP flows should get high 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 to be able 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 (4))

$$Y_{t_0}(p, d_{bl}, s_l, s_u) = d_{bl}s_l + (1-p)(d_{bl}s_u - d_{bl}s_l), \text{with } p \in [0,1], s_l, s_u \geqslant 0, d_{bl} > 0 \quad (4)$$

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 (5))

$$C_b(p) = 1 + (10p)^2, \text{with } p \in [0,1] \quad (5)$$

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))

$$C_a(p, d_{bl}, s_l, s_u, c) = C_b(p)/(Y_{t_0}(p, d_{bl}, s_l, s_u)^c), \text{ with } p \geqslant 0 \quad (6)$$

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

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

5.3 | Collected measurements

For each combination we collect statistics covering both the network-level and the game-theoretic behavior of the flows. The network-level statistics are the typical ones used to characterize flow behavior and are commonly used in other works as well[29,30]:

- Throughput per priority level, a network-level measurement, which is often of high importance in measuring TCP flow performance.
- End-to-end delay (or latency) per priority level, a network-level measurement, which is often used in delay-sensitive UDP flows to gauge performance.
- Packet drop per priority level, a network-level measurement, which is an indicator of both scheduling quality and network efficiency, in wasted effort spent on eventually dropped packets.
- Utility (packet value d_p, balance m_p, and total $d_p + m_p$), a game-theoretic measurement, which is used to gauge whether the game-theoretic mechanism is incentivizing flows to participate in PacketEconomy (see Section 7.1 for more details on this issue).

A total of 11 610 experiments have been executed, as 2322 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. Because of space considerations, we have selected to present the above described representative subset of cases.

5.4 | 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 nonintegral), as described in Section 5.2.2. 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).

6 | EXPERIMENTAL RESULTS

In this section the results of the experiments are presented, organized on a flow composition case basis. For each flow composition case (TCP-only, UDP-only, and TCP and 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 (decrease) in priority leads to an increase (decrease) of throughput and to a decrease (increase) in end-to-end delay.

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.

6.1 | The TCP-only flows case

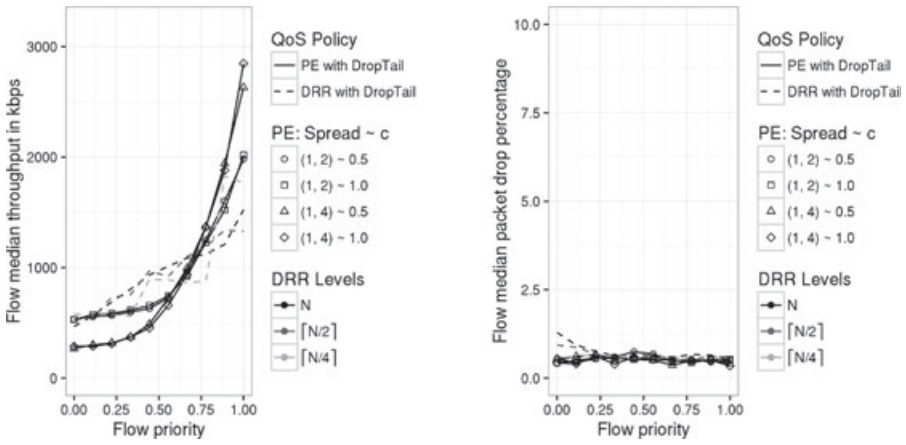
In this case we are concerned with flow throughput and packet drops.

6.1.1 | Base case

We will initially present the results of the base case where the TCP flows have a 1400 byte MSS and the channel between $R1$ and $R2$ has 0% packet error rate and 50 nanoseconds delay.

For the case where a DropTail bottleneck router queue is used, the results for throughput are presented in Figure 7A and for packet drop percentage in Figure 7B. PacketEconomy has similar performance to DRR as far as packet drop is concerned. In the case of throughput, PacketEconomy displays a nonlinear, but smooth distribution, while DRR with N and $\lceil N/2 \rceil$ levels is largely linear, however 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 and 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 and for packet drop percentage show that the use of the RED admission policy makes the distribution of throughput with PacketEconomy more linear, because of limiting higher priority flows from getting a higher proportion of throughput.



(A) Median throughput of flows in kbps. (B) Median percentage of dropped packets.

FIGURE 7 TCP-only flows case results per priority level with a DropTail bottleneck router queue, MSS of 1400 bytes, 0% packet error rate, and 50-ns delay between $R1$ and $R2$. Throughput increases with priority, as expected, and 2 spread-c combinations distribute throughput more aggressively than the other 2. Packet drop is low ($< 1\%$) and approximately the same for all priority levels. DRR indicates Deficit Round Robin; MSS, Maximum Segment Size; QoS, quality of service; TCP, Transmission Control Protocol

Deficit Round Robin is not affected significantly by RED, and the comments on DRR's behavior under DropTail hold for RED as well. Packet drop percentage with RED is, as expected, higher for both PacketEconomy and DRR, but $< 2\%$, which perform almost identically in this respect and are approximately the same for all priority levels.

6.1.2 | Effects of MSS and $R1$and $R2$channel packet error rate and delay

Changing the TCP MSS between the values examined (1400, 512, and 200 bytes) had very little effect on both throughput and dropped packet percentage, irrespective of $R1$ and $R2$ channel packet error rate and delay, with a smaller MSS leading to slightly less throughput because of increased TCP header overheads.

On the other hand, throughput was affected by both $R1$ and $R2$ channel packet error rate and delay. Both PacketEconomy and SP were affected identically; thus, the effects are due to the TCP flow control and not PacketEconomy. With 0% packet error rate, the throughput was not significantly affected for delay $\leqslant$10 milliseconds, slightly affected by compressing the distribution for delay $= 20$ milliseconds and failing to distribute the throughput for delay $\geqslant$30 milliseconds. With 1% packet error rate, the previous thresholds became $\leqslant$1, $= 10$, and $\geqslant$20 milliseconds correspondingly. Finally, with 2% packet error rate, the distribution was compressed slightly for delay $\leqslant$1 millisecond, and distribution failure was present for delay $\geqslant$10 milliseconds. In general, when the packet error rate is reasonable, typically $< 2\%$ and the additional channel delay not too high, typically $\leqslant$30% of unmodified one-way trip time, both PacketEconomy and SP continue to be able to distribute throughput effectively.

6.2 | The UDP-only flows case

In this case we are concerned with flow end-to-end delay and packet drops. Because of the large number of UDP flows used (68-194 with 100% bandwidth requirements and 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.

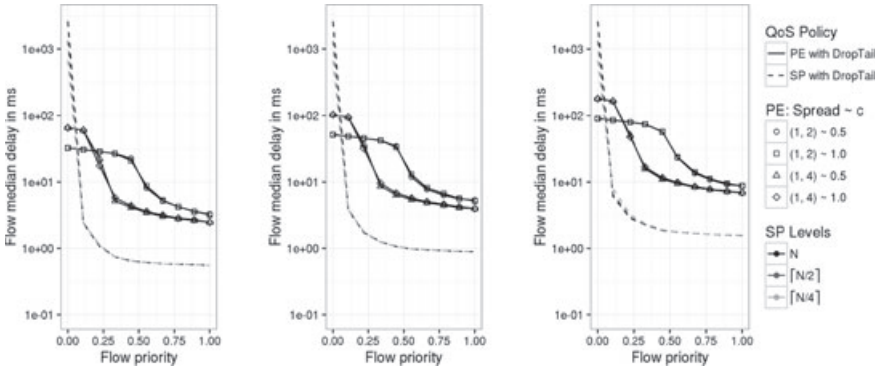
For the case where a DropTail bottleneck router queue is used, the results for end-to-end delay are presented in Figure 8. PacketEconomy has similar performance to SP as far as packet drop is concerned: In both cases the packet drop percentage is approximately the same for all priority levels and decreases as the number of flows decreases and as the size of the payload increases ($\simeq$6.25% [120 bytes, 194 flows], $\simeq$4% [240 bytes, 120 flows], and $\simeq$2.5% [480 bytes, 68 flows]).

In the case of end-to-end delay, both PacketEconomy and SP display a nonlinear 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 do 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 almost identical to the DropTail queue case shown in Figure 8. Packet drop percentage accordingly mirrors the DropTail case.

End-to-end delay for both DropTail and RED bottleneck router queues with 150% bandwidth requirements is also the same as in the case with 100% 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 and PacketEconomy has similar performance to SP. More specifically, with flows requesting 150% of the



(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 8 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 2 spread-c combinations distribute delay more aggressively than the other 2. Note: the y-axis is logarithmic. SP indicates Strict Priority; QoS, quality of service; UDP, User Datagram Protocol

available bandwidth, the packet drop percentage is approximately 38% and approximately the same for all priority levels and payload sizes, albeit showing slightly higher variability with RED versus DropTail.

6.3 | The TCP and 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, because of the large number of UDP flows used (34-77 with 50% bandwidth requirements), we only present 10 representative priority levels in the figures of this section to preserve legibility.

6.3.1 | TCP flow measurements

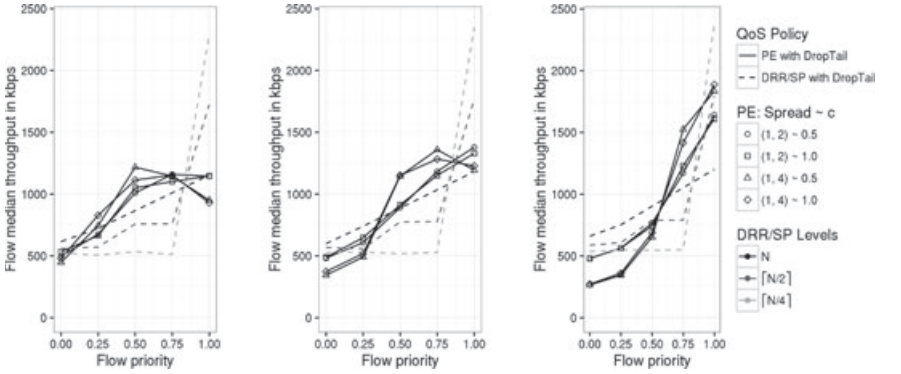
For the case where a DropTail bottleneck router queue is used, the results for TCP throughput are presented in Figure 9. In this case, PacketEconomy displays a nonlinear but smooth distribution while DRR with N levels is largely linear; however, DRR with $\lceil N/2 \rceil$ or $\lceil N/4 \rceil$ levels loses this property.

Packet drop is approximately the same for all priority levels for both PacketEconomy and DRR/SP. For PacketEconomy, it is between 0.5% and 2%, decreasing as the number of UDP flows decreases and as the size of the UDP payload increases. For DRR/SP, packet drop percentage is very low, approximately 0.07%. When using the RED admission policy instead of DropTail, the results are very similar to the DropTail case with different priority levels leading to smaller differences in throughput and slightly larger variability in packet drop percentages.

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 and 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.

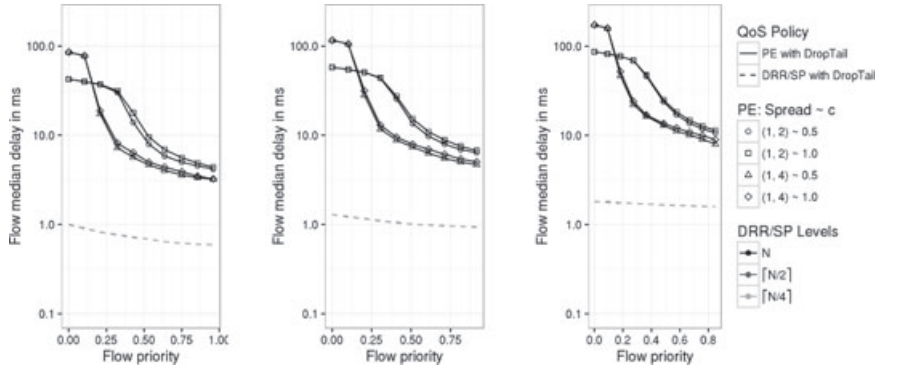
6.3.2 | UDP flow measurements

For the case where a DropTail bottleneck router queue is used, the results for UDP end-to-end delay are presented in Figure 10. In this case, PacketEconomy distributes end-to-end



(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 9 TCP and UDP flows case results for median TCP throughput per priority level with a DropTail bottleneck router queue. Throughput increases with priority, as expected, but 2 spread-c combinations distribute throughput less aggressively at high priority values than the other 2. DRR indicates Deficit Round Robin; QoS, quality of service; SP, Strict Priority; TCP, Transmission Control Protocol; UDP, User Datagram Protocol



(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 10 TCP and 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 2 spread-c combinations distribute delay more aggressively than the other 2. Note: the y-axis is logarithmic. DRR indicates Deficit Round Robin; QoS, quality of service; SP, Strict Priority; TCP, Transmission Control Protocol; UDP, User Datagram Protocol

delay smoothly with higher priority flows being delayed less than lower priority flows, but nonlinearly. Deficit Round Robin distributes delay more linearly, and the number of levels does not seem to affect the resulting delay. Packet drop is approximately the same for all priority levels for both PacketEconomy and DRR/SP. For PacketEconomy, it is between 0.8% and 2.5%, decreasing as the number of UDP flows decreases and as the size of the UDP payload increases. For DRR/SP, packet drop percentage is very low, approximately 0.03%. The UDP end-to-end delay for the RED bottleneck router queue is similar to the DropTail counterpart shown in Figure 10. However, the packet drop percentage for the RED bottleneck router queue is approximately 2.5% and does not significantly change with UDP payload sizes.

7 | GAME-THEORETIC ASPECTS

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

7.1 | Incentive to participate

The first issue is whether flows have an incentive to participate in PacketEconomy, i.e., the property of individual rationality, something that has been also investigated in the work of Efraimidis and Koutsiamanis.[15] According to that previous work, because of trades being Pareto improvements on each trading packet's benefit, we concluded that there is a Nash equilibrium in which all flows participate in the scheme.

The incentive to participate is important since flows are free to choose not to participate in PacketEconomy. Flows having an incentive to participate in PacketEconomy leads rational flows (or typically the users that configure them) to want to participate in PacketEconomy of their own free will since they have advantages to gain from their participation. In the following presentation of results, we use the percentage of total benefit gained when participating versus not participating as a metric of whether flows have an incentive to use PacketEconomy. When this value is $\geqslant 100\%$ for a flow, then it gains more benefit, i.e., better user experience, when it participates in PacketEconomy than when it does not, thus it should rationally choose to use PacketEconomy, driving its adoption.

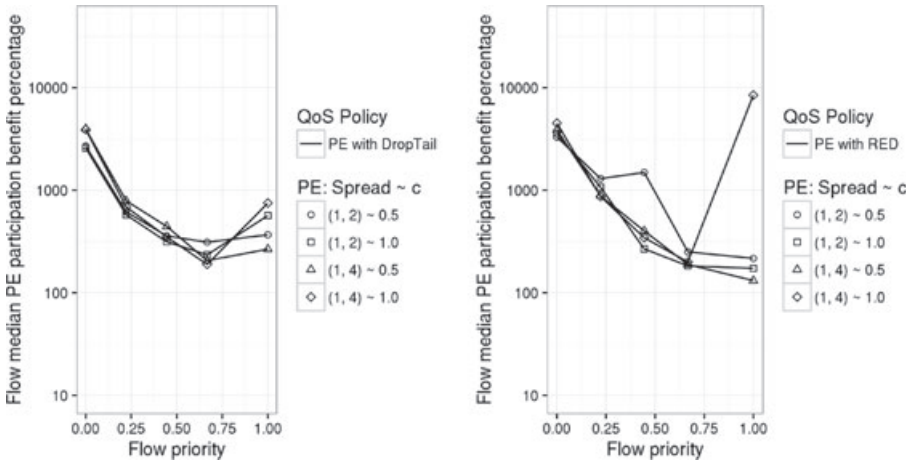
The following results illustrate that individual rationality is present in the real network implementation of PacketEconomy as well, with the caveat that in practice only a subset of flows have been tested for this property.

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. Because of the large number of flows used, and because 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.

7.1.1 | The TCP-only flows case

The incentive to participate in the TCP-only flows case is presented in Figure 11, and as it can be seen, all examined flows have a strong incentive to participate in both DropTail and RED bottleneck router queue cases.

Effects of MSS and $R1$ and $R2$ channel packet error rate and delay The MSS has no bearing on the incentive to participate. Also, when the packet error rate is 0%, incentive is maintained for all examined values of additional delay with both DropTail and RED admission policies. On the other hand, with DropTail, once the packet error rate becomes 1%, the incentive is lost for delay $\geqslant 20$ milliseconds, while with 2% packet error rate it is lost for delay $\geqslant 1$ millisecond. The incentive to participate is even lower with the RED admission policy.



(A) DropTail bottleneck router queue. (B) RED bottleneck router queue.

FIGURE 11 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. Qos indicates quality of service; TCP, Transmission Control Protocol; RED, Random Early Detection

Effectively, since the flow utility functions model delay and not packet loss, PacketEconomy miscalculates the expected utility of packet delivery under too much packet loss. We expect that this issue can be solved by adding a parameter to represent this variable into the utility function, however, we consider this to be future work.

7.1.2 | The UDP-only flows case

The incentive to participate in the UDP-only flows case is similar to the TCP-only flows case in that it can be seen that 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 is almost identical to the DropTail queue case.

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 and is also always over 100%.

7.1.3 | The TCP and 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 case and is almost always over 100% providing an incentive to participate in PacketEconomy. In some rare cases the incentive is slightly lost. We have found this to happen occasionally with RED queues and higher priority TCP flows.

One example case is for the highest priority TCP flows, when using RED, having the UDP payload is 240 bytes and 60 UDP flows and when the (s_l, s_u) spread parameter is (1, 2).

We have concluded that this happens because of the larger error in estimation of delivery time with larger size high priority packets. We expect that because of 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.

7.2 | Packet size variability

In our previous work,[15] 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 of the 2 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 favored.

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 parallelization. Therefore, a compromise was sought in which we disallowed any trades between larger buyers and smaller sellers. Although this approach somewhat decreased 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 higher-level multi-queue scheduler.

7.3 | 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 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 that 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 2 quantities with an inverse relation: Increasing the packet value

incurs higher budgetary costs, and increasing the accumulated budget negatively impacts 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 to be able to gain added benefit. 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.

7.4 | Price of anarchy/stability

A notion that is also interesting to investigate in this context is the Price of Anarchy[1,2] and its related Price of Stability.[31,32] 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 Price of Anarchy and Price of Stability 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 nontrivial. For example, fixed size router queues, probabilistic admission policies (eg, RED), adaptive flows that are affected by feedback (eg, TCP), and the interaction between flow types and packet sizes (eg, 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 (eg, 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 that, 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 it 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.

7.5 | Relation to smart market

In the seminal work of MacKie-Mason,[33] they propose a generalized Vickrey auction to be able 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.

8 | CONCLUSIONS AND FUTURE WORK

In this work we presented a realistic implementation of PacketEconomy, a distributed 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 DRR and SP 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 end points to auto-configure utility function parameters. We also envision PacketEconomy's applicability in alternate contexts, such as being used in 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 in the work of MacKie-Mason[33]) can be used instead. An additional alternate context concerns Internet of Things networks. Since PacketEconomy uses a notion of utility, which encodes a time-varying quantity, it may be useful for Internet of Things networks wherein communication comes at a premium in both terms of energy and computational complexity. By 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 that is also supported by the experimental results.

REFERENCES

1. Koutsoupias E, Papadimitriou Ch. Worst-case equilibria. *STACS '99*, Trier, Germany; 1999:404–413.
2. Papadimitriou Ch. Algorithms, games, and the Internet. *ACM STOC '01*, Heraklion, Crete, Greece; 2001:749–753.
3. Altman E, Boulogne T, El-Azouzi R, Jiménez T, Wynter L. A survey on networking games in telecommunications. *Comput Oper Res*. 2006;33(2): 286–311.
4. Nisan N, Roughgarden T, Tardos E, Vazirani VV. *Algorithmic Game Theory*. New York: Cambridge University Press; 2007.
5. Akella A, Seshan S, Karp R, Shenker S, Papadimitriou Ch. Selfish behavior and stability of the Internet: a game-theoretic analysis of TCP. *SIGCOMM '02*, Pittsburgh, Pennsylvania, United States of America; 2002:117–130.
6. Gao X, Jain K, Schulman LJ. Fair and efficient router congestion control. *SODA '04*, New Orleans, Louisiana, United States of America; 2004:1050–1059.
7. Efraimidis PS, Tsavlidis L, Mertzios GB. Window-games between TCP flows. *TCS*. 2010;411(31-33):2798–2817.
8. Shenker S. Making greed work in networks: a game-theoretic analysis of switch service disciplines. *SIGCOMM '94*. ACM Press, London, United Kingdom; 1994:47–57.
9. Yang M, Li Y, Li B, Jin D, Chen S. Service-oriented 5G network architecture: an end-to-end software defining approach. *Int J Commun Syst*. 2016;29(10): 1645–1657. DAC.2941.
10. Chen J-L, Kao S-P. QoS mechanism for virtualized wireless networks with software-defined networking. *Int J Commun Syst*. 2015;28(11):1741–1752. IJCS-13-0693.R1.
11. Sivaraman A, Cheung A, Budiu M, Kim C, Alizadeh M, Balakrishnan H, Varghese G, McKeown N, Licking S. Packet transactions: high-level programming for line-rate switches. *Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference*, SIGCOMM '16, Florianopolis, Brazil; 2016:15–28.
12. Varga A. Omnet++. *Modeling and tools for network simulation*. Berlin Heidelberg: Springer; 2010:35–59.
13. Varga A. The OMNET++ discrete event simulator. http://omnetpp.org. (Accessed February 1, 2016); 2016.
14. Varga A et al. The INET framework; 2016. Available at http://inet.omnetpp.org. (Accessed February 1, 2016)
15. Efraimidis PS, Koutsiamanis R-A. Money-based coordination of network packets. In: Spirakis PG, Serna M, eds. *Algorithms and Complexity*, LNCS, vol. 7878. Berlin Heidelberg: Springer; 2013:197–208.
16. Karp R, Koutsoupias E, Papadimitriou Ch, Shenker S. Optimization problems in congestion control. *FOCS '00*. IEEE Computer Society, Redondo Beach, California, United States of America; 2000:66–74.
17. Kelly F. Charging and rate control for elastic traffic. *European Trans. Telecommunications*. 1997;8(1):33–37.
18. Kelly FP, Maulloo AK, Tan DKH. Rate control for communication networks: shadow prices, proportional fairness and stability. *J Oper Res Soc*. 1998;49(3):237–252.
19. Low SH, Lapsley DE. Optimization flow controli: basic algorithm and convergence. *IEEE/ACM Trans Netw (TON)*. 1999;7(6):861–874.
20. Yaïche H, Mazumdar RR, Rosenberg C. A game theoretic framework for bandwidth allocation and pricing in broadband networks. *IEEE/ACM Trans Netw (TON)*. 2000;8(5):667–678.
21. Kunniyur S, Srikant R. A time scale decomposition approach to adaptive ECN marking. *INFOCOM 2001*, vol. 3, IEEE, Anchorage, Alaska, United States of America; 2001:1330–1339.
22. Li S, Sun W, Hua C. Optimal resource allocation for heterogeneous traffic in multipath networks. *Int J Commun Syst*. 2016;29(1):84–98.
23. Srikant R. *The Mathematics of Internet Congestion Control*. New York: Springer Science & Business Media; 2012.
24. Basar T, Srikant R. Revenue-maximizing pricing and capacity expansion in a many-users regime. *INFOCOM 2002*, vol. 1, IEEE, New York, United States of America; 2002:294–301.
25. Acemoglu D, Ozdaglar A. Flow control, routing, and performance from service provider viewpoint. LIDS Report; 2004.
26. Ma K, Han Q, Chen C, Guan X. Bandwidth allocation for cooperative relay networks based on nash bargaining solution. *Int J Commun Syst*. 2012;25(8): 1044–1058.
27. Cole R, Dodis Y, Roughgarden T. How much can taxes help selfish routing? *EC '03*. ACM, San Diego, California, United States of America; 2003:98–107.
28. Koutsiamanis R-A, Efraimidis PS. A heaviest hitters limiting mechanism with o (1) time complexity for sliding-window data streams. *J Internet Technol*. 2013;14(1):117–126.
29. Floyd S, Jacobson V. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans Netw*. August 1993;1(4):397–413.
30. Shreedhar M, Varghese G. Efficient fair queuing using Deficit Round-Robin. *IEEE/ACM Trans Netw*. 1996;4(3):375–385.
31. Schulz AS, Moses NS. On the performance of user equilibria in traffic networks. *14th SODA*, Baltimore, Maryland, United States of America; 2003:86–87.
32. Anshelevich E, Dasgupta A, Kleinberg J, Tardos E, Wexler T, Roughgarden T. The price of stability for network design with fair cost allocation. *45th FOCS*, IEEE, Rome, Italy; 2004:295–304.
33. MacKie-Mason JK. A smart market for resource reservation in a multiple quality of service information network. SSRN Working Paper Series; 2007.

How to cite this article: Koutsiamanis R-A, Efraimidis PS. Implementing PacketEconomy: Distributed money-based QoS in OMNET++. *Int J Commun Syst*. 2017;30:e3252. https://doi.org/10.1002/dac.3252