Market Driven Queueing

Boris Pismenny
Market Driven Queueing

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

Boris Pismenny

Submitted to the Senate
of the Technion — Israel Institute of Technology
Tamuz 5772 Haifa July 2017
The Research Thesis Was Done Under the Supervision of Assistant Professor Orna Agmon Ben-Yehuda and Prof. Assaf Schuster, in the Faculty of Computer Science.

The Generous Financial Help Of The Technion, Mellanox Technologies and the PAZI fund Is Gratefully Acknowledged.
8 Conclusion and open questions 57

A Ginseng for Bandwidth 59
A.1 System Overview .................................................. 59
   A.1.1 Host Side Design ........................................... 59
   A.1.2 Ginseng Auction Flow .................................... 60
   A.1.3 Bandwidth Controllers ................................. 61

Hebrew Abstract i
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Example of an allocation that maximizes the social welfare and its effect on work-conservation</td>
<td>15</td>
</tr>
<tr>
<td>2.1</td>
<td>Demand function and pseudo-demand function according to the multi-bids. The shaded area is the truncated pseudo-demand for those multi-bids and $Q_i$ queued bytes.</td>
<td>20</td>
</tr>
<tr>
<td>2.2</td>
<td>Aggregate pseudo-demand function according to the multi-bids. The shaded area is the aggregate truncated pseudo-demand function. On the horizontal axis: the truncated pseudo-market clearing price $\tilde{u}$ for the available capacity $Q_{round}$.</td>
<td>21</td>
</tr>
<tr>
<td>3.1</td>
<td>Overview of MDQ</td>
<td>25</td>
</tr>
<tr>
<td>4.1</td>
<td>Each block represents a packet that can be transmitted by the router only when the entire packet is received. The figure shows a pattern of packet arrivals where the payment for client $A$ is negative. In the absence of $A$ less data of client $C$ is transmitted during the round, resulting in a negative payment.</td>
<td>34</td>
</tr>
<tr>
<td>6.1</td>
<td>First scenario</td>
<td>48</td>
</tr>
<tr>
<td>6.2</td>
<td>Second scenario</td>
<td>49</td>
</tr>
<tr>
<td>6.3</td>
<td>Using the valuations in Figure 6.2a. The graph presents the social welfare obtained by one backlogged high demand client, one medium demand client who is backlogged half the time, and an increasing number of inactive low demand clients. MDQ improves the social welfare infinitely over combinations of the multi-bid auction with queueing algorithms that allocate all available bandwidth to the low demand clients.</td>
<td>50</td>
</tr>
<tr>
<td>6.4</td>
<td>The trade-off between the CPU run time of the simulation and the loss of social welfare from changes in the $MaxBatchSize$. As the CPU run time of the simulation increases, the social welfare lost is less significant. The optimal $MaxBatchSize$ is obtained at 1100 bytes. Starting from a batch size of 4800 bytes, even a round of MDQ running on the Python simulator executes in less than one second.</td>
<td>51</td>
</tr>
</tbody>
</table>
6.5 A linear regression of the run-time and an inverse of the $MaxBatchSize$. The result for a sample size of 20 has a Pearson product-moment correlation coefficient of 0.58.

6.6 Run-time and inverse $MaxBatchSize$

A.1 Overview of the Ginseng system architecture. Ginseng is a process on the hypervisor machine that communicates with client VMs over a dedicated channel. Using this channel clients submit multi-bids to the Ginseng process. The Ginseng process allocates bandwidth to all clients in accordance with these bids.

A.2 System with 1 NIC and 2 NUMA nodes. IO from NUMA node 2 has reduced performance.
List of Tables

1.1 Summary and comparison between queueing algorithms and bandwidth pricing mechanisms ........................................ 10

A.1 Average bandwidth (Gbps) from running netperf between 2 machines connected back-to-back. The bandwidth changes according to the NUMA node on which netperf is running. Running the receiver on the NUMA node connected to the NIC improves throughput by up to 35% ........ 62
Abstract

Network providers must dynamically allocate scarce physical resources among their clients to maximize benefit. Network pricing is one way for providers to maximize client benefit by allowing them to share available bandwidth according to their willingness to pay for it. The resulting allocation grants additional bandwidth to those who need it the most, while decreasing the bandwidth of those who need it the least. Existing queueing algorithms use the results of pricing schemes as weights for sharing bandwidth, which can change only in response to a change in client willingness to pay. However, network congestion, jitter and failures affecting a flow create excess bandwidth that could be used by another flow. Queueing algorithms that can share the excess bandwidth are called work-conserving. Network pricing schemes traditionally ignore work conservation, by assuming that all clients are constantly backlogged.

In this paper, we design and evaluate the Market Driven Queueing (MDQ) algorithm. By combining a queueing algorithm with a bandwidth pricing mechanism, MDQ provides the benefits of both. As a work-conserving algorithm, MDQ maximizes client benefit while improving utilization. Moreover, it requires only $O(\log(n))$ processing time per packet for traffic scheduling, where $n$ is the number of active flows. We analyze the properties of MDQ and evaluate it using simulation. Our simulation results show that MDQ improves clients’ aggregate benefit by up to 4x compared to state-of-the-art combinations of pricing and queueing algorithms. MDQ is also applicable to other scheduling problems such as distributed queues or I/O queue scheduling.
Abbreviations and Notations

\( \lambda \) : The duration of a round in seconds.

\( Q_i \) : The number of bytes in the queue of client \( i \).

\( Q \) : A sequence of all client queues (\( Q_1, Q_2, \ldots, Q_n \)).

\( Q_{\text{round}} \) : Number of bytes offered for auction each round.

\( M_i \) : Number of bids submitted by client \( i \).

\( p_{\text{sec}}^j \) : The unit price that is part of bid number \( j \) of client \( i \) for quantity \( q_{\text{sec}}^j \), measured in \( \frac{\$}{\text{sec}} \).

\( q_{\text{sec}}^j \) : The bandwidth that is part of bid number \( j \) of client \( i \), measured in \( \frac{\text{bytes}}{\text{sec}} \).

\( p_i^j \) : The unit price per round for bid number \( j \) of client \( i \) for quantity \( q_i^j \), measured in \( \$ \).

\( q_i^j \) : The quantity of bytes per round for bid number \( j \) of client \( i \).

multi-bid : A sequence of \( M_i \) bids for client \( i \) each in the form \( (p_i^j, q_i^j) \) where \( j \) goes from 1 to \( M_i \).

\( s_i^j \) : A single bid number \( j \) of client \( i \) that is part of a multi-bid of length \( M_i \).

\( s_i \) : The multi-bid of client \( i \).

GBps : Gigabytes per second

\( \theta_i(q) \) : The full valuation function of client \( i \). It represents the estimated benefit from sending \( q \) bytes in an auction round at the stable state. Measured in \( \$ \).

\( \Theta_i(q, Q_i) \) : The truncated valuation function of client \( i \). It represents the estimated benefit from sending \( q \) bytes in an auction round in the stable state, when there are \( Q_i \) bytes in the queue of client \( i \). Measured in \( \$ \).

\( \theta'_i(q) \) : The pseudo-marginal valuation function of client \( i \) for sending \( q \) bytes. Measured in \( \$ \).

\( \theta'_i(q, Q_i) \) : The truncated pseudo-marginal valuation function of client \( i \) for sending \( q \) bytes when there are \( Q_i \) bytes in the queue of client \( i \) according to the multi-bid of client \( i \). Measured in \( \$ \).

\( \upsilon_i(\text{performance}) \) : The estimated benefit from \textit{performance} for client \( i \). Measured in \( \$ \) per second.

\( \text{perf}_i(q) \) : The performance gained by client \( i \) from sending at the rate of \( q \) bytes per second.

\( d_i(p) \) : The full demand function of client \( i \), which is the amount client \( i \) would buy if the resource was sold for a price \( p \) per round. Measured in bytes.
\( \tilde{d}_i(p) \): The pseudo-demand function of client \( i \), which is the amount client \( i \) would buy if the resource was sold for a price \( p \) per unit, based only on the multi-bid of client \( i \). Measured in bytes.

\( \tilde{d}_i(p,Q_i) \): The truncated pseudo-demand function of client \( i \), which is the amount client \( i \) would buy if the resource was sold for a price \( p \) per unit and there are \( Q_i \) bytes in the queue of client \( i \). It is obtained from the multi-bid submitted by client \( i \) and the backlog of client \( i \). Measured in bytes.

\( \tilde{d}(p,Q) \): The aggregate truncated pseudo-demand function associated with all multi-bids and backlogs for each client. Measured in bytes.

\( u \): The market clearing price, which is the price when supply is equal to demand.

\( \tilde{u} \): The truncated pseudo-market clearing price, which is the price when the aggregate truncated pseudo-demand function is equal to or greater than the available capacity for auction \( Q_{\text{round}} \).

\( a_i(s,Q) \): The allocation function, which returns the number of bytes allocated by the auction to client \( i \) according to all multi-bids in \( s \) and all client queues \( Q \).

\( c_i(s,Q) \): The pricing rule, returns the price for the allocation obtained by client \( i \) in the auction according to all multi-bids in \( s \) and all client queues \( Q \).

Social Welfare: A basic game-theoretic measurement which determines the overall client satisfaction for an allocation, i.e., \( \sum_i \theta_i(a_i(s,Q)) \). Measured in dollars per second.

\( U_i(s,Q) \): The utility function of client \( i \). It is calculated by subtracting the cost of the allocation from the value obtained from the allocation.

\( pp \): Price of a packet.

\( ps \): Size of a packet.

\( rq \): Remaining bytes in queue.

\( Used_i \): The amount of data sent by client \( i \) during a traffic scheduling round.

\( MaxBatchSize \): The maximum size of a batch of packets processed together. Measured in bytes.

\( Schedule \): A maximum heap of active bids sorted according to bid price.

\( Active_i \): Set to true if client \( i \) has enqueued packets waiting to be sent.

\( Queue_i \): A queue that contains the packets of client \( i \).

\( Budget \): The number of bytes that can be sent until the end of the traffic scheduling round.

\( CurBid_i \): The current bid of client \( i \).

\( \theta_{\text{max}}(q) \): The maximum valuation among all client valuations that could be obtained for \( q \) bytes transmitted.

\( Isent_i(t_1,t_2) \): Total number of bytes sent by client \( i \) during the time interval between \( t_1 \) and \( t_2 \).
Chapter 1

Introduction

As demand for bandwidth in communication networks increases with the number of end-point devices, bandwidth hungry applications and cloud based services, so does the importance of Quality of Service (QoS). Network operators turn to pricing to manage the scarce amount of available capacity. To compete in this tough market, they must strive to improve clients’ Quality of Service (QoS) while maintaining competitive pricing [SJWHC13]. In bandwidth pricing mechanism design, bids are submitted to an auctioneer that uses an allocation rule and a pricing rule to determine how bandwidth on each outgoing link is allocated to each bidder and how much it will cost. The cost provides an incentive for clients not to overuse network resources.

Traditionally, the pricing mechanism for bandwidth is based on a fixed Flat Fee per GB of data sent [SJWHC13][SV03], particularly in clouds [Amaa][Amab], where the flat fee is changed after the total data transfer exceeds a threshold (i.e., after the first 1GB, 10TB, 40TB, etc.). Charging per GB sent provides an incentive not to use the network when it is not necessary. However, these pricing mechanisms do not provide the network operator with the information required to allocate available bandwidth to those who need it the most. Clients cannot get a discount for bandwidth when demand is low, nor is it possible to suppress excessive demand from low priority flows in favor of high priority flows when demand is high [BBE+13].

Besides preventing congestion, there are two possible objectives for a resource pricing mechanism: maximizing the efficiency of the allocation in terms of client satisfaction, or maximizing the revenue of the resource provider. We focus on maximizing the efficiency of the allocation in terms of client satisfaction, where satisfaction is defined as willingness to pay, and willingness to pay is determined via a repeated VCG auction [Vic61][Cla71][Gro73]. To measure the aggregate benefit of bandwidth allocation to all clients, we use the game theoretic notion of social welfare — their cumulative satisfaction with the bandwidth allocation resulting from the auction. We consider an allocation efficient if it maximizes the social welfare of the participants.

Without any knowledge of the client’s current workload, the provider is neither aware of the potential benefit additional bandwidth might provide nor able to compare the
benefit different clients might gain from this bandwidth. To meet the current system’s load, a client may be willing to pay more for additional bandwidth. A client may also be willing to limit or delay use of the network, in anticipation of a decline in bandwidth price. But he cannot get rewarded for it when bandwidth is priced according to a fixed flat fee.

Today, network providers focus on isolating contending flows. Weighted fair sharing algorithms such as WFQ [DKS89], GPS [PG93], DRR [SV96] and others [BZ96][GVC96] ensure isolation among flows according to share guarantees on the bandwidth of the output link. Those algorithms are also work-conserving, in that the output link is fully utilized as long as there are packets enqueued.

In weighted fair sharing, weights are chosen by the network administrator and each flow will achieve an average data rate proportional to its weight. When prices are based on a flat fee, all clients get the same weight regardless of their actual preferences. However, weighted fair sharing does not guarantee efficiency in terms of social welfare. For example, if two clients pay the same flat fee, it is fair to allocate them an equal share of bandwidth even if the first would rather receive a lower share and pay less while allocating a greater share to the second would increase the social welfare. Therefore, to achieve efficiency, it is in the interest of both client and provider that clients pay for the fine grained bandwidth they need, when they need it [ABYST12][ABYBYST14][BBE+13].

Thus, to optimize social welfare, the network provider needs more information about client preferences than the information that is made available by flat fee pricing, i.e., the client’s willingness to pay a price, chosen by the provider, for a GB sent. Generally, there are two models for sharing the client’s true private information with the provider. The first is the white box model [GHDS+11][HZPW09][HGS+11][NKG10], where client machines share their private information with the provider despite having no incentives to do so. The second is the black box model, where clients do not share private information without an incentive to do so. In our scheme, following the black box model, clients are rational and selfish: they do not share private information without an incentive.

A client’s valuation function reflects his willingness to pay for an allocation of bandwidth. In turn, the client’s willingness to pay for bandwidth reflects the value of bandwidth to the client. The valuation function is derived from the combination of two functions: the performance gained from bandwidth (in performance units for bandwidth) and the value for performance (in dollars per performance unit).

The client may itself be a service provider, with clients of its own. Such a client’s valuation for bandwidth allocation might be affected by variables unrelated to performance gain, such as the market price for the offered services.

In this paper, we design a new queueing algorithm we call Market Driven Queueing (MDQ). Our scheme combines notions from auction theory [MT04] and fair queueing [DKS89] to address the problem of how a network provider might efficiently allocate the bandwidth on an output link among selfish, rational, black box clients, while using
Processing time to transmit a packet where \( n \) is the number of active flows. MDQ calculates the bill for each client after each round of traffic scheduling. The time complexity of calculating the bill for each client is \( O(n \log(n)) \) times the number of packets sent during the round. We analyze the properties of MDQ, evaluate it using a SimPy [Mat08] implementation and simulations, and show that it provides more efficient bandwidth sharing than other pricing schemes and queueing algorithms. To summarize, our contributions are:

1. We identify the problem of traffic schedulers, which allocate all available bandwidth among active flows, but do not allocate bandwidth according to social welfare.

2. We provide a queueing algorithm that combines these two properties and maximizes the efficiency of resource utilization: Market Driven Queueing (MDQ). MDQ allocates bandwidth according to actual demand and the clients’ willingness to pay for bandwidth.

3. We analyze and simulate MDQ showing that it is efficient in optimizing social welfare.

Bandwidth pricing mechanisms take input from clients in the form of bids. If bidding according to the true valuation is a dominant strategy, then the pricing mechanism is called incentive compatible. The bids are used to produce resource allocations and to charge clients accordingly. The allocation and pricing rules could focus on maximizing either the network provider’s revenue or social welfare. The output of a pricing mechanism can be used as input for a queueing algorithm that distributes available bandwidth. Whereas a bandwidth pricing mechanism executes until an allocation is computed, the queueing algorithm is a process that continuously chooses which packet will be transmitted next according to rates/weights, and it provides clients with guarantees regarding their weighted fair share of available bandwidth. In work-conserving algorithms, available bandwidth is shared among active flows according to their respective weights. Queueing algorithms provide isolation among flows such that a malicious client cannot reduce the QoS of other clients.

Next, we discuss the respective advantages of queueing algorithms and bandwidth pricing. In the design of MDQ we combine the properties from both.

**Efficiency:** We measure efficiency according to the social welfare of all clients. To maximize the social welfare, we must share the allocated resource among those who are willing to pay the most for it. Many bandwidth pricing mechanisms maximize social welfare: Lazar and Semret’s PSP auction [LS98] and Maillé and Tuffin’s multi-bid auction [MT04] are two examples. However, no queueing algorithm known to us attempts to maximize social welfare [DKS89],[SV96], [Dev], [KSC06].

**Service Isolation:** Even under heavy load, clients should be isolated. For example, if two clients with the same willingness to pay are sharing a link, and one is maliciously
sending excessive amounts of traffic, the network provider must ensure that the other’s traffic remains unaffected.

**Truthfulness:** The true, private valuations of clients will not be disclosed to the network provider unless it is in the best interest of the client to do so. Bidding according to the true valuation should be a dominant strategy and clients should not be able to manipulate the queueing algorithm to receive a higher bandwidth allocation.

**Work conservation:** To be work-conserving, a queueing algorithm must always be fully utilized as long as there is sufficient demand. This means that a client should be able to dynamically grab free bandwidth that is not currently used by other clients. Many queueing algorithms are work-conserving: DRR[SV96], WFQ[DKS89], HTB[Dev], WRR[KSC06]. However, we are not aware of any social welfare maximizing, incentive compatible bandwidth pricing mechanisms that guarantee this property [LS98][MT04][SH04][DTN+12][EH75]. Those mechanisms assume that all clients are always backlogged. But this assumption is unlikely to hold in packet based networks where network congestion, jitter, bursty traffic and failures in one flow create excess bandwidth that could be used by another flow. We do not make this assumption in our work.

In our analysis of the algorithm we assume that is common in queueing algorithms - We assume that the link controlled by the queueing algorithm is the bottleneck and not the CPU running the queueing algorithm. In addition, we make two assumptions common in bandwidth pricing mechanisms. These assumptions are required to compute the allocation rule efficiently and to provide valid valuation functions. First, we assume that the valuation functions are differentiable, monotonic non-decreasing, continuous, and as long the valuation function is strictly increasing it must be strictly concave [Kel97][LS98][MT04]. Second, we assume that clients are myopic [ABYPBY+14], i.e., do not plan several rounds ahead, just one round ahead.

<table>
<thead>
<tr>
<th>Input</th>
<th>Queueing algorithms</th>
<th>Bandwidth pricing mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Admin sets rate/weights</td>
<td>Submits bids</td>
</tr>
<tr>
<td></td>
<td>Receives packets</td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>Transmits packets</td>
<td>Allocates bandwidth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Charges clients</td>
</tr>
<tr>
<td>Properties</td>
<td>Bandwidth guarantees</td>
<td>Social welfare</td>
</tr>
<tr>
<td></td>
<td>Work conservation</td>
<td>Revenue maximization</td>
</tr>
<tr>
<td></td>
<td>Latency bounds</td>
<td>Incentive compatibility</td>
</tr>
<tr>
<td></td>
<td>Isolation</td>
<td>Fairness</td>
</tr>
<tr>
<td></td>
<td>Fairness</td>
<td></td>
</tr>
<tr>
<td>Time Complexity</td>
<td>Processing per packet</td>
<td>Calculating an allocation and pricing rules</td>
</tr>
</tbody>
</table>

Table 1.1: Summary and comparison between queueing algorithms and bandwidth pricing mechanisms
1.1 Related Work

Auction Theory

Vickery Clarke Groves (VCG) auctions [Vic61][Cla71][Gro73] provide a framework for designing auctions in which even participants with conflicting economic interests will have an incentive to provide the auctioneer with their true valuation of auctioned items. VCG auctions do so by a combination of optimizing social welfare and charging participants according to the damage inflicted on other participants by their presence in the auction.

Various auction mechanisms, some of which resemble the VCG family, have been proposed for divisible resources, in particular for bandwidth sharing on a single link [Kel97][LS98][MT04]. Kelly [Kel97] proposed a mechanism that receives bids and calculates a price where the total resource demand and supply are in equilibrium. Kelly analyzed clients who are ”price takers”, i.e., do not anticipate their influence on prices. For such clients, Kelly’s mechanism maximizes social welfare.

The supply of the link is said to be elastic if it is not fixed in advance. Johari and Tsitsiklis [JT04] extended the analysis of Kelly’s mechanism to price anticipating clients, non-elastic bandwidth supply, and to the network context. They show that when clients are price anticipating, the social welfare could decrease by at most 25% for non-elastic supply. Johari, Manor and Tsitsiklis [JMT05] extended the analysis to elastic supply. They show that when clients are price anticipating, the social welfare could decrease by at most 34% in the case of elastic supply.

Lazar and Semert introduced the Progressive Second Price (PSP) auction [LS98], which is based on VCG. The PSP auction iteratively receives truthful bids representing a single value of the valuation function, broadcasts this value to all participants, and computes the allocation and payments using the latest values received. After a finite number of rounds the system converges to an equilibrium for the given valuations of the participants. The main drawback of this mechanism is that the convergence phase can be quite long, and that it corresponds to a signaling burst. This is especially true if participants enter or leave the auction over time, or change their valuation dynamically, meaning that the convergence phase must be repeated at each change, even when the rest of the valuations did not change.

Maillé and Tuffin introduced the multi-bid auction [MT04], which is also based on VCG. Unlike PSP, the mutli-bid auction reaches equilibrium in a one-shot mechanism, where each client submits a pseudo-valuation, which could be represented as multiple PSP bids aggregated to a single bid. The client’s pseudo-valuation is a subset of values from the valuation function that represent some points of the entire valuation. Using the pseudo-valuations, the auctioneer computes the aggregate demand function, which is used to compute the market clearing price. Bandwidth is allocated according to the market clearing price and payments are set according to the exclusion compensation
principle. The ratio between the social welfare of the multi-bid auction and the social welfare of an auction with full valuations depends on the bids the clients have submitted. In the one-shot multi-bid auction, clients could not improve this approximation, because they learn about the market clearing price only after the auction is complete. We use a repeating auction where clients can update their pseudo-valuation to be more accurate around the market clearing price; thus, the resulting social welfare approximation becomes closer to the optimal social welfare given the complete valuation functions. While reducing the number of bytes required to communicate a valuation function, the multi-bid mechanism adds complexity to the allocation computation, which depends on the total number of bids submitted. Maillé [Mai07] analyzed the Nash equilibria of the PSP auction, proving that all of them are approximations of the optimal social welfare achieved by an allocation according to the market clearing price.

Dobzinski and Nisan [DN10] have shown that all auctions that are based on VCG and whose allocation rule approximates the maximum social welfare must belong to a family of maximum-in-range (MIR) algorithms to maintain incentive compatibility. MIR mechanisms fix in advance the range of possible allocations, completely optimize the social welfare over this range, and then use the exclusion compensation principle for payments. The multi-bid mechanism does not belong to this family, because the bids are not selected from a fixed range. However, the multi-bid mechanism slightly relaxes the notion of truthfulness, by providing an ex-post $K$-Nash equilibrium using truthful bids. In such an equilibrium, a client could not improve his utility beyond $K$ by bidding untruthfully. In the multi-bid auction $K$ is the maximum absolute difference between the two valuations from the allocation of any 2 adjacent bids of some client.

Chawla et. al. [CDH+17] have shown a truthful posted price mechanism that is order oblivious, matching supply and demand of temporal resources with the goal of approximating the maximal social welfare. Such order oblivious pricing mechanisms use probabilistic assumptions about the valuations of the buyers to determine the sell price. In contrast, our algorithm makes no assumptions about the valuations of buyers. Moreover, while the pricing mechanism of Chawla et al. requires that each job’s length must be known when it arrives our algorithm requires no knowledge of a job’s demand for resources.

Queueing Algorithms

Queueing algorithms have been widely studied over the years. One of the earliest algorithms is the Token Bucket Filter algorithm of Clark et al. [CSZ92]. In TBF, there is a bucket of depth $B$ that is replenished with tokens, or credits, at a rate of $R$ tokens per second. To transmit a packet of size $BatchSize$ bytes, the same number of tokens is subtracted from the bucket. Thus, a flow that is scheduled by TBF according to a classifier has a burst of at most $B$ and can transmit at a rate of at most $R$ over long
periods. While TBF restricts the transmission rate of a single flow, it does not capture interactions between different flows going through the same router. When multiple flows traverse the same router, it is desirable to have them isolated, while also utilizing the link fully as long as there is traffic to transmit. To this end, Demers et al. proposed the Weighted Fair Queueing algorithm [DKS89]. The bandwidth allocation of WFQ provides each flow a guaranteed *fair share*, defined by a system administrator. The fair share is based on the max-min fairness principle, where (1) no flow receives more than its request (2) in no other allocation that satisfies condition 1 any flow has a higher share, and (3) condition 2 remains true even when we remove the flow with the minimal allocation. WFQ is also *work-conserving*, which means that the link is never idle while there are packets to transmit. WFQ marks each packet upon arrival with its virtual transmission time. The packet with the lowest virtual time is transmitted. Thus, each packet requires $O(1)$ time to calculate its virtual time and then $O(\log(n))$ time to find the packet with the lowest virtual time. Thus, WFQ requires $O(\log(n))$ processing per packet.

In contrast to the virtual time approach of WFQ, Shreedhar and Varghese introduced DRR [SV96], an efficient fair queueing algorithm. The DRR algorithm guarantees a fair share to each flow and it is also work-conserving, while it requires only $O(1)$ processing time per packet. To achieve this processing time, DRR walks through the flows in round-robin, processing a quantum of bytes for each flow, where the quantum is assigned according to the flow’s fair share.

Hierarchical Token Bucket [Dev] is a classful queueing discipline in Linux. It provides bandwidth guarantees and it is work-conserving. Work conservation is provided according to a hierarchical structure, where bandwidth is shared among neighboring flows before other flows are allowed to borrow. The hierarchical structure allows clients to define how to share bandwidth among their own flows before allowing others to borrow.

**Resource-as-a-Service cloud**

In a Resource-as-a-Service (RaaS) cloud [ABYST12][ABYBYST14], the cloud provider rents fine-grained resources to not-necessarily-cooperative clients running virtual-machines. Ginseng [ABYPBY+] [FBYS16] is an implementation of the RaaS cloud. Ginseng is based on a periodic auction for resources where, at the start of each round, clients submit bids for resources they need, on the basis of which the hypervisor computes resource allocations and payments for those resources. Client are first notified on the results, and only later (eight seconds into the following round in [ABYPBY+] ) do the results become effective. Ginseng as a cloud platform strives to optimize the overall social welfare from resource allocation. Ginseng for memory [ABYPBY+] has introduced the MPSP auction, which is based on the PSP auction, i.e., it approximates the optimal social welfare allocation. Ginseng for LLC [FBYS16] uses an auction that is based on the principles of VCG, such as the exclusion-compensation principle and social welfare
optimization to extract truthful bids. However, it is also an approximation, because it does not consider sharing LLC.

In auctions that allocate memory, care must be taken to avoid waste during memory exchanges. On the one hand, VMs receiving additional memory need some time to warm this memory up, and if this memory is removed too soon then it would have been better not to receive it at all.

On the other hand, VMs who lose actively used memory without any prior notification have to pay the cost of swapping that data to disk. However, bandwidth does not need to be warmed up before it is used and it is exchanged without any cost to either party. Unlike memory or cache, the actual use of bandwidth is bursty and unstable, with no cost for changing the actual allocation. To efficiently utilize bandwidth we must allow the frequent changes in client demands to occur transparently. These changes need to be based on valuation functions and without incurring an additional cost in re-calculating valuations or communicating them between the client and the auctioneer. Moreover, we must consider the effects of changes in allocation on payment calculation to maintain the incentive compatibility properties of our auction. We developed MDQ to meet these requirements, which were not present in Ginseng for other resources.

Memory valuation functions are not necessarily concave. For example, an application utilizes memory only in blocks of 64KB. If it receives a smaller allocation, it cannot utilize it. Moreover, there is a cost for managing additional memory by the kernel. Ginseng for memory introduces the MPSP auction to handle the problem of non-concave valuation functions. The performance from bandwidth is a strictly increasing function up to the saturation point, whereas performance from memory is a weakly increasing function - it may have plateaus before the saturation is reached. Thus, we do not consider non-concave or non-increasing valuation functions.

When allocating cache using state-of-the-art technology [Int], there are only a few possible allocations without sharing, which allows for an efficient implementation where bids include the client’s valuation for all possible allocations. This is not the case for bandwidth, where there are many possible allocations ranging from a few bytes per second to tens of gigabytes per second.

Special application support is needed to fully utilize dynamic memory allocation, because the OS is not aware of the application’s preferences when it is called to swap out data. This means that existing applications have to be modified to benefit fully from Ginseng for memory. However, no modifications are required for applications to benefit from dynamic allocation of bandwidth, since additional bandwidth will be utilized seamlessly.

1.2 Social Welfare and Work Conservation

We conclude the introduction with an example of the interaction between a social welfare maximizing allocation and work conservation. In the example, clients $A$, $B$ and $C$ take
(a) Valuations for 3 clients, A, B and C.

(b) The shaded area shows the social welfare maximizing allocation when all clients are backlogged. The resulting weights are 3 : 3 : 4 for clients A, B and C accordingly.

(c) The green shaded area shows the work-conserving allocation according to the weights of the social welfare maximizing allocation in 1.1b when client C is inactive. Note that this allocation does not maximize social welfare because sharing is not performed according to valuations, but rather using weights.

(d) The green shaded area shows the social welfare maximizing allocation when client C is inactive and the algorithm is work-conserving. Unlike the allocation in 1.1c, this allocation is performed according to client valuations and not according to weights.

Figure 1.1: Example of an allocation that maximizes the social welfare and its effect on work-conservation

part in the multi-bid auction for 10 GBps (see Figure 1.1a). A is willing to pay $15 for each of the first 3 GBps and $1 for each of the remaining 7 GBps. B is willing to pay $5 for each of the first 3 GBps and $2 for each of the remaining 7 GBps. C is willing to pay $4 for each of the 10 GBps. The social welfare maximizing allocation (see Figure 1.1b) will grant A 3 GBps at $5 for each of the first 3 GBps, B will get 3 GBps at $4 for each GBps, and C will take the remaining 4 GBps at $2 for each GBps. However, if C has no traffic to send, a work-conserving algorithm will need to distribute C’s 4 GBps between A and B. If the bandwidth was distributed according to the existing shares, both A and B would get an additional 2 GBps each (see Figure 1.1c), because both have the same weight. Yet this allocation does not optimize social welfare, since the optimal social welfare is achieved when B gets the additional 4 GBps (see Figure 1.1d).

The rest of the paper is organized as follows: Game theoretic notions, allocation and payment rules of the auction, are provided in Chapter 2. MDQ is presented in Chapter 3. An analysis of the properties of MDQ is presented in Chapter 4. We prove that each round of MDQ, when viewed as an auction, results in incentive compatible bidding in Chapter 5. We evaluate MDQ using simulations in Chapter 6. Potential applications of MDQ in other fields are presented in Chapter 7. Finally, conclusions and future work are presented in Chapter 8.
Chapter 2

Truncated Multi-bid Auction

Consider a communication link whose link speed is $\text{LinkSpeed}$ bytes per second. Let the capacity that can be transmitted each second $Q_{\text{round}} = \text{LinkSpeed} \times 1\text{sec}$ bytes be offered for auction. We denote the duration of the auction round by $\lambda = 1\text{sec}$, because it is convenient to consider bandwidth in units of bytes per second, in comparison to bytes per minute, hour, etc. We study a scheme to share the available capacity of $Q_{\text{round}}$ bytes among a set of $n$ clients called $\text{Clients}$ in a repeated auction. Our scheme adapts some notions of the one-shot multi-bid auction of Maillé and Tuffin[MT04] to a round based queueing algorithm.

When entering the auction, the client submits a multi-bid. The multi-bid of client $i \in \text{Clients}$ consists of a sequence of $M_i$ bids of the form $\text{ssec}_i = \{ssec^1_i, ... ssec^{M_i}_i\}$, where for all $m, 1 \leq m \leq M_i$, $ssec^m_i = (qsec^m_i, psec^m_i)$, similarly to the definition in [MT04]: $qsec^m_i$ represents bandwidth in bytes per second and $psec^m_i$ represents the number of dollars the client is willing to pay per second to get $qsec^m_i$ bandwidth.

In the stable state of the queueing algorithm, all active clients in each round, get the same allocation of bytes to send. Thus, we can represent the client’s allocated bandwidth according to the number of bytes he sends in a round. We transform the client multi-bids, to be used in our round based queueing algorithm by multiplying them by the round duration. Thus, for each client $i \in \text{Clients}, 1 \leq m \leq M_i$, $s^m_i = (q^m_i, p^m_i)$, where $q^m_i = qsec^m_i \times \lambda$ and $p^m_i = psec^m_i \times \lambda$. Finally, we denote all client multi-bids with $s = s_1,...,s_n$.

In Section 2.1 we will start with some background and definitions from Maillé and Tuffin’s. We introduce the notion of truncation according to demand in section 2.2 and we define the allocation rule for the truncated valuations in section 2.3. In the next chapter we will use this rule to choose which packet is transmitted next in the queueing algorithm. Finally, we define the pricing rule according to which we bill our clients in section 2.4. The truthfulness of the auction is based on the analysis of the multi-bid auction in [MT04], and we provide the authors’ definitions for completeness. Note, however, that unlike the multi-bid auction, the queueing algorithm is work-conserving and it shares the link according to a combination of client bids and the actual demand.
We represent the actual demand with data queued to be sent.

## 2.1 Game Theoretic Background

In this section, for completeness, we provide basic definitions for the game theoretic constructs from the multi-bid auction [MT04].

**Definition 2.1.** We denote the client’s valuation by:

$$\theta_i(q) = V_i(\text{perf}_i(q)) \times \lambda,$$  \hspace{1cm} (2.1)

where $\text{perf}_i(q)$ describes the performance client $i$ can achieve with $q$ bytes per second and no other bottlenecks, and $V_i(\text{perf}_i(q))$ describes the value perceived by client $i$ for the given level of performance. $V_i$ is the client’s private information, and it is derived from business logic.

**Definition 2.2.** A client $i \in \text{Clients}$ is said to have submitted a truthful multi-bid if

$$\forall m, 1 \leq m \leq M_i, p_{i}^m = \theta_i'(q_i^m).$$  \hspace{1cm} (2.2)

Our system is designed to make truthful bidding a dominant strategy. In Section 5, we discuss the truthfulness of our algorithm, the assumptions required for truthfulness, and their implications. If an algorithm is truthful, then the clients cannot increase their utility by manipulating their bids.

**Definition 2.3.** The demand function of client $i \in \text{Clients}$ is the amount client $i$ would buy if the resource were sold for a price $p$, defined as the function

$$d_i(p) = \begin{cases} 0 & \text{if } p > p_i^{M_i} \\ \frac{1}{\lambda} & \text{if } 0 \leq p \leq \theta_i'(0) \\ q_i^m & \text{otherwise} \end{cases}.$$  \hspace{1cm} (2.3)

According to the definition, the demand function is the inverse of the marginal valuation function. We now move to define the pseudo-demand and marginal valuation functions, which are representations of the full functions constructed on the basis of client multi-bids. Figure 2.1 shows the full demand function $d_i(p)$ and the pseudo-demand function $\bar{d}_i(p)$ constructed according to a multi-bid that contains 4 bids: $\{(1000, 50), (400, 200), (200, 300), (50, 400)\}$.

**Definition 2.4.** Consider a client $i \in \text{Clients}$ who submitted a multi-bid as defined above. The pseudo-demand function associated with the multi-bid of client $i$ is the amount $q_i^m$ client $i$ would buy if the resource were sold for a price $p_i^m$, defined as the function

$$\bar{d}_i(p) = \begin{cases} 0 & \text{if } p > p_i^{M_i} \\ \max_{1 \leq m \leq M_i} \{q_i^m : p_i^m \geq p\} & \text{otherwise} \end{cases}.$$  \hspace{1cm} (2.4)
Definition 2.5. Consider a client \( i \in Clients \) who submitted a multi-bid as defined above. The \textit{pseudo-marginal valuation function} of \( i \) is defined as the function

\[
\bar{\theta}_i'(q) = \begin{cases} 
0 & \text{if } q^1_i < q \\
\max_{1 \leq m \leq M_i} \{p^m_i : q^m_i \geq q\} & \text{otherwise}
\end{cases}
\]  

(2.5)

Using the pseudo-demand functions we have defined, we can define the aggregate pseudo-demand function and the market clearing price. The aggregate pseudo-demand represents the sum of the resource demands of all the clients according to their multi-bids. Using this function we can calculate a price under which clients are willing to share an available supply \( Q_{\text{round}} \). We call this price the pseudo-market clearing price. Maillé and Tuffin proved that allocating resources according to the market clearing price maximizes the social welfare [Mai07]. However, the auction can only deduce the pseudo-market clearing price from the multi-bids, which gives only an approximation of the true market clearing price. Figure 2.2 shows the aggregate pseudo-demand and the pseudo-market clearing price.

Definition 2.6. Consider a set of clients, each of whom submits a multi-bid as defined above. We define the \textit{aggregate pseudo-demand function} associated with those multi-bids as the function

\[
d_i(p) = \sum_{i \in Clients} d_i(p).
\]

(2.6)

Definition 2.7. The \textit{market clearing price} \( u \) is the unique price for which

\[
\sum_{i \in clients} d_i(u) = Q_{\text{round}}.
\]

(2.7)

If the aggregate demand is less than \( Q_{\text{round}} \), i.e., \( \sum_{i \in clients} d_i(u) < Q_{\text{round}} \), then there is no contention for resources and every client gets the maximum allocation he bid for.

Definition 2.8. The \textit{pseudo-market clearing price} \( \bar{u} \) is the highest price such that aggregate pseudo-demand exceeds or is equal to available capacity \( Q_{\text{round}} \). Formally, we define it as

\[
\bar{u} = \max\{p : d(p) \geq Q_{\text{round}}\}.
\]

(2.8)

2.2 Truncation According to Actual Demand

In this section we augment the game theoretic notions from section 2.1 and apply them to the client’s current demand. First, we adapt the definition of valuation to refer to actual demand in the current auction round. Consider a client \( i \in Clients \) who submitted a multi-bid as defined above. Let \( Q_i \) be the number of bytes queued by client \( i \) to be transmitted and let \( Q = (Q_1, Q_2, ..., Q_n) \). The bytes currently queued for transmission represent the client’s current demand.
Figure 2.1: Demand function and pseudo-demand function according to the multi-bids. The shaded area is the truncated pseudo-demand for those multi-bids and $Q_i$ queued bytes.

**Definition 2.9.** The truncated valuation function associated with the multi-bid and the amount of data queued by client $i \in \text{Clients}$ is the value client $i$ would get from $q$ bytes transmitted in this auction round, defined as

$$\Theta_i(q, Q_i) = \theta_i(\min(q, Q_i)).$$  \hfill (2.9)

**Definition 2.10.** The truncated pseudo-demand function associated with the multi-bid and the amount of data queued by client $i \in \text{Clients}$ is the amount $q$ client $i$ would buy if the resource were sold for a price $p$ and client $i$ could use it to transmit those queued bytes. We define it as

$$\tilde{d}_i(p, Q_i) = \min\{Q_i, \bar{d}_i(p)\}.$$ \hfill (2.10)

**Definition 2.11.** The truncated pseudo-marginal valuation function associated with the multi-bid and the amount of data queued by client $i \in \text{Clients}$ is the value client $i$ would gain from being able to transmit $q$ bytes during an auction round. We define it as

$$\tilde{\theta}_i'(q, Q_i) = \tilde{\theta}_i'(\min\{Q_i, q\}).$$ \hfill (2.11)

Figure 2.1 shows an example of a truncated pseudo-demand function (shaded area): the pseudo-demand function $\bar{d}_i(p)$ is constructed using the following multi-bids: $\{(1000, 50), (400, 200), (200, 300), (50, 400)\}$. $Q_i = 300$ is client $i$'s backlog, which represents that client’s actual demand. We truncate the pseudo-demand function according to the actual demand to get the following multi-bids: $(300, 200), (200, 300), (50, 400)$. One of the bids $(1000, 50)$ thus becomes irrelevant and is entirely removed, while another $(400, 200)$ is truncated according to $Q_i$.

**Definition 2.12.** Consider a set of clients, each of whom submits a multi-bid as defined above and queues some bytes to be transmitted. The following function defines the
Figure 2.2: Aggregate pseudo-demand function according to the multi-bids. The shaded area is the aggregate truncated pseudo-demand function. On the horizontal axis: the truncated pseudo-market clearing price \( \tilde{u} \) for the available capacity \( Q_{\text{round}} \).

The aggregate truncated pseudo-demand function associated with those multi-bids and the data queued for transmission by those clients:

\[
\tilde{d}(p, Q) = \sum_{i \in \text{clients}} \tilde{d}_i(p, Q_i). \tag{2.12}
\]

**Definition 2.13.** The truncated pseudo-market clearing price \( \tilde{u} \) is the highest price such that aggregate truncated pseudo-demand is at least as high as available capacity \( Q_{\text{round}} \). Formally, we define it as

\[
\tilde{u} = \max\{p : \tilde{d}(p, Q) \geq Q_{\text{round}}\}. \tag{2.13}
\]

### 2.3 Allocation Rule

The allocation rule is used in each round of the auction to determine the capacity allocated to each client. To allocate the resource, we need to find the truncated pseudo-market clearing price for the multi-bids and the actual queued data. If the aggregate truncated pseudo-demand function for the truncated pseudo-market clearing price is below or equal to \( Q_{\text{round}} \), then all of the capacity is allocated according to the simple allocation rule:

\[
a_i(s, Q) = \tilde{d}_i(\tilde{u}, Q_i). \tag{2.14}
\]

**Definition 2.14.** For every demand function \( d : \mathbb{R} \to \mathbb{R} \) and all \( p \in \mathbb{R} \), we define the value of the function when approaching from the right as

\[
d(p^+) = \lim_{x \to p^+, x \to p} d(x). \tag{2.15}
\]
According to equation (2.15) and equation (2.13), we get
\[ \tilde{d}(\tilde{u}^+, Q) \leq Q_{\text{round}} \leq \tilde{d}(\tilde{u}, Q), \]
(2.16)
which implies that, in order to allocate the available capacity $Q_{\text{round}}$, we need to allocate at least $\tilde{d}(\tilde{u}^+, Q)$ units with unique bids that could be fully satisfied and the remainder consisting of bids that could not be fully satisfied with price $\tilde{u}$.

If the aggregate truncated pseudo-demand function for the truncated pseudo-market clearing price is higher than $Q_{\text{round}}$, then some of the resource will remain unallocated after the simple allocation rule is applied. These "leftovers" are shared among the clients who submitted a multi-bid with price $\tilde{u}$, i.e., clients for whom $\Delta = \tilde{d}_i(\tilde{u}, Q_i) - \tilde{d}_i(\tilde{u}^+, Q_i) > 0$. Each client receives a share proportional to $\Delta$ with respect to the change in the aggregate truncated pseudo-demand function. Therefore, the complete allocation rule for any multi-bid profile $s$ is
\[ a_i(s, Q) = \tilde{d}_i(\tilde{u}^+, Q_i) + \frac{Q_{\text{round}} - \tilde{d}(\tilde{u}^+, Q)}{\tilde{d}(\tilde{u}, Q) - \tilde{d}(\tilde{u}^+, Q)} \Delta. \]
(2.17)
According to equation (2.16), we have $0 \leq \frac{Q_{\text{round}} - \tilde{d}(\tilde{u}, Q)}{\tilde{d}(\tilde{u}, Q) - \tilde{d}(\tilde{u}^+, Q)} \leq 1$. Applying this to equation (2.17), we get
\[ \tilde{d}_i(\tilde{u}^+, Q_i) \leq a_i(s, Q) \leq \tilde{d}_i(\tilde{u}, Q_i), \]
(2.18)
which means that each client receives at least the quantity for which supply exceeds demand and that it is limited by demand for the market clearing price.

The allocation rule is similar to the multi-bid auction allocation rule of [MT04]. However, we use the actual demand, represented by data queued to be sent. Our allocation rule has the same result as the multi-bid auction allocation rule when queues are full, i.e., $\forall i \in \text{clients}, q_i^{M_i} \leq Q_i$. However, when the actual demand is low, i.e., $\exists i \in \text{clients}, q_i^{M_i} \geq Q_i$, our allocation differs from the multi-bid auction in that we truncate the valuation functions according to actual demand.

For example, assume that the result of the multi-bid auction allocation rule for the multi-bids of 2 clients is to split the bandwidth equally between them. However, the multi-bid of the first client is based on expected traffic and for a few minutes that client has nothing to send. Then, our allocation rule would respond by allowing the second client to transmit at full wire speed, while the multi-bid auction would ignore the change if the first client’s remained unchanged.

In this offline version, we use the truncated aggregate pseudo-demand function to compute the allocation for each client. We sort the eligible multi-bids, then find the truncated pseudo-market clearing price via a binary search for the quantity $Q_{\text{round}}$. Finally, we compute the allocation for each client according to the allocation rule in (2.17).
A different but equivalent online approach is to compute the allocation for all clients by gradually allocating the resource according to the highest price until the entire resource has been allocated. Essentially, this is equal to finding the truncated pseudo-market clearing price. This method will be used by our queueing algorithm in Section 3.2.

Note that the allocation rule is not continuous in prices. A client willing to pay \( X + \epsilon \) (\( \epsilon > 0 \)) for \( Q_{\text{round}} \) units of bandwidth will get all \( Q_{\text{round}} \) units, even though another client is willing to pay \( X \) for \( Q_{\text{round}} \) bytes. However, if both clients bid \( X \) for \( Q_{\text{round}} \) bytes, then \( Q_{\text{round}} \) is split equally between them.

### 2.4 Pricing Rule

At the end of an auction round, prices are computed for the service received by each client. Each client \( i \in \text{Clients} \) is charged a payment \( c_i \) in dollars according to the exclusion-compensation principle\[\text{[Vic61]}\]. A client pays to cover the cost of lost value he imposes on all other clients by his presence. The direct application of the exclusion compensation principle to truncated valuations leads to:

\[
c_i(s, Q) = \sum_{j \neq i, i \in \text{Clients}} \Theta_j(a_j((\emptyset, s_{-i}), Q), Q_j) - \Theta_j(a_j(s, Q), Q_j),
\]

where \( s_{-i} \) refers to all the multi-bids except \( s_i \).

However, the multi-bids reveal only the pseudo-marginal valuation function, which is an approximation for the marginal-valuation function. The approximation leads to the pricing rule we use:

\[
c_i(s, Q) = \sum_{j \neq i, i \in \text{Clients}} \int_{a_j(s, Q)} a_j((\emptyset, s_{-i}), Q) \tilde{\theta}_i'(q, Q) dq.
\]

**Definition 2.15.** Finally, we define the truncated utility function of client \( i \in \text{Clients} \) for multi-bid profile \( s \):

\[
U_i(s, Q) = \Theta_i(a_i(s, Q), Q_i) - c_i(s, Q).
\]
Chapter 3

Market Driven Queueing

Market Driven Queueing (MDQ) connects the process of scheduling traffic flows and the process of bidding and charging for traffic allocation. MDQ consists of three layers running in parallel. The first layer, Admission Control, accepts multi-bids from new clients and from clients who changed their valuation. It calculates the guaranteed resource allocation according to the allocation rule in (2.17) applied to the non-truncated valuations. It reports expected bandwidth guarantees to all clients, and adjusts the multi-bids for the next round.

The second layer, Traffic Scheduling, accepts packets from clients, classifies them, shares the round’s available capacity $Q_{round}$ bytes among clients according to the multi-bids, logs all operations, and provides bandwidth guarantees and work conservation.

The third layer, Billing, processes the logs produced by Traffic Scheduling, calculating payments according to the allocation that resulted from each round of traffic scheduling.

Figure 3.1: Overview of MDQ
3.1 Admission Control

Each time a client joins or changes his valuation function, he submits a new multi-bid. The multi-bid is collected and processed by the admission control process to determine the guaranteed resource allocation for each client and to transform them according to the round duration $\lambda$. The new guaranteed resource allocation is then reported to each client. The queueing algorithm will provide predictable minimum bandwidth guarantees for backlogged clients according to the reported quantity. Because payments are collected only for service received by a client, clients do not pay for the resource allocation guarantees.

We calculate the guaranteed resource allocation according to the allocation rule in (2.17) applied to the non-truncated valuations. To measure the worst-case allocation, which occurs when all queues are full, we use the transformed multi-bids of all clients as input to the allocation rule, disregarding the current state of client queues. The Traffic Scheduling process will allocate available capacity according to the valuation functions and uphold the guaranteed resource allocation provided by the Admission Control process, as we show in Theorem 4.3.

A client can query the current allocation to learn whether an increase in his send rate will increase his allocation. The query interface is useful for clients who would like to send packets at a rate higher than their allocation. This interface is also useful when the guarantees are very inaccurate. Such inaccuracy may result either from sudden utilization changes or from malicious behavior. For example, a malicious client may submit a very high bid. Such bids cause the Admission Control process to announce that this client is guaranteed to receive the entire capacity if he has traffic to send. However, the malicious client will not send any traffic, rendering the announcement of guarantees irrelevant. The query interface can help overcome this problem as it will allow clients to estimate the current allocation and make short term decisions on that basis.

3.2 Traffic Scheduling

We propose an algorithm for servicing queues based on client transformed multi-bids. The transformed multi-bids are already known from the admission control process. Traffic is scheduled in rounds, where each round allocates the available capacity $Q_{\text{round}}$ between active backlogged clients according to the allocation rule in (2.17). Inactive clients have no backlog and are ignored by the algorithm. Each operation during traffic scheduling is logged and later processed by the billing process. Some of the variables need to be reset when a new round begins; this is done explicitly in the pseudo-code. However, each such variable can be reset lazily when it is first accessed during a new round.

We assume that bandwidth on the output link is the bottleneck. Each round
allocates the available capacity $Q_{\text{round}}$ bytes. Thus, if we set $Q_{\text{round}} = \text{LinkSpeed} \times \lambda$, i.e., $Q_{\text{round}} = 10 \text{ GB}$ when $\text{LinkSpeed} = 10\text{GBps}$ and $\lambda = 1\text{sec}$, then it is completed within exactly $\frac{Q_{\text{round}}}{\text{LinkSpeed}}$ seconds when we estimate $\text{LinkSpeed}$ accurately. Batching could be increased in order to sustain higher rates and reduce the load on the CPU; however there is a tradeoff between the maximum batch size, marked as $\text{MaxBatchSize}$, and the optimal social welfare provided by the allocation of $\text{Dequeue}$. In Theorem 4.1 we explore the worst case decrease in social welfare as a function of $\text{MaxBatchSize}$ and in Section 6 we use simulations to find the optimal point in this tradeoff.

Let packets sent by some client $i$ arrive at an $\text{Enqueue}$ process. (The pseudo-code of this process is presented in Algorithm 3.1.) The $\text{Enqueue}$ process first classifies the packet to a flow and queues it in $\text{Queue}_i$, where it waits to be sent on the output link. Packets may be classified by any of the packet header fields that were provisioned for the client, i.e., MAC address, VLAN ID, IP address or transport port. We use a search in a trie of these fields to find the client who owns an incoming packet. Once a packet is classified, the $\text{Enqueue}$ process will set a client as active if he was not active previously. If the current active bid is not sufficient to send a packet, because its quantity is smaller than the packet, then the processing of this packet adjusts the bid price and quantity according to the next bid to allow the transmission of this packet using the adjusted bid. We choose $\text{MaxBatchSize}$ to be at most the smallest difference between consecutive bids. Using such $\text{MaxBatchSize}$ guarantees that at most one bid boundary is crossed when a batch is sent and the later bid would still be able to handle at least one additional batch.

Algorithm 3.1 Enqueuing process - called on arrival of packet to queue the packet into client’s queue

1. function $\text{Enqueue}(\text{Packet})$
2. $i = \text{ExtractClient}(\text{Packet})$
3. $\text{Queue}_i.\text{Enqueue}(\text{Packet})$
4. if $\text{Active}_i == 0$ then
5. $\text{Active}_i = 1$
6. $ps = \text{Size}(\text{Packet})$ ▷ packet size
7. $(q_i^j, p_i^j) = \text{CurBid}_i$
8. $rq = q_i^j - \text{Used}_i$ ▷ remaining in queue
9. if $ps > rq$ then
10. $pp = \frac{p_i^j \cdot rq + p_i^{j-1} \cdot (ps - rq)}{ps}$ ▷ packet price
11. $\text{CurBid}_i = (rq + ps, pp)$ ▷ adjust current bid to include packet size
12. $\text{Schedule}.\text{Insert}(\text{CurBid}_i)$

The $\text{Dequeue}$ process is presented in Algorithm 3.2. The $\text{Dequeue}$ process is active whenever there are packets queued to be sent on the output link. Once a packet has been transmitted, this process chooses the next packet and begins to transmit it. The $\text{Dequeue}$ process operates in rounds; each round transmits up to $Q_{\text{round}}$ bytes from client queues (or until all queues are empty). The $\text{Budget}$ tracks the number of bytes
Algorithm 3.2 Dequeuing process — called to transmit a packet

1: function Dequeue()
2:     while True do
3:         Schedule = MaximumHeap() \(\triangleright\) contains active bids
4:         Budget = Q_{round} \(\triangleright\) Link capacity
5:         \(\forall i, \text{Used}_i = 0\)
6:         \(\forall i, j_i = M_i\)
7:         \(\forall i, \text{CurBid}_i = (q_i^j, p_i^j)\)
8:         while not Schedule.Empty() do
9:             \((q_i^{j_i}, p_i^{j_i}) = \text{Schedule.Extract()}\) \(\triangleright\) Get highest price bid
10:            BatchSize = Size(Head(Queue_i))
11:            if Budget < BatchSize then
12:                break
13:            end if
14:            Budget = Budget – BatchSize
15:            Send(Dequeue(Queue_i))
16:            Used_i = Used_i + BatchSize
17:            if IsEmpty(Queue_i) then
18:                Active_i = 0
19:            else
20:                nps = Size(Head(Queue_i)) \(\triangleright\) next packet size
21:                rq = q_i^j – Used_i \(\triangleright\) remaining in queue
22:                if rq == 0 then
23:                    j_i = j_i – 1 \(\triangleright\) advance to the next bid
24:                    rq = q_i^j – Used_i \(\triangleright\) remaining in queue
25:                end if
26:                if nps > rq then \(\triangleright\) adjust price to include nps
27:                    p_i^{j_i} = p_i^{j_i-1} \cdot (nps – rq) \(\triangleright\) remaining in queue
28:                    q_i^{j_i} = rq + nps
29:                end if
30:                CurBid_i = (q_i^{j_i}, p_i^{j_i}) \(\triangleright\) reset active schedule and start new round
31:                Schedule.Insert(CurBid_i)
32:            end if
33:        end while
34:    end while
35: end function
that can still be transmitted in the current round. We consider each round of Dequeue as a standalone auction, where transformed multi-bids from the admission control process are combined with the queued packets to produce the truncated functions from Section 2.2. The Dequeue process computes the allocation rule by repeatedly extracting the bid with the highest price from the heap of active bids.

Client \( i \) becomes active by sending a packet when his queue is empty. In that case, \( Active_i \) is set to 1. To avoid examining bids for empty queues, we keep track of all the active bids in a heap called Schedule sorted by maximum price. Whenever a packet of client \( i \) arrives at a previously empty queue \( Queue_i \), the latest active bid \( j \) of client \( i \), i.e., \((q_j^i, p_j^i)\) of client \( i \), is inserted to the heap and client \( i \) becomes active.

Whenever a bid \((q_j^i, p_j^i)\) is extracted from the heap for processing, we guarantee that the algorithm can either handle a full batch of packets or consume what is left of the round’s budget. Thus, for every bid extracted, at least one packet batch is sent.

During a round, for each client \( i \) we track the number of bytes sent by this client in a variable called \( Used_i \). This variable allows us to track the client’s truncated valuation. After a packet is sent, if \( Used_i \) exceeds the number of bytes specified by the currently active bid and \( Queue_i \) still has data to send, we insert the next bid \((q_j^{i-1}, p_j^{i-1})\) of client \( i \) into the heap. Otherwise, we remember the current bid \((q_j^i, p_j^i)\) in case client \( i \) queues data to be sent later during this round, in which case this bid will become active again.

The Dequeue process computes the allocation rule by repeatedly processing the highest bid from the heap of active bids. During the processing of a bid of client \( i \), Dequeue will service a single packet of client \( i \) and advance the client’s current bid \((CurBid_i)\) to the next bid if the quantity specified in the current bid has been consumed. A throughput oriented implementation could use batching to process multiple packets together based on a single bid to reduce the overhead per packet. While batching can improve the throughput of the implementation, it may increase latency of other more important packets and reduce the efficiency of the social welfare optimization by Dequeue. Latency of important packets may increase, because the transmission of a batch of packets cannot be interrupted. Social welfare may decline, because during some batch transmission, a more valuable batch may arrive, but it cannot be transmitted in the same round. We discuss this further in chapter 4.

Overall, when clients’ valuations remain constant and clients do not become active or inactive in Dequeue, then each traffic scheduling auction round repeatedly allocates the capacity \( Q_{round} \) bytes according to the allocation rule. This creates the illusion of a single auction allocating bandwidth of \( LinkSpeed \) gigabytes per second over a long period. However, payments are collected for each round and allocations will change according to the client bids and demand for bandwidth.

When two or more clients submit a bid with the same price, this results in a price conflict. Algorithm 3.2 and the allocation rule in Eq. (2.17) are agnostic to price conflicts on prices below the market clearing price. When there is a price conflict for a
price equal to or above the market clearing price, the resource needs to be allocated among the conflicting clients in proportion to their demand for bandwidth at this price (see the allocation rule in Eq. (2.17)). Let $q_{\text{conf}}$ be the sum of $q$’s for all bids with the conflicting price. During $\text{Enqueue}$ such bids will be combined to a single super-bid $(p, q_{\text{conf}}, \text{proportions})$, where $\text{proportions}$ indicates the proportion of each client’s bid in $q_{\text{conf}}$. When $\text{Dequeue}$ encounters such a bid, it will distribute what is left from $\text{Budget}$ to those clients according to these proportions. For clarity, we assume in the pseudo-code of Algorithm 3.2 that there are no 2 bids with the same price.

### 3.3 Billing

Payments are calculated for each round of $\text{Dequeue}$ according to the pricing rule in Eq. (2.20). Clients are charged for service received, which is the number of bytes sent by each client during a round.

We calculate the bill for each client using a log of packet arrival and transmission times, with an additional indication for when the round is complete. To calculate the bill of client $i$ according to equation (2.20), we need to find the social welfare of the packet schedule had client $i$ not participated in the traffic scheduling round. To do so, we simulate $\text{Dequeue}$ without the packets of client $i$ by using the link speed to calculate when packet transmission is complete. The simulation lasts as long as the actual round and continues even if no client is backlogged. Note that during the time when no client is backlogged, the link is inactive and no social welfare is gained. We calculate client $i$’s final bill for the round just as we did in equation (2.20), by subtracting the social welfare of the simulated schedule from the social welfare of the actual packet schedule without client $i$. Note that while this formally represents the bill, it is not really the social welfare that would have been achieved without client $i$, because of the feedback loop [Fei16], by which the social welfare might be affected by the reactions of other clients. For example, other clients could have reacted to the absence of client $i$ by increasing their packet rate.

The time complexity of calculating the bill is the same as running $\text{Dequeue}$ for $n$ times, where $n$ is the number of clients. Since the time complexity of calculating the payment is greater than the time complexity of the allocation, we suggest the following optimizations. First, when the system is in a stable state, and client truncated valuations remain the same, then each round has the same allocation and all participants pay the same price as long as backlogs are sufficiently long. In that case, we could cache the bill of each client and return it without further calculation. Second, we could run the bill calculation in parallel for each client. Third, we could run the bill calculation in parallel to the round itself by allowing each calculating thread to go over the log while it is being created, effectively calculating the bill in parallel to traffic scheduling.

Finally, there are interfaces through which clients can interact with the billing process. Clients can query the billing process at any time to receive more information.
regarding their bill. Clients can also ask the billing process to set a limit to prevent them from exceeding their budget. Note that payments may not be calculated immediately at the end of the traffic scheduling round, but rather at a later point in time. Thus, some time may pass between the moment when the budget is exceeded and the client is notified.
Chapter 4

Analysis of the Queueing Algorithm

In this section we analyze the properties of MDQ, focusing on the stable state when clients’ valuations remain constant and clients do not become active in Dequeue. We start from observations regarding traffic scheduling during a round of Dequeue. Then, we proceed to analyze the worst case run time of Dequeue required to process each packet. Finally, we show that the number of bytes sent each round is higher than or equal to the result of the allocation rule.

Definition 4.1. Let $\theta_{\text{max}}(q)$ be the maximum valuation for $q$ bandwidth. Formally,

$$\theta_{\text{max}}(q) = \max_{i \in \text{clients}} \theta_i(q).$$

Definition 4.2. A client $i$ is backlogged during an interval $(t_1, t_2)$ of an execution if the queue of client $i$ is never empty during $(t_1, t_2)$.

We observe that the packet schedule during a round of Dequeue where not all clients are backlogged may not be optimal, because during packet transmission a packet from a client with higher valuation can arrive. The optimal allocation would anticipate this packet and wait for it instead of transmitting suboptimal packets. However, the social welfare obtained by Dequeue is always at most $\theta_{\text{max}}(\text{MaxBatchSize})$ lower than the optimal algorithm, because after Dequeue finishes transmitting a packet it will mimic the packet schedule of the optimal algorithm. This means that all payments for client $i \in \text{Clients}$ are bounded in the range $(-\theta_{\text{max}}(\text{MaxBatchSize}), \theta_{\text{max}}(\text{MaxBatchSize}) + \Theta_i(a_i(s,Q),Q_i))$.

In an unstable state a negative payment is possible where the system needs to pay a client due to the social welfare lost in his absence. This could happen when a high-valuation packet could not be sent in the absence of this client. We can see an example of this in Figure 4.1, which shows the arrival times and sizes of packets belonging to 3 clients participating in MDQ. The pseudo-marginal valuation of client $C$ is the
Figure 4.1: Each block represents a packet that can be transmitted by the router only when the entire packet is received. The figure shows a pattern of packet arrivals where the payment for client A is negative. In the absence of A less data of client C is transmitted during the round, resulting in a negative payment.

highest, followed by the pseudo-marginal valuation of client A, with the lowest valuation belonging to client B. Thus, when active, client C should receive all the available bandwidth. Assume that at the start of the round clients A and B are backlogged and client A is constantly transmitting, while client C starts transmitting during the round.

We observe that the first full packet of client C arrives shortly before the transmission of the second packet of client A is complete. This allows MDQ to start transmission of the packets of client C shortly after they arrive. However, in the absence of client A, the packets of client B are constantly transmitted. In that case, the first full packet of client C arrives shortly after the transmission of a new packet of client B has begun. Thus, the transmission of the first packet of C will be delayed when client A is absent, resulting in a reduced number of bytes sent by client C.

**Theorem 4.1.** Any allocation during a round of MDQ has at least the same social welfare as the optimal allocation OPT minus $\theta_{\text{max}}(\text{MaxBatchSize})$.

**Proof.** We prove the claim by induction over the packets transmitted in MDQ.

**Base:** The first batch of packets is of at most $\text{MaxBatchSize}$ bytes. Assume that immediately after the bid in MDQ was extracted from the heap, a packet from a client whose valuation is equal to $\theta_{\text{max}}$ arrives. OPT transmits this packet. As a result, the social welfare of OPT is higher than the social welfare of MDQ by at most $\theta_{\text{max}}(\text{MaxBatchSize})$.

**Step:** Assume the claim is true for $n \times \text{MaxBatchSize}$ bytes transmitted by MDQ. When MDQ transmits the next $\text{MaxBatchSize}$, it can transmit the batch with the highest-value that already arrived, while OPT can also take one more batch which arrives in the middle of the step, instead of the batch sent by MDQ. In the worst case, the batch sent by OPT is from a client whose valuation is equal to $\theta_{\text{max}}$. As a result, the boundary on the difference between the social welfare of OPT and the social welfare of MDQ did not change after the transmission of the next $\text{MaxBatchSize}$.

According to Theorem 4.1, when rounds get shorter, the difference for the social welfare per second increases. Thus, longer rounds will improve the algorithms social
welfare per second.

When all clients who participate in the round remain backlogged until the round is finished, then the allocation according to Dequeue is optimal, as it follows an allocation according to truncated valuations where no packet arrivals can change the allocation. If these clients remain backlogged during the simulation of the bill calculation, then the algorithm can be analyzed based on truncated valuations and it has the incentive compatibility properties of the multi-bid auction [MT04], as we show in Section 5.

**Definition 4.3.** The work required to process a packet is defined as the maximum of the time complexities to enqueue or dequeue a packet.

**Theorem 4.2.** The queueing algorithm requires $O(\log(n))$ processing time per packet, if for each client $i \in \text{Clients}$ and bid $j_i$ it holds that $(q_i^{hi}, p_i^{hi})$, $q_i^{hi} > \text{MaxBatchSize}$. \[ \text{Proof.} \] Enqueuing a packet requires finding the client queue associated with it. Packet classification is accomplished in $O(1)$ time by searching a trie of packet classifiers using packet headers fields, enqueueing the packet to the end of the client’s queue ($O(1)$ time), and possibly inserting the current bid to the maximum heap $\text{Schedule}$ ($O(\log(n))$ time). Dequeueing a packet requires determining the next queue to service by extracting the next maximum price bid from the heap $\text{Schedule}$ ($O(\log(n))$ time), then doing a fixed number of operations (per packet) to update usage and budget, and possibly inserting the next bid to the maximum heap $\text{Schedule}$ ($O(\log(n))$ time). If for all bids $(q_i^{hi}, p_i^{hi})$, $q_i^{hi} > \text{MaxBatchSize}$, we are guaranteed to send at least one packet for each processed bid except the last packet in a round, which will get serviced the next time any of its owner’s bids are processed by Dequeue. Multiple per client counters need to be reset when a new round begins. We defer their reset until the first time they are used in a new round. This requires checking the round number before accessing them. Note that we do not take the time complexity of price calculation into consideration, because it is accomplished by a different process. \[ \square \]

**Definition 4.4.** Let $I_{sent_i}(t_1, t_2)$ be the total number of bytes sent on the output line by client $i$ in the interval $(t_1, t_2)$.

We first show that when client $i$ is backlogged at the start of a round, then by the end of the round he sends at least his guaranteed allocation.

**Theorem 4.3.** Each client is allocated bandwidth that is more than or equal to his truncated valuation during the round. Formally, let $s$ be the transformed multi-bid profile of all clients, for all executions of the queueing algorithm, and let there be a time interval $(t_1, t_2)$ that contains a complete round of MDQ. It holds that

\[ \alpha_i(s, Q_i) \leq I_{\text{sent}_i}(t_1, t_2). \] (4.1)
Proof. At any given time during the auction round there is a bid of client $i$ in Schedule because client $i$ is always backlogged. In the worst case, all clients are backlogged during this round. Let $s'$ denote all the bids in $s$ that are lower than $\tilde{u}$, which is the price where aggregate truncated pseudo-demand exceeds the available capacity $Q$. In each round of Dequeue, all bids of $s'$ will be extracted from Schedule since $\sum_{(p,q) \in s'} q \leq Q$. Since client $i$ is backlogged, he will be eligible to send at least $\tilde{d}_i(\tilde{u}^+, Q_i)$ bytes in the auction round. It remains to address the bid $\tilde{u}$: if client $i$ has no such bid, then he already sent $a_i(s, Q_i)$ bytes. However, if client $i$ has such a bid, i.e., $\exists k, 1 \leq k \leq M_i, (p^k_i, q^k_i) \in s$ such that $p^k_i = \tilde{u}$, then this is the bid with the highest price: it will be extracted from Schedule and client $i$ will get to send the number of bytes remaining in the budget of this round. In conclusion, client $i$ sent at least $a_i(s, Q_i)$ during the auction round.
Chapter 5

Incentive Compatibility

In this section we present the proof from Maillé and Tuffin [MT04] that the multi-bid chosen by a client cannot achieve better results than simply revealing that client’s true valuation regardless of the multi-bids submitted by others. To simplify the notations, we omit the transformation between client multi-bids, which are measured in bandwidth and $ per second, and the transformed multi-bids which are measured in bytes and $ respectively. In our case bidding truthfully means bidding with prices equal to the marginal valuation \( \forall m, 1 \leq m \leq M, p_i^m = \theta_i'(q_i^m) \). During our analysis we focus on the stable state when clients’ valuations remain constant and clients do not become active in Dequeue. Only in the stable state do the truncated valuations represent the actual allocation. This limitation applies also for pricing, where a stable state in the absence of a client is required for analysis according to truncated valuations.

**Definition 5.1.** Let \( C_i \) be the maximum gap in value between 2 consecutive bids of client \( i \). Formally,

\[
C_i = \max_{1 \leq m < M_i} \left\{ \frac{q_i^m}{q_i^{m+1}} \int (\Theta_i(q) - \Theta_i(q_i^m))dq \right\},
\]

(5.1)

with \( q_i^{M_i+1} = 0 \).

Similarly to the multi-bid auction, we show that truthful bidding is a dominant strategy, up to \( C_i \) for any client \( i \). We do not discuss the truthfulness of unstable states. However, following Theorem 4.1, we know that the inefficiency of the MDQ compared to the OPT allocation is bounded by \( \theta_{\text{max}}(\text{MaxBatchSize}) \) even for unstable states. Assuming the worst-case, MDQ still ensures that truthful bidding is a dominant strategy, up to \( C_i + \theta_{\text{max}}(\text{MaxBatchSize}) \).

The proof follows the notation and steps of Maillé and Tuffin [MT04]. We start by comparing the truncated client valuation obtained by any multi-bid with the truncated client valuation of the truthful multi-bid. Then, we proceed to consider utility functions.
and we conclude with a comparison of utility obtained by non-truthful and truthful bidding.

First we show a lemma that will be helpful later.

**Lemma 5.2.** For any quantity \( y \geq 0 \) and any multi-bid profile \( s \),

\[
\int_y^{a_i(s,Q)} \tilde{\theta}'_i(q, Q_i) dq \geq \tilde{u} \cdot (a_i(s, Q) - y),
\]

(5.2)

where \( \tilde{u} \) is the truncated pseudo-market clearing price corresponding to \( s \) and the currently queued data.

**Proof.** There are 2 cases:

- \( y \leq a_i(s, Q) \). Then, because \( \tilde{\theta}'_i \) is a non-increasing function, we know that

\[
\tilde{\theta}'_i(y, Q_i) \geq \tilde{\theta}'_i(a_i(s, Q), Q_i).
\]

Thus we get,

\[
\int_y^{a_i(s,Q)} \tilde{\theta}'_i(q, Q_i) dq \geq \tilde{\theta}'_i(a_i(s, Q), Q_i) \cdot (a_i(s, Q) - y).
\]

According to equation (2.18) we get

\[
a_i(s, Q) \leq \tilde{d}_i(\tilde{\theta}, Q_i).
\]

Thus, since \( \tilde{\theta}'_i \) is a non-increasing function, we get

\[
\tilde{\theta}'_i(a_i(s, Q), Q_i) \geq \tilde{\theta}'_i(\tilde{d}_i(\tilde{\theta}, Q_i)).
\]

By definition of the truncated pseudo-demand function (2.10) and the truncated pseudo-marginal valuation function, we get

\[
\tilde{\theta}'_i(\tilde{d}_i(\tilde{\theta}, Q_i), Q_i) = \tilde{u}.
\]

Finally, we get

\[
\int_y^{a_i(s,Q)} \tilde{\theta}'_i(q, Q_i) dq \geq \tilde{u} \cdot (a_i(s, Q) - y).
\]

- \( y > a_i(s, Q) \). Then, because \( \tilde{\theta}'_i \) is a non-increasing function, we know that

\[
\tilde{\theta}'_i(y, Q_i) \leq \tilde{\theta}'_i(a_i(s, Q), Q_i).
\]
Thus we get,

\[ \int_{a_i(s,Q)}^{y} \tilde{\theta}'_i(q,Q_i) dq \leq \tilde{\theta}'_i((a_i(s,Q))^+, Q_i) \cdot (y - a_i(s,Q)). \]

According to equation (2.18) we get
\[ a_i(s,Q) \geq \tilde{d}_i(\tilde{u}^+, Q_i). \]

Thus, since \( \tilde{\theta}'_i \) is a non-increasing function, we get
\[ \tilde{\theta}'_i(a_i(s,Q), Q_i) \leq \tilde{\theta}'_i(\tilde{d}_i(\tilde{u}^+, Q_i), Q_i). \]

By definition of the demand function (2.3) we get
\[ \tilde{\theta}'_i(\tilde{d}_i(\tilde{u}^+, Q_i), Q_i) \leq \tilde{u}. \]

Thus, we get
\[ \int_{a_i(s,Q)}^{y} \tilde{\theta}'_i(q,Q_i) dq \leq \tilde{u} \cdot (y - a_i(s,Q)). \]

Finally, we get
\[ \int_{y}^{a_i(s,Q)} \tilde{\theta}'_i(q,Q_i) dq \geq \tilde{u} \cdot (a_i(s,Q) - y). \]

**Theorem 5.1.** If a client \( i \) submits a truthful multi-bid \( s_i \), then every other multi-bid, \( t_i \) (truthful or not) corresponds to an increase in client valuation that is less than
\[ \int_{\tilde{d}_i(\tilde{u}^+, Q_i)}^{\tilde{d}_i(\tilde{u}, Q_i)} (\tilde{\theta}'_i(q,Q_i) - \tilde{u}) + \tilde{u} \cdot (a_i(t_i, s_{-i}, Q) - a_i(s_i, s_{-i}, Q)). \]

Formally we show that,
\[ \tilde{d}_i(\tilde{u}, Q_i) - \int_{\tilde{d}_i(\tilde{u}^+, Q_i)}^{\tilde{d}_i(\tilde{u}, Q_i)} (\tilde{\theta}'_i(q,Q_i) - \tilde{u}) + \tilde{u} \cdot (a_i(t_i, s_{-i}, Q) - a_i(s_i, s_{-i}, Q)) \geq \Theta_i(a_i(t_i, s_{-i}, Q), Q_i) - \Theta_i(a_i(s_i, s_{-i}, Q), Q_i). \]

**Proof.** Now, we begin by showing that the difference between the valuations is bounded.

First, if we multiply the equation by minus 1, we obtain that
\[ \Theta_i(a_i(s_i, s_{-i}, Q), Q_i) - \Theta_i(a_i(t_i, s_{-i}, Q), Q_i) \geq \int_{\tilde{d}_i(\tilde{u})}^{\tilde{d}_i(\tilde{u}^+)} (\tilde{\theta}'_i(q,Q_i) - \tilde{u}) + \tilde{u} \cdot (a_i(s_i, s_{-i}, Q) - a_i(t_i, s_{-i}, Q)). \]
Observe that

$$\Theta_i(a_i(s_i, s_{-i}, Q), Q_i) - \Theta_i(a_i(t_i, s_{-i}, Q), Q_i) = \int_{a_i(t_i, s_{-i}, Q)} a_i(s_i, s_{-i}, Q) \Theta'_i(q, Q_i) dq. \quad (5.5)$$

Thus, it is sufficient to show that

$$\int_{a_i(t_i, s_{-i}, Q)} a_i(s_i, s_{-i}, Q) \Theta'_i(q, Q_i) dq \geq \tilde{u} \cdot (a_i(s_i, s_{-i}, Q) - a_i(t_i, s_{-i}, Q)) - \int_{a_i(t_i, s_{-i}, Q)} a_i(s_i, s_{-i}, Q) \tilde{\Theta}'_i(q, Q_i) dq. \quad (5.6)$$

There are 3 cases:

- If $a_i(s_i, s_{-i}, Q) > a_i(t_i, s_{-i}, Q)$, then by the definition of the truncated valuation function (Definition 2.9), and by definition of the truncated pseudo-marginal valuation function (Definition 2.11),

  $$\int_{a_i(t_i, s_{-i}, Q)} a_i(s_i, s_{-i}, Q) \Theta'_i(q, Q_i) dq \geq \tilde{u} \cdot (a_i(s_i, s_{-i}, Q) - a_i(t_i, s_{-i}, Q)). \quad (5.7)$$

According to lemma (5.2) we get

  $$\int_{a_i(t_i, s_{-i}, Q)} \tilde{\Theta}'_i(q, Q_i) dq \geq \tilde{u} \cdot (a_i(s_i, s_{-i}, Q) - a_i(t_i, s_{-i}, Q)). \quad (5.8)$$

- If $a_i(s_i, s_{-i}, Q) \leq a_i(t_i, s_{-i}, Q)$ and $\tilde{u} \geq \Theta'_i(0, Q_i)$, then since $\Theta'_i(0, Q_i) \geq \Theta'_i(q, Q_i)$ for any $q$, we get

  $$\int_{a_i(t_i, s_{-i}, Q)} \Theta'_i(q, Q_i) dq \geq \Theta'_i(0, Q_i) \cdot (a_i(s_i, s_{-i}, Q) - a_i(t_i, s_{-i}, Q)). \quad (5.9)$$

According to the assumption of this case, $\Theta'_i(0, Q_i) \leq \tilde{u}$. Thus,

  $$\Theta'_i(0, Q_i) \cdot (a_i(s_i, s_{-i}, Q) - a_i(t_i, s_{-i}, Q)) \geq \tilde{u} \cdot (a_i(s_i, s_{-i}, Q) - a_i(t_i, s_{-i}, Q)). \quad (5.10)$$

- If $a_i(s_i, s_{-i}, Q) \leq a_i(t_i, s_{-i}, Q)$ and $\tilde{u} < \Theta'_i(0, Q_i)$, start from our original formula,

  $$\Theta_i(a_i(s_i, s_{-i}, Q), Q_i) - \Theta_i(a_i(t_i, s_{-i}, Q), Q_i). \quad (5.11)$$
After adding and subtracting $\Theta_i(\tilde{d}_i(\tilde{u}, Q_i))$, we get

$$\Theta_i(a_i(s_i, s_{-i}, Q), Q_i) - \Theta_i(\tilde{d}_i(\tilde{u}, Q_i)) + \Theta_i(\tilde{d}_i(\tilde{u}, Q_i)) - \Theta_i(a_i(t_i, s_{-i}, Q), Q_i).$$

(5.12)

By the definition of the truncated valuation function (Definition 2.9), and the definition of the truncated pseudo-marginal valuation function (Definition 2.11), we know that the marginal valuation function is equal to or greater than the pseudo-marginal valuation function. Thus, the previous statement is equal to or greater than the following statement,

$$\tilde{\theta}_i(a_i(s_i, s_{-i}, Q_i) - \tilde{\theta}_i(\tilde{d}_i(\tilde{u}, Q_i)) + \tilde{\theta}_i(\tilde{d}_i(\tilde{u}, Q_i)) - \tilde{\theta}_i(a_i(t_i, s_{-i}, Q), Q_i).$$

(5.13)

Represent the subtraction as an integral,

$$a_i(s_i, s_{-i}, Q) \int_{\tilde{d}_i(\tilde{u}, Q_i)} \tilde{\theta}_i'(q, Q_i) dq + \tilde{\theta}_i(\tilde{d}_i(\tilde{u}, Q_i)) - \tilde{\theta}_i(a_i(s_i, s_{-i}, Q)) + \tilde{\theta}_i(\tilde{d}_i(\tilde{u}, Q_i)) - \tilde{\theta}_i(a_i(t_i, s_{-i}, Q)).$$

(5.14)

Applying lemma (5.2) on the right integral, we get a statement that is equal to or smaller than the previous statement,

$$a_i(s_i, s_{-i}, Q) \int_{\tilde{d}_i(\tilde{u}, Q_i)} \tilde{\theta}_i'(q, Q_i) dq + \tilde{\theta}_i(\tilde{d}_i(\tilde{u}, Q_i)) - \tilde{\theta}_i(a_i(s_i, s_{-i}, Q)) + \tilde{\theta}_i(\tilde{d}_i(\tilde{u}, Q_i)) - \tilde{\theta}_i(a_i(t_i, s_{-i}, Q)).$$

(5.15)

Insert $-\tilde{u}$ to the remaining integral and add $\tilde{u} \cdot (a_i(s_i, s_{-i}, Q) - \tilde{d}_i(\tilde{u}, Q_i))$,

$$a_i(s_i, s_{-i}, Q) \int_{\tilde{d}_i(\tilde{u}, Q_i)} (\tilde{\theta}_i'(q, Q_i) - \tilde{u}) dq + \tilde{u} \cdot (\tilde{d}_i(\tilde{u}, Q_i) - \tilde{d}_i(\tilde{u}, Q_i)) + \tilde{u} \cdot (\tilde{d}_i(\tilde{u}, Q_i) - a_i(t_i, s_{-i}, Q)).$$

(5.16)

Eliminate positive and negative $\tilde{u} \cdot \tilde{d}_i(\tilde{u}, Q_i)$,

$$a_i(s_i, s_{-i}, Q) \int_{\tilde{d}_i(\tilde{u}, Q_i)} (\tilde{\theta}_i'(q, Q_i) - \tilde{u}) dq + \tilde{u} \cdot (a_i(s_i, s_{-i}, Q) - a_i(t_i, s_{-i}, Q)).$$

(5.17)

Applying equation (2.18), we get a statement that is equal to or smaller than the
previous statement,
\[
\hat{d}_i(\tilde{u}^+,Q_i) \int \tilde{d}_i(\tilde{u},Q_i) (\tilde{\theta}'_i(q,Q_i) - \tilde{u})dq + \tilde{u} \cdot (a_i(s_i, s-i, Q) - a_i(t_i, s-i, Q)). \tag{5.18}
\]

**Theorem 5.2.** If a client \( i \) submits a truthful multi-bid \( s_i \), then every other multi-bid, \( t_i \) (truthful or not) corresponds to an increase in client utility that is less than
\[
\hat{d}_i(\tilde{u},Q_i) \int \tilde{d}_i(\tilde{u},Q_i) (\tilde{\theta}'_i(q,Q_i) - \tilde{u})dq. \tag{5.19}
\]

**Proof.** In theorem 5.1 we showed that the valuations are bounded. Now, we begin by showing that the payment of client \( i \) is bounded. Formally, we show that
\[
c_i(s,Q) \leq \tilde{u} a_i(s,Q). \tag{5.20}
\]

According to the definition of the pricing rule (2.20),
\[
c_i(s,Q) = \sum_{j \neq i, i \in \text{Clients}} a_j(0,s-i) \int \tilde{\theta}'_i(q,Q_i) dq. \tag{5.20}
\]

Applying lemma (5.2), we get a statement that is greater or equal than the previous statement,
\[
\sum_{j \neq i, i \in \text{Clients}} \tilde{u} \cdot (a_j(0,s-i) - a_j(s)). \tag{5.21}
\]

Since the pricing rule is according to the exclusion compensation principle, we know that,
\[
a_i(s,Q) = \sum_{j \neq i, i \in \text{Clients}} (a_j(0,s-i,Q_i) - a_j(s,Q_i)). \tag{5.23}
\]

Applying it to the previous statement we get an equal statement,
\[
\tilde{u} \cdot a_i(s,Q). \tag{5.22}
\]

Thus, we get that,
\[
c_i(s,Q) \geq \tilde{u} \cdot a_i(s,Q). \tag{5.23}
\]

Now we put it all together, applying equation (5.23) and Theorem (5.1) to the utility function (2.21). First, according to the definition of the utility function,
\[
U_i(s,Q) - U_i(t_i,s-i,Q) = \Theta_i(a_i(s,Q),Q_i) - \Theta_i(a_i(t_i,s-i,Q),Q_i) + c_i(s,Q) - c_i(t_i,s-i,Q). \tag{5.24}
\]

According to Theorem (5.1) we get the following statement that is equal to or smaller
than the previous statement,
\[
\tilde{d}_i(\tilde{u},Q_i) \int_{\tilde{d}_i(\tilde{u}^+,Q_i)} (\tilde{\theta}'_i(q,Q_i) - \tilde{u})dq + \tilde{u} \cdot (a_i(s_i,s_{-i},Q) - a_i(t_i,s_{-i},Q)) + c_i(s,Q) - c_i(t_i,s_{-i},Q).
\] (5.25)

Applying equation (5.23), we get a statement that is equal to or smaller than the previous statement,
\[
\tilde{d}_i(\tilde{u},Q_i) \int_{\tilde{d}_i(\tilde{u}^+,Q_i)} (\tilde{\theta}'_i(q,Q_i) - \tilde{u})dq + \tilde{u} \cdot (a_i(s_i,s_{-i},Q) - a_i(t_i,s_{-i},Q)).
\] (5.26)

Finally, eliminating, \(\tilde{u} \cdot (a_i(s_i,s_{-i},Q) - a_i(t_i,s_{-i},Q))\), we get a statement that is equal to or smaller than the previous statement,
\[
- \tilde{d}_i(\tilde{u},Q_i) \int_{\tilde{d}_i(\tilde{u}^+,Q_i)} (\tilde{\theta}'_i(q,Q_i) - \tilde{u})dq.
\] (5.27)

**Theorem 5.3.** If a client \(i\) submits a truthful multi-bid \(s_i\), then every other multi-bid, \(t_i\) (truthful or not) corresponds to an increase in client utility that is less than \(C_i\). Formally, we show that
\[
U_i(t_i,s_{-i},Q) - U_i(s,Q)dq \leq \max_{1 \leq m < M_i} \left\{ \int_{q_{i,m}^m}^{q_{i,m+1}} (\Theta'_i(q) - \Theta'_i(q_{i,m}))dq \right\}
\] (5.28)

**Proof.** According to Theorem 5.2
\[
U_i(t_i,s_{-i},Q) - U_i(s,Q)dq \leq \tilde{d}_i(\tilde{u},Q_i) \int_{\tilde{d}_i(\tilde{u}^+,Q_i)} (\tilde{\theta}'_i(q,Q_i) - \tilde{u})dq.
\] (5.29)

For every client \(i\), there are 2 cases:

- \(\exists m, 0 \leq m \leq M_i, \tilde{u} = p_{i,m}^m\). Thus, by definition \(\tilde{u}^+ = p_{i,m+1}^m\). Thus, we continue
from 5.29 as follows:

\[
\tilde{d}(\tilde{u}, Q_i) \int \tilde{d}(\tilde{u}, Q_i) (\tilde{\theta}'(q, Q_i) - \tilde{u}) \leq \int \tilde{d}(p_i^m, Q_i) (\tilde{\theta}'(q, Q_i) - \tilde{u}). \tag{5.30}
\]

By definition \(p_i^m = \Theta'_i(q_i^m)\). Thus,

\[
\tilde{d}(p_i^m, Q_i) \int \tilde{d}(p_i^m+1, Q_i) (\tilde{\theta}'(q, Q_i) - p_i^m) = \tilde{d}(p_i^m+1, Q_i) (\tilde{\theta}'(q, Q_i) - \Theta'_i(q_i^m)). \tag{5.31}
\]

- Otherwise, the truncated pseudo-market clearing price \(\tilde{u}\) does not appear in the prices of the client’s multi-bid. Thus, by definition \(\tilde{u}^+ = \tilde{u}\) and we get that \(\tilde{d}(\tilde{u}^+, Q_i) = \tilde{d}(\tilde{u}, Q_i)\). Thus,

\[
\tilde{d}(\tilde{u}, Q_i) \int \tilde{d}(\tilde{u}, Q_i) (\tilde{\theta}'(q, Q_i) - \tilde{u}) \leq \int \tilde{d}(\tilde{u}, Q_i) (\tilde{\theta}'(q, Q_i) - \tilde{u}). \tag{5.32}
\]

Thus, we conclude that:

\[
\tilde{d}(\tilde{u}, Q_i) \int (\tilde{\theta}'(q, Q_i) - \tilde{u}) \leq \max_{1 \leq m < M_i} \left\{ \int (\Theta'_i(q) - \Theta'_i(q_i^m)) dq^m \right\}. \tag{5.33}
\]

**Conclusion 5.3.** Since submitting a truthful multi-bid is the best action up to a difference of \(C_i\) for client \(i\) independently of other clients’ actions, then we can say that such a multi-bid is an ex-post \(C_i\)-dominant strategy for \(i\), because even if clients were told the true allocation after bidding, they would not regret their actions, up to \(C_i\) (see [HKDMT04]). Thus, if all clients submit truthful multi-bids, we get an ex-post \(K\)-Nash equilibrium, where \(K = \max_{i} \{C_i\}\), in the sense that no client could have improved his utility by more than \(K\) otherwise.
Chapter 6

Simulation Results

We evaluated MDQ using the SimPy [Mat08] discrete-event simulator. We use SimPy to compare the social welfare achieved by our algorithm against other schedulers and pricing schemes. In SimPy a queueing algorithm is implemented by selecting which flow’s packets are serviced next. The packets are generated according to each flow’s resource profile with standard deviation set to one-tenth of the mean and according to the standard distribution. In our simulations clients are also service providers, with their own customers. Thus a client’s valuation for bandwidth allocation is also affected by variables unrelated to performance gained by additional bandwidth, such as the market price for the services the client offers and the types of customers that request those services.

6.1 Alternative Queueing Algorithms and Pricing Schemes

We compared MDQ against the following combinations of queueing algorithms and pricing schemes:

**Token Bucket Filter with Flat Fee (TBFFF)**, where each client is guaranteed an equal constant rate on the output link. The rate of each client is equal to the total capacity divided by the number of clients. This is the simplest approach. The network provider is unaware of client valuations, distributing bandwidth equally while preventing clients from sharing excess bandwidth.

**Weighted Fair Queueing with Flat Fee (WFQFF)**, where each client has an equal share of the output link. Similarly to TBFFC, the provider is unaware of client valuations, which results in equal shares for all clients. However, unlike TBFFC, the provider allows clients to share excess bandwidth proportionally to their shares of the output link.

**Token Bucket Filter with multi-bid Auction (TBFMB)**, where each client is guaranteed a constant rate on the output link based on the allocation rule of a multi-bid auction applied to the fully backlogged truncated valuations. Using the multi-bid auction, the provider can learn the clients’ pseudo-valuations and allocate bandwidth...
such that social welfare is maximized when all clients are backlogged. Whenever a valuation is modified by a client, the client sends new bids, which trigger a new allocation, and similarly to TBFFF, the provider prevents clients from sharing excess bandwidth.

**Weighted Fair Queueing with multi-bid Auction (WFQMB)**, where each client has a share according to the allocation rule of the multi-bid auction applied to the fully backlogged truncated valuations. Whenever a valuation is modified by a client, the client sends new bids, which triggers a new allocation. By using the multi-bid auction, the provider can learn the client valuations and allocate bandwidth such that social welfare is maximized when all clients are backlogged. However, this combination of a queueing algorithm with the multi-bid auction is not truthful, because the allocation might be sub-optimal whenever there is excess bandwidth. Excess bandwidth occurs when some client’s transmission rate falls below his allocation, and a sub-optimal allocation occurs when excess bandwidth is redistributed by WFQ according to the multi-bid allocation rule, which assumed that all clients are backlogged. For the purpose of this comparison we chose to ignore this issue.

**Optimal (OPT)** allocation, where the full client valuation functions, changes made to them, and the packet arrival times are known in advance. The queueing algorithm that maximizes the social welfare serves as an upper bound for all other queueing algorithms.

### 6.2 Main Results:

Our main results are as follows:

1. MDQ and queueing algorithms combined with the multi-bid auction provide improved social welfare compared to queueing algorithms combined with flat fee pricing.

2. MDQ is work-conserving and it maximizes social welfare even when not all clients are backlogged, sharing the additional bandwidth according to the valuation functions and providing improved social welfare compared to queueing algorithms combined with multi-bid auction pricing.

3. In MDQ there is a trade-off between the social welfare of the allocation and the run time of a single round of the algorithm. This trade-off allows for an efficient implementation at the cost of some social welfare.

### 6.3 Comparing Social Welfare

We evaluated the social welfare obtained by using MDQ vs. each of the other methods listed in section 6.1 using two sets of valuation functions in two experiment scenarios. The basic experiment scenario consists of calculating the social welfare where the
number of clients gradually increases from one to 19 and each client handles ten different valuations one after the other. For each client the order of the valuations was chosen at random from a distribution of one high demand valuation, two medium demand valuations, and seven low demand valuations. This distribution is inspired by the Pareto distribution, which is a well known economic distribution [FBYS16][ABYST11]. When we repeat a scenario with more clients, then clients who previously participated had the same order of valuations in both scenarios. We repeated the experiment in two variations. In the first, we used the valuations that appear in Figure 6.1a and all clients are backlogged. In the second, we used the valuations that appear in Figure 6.2a, but only the high demand client was backlogged all the time, the medium demand client was backlogged for half of the time, and the low demand client was backlogged for one percent of the time. Note that the low demand client bid the highest price for up to ten percent of available bandwidth, but did not bid for any additional bandwidth.

**Payments:** In all experiments, clients using MDQ paid between 0.16 to 0.6 of their valuation for actual obtained bandwidth. However, this is not the case for the other algorithms. In the case of Flat Fee, all clients paid a fixed flat sum regardless of their valuation or current requirements for bandwidth. Sometimes this sum was higher than the client’s valuation; however, clients cannot avoid this cost unless they are willing to decline to provide services when demand is low [BYABYT16]. In the case of algorithms combined with a multi-bid auction, clients pay according to the exclusion compensation principle with the fully backlogged truncated valuations. However, when some clients are not backlogged or even inactive, then using TBFMB, these clients are charged a price that is higher than their actual truncated valuations. Moreover, payments according to the exclusion compensation principle might not reflect the allocation of the work-conserving multi-bid algorithm WFQMB, because the excess bandwidth from inactivity is redistributed sub-optimally, as discussed above.

**First Scenario - Flat Fee vs. Multi-Bid Auction:** In the first scenario, all queueing algorithms combined with multi-bid auctions allocate bandwidth correctly according to the valuation of new clients. However, queueing algorithms combined with flat fee pricing cannot distinguish the new client valuation from any other client valuation, leading to reduced social welfare. In Figure 6.1b, we can see how a combination of the multi-bid auction with queueing algorithms improves the social welfare by up to 4.43x over combinations of flat fee pricing and queueing algorithms.

**Second Scenario - Truncated Valuations:** In the second scenario, only MDQ is able to keep up with the optimal social welfare in this scenario, mainly because all the other queueing algorithms have wrong weights due to inactivity of clients. In Figure 6.2b, we can see that MDQ improves the social welfare by up to 5.3x over combinations of the multi-bid auction with queueing algorithms. For combinations of flat fee pricing with queueing algorithms (TBFFF and WFQFF), the wrong weights result in reduced social welfare, especially when the queueing algorithm is not work-conserving (TBFFF). This problem is even worse for combinations of the multi-bid
(a) Valuation functions

Figure 6.1: First scenario

(b) Social welfare obtained by an increasing number of clients. Multi-bid based queueing algorithms improve the social welfare by up to 4.43x over flat fee based queueing algorithms.

Figure 6.1: First scenario
(a) Valuation functions.

(b) Social welfare obtained by an increasing number of clients. MDQ improves the social welfare by up to x5.3 over combinations of the multi-bid auction and queueing algorithms.

Figure 6.2: Second scenario
Figure 6.3: Using the valuations in Figure 6.2a. The graph presents the social welfare obtained by one backlogged high demand client, one medium demand client who is backlogged half the time, and an increasing number of inactive low demand clients. MDQ improves the social welfare infinitely over combinations of the multi-bid auction with queueing algorithms that allocate all available bandwidth to the low demand clients.

Extreme Case of Truncated Valuations: We observe that if low demand valuation clients were to cease all their activity completely instead of being backlogged for one percent of the time, then the results would be drastically different. Using the valuations of the second scenario as presented in Figure 6.2a, we tested what happens when there is a single high demand valuation client and a single medium demand valuation client, and an increasing number of low demand valuation clients. The high demand client is constantly backlogged, the medium demand client is backlogged for half of the time, and the low demand clients are completely inactive. During this scenario there are no changes in clients’ valuations.

In Figure 6.3, we can see that MDQ improves social welfare infinitely over combinations of the multi-bid auction with queueing algorithms. This happens because once there are ten low demand clients, each of whom offers the highest price for up to ten percent of total bandwidth, then all combinations of the multi-bid auction with queueing
Figure 6.4: The trade-off between the CPU run time of the simulation and the loss of social welfare from changes in the $MaxBatchSize$. As the CPU run time of the simulation increases, the social welfare lost is less significant. The optimal $MaxBatchSize$ is obtained at 1100 bytes. Starting from a batch size of 4800 bytes, even a round of MDQ running on the Python simulator executes in less than one second.

algorithms assign all the available bandwidth to the low demand clients, leaving medium and high demand clients with nothing. However, since low demand clients are inactive, the result is complete link inactivity even though high and medium demand clients have packets to send.

Starting from 6 low demand clients, the WFQMB algorithm assigns a zero weight to the medium demand client, thus assigning all the bandwidth to the high demand client. This results in the same social welfare when there are between six and ten low demand clients. Starting from ten low demand clients, the WFQMB algorithm ignores high and medium demand clients, because their weight is zero. The WFQFF algorithm always assigns an equal weight to the medium and high demand clients, resulting in suboptimal but constant social welfare. The TBFFF algorithm splits the available rate equally in a non-work-conserving manner between all clients. As a result, high and medium demand clients get decreasing amounts of bandwidth as the number of clients increases.

### 6.4 Social Welfare and Run-time Trade-Off

In MDQ there is a trade-off between the social welfare of the allocation and the run time of a single round of the algorithm. This trade-off is based on the maximal amount of batching ($MaxBatchSize$) in Dequeue. On the one hand, when $MaxBatchSize$
is increased, then more bytes are sent based on a single iteration of the inner loop of Dequeue. Thus less CPU processing is required for each byte of data transmitted. On the other hand, as expressed in Theorem 4.1, when MaxBatchSize is increased, then the social welfare obtained by the Dequeue allocation is decreased by at most $\theta_{\text{max}}(\text{MaxBatchSize})$. This trade-off allows for an efficient implementation at the cost of social welfare.

To demonstrate this trade-off, we performed 20 simulations of MDQ with MaxBatchSize increasing from 500 to 15000 bytes in steps of 100 bytes, where in each simulation we used ten multi-bids constructed from the uniform distribution with the maximum price of $2 per byte. Each multi-bid was constructed by starting with a price of $2 per byte and gradually decreasing the price by a random amount between 0 and 1 and incrementing the quantity by a random amount until the price dropped to zero. For each such simulation of MDQ with a certain MaxBatchSize, we measured the CPU run-time of 50 rounds of the simulation and the maximum social welfare that could be obtained if there was no batching.

We observe that the relationship between the run-time of the simulation rounds and MaxBatchSize is not linear but rather inverse. According to our algorithm, each round sends a fixed number of bytes (Budget) regardless of MaxBatchSize. Increasing the batch size will decrease the total number of bids which require processing to transmit Budget bytes. The run-time is dominated by the number of bids processed. Overall, the run-time is determined by the formula $\text{Budget}/\text{MaxBatchSize}$. Since, Budget is constant, we get that the graph resembles $1/x$.

We present the results of these simulations in Figure 6.4. In this figure we see that the decrease in social welfare becomes less significant as the run-time of the simulation increases (due to the use of a lower MaxBatchSize). We emphasize two points in the graph. First, the typical knee batch size for our simulator is 1100 bytes. Second, using a batch size above 4800 bytes results in each round of MDQ completing within one second. Finally, in Figure 6.5, we show a linear regression of the run-time and an inverse of the MaxBatchSize using a sample size of 20. The results are consistent with the hypothesis that the run time is inversely proportional to the MaxBatchSize.
Figure 6.5: A linear regression of the run-time and an inverse of the MaxBatchSize. The result for a sample size of 20 has a Pearson product-moment correlation coefficient of 0.58.

Figure 6.6: Run-time and inverse MaxBatchSize
Chapter 7

Applications of MDQ

MDQ schemes could be applied to other problems of queueing/scheduling in multi-tenant systems where resources are scarce and pricing could be used to derive client preferences to optimize social welfare. This is the case for network and cloud service providers whose clients compete for bandwidth on an outgoing link. MDQ could also be used for QoS in distributed messaging systems such as Kafka [WKS+15]. In Kafka, messages are sent over TCP, and QoS is enforced by the broker at the application/transport layer by reducing the rate of message processing for a client who is violating its QoS policy. Kafka producers push messages towards the broker, to be written into the distributed log of messages. Meanwhile, Kafka consumers operate in a polling mode, where they request to read a message starting at an offset from the broker. For Kafka, MDQ can be used by the broker at the application layer to choose which client log message to service at any given moment. By operating at the application layer, the MDQ pricing scheme can focus on charging for the application data the client sent, ignoring TCP retransmissions, acknowledgments and other lower protocol layer messages. This will maximize the social welfare of client applications, regardless of the transport layer and without any modification to the existing Kafka API, wire protocol or architecture.

Another application is storage in general and specifically Network Attached Storage (NAS), where clients compete for available disk read/write bandwidth. By using MDQ at the application layer it is possible to maximize the social welfare derived by clients from storage.
Chapter 8

Conclusion and open questions

MQD efficiently allocates bandwidth to selfish black-box clients while maximizing their aggregate benefit. MDQ can operate at different layers of the software stack to maximize social welfare at different granularities and for different applications. MDQ is inspired by the multi-bid auction and work-conserving queueing algorithms and it provides the benefits of both worlds. MDQ works by sharing bandwidth among flows according to their truncated pseudo-valuations, using multi-bids obtained from clients during admission and actual traffic passing through the traffic scheduler. The traffic scheduler of MDQ consists of rounds, where each round is equivalent to an incentive compatible auction.

MDQ maximizes social welfare even when some client flows are idle, improving overall system utilization, while using only $O(\log(n))$ processing time per packet to perform packet scheduling. MDQ avoids the high signaling cost of executing VCG auctions in quick succession, while still providing incentive compatibility.

Payments in each round of MDQ are computed based on exactly $n$ simulations of traffic scheduling, which are significantly more expensive to compute compared to packet scheduling itself. Moreover, in an unstable state, the allocation calculated by the simulation can be higher or lower than the client’s multi-bid by up to $MaxBatchSize$ bytes. Thus, payments can be up to $b_{\max}(MaxBatchSize)$ higher or lower than the client’s valuation.

We suggest a few ideas for future research:

1. An exploration of the algorithm’s incentive compatibility in an unstable state could provide more insight into such algorithms, and their game-theoretic properties.

2. An extension of the algorithm’s analysis to a full network could reveal the interactions between valuation-aware and non-valuation-aware queueing algorithms [SK09][NML08].

3. An efficient implementation in software and hardware could be used to evaluate the benefits of MDQ in today’s complex multi-tenant data centers and clouds.
Appendix A

Ginseng for Bandwidth

In this section we describe our efforts and experience of working towards a Ginseng-BW system for bandwidth allocation using the Mellanox Connectx-3 Pro adapter. We worked on Ginseng-BW before moving to the Market Driven Queueing algorithm. Ginseng-BW auctions bandwidth on cloud computer platforms. It uses a repeated multi-bid mechanism [MT04], which is incentive-compatible.

As with MDQ, and as in other bandwidth pricing mechanisms such as those of Kelly [Kel97], Lazar and Semert[LS98] and Maillé and Tuffin[MT04], in Ginseng-BW we assume that the valuation functions are strictly concave non-decreasing and continuously differentiable functions. We further assume that, as long as the valuation is strictly increasing, it must also be strictly concave. These properties are required to compute the allocation rule efficiently and to provide valid valuation functions. Additionally, for incentive-compatibility of the iterative auction, we assume that all clients are myopic, i.e., do not plan several rounds ahead, just one round ahead.

A.1 System Overview

We enhanced a system called Ginseng [Pos13][FBYS16] to auction bandwidth using a multi-bid auction [MT04]. The architecture of Ginseng-BW is described in Figure A.1. Ginseng-BW is composed of a server, running a number of virtual machines (VMs) that can be referred to as guests. The server OS, referred to as the host or hypervisor, also runs a process called Ginseng-BW. This process manages bandwidth allocation to the guests according to the results of a repeated auction in which the guests can participate in order to win extra bandwidth, on top of their base bandwidth allocation.

A.1.1 Host Side Design

Based on the incentive compatibility property of the multi-bid auction, it is in the best interest of clients to provide bids representing data points of their valuation functions. Ginseng operates in rounds; In each round Ginseng announces a new auction, collects the bids from the guests, and notifies the clients of the auction’s result. The flow of
an auction round is described in Section A.1.2. Ginseng communicates with the guests via an out-of-band communication protocol which relies on MOM [Lit] and does not consume any resources from the bandwidth allocated to the client VM. Ginseng is a privileged process on the host operating system and it can use one of many possible bandwidth controllers, which are described in Section A.1.3.

### A.1.2 Ginseng Auction Flow

In Ginseng, bandwidth allocations change every round. The guest rents bandwidth for the full duration of one round. Here we describe one auction round and compare the memory, LLC and bandwidth Ginseng implementations.

A round in Ginseng for memory [Pos13] takes as much as 12 seconds, where 3 seconds are used to wait for clients to provide bids, 1 second is used to compute the auction results, and an 8-second adjustment period allows for clients to prepare for a change in allocation. However, with bandwidth, similarly to LLC [FBYS16], we wanted to increase the responsiveness of the system by running the multi-bid auction [MT04] every second, based on the available bids. This change had an impact on the structure of the auction. Towards a shorter auction, we removed the announcement stage entirely, allowing clients to submit multi-bids at any time, with new multi-bids influencing the next auction round. We also removed the adjustment period entirely, since unlike memory, reducing bandwidth has no immediate side-effects. Thus, guests are notified immediately regarding auction results and the allocation is effective immediately after it is computed. Overall, the auction is executed once every second based on the latest multi-bids provided by clients, and the auction results are reported immediately.
Initialization. Not all bandwidth is allocated based on an auction; A minimal amount of bandwidth is reserved for each guest as a basic level of service, similarly to Q-Clouds [NKG10], Gingko [GHDS11], Ginseng for memory [Pos13] and Ginseng for LLC [FBYS16]. However, this basic level of service is not necessary and it could be removed entirely without any impact on Ginseng for bandwidth. During initialization Ginseng declares the total amount of bandwidth available for auction.

Bidding Interested clients provide multi-bids for bandwidth. The multi-bids are used to represent the clients’ willingness to pay for bandwidth. Previous multi-bids provided by clients are stored at the auctioneer, and clients can provide updates to these bids or reset them with new multi-bids when their valuation for bandwidth has changed. The accuracy of the allocation by the auction increases as the number of multi-bids near the market clearing price increases. To increase accuracy clients are encouraged to provide a large number of multi-bids; however, communicating all of these multi-bids might require several rounds. Thus, clients need to split their multi-bids and to send them gradually to the auctioneer.

A.1.3 Bandwidth Controllers

We experimented with a number of possibilities for controlling the bandwidth allocation in both software and hardware.

In Ginseng-BW, we use the Hierarchical Token Bucket (HTB) [Dev], which is today’s state-of-the-art work-conserving classful queueing discipline. However, we could not use it fully because we cannot rely on HTB to redistribute bandwidth according to valuations, as we do in MDQ. We therefore configured HTB to prevent sharing entirely.

Using HTB has several benefits. It is implemented in software, so it allows for easy migration of VMs. In addition, software queueing disciplines can support any packet encapsulation, which is particularly important in today’s data-center environment. Finally, software queueing can benefit from interposition, allowing the hypervisor to offer services such as firewalls and encryption to client VMs.

However, the performance and scalability of HTB are significantly limited due to the number of locks that are required to operate the hierarchical queueing discipline, where each lock wastes a significant amount of the budget available for line rate transmission. Moreover, HTB has probably not been tested with a 40Gbps NIC.

The Mellanox Connectx3-Pro [Mel] can provide QoS for clients using SRIOV [SIG] based on the ETS standard [IEE]. The SRIOV technology is a hardware based virtualization solution that improves both performance and scalability by allowing a single physical PCIe function to expose multiple virtual functions (VF). Devices that support SRIOV can present themselves to the system as multiple virtual devices. Thus each

1It is not possible to configure bandwidth beyond 34359Mbps.
guest VM can get direct access to hardware resources. By using ETS, the hypervisor can configure how to share bandwidth among different virtual functions. Hardware QoS can provide high performance, with almost no overhead for software. However, the HW interface has limited flexibility. First, the granularity of bandwidth is limited to Mbps, while with HTB it is in bits per second. Second, there are only 8 possible traffic classes in Connectx-3 Pro. Third, to use QoS with SRIOV, hardware must be able to identify with certainty that a packet originated from the VM. To that end, unique VLANs must be configured for each VM. Without the use of VLANs, hardware QoS is not supported. Finally, interposition with SRIOV is not possible and VM migration with SRIOV is complicated, since each physical NIC has a different interface.

<table>
<thead>
<tr>
<th>Sender/Receiver</th>
<th>Local</th>
<th>Remote</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td>20.1</td>
<td>11.2</td>
</tr>
<tr>
<td>Remote</td>
<td>19.9</td>
<td>13.7</td>
</tr>
</tbody>
</table>

Table A.1: Average bandwidth (Gbps) from running netperf between 2 machines connected back-to-back. The bandwidth changes according to the NUMA node on which netperf is running. Running the receiver on the NUMA node connected to the NIC improves throughput by up to 35%.

In addition to controlling bandwidth, NIC IO interrupts can significantly degrade performance. We experimented with 3 approaches for handling interrupts. First, we moved all the NICs interrupts to a separate core. This core was a bottleneck when the system encountered heavy load. Moreover, the NUMA topology (see figure A.2) of the system and the position of the IO core can influence potential guest throughput, because some guest traffic will have to pass over the QPI bus, while other guests can access the NIC directly. In Table A.1, we show the results of running netperf between 2 machines connected back-to-back. When the netperf receiver runs on the NUMA node connected to the NIC, throughput improves by up to 35%.

Second, we attempted to use a few IO cores on each NUMA node, where each IO core will handle the interrupts of some guests on that NUMA node. This is similar to ELI [GAH+12] and ELVIS [HGL+13]. With this approach the IO cores are not the bottleneck; however, they cannot be utilized for anything else due to the frequent interrupts that occur on these cores. Also, running a client VM on these cores would be unfair towards the clients running on these cores.

Finally, it is possible to assign client interrupts to the core on which their machine is running. Thus, clients incur an exit for every interrupt they receive. With this approach, no cores are dedicated for IO, allowing for more reliable accounting of client IO. However, NUMA locality is still a problem. To overcome the issue of NUMA locality, we could attach a NIC to each NUMA node and run Ginseng on each NUMA node to allocate resources on that NIC.

---

2interposition - providing hypervisor network services to guest machines.
Figure A.2: System with 1 NIC and 2 NUMA nodes. IO from NUMA node 2 has reduced performance.
Bibliography


Aiming to optimize the scheduling of deliveries, the thesis presents a novel scheduling algorithm that utilizes a token bucket filter to manage delivery requests efficiently. The algorithm ensures that delivery requests are served in a way that maximizes utilization of available resources. The token bucket filter is used to control the rate at which delivery requests are processed, allowing for fair allocation of delivery slots.

The thesis also introduces a novel concept of weighted fairness, which takes into account the differing importance of various delivery requests. This is achieved by assigning weights to each request based on its significance, ensuring that more important requests are served first. The weighted fairness approach is implemented using a weighted token bucket filter, which enables the system to dynamically adjust the delivery rate based on the weights assigned to each request.

The overall goal of the thesis is to develop a scheduling algorithm that not only ensures fair allocation of delivery slots but also maximizes the overall efficiency of the delivery process. This is achieved through the use of sophisticated scheduling techniques and algorithms that are designed to work in real-time, allowing for dynamic adjustment based on the current state of the system.

The implementation of the algorithm is tested in various scenarios, and the results show significant improvements in delivery efficiency and fairness. The thesis also discusses the implications of these findings for future research in the field of logistics and scheduling.
токיירה

ברשתות, רוחב פס הוא האחראי על ייצור הפסקים של הצלחתה. סקיפ שהירח לשגד הגינה לשכבות
במתחוור, בכדיגל יירה את השפעת ברוחב פס העומד לרשותם. בנגנונכם של הירח לשגד
רזרח נколо lesbian יירה לשמידה. קבינהتنظימה של זירה בית מתח אחר זה לתחום
רוחב פס שלושה ש darf לא מאפרת את הלקחות או הרוחב לפלאות חברה, מתוכ
שומע רצייב או.

יש נשים ספורטיביות למתחוור רוחב פס: מקסימום היעילות של הלקחות, מקסימום הרוחב של הרלקחות
ומקסימום הרוחב של גידולי הنتشر. אנתגר בחרים של מקסימום היעילות של הרלקחות
ומקסים של הלקחות. קולק😂 מרגיע את הרוחב של גידולי הלקחות, הרוחב של גידולי הלקחות
מה מופנים של גידולי רוחב הפס. ארוגים מדרידים את صفיע הרוחב בשיער הקצות
לאל חוויות תוצרת הפנים לשגרה, גיביתם עמדה במקסימום.

קנה "הרוחב הצקות" מצורף בשק הבית.

 неск היית שירח הוא גודל משנינו לתוכים את הרלקחות של הלקחות של רוחב הפס, לעד מי
רוחב פס הניהたり ללקחות שבאילים והן יותם בחולות פעילויות של מקסימום של גידולים, יותים חום
ברוחב פס של שם של מדרידים בחקלאים, יותים חום לחופם, יותים חום רוחב פס פ
ידנית "הרוחב הצקות" מצורף בשק הבית.

скаיפ שהירח לשגד מפעלים נוספים את הרלקחות של הלקחות של רוחב הפס, לעד מי
מחק את תואר של ליקות פאקטים (PACKETS), בходит צהוב וירובים. עניין של הרלקחות
סיכ ואלו שקט קיים בתקコスト שאותו
 IEnumerable לשחרור הפנים שירח הוא השתק. בנהית, אט שונה, עליון פגショ
לאל חשים, אט של שחק משחק במקסימום של גידולים, רוחב הפס פ
ידנית "הרוחב הצקות" מצורף בשק הבית.

앗 ספק שירח הוא גודל משנינו לתוכים את הרלקחות של הלקחות של רוחב הפס, לעד מי
ם יותם בחולות פעילויות של מקסימום של גידולים, יותים חום
רוחב פס של שם של מדרידים בחקלאים, יותים חום לחופם, יותים חום רוחב פס פ
ידנית "הרוחב הצקות" מצורף בשק הבית.

מחק את תואר של ליקות פאקטים (PACKETS), בходит צהוב וירובים. עניין של הרלקחות
סיכ ואלו שקט קיים בתקakest שאותו
 IEnumerable לשחרור הפנים שירח הוא השתק. בנהית, אט שונה, עליון פגש
לאל חשים, אט של שחק משחק במקסימום של גידולים, רוחב הפס פ
ידנית "הרוחב הצקות" מצורף בשק הבית.

ם יותם בחולות פעילויות של מקסימום של גידולים, יותים חום
רוחב פס של שם של מדרידים בחקלאים, יותים חום לחופם, יותים חום רוחב פס פ
ידנית "הרוחב הצקות" מצורף בשק הבית.

Tambah מפקח על שחרור הפנים של הלקחות של הלקחות של רוחב הפס, לעד מי
מחק את תואר של ליקות פאקטים (PACKETS), בходит צהוב וירובים. עניין של הרלקחות
סיכ ואלו שקט קיים בתקakest שאותו
 IEnumerable לשחרור הפנים שירח הוא השתק. בנהית, אט动人, עליון פגש
לאל חשים, אט של שחק משחק במקסימום של גידולים, רוחב הפס פ
ידנית "הרוחב הצקות" מצורף בשק הבית.

ם יותם בחולות פעילויות של מקסימום של גידולים, יותים חום
רוחב פס של שם של מדרידים בחקלאים, יותים חום לחופם, יותים חום רוחב פס פ
ידנית "הרוחב הצקות" מצורף בשק הבית.

מחק את תואר של ליקות פאקטים (PACKETS), בходит צהוב וירובים. עניין של הרלקחות
סיכ ואלו שקט קיים בתקakest שאותו
 IEnumerable לשחרור הפנים שירח הוא השתק. בנהית, אט动人, עליון פגש
לאל חשים, אט של שחק משחק במקסימום של גידולים, רוחב הפס פ
ידנית "הרוחב הצקות" מצורף בשק הבית.

ם יותם בחולות פעילויות של מקסימום של גידולים, יותים חום
רוחב פס של שם של מדרידים בחקלאים, יותים חום לחופם, יותים חום רוחב פס פ
ידנית "הרוחב הצקות" מצורף בשק הבית.

מחק את תואר של ליקות פאקטים (PACKETS), בходит צהוב וירובים. עניין של הרלקחות
صيدיאבאיר בטן, יותים חום��ה שירח הוא השתק. בנהית, אט动人, עליון פגש
לאל חשים, אט של שחק משחק במקסימום של גידולים, רוחב הפס פ
ידנית "הרוחב הצקות" מצורף בשק הבית.
המחקר בוצע בחנאות פ הרב, משה אורט ואockey בריוחד ורפורם אסק שוסט, בפכתלמה תמריע קר.

המחשוב

הכרת תודה לטכניון, מלואנוק טכנולוגיה,CKER פי על מיכומ ומehler יד.
ניהול תורמים מוגבה שוק

היבורים על מקרר

לפי מילוי חלקי של הדרישות קבוצת החוגרא מעסיק לעידוד
במודעי המחשב

بورיס פייסנגי

רגוש לצלט הcrastנגי ... מנין סטנגלגון ישראל
תמדח ותעש"י הורף יולי 2017
ניהול תורמים מתנה שוק

בורייס פיסנני