Utility-Based Bandwidth Allocation and Link Scheduling in Wireless Networks: Linear Programming and Market-Oriented Approaches

A Thesis presented

by

Qicheng Ma

to

The Department of Computer Science

and

The Department of Economics

in partial fulfillment of the honors requirements

for the degree of

Bachelor of Arts

Harvard College

Cambridge, Massachusetts

April, 2006

Thesis Advisors:

Professor David Parkes

Professor Matt Welsh

Professor Jerry Green

Qicheng Ma

Utility-Based Bandwidth Allocation and Link Scheduling in Wireless Networks: Linear Programming and Market-Oriented Approaches

Abstract

In this thesis, we address the problem of bandwidth allocation and link scheduling in order to maximize aggregate user utilities in wireless ad-hoc networks. Unlike previous works that aimed to maximize aggregate throughput or some artificial utility metrics, our goal will be to optimize the true aggregate utility as reported by the applications themselves.

We first develop and implement two centralized scheduling algorithms: The OPT algorithm solves the optimal TDMA schedule directly by formulating an Integer Program and invoking the CPLEX optimizer. The ORL-CSMA algorithm first solves a relaxed version of the full optimization problem to obtain an upper-bound to the optimal schedule, then employs a best-effort CSMA scheme to ensure schedulability. Moreover, we develop a market-oriented approach MARKET with a tatonnement process and demonstrate its ability to effectively price bottleneck resource in the network and thereby approximate the optimal solution with much reduced complexity.

Finally we present detailed study of the behavior of MARKET algorithm, and performance comparisons between all three algorithms developed, as well as the NAIVE-CSMA scheduler. Our experimental results suggest that: 1) OPT, though able to achieve the most amount of utility, is computationally intractable in practice; 2) ORL-CSMA approximates the optimal results well in practice within reasonable run time; 3) MARKET offers further approximation of the optimal solution, with even less runtime complexity and other properties that makes it a suitable candidate for a realtime online scheduler in practical settings.

Contents

List of Figures

List of Tables

Acknowledgments

I'd like to thank Professor David Parkes, Professor Matt Welsh and Professor Jerry Green for providing invaluable advice and guidance in the progress towards the completion of this thesis.

1

Introduction

1.1 Motivation

Traditionally, bandwidth allocation and link scheduling in wireless ad-hoc networks are often done either in a pre-specified coordination schedule or in a simple local tiral-and-backoff model [20]. The latter requires unnecessary overhead due to contention and collision, while the former is not adaptive enough to fit real-time usage patterns. Recent research has proposed algorithms to schedule network usage in response to specific flow demands in order to optimize total throughput objective [8, 11, 13, 15]. However, aggregate throughput does not faithfully reflect the ultimate true value of network usage since it fails to distinguish different missions that a same amount of throughput could accomplish.

Recent development in wireless network technology has sprung up a lot of interesting new applications supported by a large range of technology, with varying bandwidth consumptions and varying levels of quality of service. Furthermore, wireless ad-hoc networks of increasing scale are formed to support several layers of services over one network infrastructure. Consider an example of sensor networks deployed in hospital environment (e.g. CodeBlue [24]). Different services such as patient tracking, doctor paging and vital sign monitoring all have different levels of priorities and values that are not scaled linearly with their bandwidth consumption. Moreover, one service may be able to operate at different bandwidth levels to provide different levels of quality or fidelity (e.g. media streaming over a PDA-network). All these usage scenarios propose the need for the resource scheduler to be aware of application- and user- specific *utility* instead of network level *bandwidth*. Hence we propose to consider the optimization of aggregate application

utility as the central goal of a new generation of resource schedulers.

In this thesis, we consider the introduction of utility optimization to the most-studied complex of bandwidth allocation and link scheduling in response to flow demands in multi-hop networks, coupled with wireless interference constraints. We will augment on previous models of multi-hop multi-flow scheduling [13] to formulate the Optimal Utility Scheduling problem. Attempts will be made to gauge and overcome the complexity of the problem by developing several algorithms with successive relaxations and approximations.

From an economic perspective, a wireless ad-hoc networks can be viewed as a complex market system in which a variety of users and applications act as net-utility optimizing agents to form the demand basis while nodes and links resources become the limited supply. Wireless interference translates into complex externalizes that are idiosyncratic and highly combinatorial in nature. The main focus of this thesis will be to present a computational market-oriented approach to solve the optimal utility scheduling problem with approximation.

The ultimate goals of this thesis are to provide a theoretical framework for the problem of optimal-utility allocation and scheduling in ad-hoc networks in the presence of wireless interference, and to contribute towards a practical resource scheduler that is decentralized, application-aware, utility-oriented and interference-friendly.

1.2 Related Works

A body of research papers have looked at the characterization of network capacity and link scheduling problem in multi-hop networks. Some [7, 16] are concerned with the theoretical oblivious link utilization under a range of unknown demand vectors, while in this thesis we take specific demands as given and aim to find optimized solutions. Others mainly explored the problem of optimizing total bandwidth or throughput in multi-hop network given flow demands, including [8, 11, 13, 15], mostly employing LP's to formulate the optimization problem or develop theoretical bounds.

Jain et al. [13] developed the conflict graphs to model interference and used a multi-commodity flow formulation to solve the optimal throughput problem; they also developed upper-bound and lower-bound for the optimal solution. Their work has largely inspired and influenced the work in this thesis. In their own formulations of the problem, [13] and [11] showed that link scheduling to achieve optimal throughput under interference constraints is NP-complete in general.

Radunovic and Boudec [19] argued that total throughput may not be the right objective to optimize. They introduced a class of pseudo-utility functions as performance objectives, which will lead to balance between performance and fairness. In this thesis, we treat utilities as first class objects that by themselves are the goal of optimization, rather than serving as a mean or proxy to achieve fairness.

The notion of utilities of application or users are also considered in many papers. Andrews et al. [2] considered the optimal utility problem in the situation of multiple user sharing one single radio medium. Yang and de Veciana [28] considered both user and network utility to form a dual optimization problem, which hinted on the use of market optimization approaches.

Market-oriented computational models have been developed to solve various resource allocation problems such as network transportation, multi-commodity flow problem [27], power load management[29] or any general market equilibrium problems [9]. Despite the negative results in general equilibrium theory about the stability and convergence of the competitive market equilibrium [4, 23], the tatonnement process tends to work quite well in practice in certain types of computational markets. Compared to previous market-oriented models, the distinguishing features of the market approach developed in this thesis include: the usage of virtual goods to capture externality due to interference, the high complexity of individual demand functions, and the difficulty of convergence in a discrete price and demand space with complementarity of goods.

Other authors have focused on the topic of wireless interference. Although richer partial interference models developed by Padhye et al. [18] model practical measurements of link quality [1] better, we choose to assume a simpler boolean interference model considering the already high complexity of our problem. Tan and Guttag [25] discussed the

scheduling problem in the multiple-to-one interference setting between wireless devices and wireless access points. Here we do not assume a central wireless access point but instead investigate the scheduling problem in ad-hoc network formed by peer devices.

To summarize, although there are an abundance of related works in the fields of multi-hop network routing and scheduling, wireless radio interference, utility-based optimization, and market-oriented computations, there are very few that lie in the intersection of all four. This thesis makes an honest attempt to consider a comprehensive problem involving all these four elements: the application of market-oriented approaches to the problem of bandwidth allocation and link scheduling in a wireless ad-hoc network to optimize user-specified utility in the presence of interference.

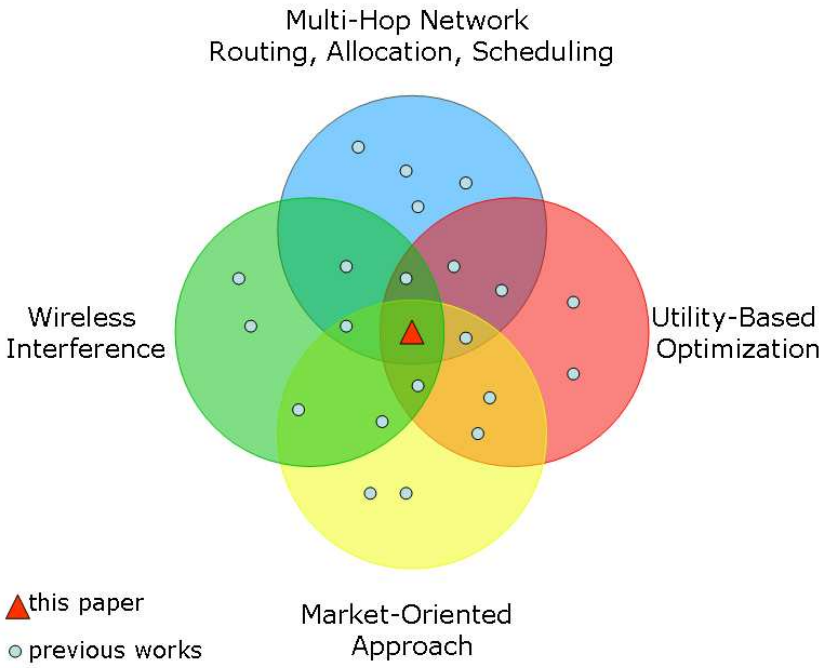


Figure 1.1: **Relation to previous works**. *This thesis attempts to combine research interests of several fields. (number of papers not to scale)*

1.3 Organization of this Thesis

In Chapter 2 we will present a general formulation of the Optimal Utility Scheduling problem and discuss its various interpretations and generalizations. Chapter 3 formulates the problem directly as a brute-force 0-1 integer linear program, OPT, which has optimal utility output but suffers from extreme complexity. Chapter 4 presents the Optimally Rate-Limited CSMA algorithm (ORL-CSMA) that first solves a relaxation of the OPT integer program to compute an upper-bound, and then uses a CSMA-like protocol to ensure schedulability. The main focus of the thesis will be in Chapter 5, in which we develop a market oriented allocation protocol MARKET to compute a bandwidth allocation as a rate-limiter. We will look at the unusual characteristics of our market in comparison to classic economic markets and previous computational market models in detail, and discuss the choices of various components and parameters of the MARKET algorithm. In Chapter 6, we present comparative performance analysis of the algorithms we developed in a wide range of setups. The final Chapter draws conclusions from the theoretical and experimental results and discuss future areas of research.

2

Theoretical Framework

In this chapter we will present a general formulation of the *Optimal Utility Scheduling* problem and discuss its various interpretation and generalization to encompass a large range of setups. Then we summarize previous results regarding the complexity of the problem. The problem formulation presented in this chapter will directly guide the development of the optimal schedule solver presented in the next chapter.

2.1 Problem Formulation

Our formulation is based on the model of connectivity graph and conflict graph developed by Jain et al. [13] augmented with the notions of applications and utilities.

For a given wireless network with n nodes, the *connectivity graph* C is a *directed* graph defined by $\langle N, L\rangle$ where $N = \{1, 2, ..., n\}$ is the nodes representing wireless devices and $L = \{l_{ij} : \text{there exists a link from } i \text{ to } j\}$ is the set of *directed* links among them. Each link has its capacity, $Cap(l_{ij})$, which is the maximum achievable date rate *if the link* $l_{i,j}$ *is active during the entire time span* T (defined later).

A set of applications $A = \{a_1, a_2, ..., a_m\}$ are contending to use the network. Each application a is defined by a tuple $\langle s, d, u(.)\rangle$, which specifies a source node s, a destination node d and a utility function $u(.)$ defined over the bandwidth of a flow allocated to it from s to d. The minimum requirements for a utility function is that it passes through (0,0) and is non-decreasing. We will discuss further restriction of utility functions such as concavity and piecewise-linearity in later sections.

A flow f is an assignment of bandwidth traffic to each link in L. In our convention,

superscripting a flow denotes the index of application it belongs to and subscripting a flow denotes the corresponding link component of the flow (i.e. f_{ij}^k means the bandwidth consumption on link l_{ij} for application a_k). The following *Flow Conservation Constraints* must be satisfied:

$$\forall i, j \in N, k : a_k \in A, \qquad f_{ij}^k \geq 0 \tag{2.1}$$

$$\forall i \in N, k : a_k \in A, \sum_{j:l_{ji} \in L} f_{ji}^k - \sum_{j:l_{ij} \in L} f_{ij}^k = \begin{cases} -\|f^k\| & \text{, if } i = s_k; \\ \|f^k\| & \text{, if } i = d_k; \\ 0 & \text{, otherwise.} \end{cases} \tag{2.2}$$

Equation (2.2) states that for every application a_k and every non-terminal node i ($i \neq s_k$ and $i \neq d_k$), the total inflow to node i must be equal to the total outflow for application a_k; whereas the source node and the destination node will have positive net outflow or positive net inflow respectively, and its magnitude is equal to the overall bandwidth of the flow, denoted as $\|f^k\|$.[1] The application's utility is therefore $u_k(\|f^k\|)$. Moreover, we call the complete flow assignment for every application the *flow vector*, denoted $f^A = (f^1, ..., f^m)$ where $A = \{a_1, ..., a_m\}$.

To model interference, define the *conflict graph* of the network to be an *undirected* graph $F = \langle V_F, E_F \rangle$, whose vertices $V_F = L$ correspond to the links in the connectivity graph. An edge $\langle l_{ij}, l_{pq} \rangle \in E_F$ means that the two links interfere with each other and cannot be active simultaneously.[2] The conflict graph is defined between links rather than between nodes to allow a richer model of interference (discussed later in section 2.2.3). However, unlike [18] which explores a more fine-grained partial interference model, we make the simplifying assumption that links either interfere or they do not.

The *time span* T is the index set of all possible time values in an *epoth*, which is a typical period of communication patterns that can be repeated over and over again. It can be either continuous $[0, 1]$ or discrete $\{1, 2, ..., t\}$. The former can be interpreted as a proportion of the entire medium resource, while the later can be viewed as discrete time slots in an epoch in a Time Division Multiple Access (TDMA) scheme. For every element in the set T, a decision

[1] This is not to be confused with the norm when f is considered an l-dimensional vector, where l is the number of links.

[2] Note that we do not draw an edge from a vertex (link) to itself in the interference graph, but we do draw an edge between a pair of reverse links l_{ij} and l_{ji}.

has to be made as to which subset of the link L will be active for which application. Or alternatively, for every link in L a decision has to be made as to which subset of the entire time span T it will be active, and which application's data it will carry.

Hence, we define a *schedule* to be a function $S : A \times L \mapsto \{0,1\}^T$ that assigns each application to a certain subset of the the total time span T for every link. e.g. $S(a,l) = \{1,3,5\}$ means in time slots 1,3,5 link l will be active transmitting application a's data. We say a schedule S is *feasible* if the following *Schedulability Constraint* is satisfied:

$$\begin{aligned} &\forall l_{ij} \in L, a_{k_1}, a_{k_2} \in A : a_{k_1} \neq a_{k_2} \\ &\Rightarrow S(a_{k_1}, l_{ij}) \cap S(a_{k_2}, l_{ij}) = \phi \end{aligned} \tag{2.3}$$

$$\begin{aligned} &\forall l_{ij}, l_{pq} \in L, a_{k_1}, a_{k_2} \in A : \langle l_{ij}, l_{pq} \rangle \in E_F \\ &\Rightarrow S(a_{k_1}, l_{ij}) \cap S(a_{k_2}, l_{pq}) = \phi \end{aligned} \tag{2.4}$$

Equation (2.3) states that no two applications can occupy the same link at the same time, and (2.4) states that no two links that interfere with each other can be active at the same time. Note that by construction, no link can be active for a time period longer than the total time span since the value of S is always a subset of T.

We say that a schedule S *implements* a flow vector $f^A = (f^1, ..., f^m)$ iff S is feasible and the following is satisfied:

$$\forall i, j \in N, k : a_k \in A, f_{ij}^k \leq Cap(l_{ij}) \cdot \frac{\|S(a_k, l_{ij})\|}{\|T\|} \tag{2.5}$$

where $\|.\|$ denotes the size of a set.[3] This equation means that the achieved bandwidth of an application on a link is no more than the capacity of the link scaled by the proportion of time the application is scheduled to occupy that link. We say a flow vector f^A is feasible if it can be implemented by some feasible schedule S.

Finally, the *Optimal Utility Scheduling Problem* is the following: given the connectivity graph C, the conflict graph F and a list of applications A, compute a feasible schedule S

[3]Or the *measure* of a set of continuous time range. e.g. $\|[0, 0.3] \cup [0.7, 0.9]\| = 0.5$.

that maximizes the aggregate utility of all applications:

$$\max_{f^A,S} \sum_{k:a_k \in A} u_k(\|f^k\|) \quad \text{s.t. } f^A \text{ and } S \text{ satisfy (2.1)-(2.5)} \tag{2.6}$$

2.2 Interpretation and Generalization

2.2.1 Links and Capacities

The connectivity graph and capacity vector together characterize the network topology and link quality. The existence of a link l_{ij} in the connectivity graph only means that data can be transmitted via a direct wireless channel from i to j, while $Cap(l_{ij})$ quantifies the maximum achievable data rate if the wireless channel is dedicated only to the transmission from i to j, free of interference from other links, during the entire time span T. A simple idealized network model would assume that a link exists between every pair of nodes that lie within the nominal radio range of each other, and that they all have the same capacity as specified by the radio hardware (which we will adopt in our simulations). In practical networks, the network topology can be formed by neighborhood discovery techniques such as periodic beaconing, broadcast probing or traffic snooping (surveyed in [22]). The capacity of a link can either be estimated by the nominal data rate of the wireless hardware calibrated by some empirical formula, or experimentally determined by measuring sustainable pairwise bandwidth with all other links inactive [1, 18].

Since links in the connectivity graph are directed, our network model does not assume link symmetry. A pair of reverse links l_{ij} and l_{ji} do not have to exist in L simultaneously. Even if they do, they can have different capacity limit to constitute link asymmetry. Furthermore, they can have interference with a different set of other links under some interference models (see section 2.2.3).

2.2.2 Time and Schedule

The notion of time span in our model requires a little more elaboration. It is not the time length in a normal sense as measured by seconds or milliseconds, but rather a set of values to index a collection of individually schedulable parts of one radio resource. The choice of

T depends primarily on the Media Access Control (MAC) layer of the wireless network.

For example, in a Time Division Multiple Access (TDMA) scheme the time span is a discrete set $T = \{1, 2, ..., t\}$ that corresponds to t time slots in one epoch. The number of rounds t in an epoch, and the physical time length of each epoch are parameters of the TDMA protocol. This abstraction of time span allows us to talk about the bandwidth of an individual application in *relative* magnitude as a portion of the overall raw data rate. Consider a TDMA scheme with a raw data rate of 200kbps, 100ms epoch (frame) length, time-slotted into 10 rounds. Then an application that has been assigned to 3 time slots on a link will have an effective bandwidth of $200\textit{kbps} \times \frac{3}{10} = 60\textit{kbps}$. If this 3-time-slots-out-of-10 schedule is repeated over time, then 60*kbps* will be the sustainable bandwidth that the application experiences. If T is discrete, a schedule can be compactly represented as a 2-dimensional schedule table of the form $L \times T \mapsto A$ or a 3-dimensional indicator array $L \times A \times T \mapsto \{0, 1\}$.

In the case where the medium resource is infinitely divisible (or close to that in effect), then we may take T to be continuous, and without loss of generality let it be the unit interval $[0, 1]$. This fits a Carrier Sense Multiple Access (CSMA) protocol, in which transmission may begin and end at any continuous time. In this continuous case it is more complex to represent a schedule since it has to specify the activity for every (link, application, time) tuple. If we assume that reordering of the link assignment besides along the time axis is irrelevant, a schedule can be expressed as a weighted linear combination of independent sets of the conflict graph[13].[4] Although the use of continuous weights seems to bring computational relief because solving Linear Program is easier than solving Integer Program, there are in fact an exponential number of independent sets and enumerating them is NP-complete.

It is worth noting that the index set T can be generalized to correspond to other ways of dividing the medium resource along the time axis. For example, if multiple frequencies are in use and nodes are capable of operating at different frequencies simultaneously, then letting T be the list of available frequencies will lead to a problem of scheduling applica-

[4]Recall that an independent set of an undirected graph is a set of vertices without any edges between any pair among them.

tions over frequency channels instead of over time. A combination of frequency division and time division can also be used (e.g. in GSM network).

2.2.3 Interference Models

In general, the conflict graph can be totally independent of the connectivity graph, and can be measured empirically by simultaneously activating every pair of links and see whether the transmission rates degrade. Although Padhye et al. [18] showed that realistic interference would manifest as gradual degradation in both links' transmission rates, their partial interference model is unreliable to extrapolate to multi-lateral interference, and will lead to extra complexity in our problem. Instead, we will make the simplifying assumption that we will treat partial interference between two links as total interference and will not allow them to become active at the same time.

When empirical interference measurements are not available, we can generate the interference graph from the connectivity graph by making assumptions about the link protocols. Different models of interference can be adopted in our model by different ways of construction. Similar to the classification of interference conditions provided by Ramanathan [20], we will mainly consider the following interference conditions:

- **Hardware Interference**: Because a wireless device cannot transmit or receive two signals at the same time,[5] two links that involve a same node in the connectivity graph interfere with each other. Formally, $i = p$ or $i = q$ or $j = p$ or $j = q \Rightarrow \langle l_{ij}, l_{pq} \rangle \in E_F$.

- **Reception Interference**: The most common interference in wireless networks occurs when two radio signals interfere with each other and prevent the receiver from decoding either of the two. Two links will interfere with each other if the receiver of one link is in the range of the sender of the other link. Formally, $l_{iq} \in L$ or $l_{pj} \in L \Rightarrow \langle l_{ij}, l_{pq} \rangle \in E_F$. This condition sufficiently describes the interference relationship in a protocol where only the receiver is required to be free of signal inter-

[5] Although this may be possible with e.g. multiple antennas, the possibility of multiple wireless channels should be factored into the time index T as described in section 2.2.2. Thus, a node cannot handle more than one operation at the same "time" as defined by set T.

ference, i.e. if the sender does not need to receive an acknowledgement (ACK) from the receiver.

- **Acknowledgement Interference**: In some protocols the sender must also receive an un-interfered ACK back from the receiver in order to complete the transmission (otherwise it will re-transmit again). In this case, two links l_{ij} and l_{pq} interfere if any of the cross links exists (l_{ip}, l_{ij}, etc. are in L).

Thus we have the following sequence of increasingly restrictive interference models:

- **Level-0 Interference Model**: Includes only the first condition.
- **Level-1 Interference Model**: Includes only the first two conditions.
- **Level-2 Interference Model**: Includes all three interference conditions above.

Figure (2.1) illustrates whether two links l_{ij} and l_{pq} interfere under Level-1 and Level-2 interference mode (in Level-0 interference model they do not interfere since all end-points are different).

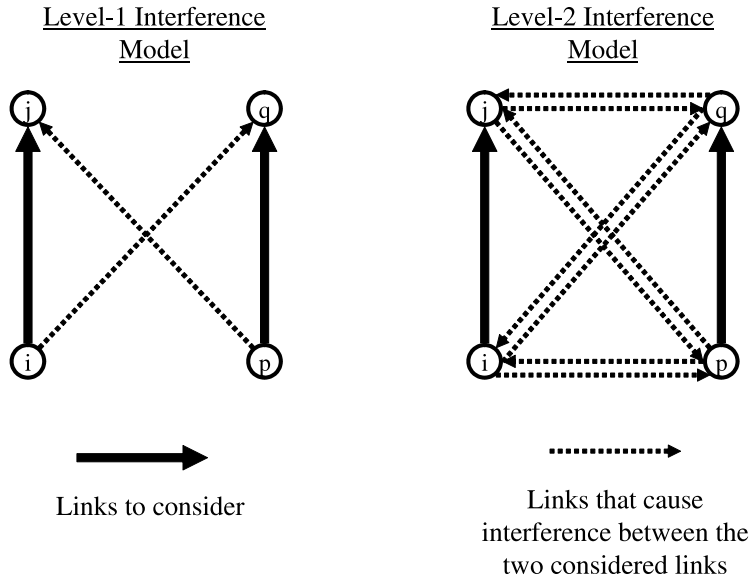


Figure 2.1: **Level-1 and Level-2 Interference Models**. *The bold-arrow links will interfere with each other if any of the dotted-arrow links exist.*

2.2.4 Utility Functions

The addition of application utility functions is the most significant contribution of this thesis compared to previous research by Jain et al. [13] and etc. The utility function is a map from bandwidth to the application's value for being able to achieve a given average bandwidth level. In an economic setting, this would be the virtual monetary value that the user would be willing to pay in exchange for network usage at the level of the specified bandwidth. Although Radunovic and Boudec [19] proposed using utility function of specific forms such as a power function or the logarithm of the bandwidth, they merely use it as a mean to improve fairness properties over the traditional throughput-maximizing allocation. This is not the case here – the utility reported by the applications are their ultimate *true values* for network usage and optimizing aggregate utility is the end goal instead of a mean to achieve fairness. The advantage of maximizing aggregate utility rather than aggregate throughput is that it allows applications to specify rich preferences over different bandwidth consumptions and allows the network scheduler to prioritize applications of different importance in order to dedicate limited network resources to achieve maximum value.

There are several properties of utility functions considered in our model: First we require that it pass through (0,0) by convention. Second it must be non-decreasing (otherwise the application may just step down to operate at a lower bandwidth than it is allocated). Finally we restrict it to be in the class of piecewise linear functions, including ones with discontinuous jumps. The advantages for using piecewise linear utility functions are the following:

- They have compact representations. One way to specify a piecewise-linear function is to give a list of turning points (x_i, y_i) in ascending x coordinate, and two extra slopes $preSlope$ and $postSlope$ before the first point and after the last point, which we will use in our implementation.
- They are simple yet general enough to approximate any reasonable functions.
- Every piecewise linear function can be reduced into a collection of constraints in a

linear program using only linear operations, inequality operators and slack variables. This makes it possible to incorporate piecewise linear utility functions in an (integer) linear program to solve for the optimal schedule. The actual reduction is omitted here since it is automatically handled by the CPLEX optimizer [12].

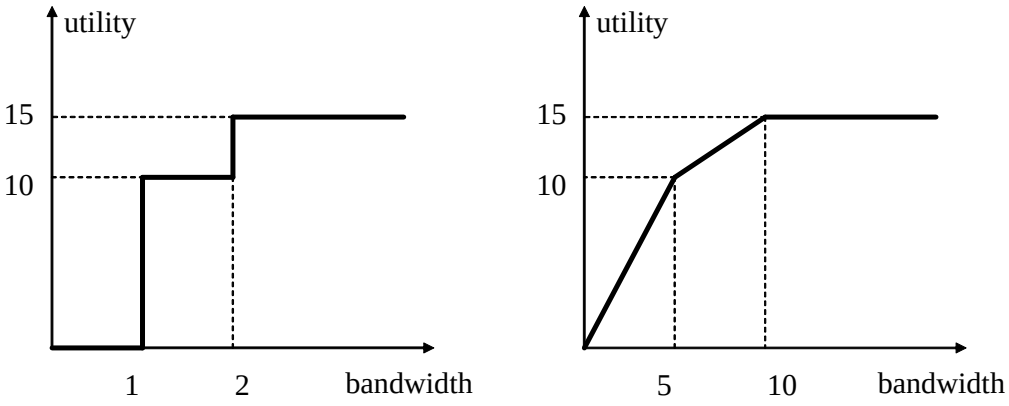


Figure 2.2: **Examples of piecewise-linear concave utility functions**. *The first one makes step jumps at discrete thresholds; the second one grows over a range of continuous bandwidths but changes its slopes. The first one is considered concave only when bandwidth is discrete instead of continuous.*

Although it is not a requirement in our optimization problem for the utility functions to be concave, practical applications usually have diminishing marginal returns on bandwidth and thus have concave utility functions. Figure 2.2 shows two examples of piecewise linear concave utility functions. The first utility function (0,0)-(1,10)-(2,15) represents an application with two modes of operations and a decreasing marginal utility of each additional unit of bandwidth. We will use this typical utility function extensively in later experimental sections.

2.2.5 Application Classes

Our model of an application is one that derives utility from sending date from one source to one destination at a certain sustainable data rate. This suits the class of applications that care mostly about average bandwidth, but does not quite fit other classes of applications that care about other performance metrics such as latency, accumulated traffic, or future bandwidth reservation. Here we present a list of application scenarios, ordered by

decreasing compatibility with our model.

- **Signal Monitoring**: Applications such as vital sign monitoring in medical environment [24] are typical applications that fit our model. For example, a patient sensor device can either report a heart rate in low bandwidth mode or record and send detailed cardiography at high bandwidth mode. Furthermore, different specification of application utilities can facilitate prioritization of urgent emergency-related applications vs routine maintenance applications.

- **Voice/Video Streaming**: Media Streaming applications can potentially be deployed on high-bandwidth network such as PDA or other mobile ad-hoc network. The media player can operate at various bandwidth levels to playback at different quality, and the user derives more utility as the sound/image resolution increases.

- **File Transfer**: A file transfer user is likely to say that "my utility will be U if I can transfer S amount of data by time Y, or 0 otherwise". In principle, the last packet transferred by the application will carry the entire utility amount U, and since the previous traffic carries zero utility, they will not be scheduled at all. In practice, we can amortize the lump-sum utility onto the data and spread it over the time remaining until deadline. A utility function of the form

$$u(b) = \begin{cases} \frac{Us_r}{S} \cdot b & \text{, if } bt_r \geq s_r; \\ 0 & \text{, otherwise.} \end{cases}$$

  will be appropriate, where s_r is remaining size to transfer and t_r is the remaining time till deadline.

- **Multi-Cast Broadcast**: Since our network model assumes multi-path uni-cast transmission, an advertiser application that derives utility from reaching a number of receivers through multi-cast broadcast cannot readily fit into our model. The application can certainly divide up the multi-cast into multiple single-casts, but it would suffer from inefficiency because duplicate data is sent in the main trunk of the multi-cast tree. This is a limitation of our network model because a multi-cast flow does not satisfy the flow conservation constraint.

- **Conference Calling**: Unlike the previous class of voice/video streaming applications that derive utility by-the-seconds, a corporation may decide to hold a conference call over wireless ad-hoc network in future and requires a guaranteed level of bandwidth over a minimum period of time (in order for its members to reach some conclusion). Our model currently cannot support such utility that is contingent upon future bandwidth reservation since the scheduler solves for short-term schedules only.[6]

- **Latency-Sensitive Applications**: Since we consider the re-ordering of round-schedule within an epoch to be irrelevant, it is possible for the links on a multi-hop path to be activated out of order. For a multi-hop flow path of hop-count d, it may take one package d *epoches* to reach its destination in the worst case when the links are activated in the exact reverse order in a schedule, whereas in the best case it takes only d *rounds* when the links are activated in the correct order (for an example see section 3.3). Because in the long run the average throughput are the same for these two extreme cases, our optimization problem does not take latency into consideration. Applications can only specify utility that depends on the average bandwidth level but not latency.

Our model covers several major classes of applications and we hope to extend it to take into account more application models in future work.

2.3 Complexity Results

Many similar scheduling problems have been shown to be NP-hard [3, 10, 21]. In particular, Jain et al. [13] established that it is NP-hard to compute the the throughput-maximizing schedule in the presence of interference, and moreover it is NP-hard to produce an approximation within a constant ratio of the optimum solution. Since their formulation of the maximum throughput scheduling problem is a special case of our above formulation with continuous time span and identity utility function, the Optimal Utility Scheduling problem

[6] Unless, of course, the epoch length is long enough to be bigger than the length of the reservation.

formulated above is NP-hard. Furthermore, the addition of utility function will introduce additional complexity into the problem. Nevertheless we will still design and implement an optimal schedule solver in the next chapter to examine the the practical (in)tractability of this approach and also to serve as a performance benchmark for other algorithms that we will subsequently develop.

2.4 A Market Economics Interpretation

The complexity of a wireless network, especially the supply-and-demand relationship between nodes, links and applications, closely resembles a complex economy. On a high level, units of wireless resources can be considered as goods in the system, produced by nodes or links and consumed by users and applications. This forms a market that's highly combinatorial in nature with several prominent characteristics:

- **Combinatorial goods**: Because most often applications will need more resources (e.g. nodes and links) in order to form a useful flow to accomplish its goal, the construction and interaction of resource bundles is very important in this market. The dependencies between resources constitute *complementarity* on the lower good level, yet the possibility to use alternative path provides *substitutability* on the higher flow level.

- **Non-continuous demand function**: In addition to the discretization of quantities of goods, the complementary nature of resources implies that a small change in the prices may cause an application to change to a completely different path and lead to big change in the demand function.

- **Highly complex externality structure**: Interference in wireless network leads to negative externality in this virtual market because activation of one link prevents other interfering links from being utilized. In classic economic models externality is mostly non-idiosyncratic (e.g. emission of pollutant). The fact that a link interferes with some links but not others makes the externality presented in the system idiosyncratic. Furthermore, because the conflict graph can be arbitrary in principle, the types

of externalities can be exponential in the number of links. Such complex externality structure has not been extensively studied in economic literature.

- **Fixed supply**: Most resources in the wireless network (such as wireless medium, frequency channels) have limited amount of supply in the model we consider. In some network scenarios, the nodes may be able to adjust their energy consumption level to produce variable amount of bandwidth. However, such a situation is outside the scope of this thesis.

In this section, we have only scratched the surface of the market-economic interpretation of scheduling in wireless networks. In chapter 5 we will develop a market-oriented approach to the optimal utility scheduling problem and examine many of the above issues in greater detail.

3

An Optimal Schedule Solver

3.1 Design

In this section we develop an optimal schedule solver, OPT, for solving the discrete version of the Optimal Utility Scheduling problem with time span $T = \{1, 2, ..., \bar{t}\}$. The algorithm takes the connectivity graph, the interference graph and a list of applications as input, and seeks the optimal schedule directly by solving a brute-force 0-1 Mixed Integer Program (MIP) as presented in the above problem formulation using Ilog's CPLEX optimizer [12]. The OPT algorithm effectively combines the functions of bandwidth allocation, multi-hop routing and link-scheduling all in one. Figure 3.1 shows its design.

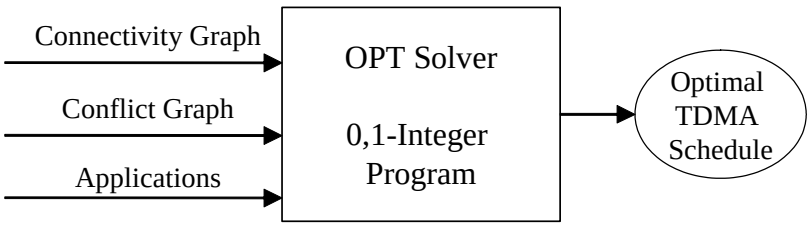


Figure 3.1: **OPT algorithm wiring diagram**.

3.2 Implementation

The inputs of the algorithm include: the adjacency matrix of connectivity graph, $\mathit{link}(i, j)$, the link capacity vector, $\mathit{cap}(i, j)$, the adjacency matrix of the conflict graph, $\mathit{conflict}(i, j, p, q)$, the source $\mathit{src}(a)$, destination $\mathit{dest}(a)$ and utility $u(a)$ for each application. The outputs include: a 4-dimensional array representation of the optimal schedule $\mathit{sch}(a, i, j, t)$ and the flow $\mathit{flow}(a, i, j)$ that it implements. For clarity of presentation in this section, we will

follow the convention that index a always iterates through applications, i, j, p, q through nodes and t through time periods. Figure 3.2 shows the mixed integer program used in OPT . Though brute-force as it may seem, there appears to be no better way to write down the problem considering its highly combinatorial nature.

$$\max \sum_a eval(\mathbf{u}(a), \mathbf{b}(a))$$

over variables

$$\mathbf{sch}(a,i,j,t) \in \{0,1\}, \mathbf{flow}(a,i,j) \geq 0, \mathbf{b}(a) \geq 0 \quad \text{for } a \in A, i,j \in N, t \in T$$

subject to

$$(I) \forall a,i,j,t : \mathbf{sch}(a,i,j,t) \leq \mathbf{link}[i][j]$$

$$(II) \begin{cases} \forall i,j,t : \sum_a \mathbf{sch}(a,i,j,t) \leq 1 \\ \forall i,j,p,q,t \text{ s.t. } \mathbf{conflict}(i,j,p,q) : \\ \sum_a \mathbf{sch}(a,i,j,t) + \sum_a \mathbf{sch}(a,p,q,t) \leq 1 \end{cases}$$

$$(III) \begin{cases} \forall a,i,j : \mathbf{flow}(a,i,j) \leq \frac{\mathbf{cap}(i,j)}{\|T\|} \sum_t \mathbf{sch}(a,i,j,t) \\ \forall a : b(a) = \sum_j \mathbf{flow}(a, \mathbf{src}(a), j) = \sum_j \mathbf{flow}(a,j,\mathbf{dest}(a)) \\ \forall a,i \text{ s.t. } i \neq \mathbf{src}(a), i \neq \mathbf{dest}(a) : \\ \sum_j \mathbf{flow}(a,j,i) = \sum_j \mathbf{flow}(a,i,j) \end{cases}$$

Figure 3.2: **Integer Linear Program used in OPT.**

Block (I) encodes the schedule as $\mathbf{sch}(a,i,j,t) \in \{0,1\}$, indicating whether application a is scheduled on link l_{ij} at time t, and restricts it according to the connectivity graph. This provides the integer component of the linear program. A schedule indicator without the corresponding links ($\mathbf{link}[i][j] = 0$) will be forced to 0 and automatically dropped by the CPLEX pre-solver. Block (II) encodes the schedulability constraints 2.3 and 2.4 and block (III) encodes the flow conservation constraint 2.2. Finally we evaluate each piecewise linear utility function $\mathbf{u}(a)$ at the resulting bandwidth $\mathbf{b}(a)$ and maximize for the overall utility. CPLEX solver provides equivalent of $eval$ construct and will automatically reduce them into additional constraints to add to the MIP.

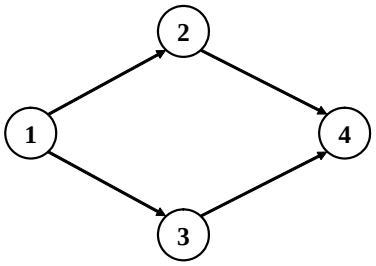
Note that $\frac{\mathbf{cap}(i,j)}{\|T\|}$ is a constant factor that scales the number of allocated time periods on a link to the actual achieved bandwidth. For simplicity, we will simply set this constant

to 1 for all links (so that every link has the same bandwidth that's equal to the number of time periods in some unit, and activating one link for one period of time during the entire period will lead to one unit of bandwidth).

Our OPT program will take an instance of the optimal utility scheduling problem and create an instance of MIP object for the CPLEX optimizer. The latter is invoked with all default parameters except for the time-out limit specified by command line and the *optimality tolerance* being set to be 5%, which instructs the optimizer to stop searching for better integer solution when the candidate solution is within 5% of the optimal upper-bound, computed by solving the dual as non-integer Linear Program. Because MIP is NP-complete and requires exhaustive branch-and-search, often times there is a sizable gap between the optimal integer solution and non-integer solution to its dual problem, in which case the optimizer may have already found the optimum solution but has not yet *proven* its optimality before exhausting all the rest of the possibilities. An optimality tolerance of 5% is used to balance optimality and running time based on previous experiences with CPLEX.

3.3 A Stylized Example

Here we present a stylized example to illustrate how the OPT scheduler works and highlight some of the advantages of utility-based maximization. Consider the simple network topology shown in Figure 3.3 with 4 rounds in an epoch and assume the level-0 interference model. Consider a class of identical applications arriving one after another over time, all wishing to transmit data from node 1 to node 4 with the utility function shown in figure 2.2-(a). The network capacity can allow up to two such applications running at full bandwidth (2) and achieve a total utility of 30. Due to interference constraints the route 1-3-4 and 1-2-4 will be running in opposite phase to each other. As a_3 enters, the optimal strategy will be to allow two applications to run at half speed, sharing one of the two possible routes, and let the other application run at full speed. The resulting utility is 15+10+10=35. Similarly a_4 will share the same route with a_2 and now all four applications are all running at low bandwidth mode with 40 total utility. All subsequent applications will be dropped (or alternatively all applications will have equal probabilities to be scheduled, 4 at a time).



1-2 apps

time	1	2	3	4
l_{12}	a_1		a_1	
l_{13}		a_2		a_2
l_{24}		a_1		a_1
l_{34}	a_2		a_2	

3 apps

time	1	2	3	4
l_{12}	a_1		a_3	
l_{13}		a_2		a_2
l_{24}		a_1		a_3
l_{34}	a_2		a_2	

4 apps

time	1	2	3	4
l_{12}	a_1		a_3	
l_{13}		a_2		a_4
l_{24}		a_1		a_3
l_{34}	a_2		a_4	

Figure 3.3: **Optimal schedules on a simple network with increasing number of identical applications from 1 to 4.**

Several interesting features of the OPT scheduler to be noted are:

- With a few applications running at high bandwidth mode, the network quickly reaches its transmission capacity limit (at 2 apps). However, this is a *pseudo-saturated* stage in the sense that although link usage and total throughput are already maximized, more utility could be extracted as more applications enters.

- As demand increases past the pseudo-saturation stage, OPT will try to first downgrade the quality of existing applications to accommodate new applications, if the trade off will increase overall utility. Contrast this with traditional admission control approach which simply drops new application when the network capacity is reached. OPT extracts more potential utilities until the network is fully saturated, and hence utilizes link resources more efficiently.

- After utility saturation, OPT reduces to admission control and drops further applications. Additional applications will not have negative impact on the network performance[1]. Contrast this with a naive greedy CSMA scheme where all applications

[1]Perhaps with the exception of increasing the complexity of the optimization problem and possibly reducing the quality of the optimal solution found when time-out limit is reached.

attempt to transmit at the maximum rate possible, and as a result overwhelming congestion and backing-off will negatively impact the overall performance after a critical point.

- As the total utility increases and approaches an optimum (40 in this case), the average utility dropped gradually (from 15 to 11.7 to 10). Hence OPT supports graceful degradation and achieves good balance in quality of services.

- Although in this example the optimal utility solution also happens to be an optimal throughput solution, this is not necessarily the case in general. Consider a hypothetical example of two applications, one with 10 utility operating at bandwidth 4 and the other with 100 utility operation at bandwidth 1 in the above network.[2]

- OPT does *not* concern with reordering the schedule to minimize latency. In the last schedule with 4 applications, the latency for a_2 and a_4 appears to be 3 rounds since information has to travel through l_{13} before l_{34}.[3] Although a fix-up step can be applied after the optimization stage to reduce latency (e.g. exchanging a_2 and a_4 for l_{34} will reduce both latencies to 1 round), but latency metric does not factor into the optimization stage. Hence applications can only specify utilities that are dependent on the average sustained bandwidth but not on latency.

[2]Such an example is used to described a high-priority or urgent task vs. a backgrounded and less important task. e.g. an Emergency Room signal vs. a security video streaming.

[3]To see this more clearly, concatenate two of the same schedules one after another. Say a given package for a_2 is transmitted on l_{13} in period 2, then it needs to wait until period 1 of the *next* epoch to be transmitted over l_{34} – resulting a latency of 3.

4

An Approximately Optimal Rate-Limited CSMA Scheduler

4.1 Design

Most of the complexity explosion in the OPT solver is due to the direct 0-1 encoding of the full schedule. It would be nice if we could somehow compute the optimal flow-vector directly and then construct a corresponding optimal schedule. However, [13] suggests that it is not possible to ensure schedulability of a given flow vector without actually finding an underlying schedule. Alternatively [13] provides methods for 1) computing the lower-bound of the optimal flow-vector by imposing *sufficient* conditions of its schedulability, assuming it is a linear combination of a non-exhaustive list of independent sets of the conflict graph, and 2) computing the upper-bound of the optimal flow-vector by imposing *necessary* conditions of schedulability by adding *clique constraints* and *odd-hole constraints* that must be satisfied if the flow vector were schedulable). In light of this, we develop the ORL-CSMA algorithm to compute an upper-bound solution of the flow vector, and then make sure of schedulability by a best-effort CSMA scheme that is guided by the upper-bound flow vector. Although it is demonstrated in [13] that the upper-bound solution does not converge to the optimal solution as close as the lower-bound solution does, we choose the upper-bound solution to follow because the CSMA scheduler can only reduce the bandwidth and utility but cannot increase it.

The basic structure of ORL-CSMA is to divide the big problem into two parts – an *allocator* that computes an allocation in the form of a flow vector, and a *scheduler* that schedules

links with best-effort according to the intermediary flow vector (see Figure 4.1).

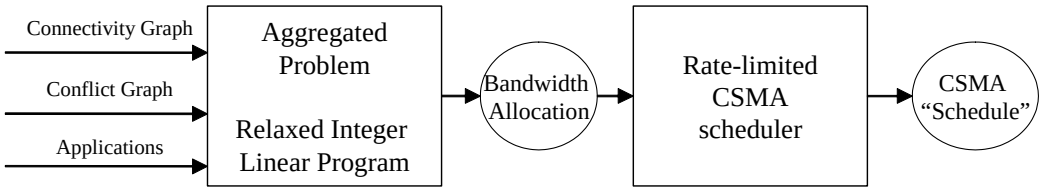


Figure 4.1: **ORL-CSMA wiring diagram**. *The first stage optimizer solves a relaxed aggregate version of the original MIP. The output is an upper-bound flow vector to the full optimal problem and may not necessarily be schedulable. In the second stage the flow vector serves as a rate-limiter to a modified CSMA scheduler, which attempts to implement the flow vector in best-effort.*

4.2 Computing Upper-bound Flow Vector

The *clique constraint* is the following[1]: A clique Q in the conflict graph is a subset of vertices with edges between every pair of them, i.e. a set of links that mutually interfere with each other. It is clear that at any given time period only one link in a clique can be active, and consequently the total number of active time periods of all links in a clique must be less than the total number of time periods. Formally, if

$$\forall l_{ij}, l_{pq} \in Q : \langle l_{ij}, l_{pq} \rangle \in E_F$$

then

$$\sum_{l_{ij} \in Q} \sum_{a} \left\lceil f_{ij}^{a} \frac{\|T\|}{Cap(l_{ij})} \right\rceil \leq \|T\| \tag{4.1}$$

Again the scaling factor $\frac{\|T\|}{Cap(l_{ij})}$ translates the bandwidth assignment to the number of time periods used. Also it suffices to consider maximal cliques because a constraint for a non-maximal clique Q_1 are subsumed by the constraint for a maximal clique Q_2 if $Q_1 \subset Q_2$.

Similarly, the *odd-hole constraint* is derived from an odd-hole H, which is a a circle of odd number of links in the conflict graph. Since at any given time period at most half ($\frac{\|H\|}{2}$) of them can be active, the aggregate number of active time periods for all links in the hole over the entire epoch must be smaller than $\left\lfloor \frac{\|H\|}{2} \right\rfloor \cdot \|T\|$. Odd-holes offer non-trivial information compared to even-holds because the number is then rounded to a smaller

[1]This is a restatement from [13].

integer value. Formally, if

$$H = \{l_{i_0 j_0}, l_{i_1 j_1}, ..., l_{i_{h-1} j_{h-1}}\}$$
$$\forall 0 \leq k < h : \langle l_{i_k j_k}, l_{i_{k\oplus 1} j_{k\oplus 1}} \rangle \in E_F$$

where $\oplus$ is addition mod h, then

$$\sum_{l_{ij} \in H} \sum_{a} \left\lceil f_{ij}^{a} \frac{\|T\|}{Cap(l_{ij})} \right\rceil \leq \left\lfloor \frac{\|H\|}{2} \right\rfloor \cdot \|T\| \tag{4.2}$$

Notice that the use of the floor and ceiling function makes the constraint tighter and leads to a more refined upper-bound.

Because enumerating all maximal cliques and maximal holes in the conflict graph is in general NP-complete, we cannot possibly add all clique constraints and all odd-hole constraints. In fact, even if we add all of them there is no guarantee that the upper-bound solution approaches the optimal solution. Therefore, we employ the following randomized process to enumerate a sufficient number of cliques by randomly growing them:

1. Pick a random link. That by definition is a trivial clique.
2. From the list of vertices that could join the current clique to form a new clique, i.e links that interfere with *every* link in the current clique, pick a random one to join.
3. Repeat the above step until no more links can join. At this point we have a maximal clique.

And similarly for odd-holes:

1. Add a random link to an initially empty list.
2. Add a random link that is not in the list and interfere with the last link in the list.
3. If the current list "wraps around", i.e. the last link in the list interfere with the first link in the list, and has an odd number of elements, save the list as a candidate odd-hole.
4. Repeat steps 2 and 3 until no more link can be added.

5. Return an odd-hole randomly chosen from all candidates saved in step 3.

Note that we do not stop at the first odd-hole found in step 3 but rather save it for later. This will help discover odd-holes of larger size.

Finally, we have the following optimization problem for the upper-bound flow vector solution as the first stage of ORL-CSMA algorithm, shown in figure 4.2. Note that the relaxation step is done by summing over the time dimension of the 0-1 representation of the schedule to get the number of time periods assigned to each application on each link, i.e. $\mathit{rounds}(a,i,j) = \sum_t \mathit{sch}(a,i,j,t)$

$$\max \sum_a eval(\mathbf{u}(a), \mathbf{b}(a))$$

over variables

$$\mathit{flow}(a,i,j) \geq 0, \mathit{rounds}(a,i,j) \in \{0,1,...,\|T\|\}, \mathbf{b}(a) \geq 0 \quad \text{for } a \in A, i,j \in N$$

subject to

$$(I) \forall a,i,j : \mathit{rounds}(a,i,j) \leq \|T\| \cdot \mathit{link}(i,j)$$

$$(II) \begin{cases} \forall i,j,t : \sum_a \mathit{rounds}(a,i,j) \leq \|T\| \\ \forall i,j,p,q \text{ s.t. } \mathit{conflict}(i,j,p,q) : \\ \sum_a \mathit{rounds}(a,i,j) + \sum_a \mathit{rounds}(a,p,q,t) \leq \|T\| \end{cases}$$

$$(III) \begin{cases} \forall a,i,j : \mathit{flow}(a,i,j) \leq \frac{Cap(l_{ij})}{\|T\|} \cdot \mathit{rounds}(a,i,j) \\ \forall a : \mathbf{b}(a) = \sum_j \mathit{flow}(a, \mathit{src}(a), j) = \sum_j \mathit{flow}(a, j, \mathit{dest}(a)) \\ \forall a, i \text{ s.t. } i \neq \mathit{src}(a), i \neq \mathit{dest}(a) : \\ \sum_j \mathit{flow}(a,j,i) = \sum_j \mathit{flow}(a,i,j) \end{cases}$$

Repeat several times: Find a clique Q

$$(IV) \sum_{l_{ij} \in Q} \sum_a \mathit{rounds}(a,i,j) \leq \|T\|$$

Repeat several times: Find an odd-hole H

$$(V) \sum_{l_{ij} \in H} \sum_a \mathit{rounds}(a,i,j) \leq \left\lfloor \frac{\|H\|}{2} \right\rfloor \cdot \|T\|$$

Figure 4.2: **Relaxed Integer Linear Program in the first stage of ORL-CSMA..**

The blocks (I) through (III) are the direct corollaries of the corresponding constraints in

OPT by summing over time dimension. Block (IV) and (V) are repeated several times to include many clique and hole constraints.

Notice this is still an *integer* program because the variables $\mathit{rounds}(a, i, j)$ are discrete. A further relaxation can be done by dropping the integer constraints on the *rounds* variables, which will reduce the problem to a much easier non-integer Linear Program.[2] In our implementation we did not opt for such further relaxation because it will relax the upper-bound even further. Because the integer version above already offered substantial computational relief from the full OPT version, it is unnecessary to make further relaxation risking an looser upper-bound.

4.3 Modified Rate-Limited CSMA Scheduler

Carrier Sense Multiple Access (CSMA) is a popular medium access control protocol used in modern wireless communications [20]. In CSMA, a device wishing to start transmission first listens to the radio channel to see if it is already occupied by other transmissions. If the channel is free, it begins transmission; if not, it will back off for a period of time and try again. Typically the back-off time limit will increase over time as a measure of congestion control, and the transmission attempt is reported to fail when a certain time-out limit has passed since the first attempt or after a fixed number of retries.

As a MAC protocol, CSMA only dictates how and when a transmission occurs on a *given link*. It is up to the higher-level routing protocol to decide what *path* data should follow. Since the upper-bound flow vector outputted by the first stage optimization problem contains both routing path and bandwidth information, we will need to have a combined routing-and-MAC layer that translates the flow vector down to a feasible schedule. Therefore, we develop the following routing-and-scheduling algorithm that follows the routing path of the flow vector and uses a CSMA scheme on the link level, with a rate limit equal to the bandwidth of the corresponding link in the flow vector. We call it (Approximately) Optimal Rate Limited CSMA. In ORL-CSMA:

- **Every node** will transmit data only in units that are equivalent to the time length of

[2]Non-integer linear program is in P.

one round in the TDMA schedule, we call this a "package" [3]. There will be a very short period of "contention period" at the beginning of each round during which different nodes will perform carrier sensing and start transmission. The first node to sense an unoccupied channel will immediately begin occupying it and subsequent nodes will sense the activity and back off to a random time in the next contention period. We make the simplifying assumption that the contention period is negligible in length compared to the round length, and hence the beginnings and ends of all transmissions are aligned round boundaries (more or less).[4] This process of contention and transmission is illustrated in figure 4.3.

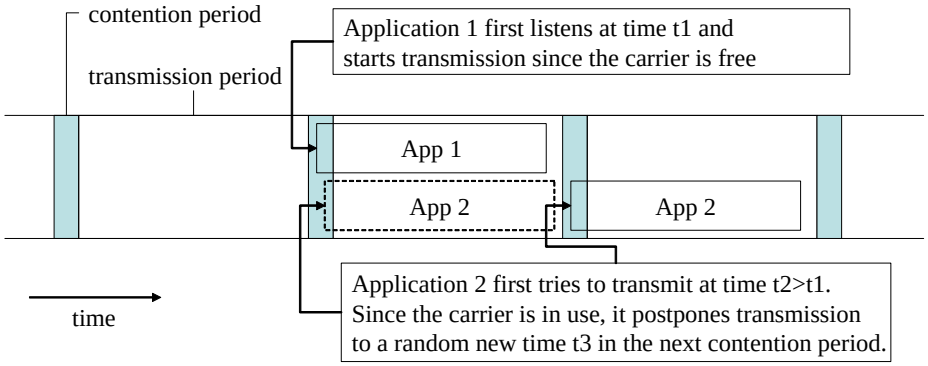


Figure 4.3: **Contention process in second stage of ORL-CSMA.**. *Two applications are ready to transmit on two interfering links. The length of contention period in above picture is exaggerated – in reality we assume that the length of contention period is negligible compared to the length of a round.*

- **Source node** will issue packages addressed to its neighbors at a rate and fashion specified in the upper-bound flow-vector. In the example upper-bound flow-vector shown in Figure 4.4, source node s will issue 1 package to a and 3 packages to b during one epoch. It is acceptable that it simply issues those attempts at the beginning of each epoch and lets the random contention process decide which one goes first.

[3] Although it could be made up of several $packets$ in the normal sense

[4] An alternative interpretation is that the center scheduler in fact simulates this randomized CSMA process and announces the simulation result as a TDMA schedule to be implemented by the nodes.

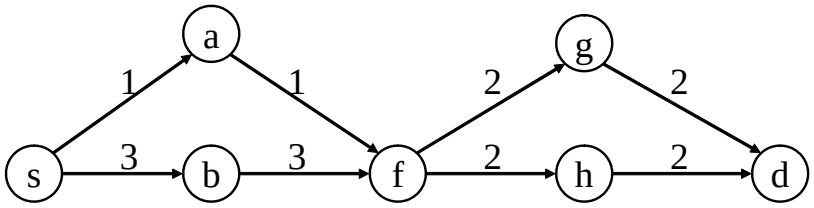


Figure 4.4: **Examples of a multi-path flow-vector**. *Node f has multiple outflows and will forward incoming packages to g or h according to a dispatch schedule.*

- **Forwarding node**, upon receiving successful transmission from its neighbors, will issue an attempt to retransmit the package at a random time in the immediately following contention period to the next downstream neighbors according to the flow-vector. For forwarding nodes with single outflow (such as a, b, g, h) there is no ambiguity as of where to forward next. For forwarding nodes with multiple outflows (and perhaps multiple inflows) such as f, it uses the following rules to decide which node to forward next:

- A **dispatch schedule** is maintained by each forwarding node (for every application) as a random permutation of its outflows (e.g. one possibility of node f's dispatch schedule is (g, h, g, h)) and it will forward incoming packages according to the dispatch schedule. The idea behind this is that we want to avoid determinism resulted from static mapping from a unit inflow to a unit outflow. The dispatch schedule can be fixed over time (since the order of incoming packages are already randomized) or re-permutated every epoch. This strategy ensures that the resulting CSMA schedule mimics the flow-vector to the best extent.

- A package will be **dropped** when it has not reached the destination node after a period of expiration time (20 epochs in our code).

Implementation-wise, we simply keep a global queue of all in-transit packages, together with (*round_number*, *application*, *source*, *destination*, *current_node*, *next_node*, *expiration*) information. The queue is prioritized by *round_number* but the order among packages with the same *round_number* is random (this effectively simulates the random back-off and contention). The simulator simply repeats processing the first element of the

queue, either forwarding it (*round_number++*; *current_node* = *next_node*; *next_node* = $dispatch(\textit{app}, \textit{next\_node})$; $requeue$) or backing off (*round_number++*; $requeue$) or dropping it (if *round_number* > *expiration*).

At the end of the simulation, we count the number of packages that actually arrived at the destination for each application in each epoch,[5] and compute the expected bandwidth and expected utility. The first few epochs will suffer from *cold-start* effect: On one hand, it may take a package several epochs to reach the destination, which reduces throughput for the first few epochs; on the other hand, the lack of previously stalled in-transit packages may have the opposite effect. We run the simulation for sufficiently large number of epochs (1000 in our code) to average out the cold-start effect and estimate the expectation in the steady state.

4.4 Effectiveness of Optimal-Rate-Limiter for CSMA

In NAIVE-CSMA, applications will keep trying to send more packages up to the maximal bandwidth, which causes extra collisions and contentions on the medium. As demand over-saturates, the contention overhead is so overwhelming that the achieved bandwidth and network usage level actually decreases over a certain critical level. The optimal rate limiter tries to get rid of this problem. To confirm this, we compare ORL-CSMA to NAIVE-CSMA with no rate-limit (i.e. the smaller of the the number of periods and the maximum bandwidth level of an application is the "rate-limiter"). We will postpone presenting the data until later sections (see figures 6.3 and 6.4 for the bandwidth and utility comparison between ORL-CSMA and NAIVE-CSMA.)

Because our implementation of the CSMA simulator assumes perfect detection of carrier status, the effect of collisions is eliminated. Even so, NAIVE-CSMA suffers at high demand compared to ORL-CSMA because link resources are used un-planned and may have been wasted to deliver packages that are more likely to be dropped later. In practice, when detection of carrier status is imperfect, collisions may occur when an application

[5] Although the choice of epoch boundary at the destination node is somewhat arbitrary, this choice does not matter because the averaging over epochs.

starts transmitting when the medium is in fact in use. Under such circumstances we expect NAIVE-CSMA to suffer even more compared to ORL-CSMA.

5

A Market-Oriented Bandwidth Allocation Protocol

The optimal utility allocation problem can also be viewed as a complex economic system well suited for general equilibrium theory, which in a nutshell states the following: A collection of self-interested agents (both consumers and producers) engage in an exchange economy. Each of them has an initial endowment of goods and wealth and acts according to the goal of maximizing its own utility, which is a function of its final consumption of goods and wealth at the end of the exchange. A set of prices are defined over the goods being exchanged and all agents respond to the prices by choosing demand for each good (positive for buyers and negative for sellers). When the prices are set such that the aggregate demand for each good is equal to the total initial endowment the market reaches a *general equilibrium*.

Many market-oriented approaches have been derived from the general equilibrium theory to address various resource allocation problems (see works by Cheng and Wellman [9], Wellman [26, 27], Ygge and Akkermans [29]). Most of them employ a process called *tatonnement*, whereby prices and demands respond to each other through a series of interim auctions until equilibrium is reached. A list of auctioneers will each be in charge of setting the price and monitoring the excess demand for one particular good. The price vector starts with some arbitrary initial values. In each iteration, the current price vector is announced and all agents response to the prices and submit their demands to the auctioneers. Then *one* auctioneer will change its price incrementally according to some update rule in order to drive excess demand for that good towards zero. The new price vector is

announced in the next iteration and the process repeats until the price vector converges and all excess demands tend to zero.

We will adopt the tatonnement market process in our market-oriented approach to the optimal utility bandwidth allocation problem, which will be termed MARKET. It replace the first stage of upper-bound computation in ORL-CSMA. The resulting scheduling algorithm (MARKET allocator + best-effort CSMA scheduler) will also be called MARKET when no ambiguity arises. Figure 5.1 provides an overview of the MARKET algorithm.

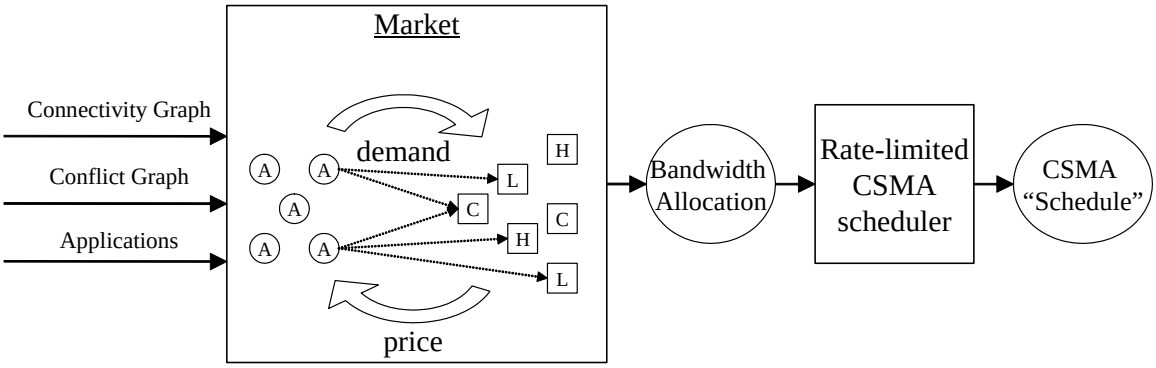


Figure 5.1: **MARKET protocol overview**. *A market is set up between the applications and various types of auctioneers controlling different resources. This replaces the first stage of* ORL-CSMA. *After iterative rounds of price-setting and demand-computation the market converges to some equilibrium. The equilibrium allocation of the market is passed on to the best-effort CSMA scheduler as the rate limiter.*

5.1 Market Design

In our model, each application is naturally a consumer agent in the market, ready to consume network resources in order to complete its function and maximize utility. The producers in the market are the virtual owners of various types of resources, which will become apparent after define the goods in the market.

The meaning of goods in our market is much more interesting and essential to the design of the market. The natural definition of goods as "usage of a wireless link for one unit of time" will not work well because the essential interference interaction between links is loss due to the fact that different goods have their own markets and work more or less independently of each other. We would like our notion of goods to capture not only tangible resources like link usage, but also more importantly the less tangible resources

of a "free wireless medium" that is used up by interference. In light of the additional constraints added to the first stage ORL-CSMA optimization problem to capture wireless interference, we will consider cliques and odd-holes as goods as well.

Formally, a good in our market is defined as *a set of links*, which we will call an *interference group*. Intuitively, a good indexed by the the interference group g is a license or permission to use *any one* of the links in g for one period of time. Conversely, in order to be able to use a physical link l in the final allocation, an application has to purchase *all* of the goods whose interference group l is a member of. Analogous to the taxation of gas emissions as a way of pricing externality in economics, defining interference groups as concrete goods in the market captures the negative externality of using a link due to wireless interference. In this "interference licensing" scheme, the cost of preventing others from transmitting due to interference is then internalized into the effective cost of the link causing interference.

We will consider the following types of goods in our market:

- A **Link Pair** type good consists of a pair of forward and backward links $L = \{l_{ij}, l_{ji}\}$. Since only one of them can be active during any period, the supplied quantity of this goods is the total number of periods $\|T\|$.

- A **Clique** type good is a set of links that form a clique Q in the interference graph. The supplied quantity is also the number of time periods $\|T\|$ since they mutually forbids each other in every period.

- An **Odd-Hole** type good is a set of links that form an odd-hole H in the interference graph. The quantity of this good $\left\lfloor \frac{\|H\|}{2} \right\rfloor \cdot \|T\|$ since no more than half of the links can be active at the same time.

The introduction of virtual clique and odd-hole goods in addition to the physical link-pair goods in order to capture the effect of interference is a novice contribution of this thesis compared to previous market-oriented solutions to the similar network transportation and multi-commodity flow problem [27]. It will turn out later that these interference-embodying goods (clique goods in particular) are the most demanded bottleneck goods in

the market.

Unlike in a classical market where the producers actively choose the quantity of goods to produce, our producers for the goods defined above will simply produce a fixed supply of the goods as dictated by network topology and interference characteristics. We will use s_i to denote the auctioneer in charge of selling good g_i. We will also call them "link auctioneer", "clique auctioneer" and "hole auctioneer" according to the type of goods they are selling.

Suppose there are a total of m physical links $\{l_1, ..., l_m\}$ and k goods $\{g_1, ..., g_k\}$,[1] we use the k-dimensional vectors $\mathbf{p}$, $\mathbf{q}$ and $\mathbf{x}$ to denote the *price vector*, *supply vector* and *demand vector* for the goods, respectively. On the other hand, because an application works naturally by first selecting a set of physical links that it wants to use and then procuring all the goods required to operate those links, we will also define the *effective link price vector* $\mathbf{p}^L$ and the *link demand vector* $\mathbf{x}^L$ as an alternative representation of the prices and demands. The effective link price p_i^L for each link l_i is the sum of prices of all goods that the link is a member of:

$$p_i^L = \sum_{1 \le j \le k : l_i \in g_j} p_j \tag{5.1}$$

In response to these effective link prices, each application decides how many of each link it wants to use (in order to form a flow), which we will call the *application-level link demand vector* $\mathbf{x}^{La}$. The component x_i^{La} denotes the quantity of physical link l_i that application a demands. The sum of $\mathbf{x}^{La}$ across all applications becomes the *aggregate link demand vector*, denoted $\mathbf{x}^L$. The good-based demand is related to the link-based demand by:[2]

$$x_j = \sum_{1 \le i \le m : l_i \in g_j} x_j^L \tag{5.2}$$

---

[1] Note that k is not necessarily greater than m, because every pair of *two* directed links become one good, and the number of clique goods and odd-holes could vary.

[2] Note that the directions of aggregation are opposite in the case of demand and price. i.e. Link prices are obtained by summing over goods prices, whereas goods demands are obtained by summing over link demands.

Equations (5.1) and (5.2) can be concisely written in vector notation:

$$\mathbf{p}^L = \mathbf{M} \cdot \mathbf{p} \tag{5.3}$$

$$\mathbf{x} = \mathbf{M}^T \cdot \mathbf{x}^L \tag{5.4}$$

where $\mathbf{M}$ is an $m \times k$ matrix with $M_{ij} = 1$ if l_i is a member of g_j and 0 otherwise, and $\mathbf{M}^T$ is the transpose of $\mathbf{M}$.

To summarize, $\mathbf{p}$, $\mathbf{x}$ and $\mathbf{q}$ are the k-dimensional price, demand and supply vector defined over k goods, and $\mathbf{p}^L$ and $\mathbf{x}^L$ are the m-dimensional effective link price and link demand vector defined over m links. Adding superscript a to the demand ($\mathbf{p}^a$ and $\mathbf{p}^{La}$) denotes application a's contribution to the demand. In addition, since the MARKET protocol will consist of a sequence of iterated auctions, we will use e.g. $\mathbf{p}(t)$ to denote the value of price vector at iteration t.

Finally we present a high-level description of the MARKET protocol as the following:

1. $\{g_1, ..., g_k\} \leftarrow$ *CHOOSE_GOODS*.
2. $t \leftarrow 0$, $\mathbf{p}(0) \leftarrow \mathbf{0}, H(0) \leftarrow \phi$.
3. Price vector $\mathbf{p}(t)$ is announced.
4. Each application responds to $\mathbf{p}(t)$ by doing:
   (a) Translate [3] good-price $\mathbf{p}(t)$ to effective link price $\mathbf{p}^L(t)$.
   (b) Compute its link demand vector $\mathbf{x}^{La}(t)$ = *APP_DEMAND*(a,$\mathbf{p}^L(t)$).
   (c) Translate link demand $\mathbf{x}^{La}(t)$ to good demand $\mathbf{x}^a(t)$.
   (d) Submit quantity demanded for each good $x_j^a(t)$ to the auctioneer s_j for good g_j.
5. Each auctioneer s_j sums over all bids from applications to get the aggregate demand $x_j(t)$ for good g_j.
6. If *CONVERGE?*$(H(t), \mathbf{p}(t), \mathbf{x}(t))$ output $\mathbf{x}^{La}(t)$ as the link allocation for each application a.

[3] The translation could also take place in step 3 and $\mathbf{p}^L(t)$ is also announced.

7. Otherwise *one* auctioneer s_j is selected,[4] which then updates the price for its good. All other prices remain the same. $\mathbf{p}(t+1) \leftarrow$ *UPDATE_PRICE*$(j, \mathbf{p}(t), \mathbf{x}(t))$.

8. $H(t+1) \leftarrow H(t) \cup \{\mathbf{p}(t), \mathbf{x}(t)\}$, $t \leftarrow t+1$, go to step 3.

Note that $H(t)$ is the history of price vectors and demand vectors prior to time t. Upon termination of the protocol, the equilibrium link demand for each application $\mathbf{x}^{La}$ becomes the final *link allocation*, which specifies how many of each link every application is allowed to use during the entire epoch of $\|T\|$ rounds. Under the simplifying assumption that $\frac{Cap(l_i)}{\|T\|} = 1$ for all links, one unit of link allocation is equivalent to one unit of flow along that link. Thus the final link allocation $\mathbf{x}^L$ is simply a flow vector,[5] which can be passed on to the next stage CSMA scheduler as the rate-limiter.

The four main modules of the market – the initial *CHOOSE_GOODS* function, the individual demand function *APP_DEMAND*, the price update rule *UPDATE_PRICE* and the convergence condition *CONVERGE?* – will be discussed in greater details in the following sections. But first, let's begin by considering the following example market shown in Figure (5.2). In this market, the network topology is taken from a randomly generated graph (see figure 6.1(a)) and we assume the level-0 interference model. The applications are generated with random source and destination and identical utility function (0,0)-(1,10)-(2,15) shown in figure 2.2(a). We have included all link auctioneers (L0 to L9) and all clique auctioneers (Q10 to Q16).[6] The supplied quantity of each good is equal to the number of time periods (10). For simplicity of presentation, no hole auctioneers are added.

In the initial iteration, all prices are set to zero. Each application demands the maximum units of flow from its source to its destination (2 in this case because of the domain of the utility function). It then assembles all links along the path and demands the appropriate number of goods required to operate those links. For example, application 8 wants

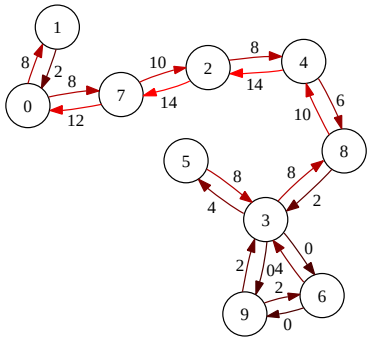
[4]Only auctioneers who want to change their prices are considered. *CONVERGE?* should be true if no auctioneer wants to change its price.

[5]Applications will choose a link demand that satisfies the flow conservation constraint, as we will explain in the next section.

[6]In a small network like this it is possible to enumerate all maximum cliques, but in general it is not feasible in a larger network.

App#	s	d
0	4	0
1	4	1
2	0	8
3	5	4
4	4	0
5	6	1
6	1	2
7	4	8
8	9	5
9	2	1
10	0	5
11	5	1
12	8	7
13	0	4
14	5	3
15	7	4
16	9	6
17	6	8
18	5	3
19	8	2

(a)



(b)

Auctioneer	Type	Member-links
Seller 0	(LINK):	<0,1> <1,0>
Seller 1	(LINK):	<0,7> <7,0>
Seller 2	(LINK):	<2,4> <4,2>
Seller 3	(LINK):	<2,7> <7,2>
Seller 4	(LINK):	<3,5> <5,3>
Seller 5	(LINK):	<3,6> <6,3>
Seller 6	(LINK):	<3,8> <8,3>
Seller 7	(LINK):	<3,9> <9,3>
Seller 8	(LINK):	<4,8> <8,4>
Seller 9	(LINK):	<6,9> <9,6>
Seller 10	(CLIQ):	<3,6> <3,9> <6,3> <6,9> <9,3> <9,6>
Seller 11	(CLIQ):	<3,5> <3,6> <3,8> <3,9> <5,3> <6,3> <8,3> <9,3>
Seller 12	(CLIQ):	<0,7> <2,7> <7,0> <7,2>
Seller 13	(CLIQ):	<3,8> <4,8> <8,3> <8,4>
Seller 14	(CLIQ):	<0,1> <0,7> <1,0> <7,0>
Seller 15	(CLIQ):	<2,4> <4,2> <4,8> <8,4>
Seller 16	(CLIQ):	<2,4> <2,7> <4,2> <7,2>

(c)

Figure 5.2: **An example market**. *(a) List of applications, with random source and destination and utility function shown in figure 2.2(a); (b) A random network topology identical to 6.1(a), numbers on links show the aggregate link demand at $t = 0$; (c) List of auctioneers (no hole-auctioneers for simplicity).*

to use 2 units of link $\langle 9, 3\rangle$ and 2 units of link $\langle 3, 5\rangle$ in order to form a flow of bandwidth 2 from node 9 to node 5. Link $\langle 9, 3\rangle$ requires goods L7, Q10 and Q11 to operate, and link $\langle 3, 5\rangle$ requires goods L4 and L11 to operate. Therefore, application 8 demands 2 units of L4, L7, Q10 and 4 units of Q11. Note that good Q11 is requested 4 times instead of 2 because both links are members of Q11, and they must have separate licenses to block the interference group.[7]

The aggregate link demands ($\mathbf{x}^L$) are shown directly on each links. Most demanded links are shown in red color. It is easy to see that a congested part of the network is the path from 4 to 0. This is due to the combination of this particular traffic shape and the network topology (with a long-extended one-way path). A more subtle congested part of the network is the star-shaped links around node 3. Although individual link demands are low, the sum of them (26) far exceeds the supply for clique good L11. This demonstrates why link goods only are insufficient to capture congestions due to interference, whereas

[7]This corresponds to the fact that in order to transport 2 packages from 9 to 5 along the route $9 \to 3 \to 5$, it must be the case that node 3 be active for 4 periods of time and therefore all other links in group Q10 must be blocked for 4 periods.

goods of bigger interference group (cliques) are apt in doing so. We will see later that the MARKET protocol is able to effectively price the bottleneck goods that correspond precisely to these congestions.

We will follow this example throughout the remaining sections in order to illustrate some fine points of the internal workings of the MARKET algorithm.

5.2 Individual Demand Function (Single-App Optimal Flow)

Given the price vector $\mathbf{p}^L$, the sub-problem facing each application is to find the optimal link demand in order to form a flow that gives the maximum net utility:

$$\mathbf{x}^{La}(t) \in \arg\max_{\mathbf{x}^L \in X^L} u_a(bw(\mathbf{x}^L)) - \mathbf{x}^L \cdot \mathbf{p}^L(t) \tag{5.5}$$

Note that $\mathbf{x}^L$ here is simply a dummy variable for the $\arg\max$ operation, not to be confused with the global aggregate link demand. $bw(.)$ denotes the bandwidth of $\mathbf{x}^L$ (treated as a flow vector), and the dot-product computes the total price of the link allocation $\mathbf{x}^L$ the application has to pay under the current link price vector. The dual meanings of $\mathbf{x}^L$ as both the link demand and the corresponding flow vector are due to the assumption that $Cap(l_i) = \|T\|$ for all links.[8]

Obviously the set of all possible link demands that the application is choosing from (X^L) should be limited by the flow conservation constraints – choosing any extra links that are not needed to form a flow will not increase the bandwidth or utility, but will only increase the price. However, the important question becomes whether we should impose the constraints that an application cannot demand more than the total supplied quantities of goods (i.e. $\mathbf{M}^T \cdot \mathbf{x}^L \leq \mathbf{q}$). It turns out that adding these supply constraints is equivalent to imposing the clique and hole constraints from the ORL-CSMA first stage optimization in this single-application case, since the supplied quantities of clique goods and hole goods correspond exactly to the clique and hole constraints in ORL-CSMA. When the clique and hole constraints are observed, the complexity of the sub-problem becomes

[8] In general, an arbitrary flow vector $\mathbf{f}$ can be translated to a link demand by scaling the component for link l_i by a factor of $\frac{\|T\|}{Cap(l_i)}$ and rounding up to the next integer (as done in the first stage of ORL-CSMA).

potentially as hard as the original global optimization problem.[9] Therefore, in order to limit the complexity of the single-application sub-problem, we will not bound the demand function by the global supply of goods.[10]

If we do not add the supply constraints, it is possible for an application to request more than the supplied quantity of some goods. This is not a problem in particular because the price of over-demanded goods will be increased until eventually the demand is no more than the supply (price adjustment rule will be explained in detail in the next section). Thus, without considering the supply constraints, the single-application optimal flow problem reduces to a classic shortest distance problem with the effective link prices $\mathbf{p}^L(t)$ as the distance metric. Each application will simply compute the shortest distance (minimum cost in this case) path between its source and destination nodes, and will request multiple units of the links along that path. The number of units request will be chosen to maximize net utility, and in the case of concave utility function, the application will keep increasing this number until the cost is higher than its marginal utility.

Note that under such single-application demand algorithm, an application will always choose several units of a *single* minimum-cost path, and hence split-path flow will not be induced. Although this is not as desirable as we would like, it is inevitable because every application makes its independent response to prices without knowledge of other applications' demands. This is a weakness of MARKET compared to the global omniscient optimizer in OPT or ORL-CSMA.

5.3 Price Updating Algorithm

We consider the following price updating rules

- **Simple Reinforcement / Negative-Feedback Update Rule:** An auctioneer will simply push the price up or down by a small increment δ depending on whether the

[9]Consider a reduction of the global multi-application optimization to this single-application optimization by adding a virtual source node linking to the original source nodes of all applications, and linking all destination nodes to a new virtual destination node. With a modified utility function, we have reduced the original global optimization problem to a single-application optimal flow problem.

[10]Note that the complexity of the individual demand function is not typically a concern in economic settings or in previous works of computational market approaches.

goods are over-demanded or under-demanded. That is, $p_j(t+1) \leftarrow p_j(t) + \delta$ if $x_j(t) > q_j$ and $p_j(t+1) \leftarrow p_j(t) - \delta$ if $x_j(t) < q_j$. Proportional updates can also be used ($p_j(t+1) \leftarrow p_j(t) + \delta(x_j(t) - q_j)$ with smaller δ). However, in our experiments a fixed increment tends to lead to less fluctuation, probably because proportional changes tend to overshoot the efficient prices due to the failure to expect discontinuous changes of the demands around the efficient prices.

- **Gradient Descent:** If we back up a step, the price-demand interaction could be viewed as a global search problem to find a set of efficient prices $\mathbf{p}^*$ to minimize the error between the demand function and the fixed supply. Viewed as such, local search techniques such as gradient descent can be employed to search for the optimal prices. Ygge and Akkermans [29] found that a Newton-style price adjustment rule leads to faster convergence compared to simple price updates in their market settings. However, in our setting, because the auctioneers do not have more information of the excess demand function other than its value at the current price levels, the gradient of the demand function must be estimated by letting each auctioneer explore the neighborhood prices $p_j(t) \pm \delta$. The re-evaluation of the excess demand function will add much overheads to our protocol and will also prevent its decentralization. Further difficulties may arise due to the discontinuity of the demand function because of interference and complementarity of goods. Therefore we will not use this rule here.

The first simple update rule turns out to be quite effective empirically in terms of reaching a good output quality. However, its convergence property presents a challenge, which we will see in the next section.

5.4 Market Convergence

5.4.1 Theory

Ideally, we would like the market to converge to an equilibrium where supply equals demand for all goods. However, such strong convergence condition is overly strict and unlikely to be met in our market setup, because it is impossible for all goods (links, cliques, holes) to be used up exactly as their supplies allow. Since the supplied quantities of each link-pair is the number of time periods, demand equaling supply for all link-pair goods implies that all links are active for all time,[11] which is impossible due to interference. Under the simple incremental price adjustment rule, we expect that prices for under-demanded goods will be driven to zero over time, while prices for popular goods (or bottleneck goods) to be non-zero. Even if we ignore zero-priced goods when considering market clearing, the demands for non-zero-priced goods still may not converge. Hence, weaker conditions of convergence and termination should be sought. We will consider the following alternatives:

- **Near-Convergence:** One option is to relax the convergence conditions by allowing a small range of error when considering supply "equals" demand. e.g. When a good has quantity q of supply, the auctioneer may consider demands in the range $[q-2, q]$ to be "cleared" and stop adjusting prices. This introduces an additional stable zone to the demand-supply interaction. Note that extending the range upward is undesirable since the supply quantity bound corresponds to a necessary condition of the schedulability of the resulting flow, so violating the supply bound will guarantee a non-schedulable flow. Extending the range downward is acceptable since the output is just an upper-bound rate-limiter to be passed to the CSMA scheduler.

- **Pseudo-Convergence:** In economics theory, the general results about the stability of price adjustment processes in general equilibrium models are negative. Scarf [23] observed examples where the price and demand state of a market tends to form a cycle without converging to the unique competitive equilibrium. In a discrete price

[11] More precisely, for every pair of connected nodes, one of the two directional links between them are active.

and demand space in our setting, we expect the market to fluctuate among a limited number of states after a certain number of initial iterations. Indeed this cycling behavior is frequently observed in our experiments. When the market falls into a trap and cycles through only a limited number of states, we say that it reaches *pseudo-convergence*, and the set of states that it cycles through form a *dynamic equilibrium*.[12] When the market reaches pseudo-convergence, some linear combination of the equilibrium states, or a state randomly selected from the equilibrium states according to their empirical distribution, can be outputted to the next stage CSMA scheduler.

5.4.2 Example

As a first attempt, we will allow our example market shown earlier to develop without relaxing the market clearing rule. With $\delta = 0.02$, the evolution of price vector and demand vector are plotted in figures 5.3 and 5.4. Initially when prices start at zero, demand for most goods is higher than the supply, thus the prices for all goods increase uniformly, causing demand to drop. The demand for less popular goods (such as L4) first drop below the supply line (10) and cause their prices to decrease back down to zero while other prices continue to rise. Over time, the prices for the most-demanded goods continue to increase and stabilize, while the prices for less popular goods fall back to zero, as we have expected. After about 1000 iterations, the price for Q11 stays as constant because demand is exactly equal to supply, whereas prices for Q12, Q15, Q16 follow a small "random walk" around about 1.5 and 2.5 as the demand fluctuates above and below the supply. Notice that the four goods with non-zero prices eventually (Q11 Q12 Q15 Q16) correspond exactly to the congested part of the network (see figure 5.2). This demonstrates the ability of the MARKET algorithm to effectively price bottleneck resources in the network. The initial and final demand vectors are also plotted later in figure 5.4.3.

Compared to the price vector which is more or less stabilized over time, the demand vector appears to be much more volatile and chaotic, even after 1000 iterations. In order to understand the behavior of the demand vector, we list the cumulative counts of each *unique* demand vector in table 5.1 and plot the composition of demand vectors for every

[12]The exact criteria and algorithm to detect pseudo-convergence will be discussed soon.

100 iterations in figure 5.5. As we can see, the initial demand vector stayed constant for more than 100 iterations due to the slow change in prices ($\delta = 0.02$). After that the market transitions through several intermediary demand vectors that appear only briefly over the course of the iterations. Interestingly, Demand Vector 3 appears to be a local attracting point of the market state, which stayed for about 300 iterations from 600-900, but was eventually replaced by others.[13] Demand Vectors 1 and 2 become increasingly dominant in terms of their empirical frequencies, and eventually become the final two demand states that the market cycles through.[14] Although we have only plotted the graph until iteration 2400, the two remain as the only demand vectors until the end of data collection. Therefore, we conclude that this example market reaches pseudo-convergence with two equilibrium demand vectors, which we will term the "Equilibrium(+)" and "Equilibrium(-)" demand vectors.

[13]Such meta-stable state is to be avoided by the equilibrium detection algorithm that we shall develop.

[14]Although each of the two have many corresponding price vectors.

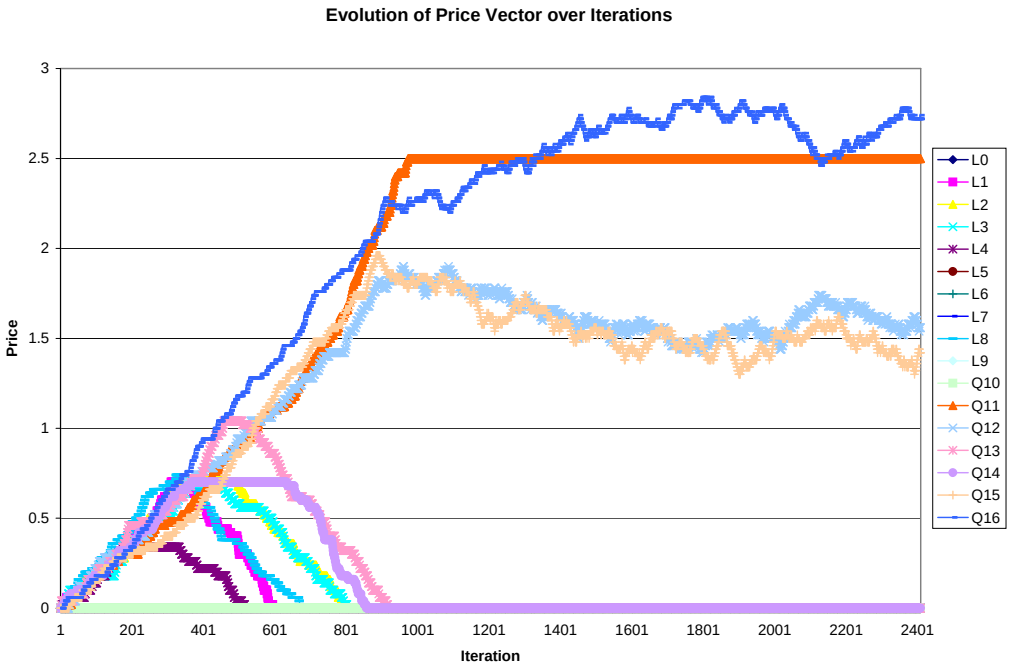


Figure 5.3: **Evolution of Price Vector over Iterations**.

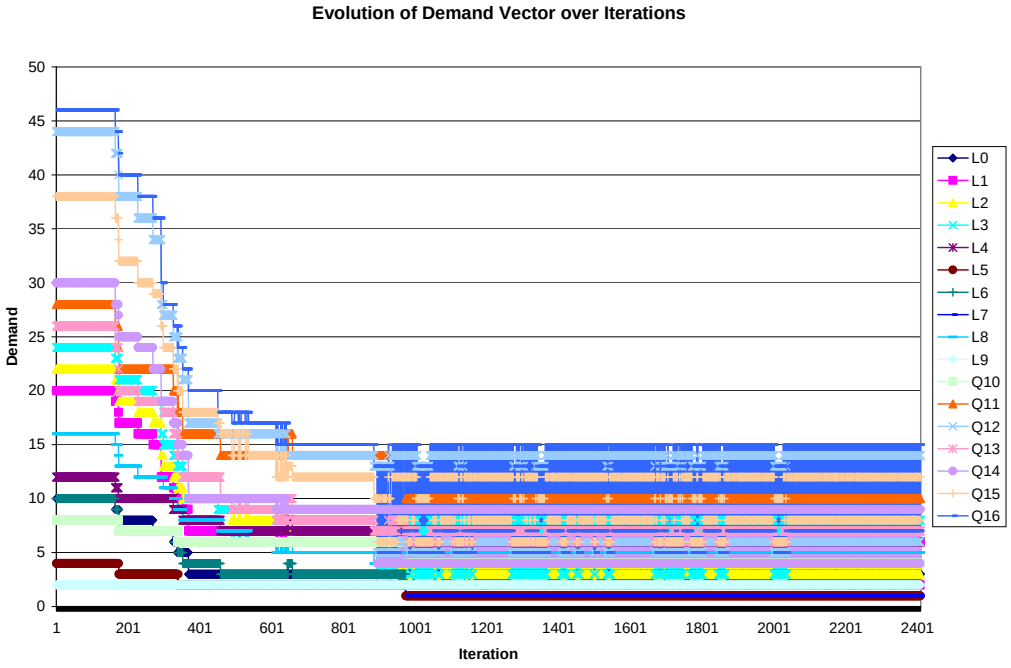


Figure 5.4: **Evolution of Demand Vector over Iterations**.

Rank	Frequency	Demand Vector
1	953	(3,6,7,8,6,1,2,1,5,2,4,10,14,7,9,12,15)
2	852	(2,2,3,4,6,1,2,1,5,2,4,10,6,7,4,8,7)
3	270	(3,6,7,8,7,2,3,2,5,2,6,14,14,8,9,12,15)
4	203	(2,2,2,3,6,1,2,1,4,2,4,10,5,6,4,6,5)
5	164	(10,20,22,24,12,4,10,2,16,2,8,28,44,26,30,38,46)
6	106	(3,7,8,9,7,2,3,2,6,2,6,14,16,9,10,14,17)
7	94	(3,6,6,7,6,1,2,1,4,2,4,10,13,6,9,10,13)
8	82	(3,7,10,10,8,2,4,2,8,2,6,16,17,12,10,18,20)
9	60	(3,7,9,9,7,2,3,2,7,2,6,14,16,10,10,16,18)
10	53	(8,17,19,21,10,3,7,2,13,2,7,22,38,20,25,32,40)
11	42	(8,16,18,20,10,3,7,2,12,2,7,22,36,19,24,30,38)
12	30	(2,2,2,3,7,2,3,2,4,2,6,14,5,7,4,6,5)
13	28	(7,12,13,15,10,3,7,2,11,2,7,22,27,18,19,24,28)
14	27	(3,6,6,7,7,2,3,2,4,2,6,14,13,7,9,10,13)
15	23	(7,15,17,19,10,3,7,2,12,2,7,22,34,19,22,29,36)
16	16	(5,9,10,12,8,2,4,2,8,2,6,16,21,12,14,18,22)
17	14	(5,10,11,13,9,2,5,2,9,2,6,18,23,14,15,20,24)
18	14	(3,6,7,8,8,2,4,2,6,2,6,16,14,10,9,13,15)
19	12	(6,11,12,14,9,3,6,2,10,2,7,20,25,16,17,22,26)
20	10	(2,2,3,4,7,2,3,2,5,2,6,14,6,8,4,8,7)
21	9	(9,19,21,23,11,4,9,2,15,2,8,26,42,24,28,36,44)
22	7	(3,7,9,9,8,2,4,2,8,2,6,16,16,12,10,17,18)
23	6	(7,12,14,16,10,3,7,2,12,2,7,22,28,19,19,26,30)
24	1	(9,18,20,22,10,4,8,2,14,2,8,24,40,22,27,34,42)

Table 5.1: **Demand Vector frequency count**. *Total number of occurrences for each unique demand vector appeared over the course of 3000+ iterations.*

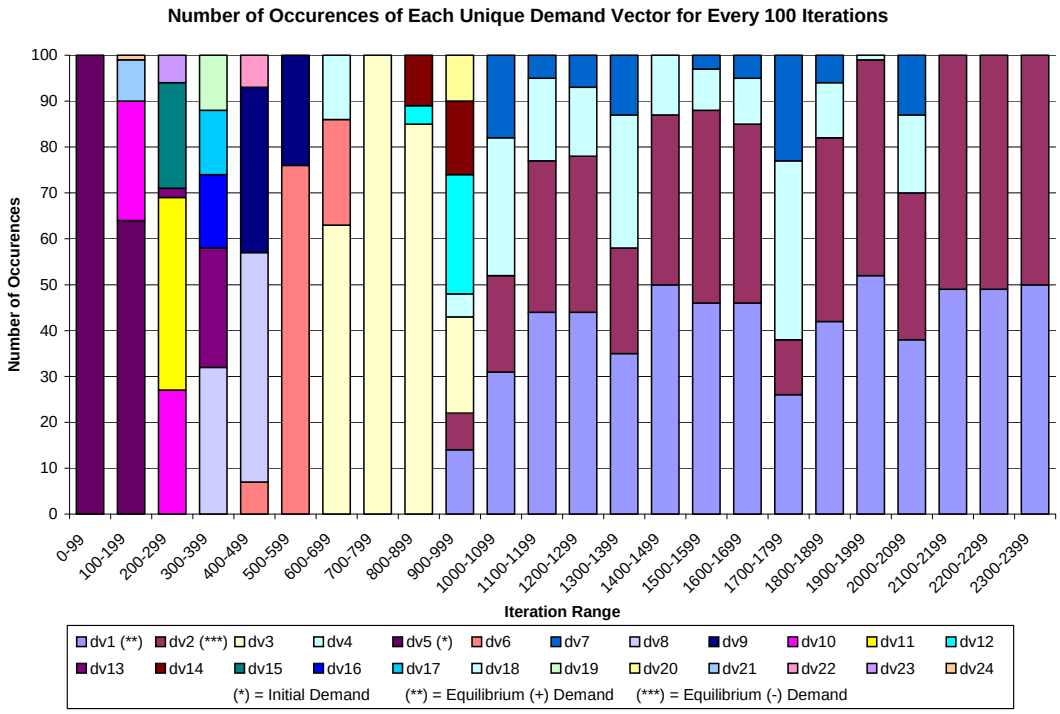


Figure 5.5: **Number of occurrences for each unique demand vector during every 100 iterations**. *"Equilibrium(+)" and "Equilibrium(-)" are the two dominant demand vectors that will eventually form a dynamic equilibrium.*

Figure 5.6 shows a high-level visualization of the market dynamics: the x-axis and y-axis are the L-2 norms of the 17-dimensional price vector and demand vector, respectively ($\|\mathbf{p}(t)\|$ and $\|\mathbf{x}(t)\|$); the z-axis is the number of iterations t. Thus every market state (in $\mathbb{R}^{17} \times \mathbb{R}^{17}$) is mapped to a point in $\mathbb{R}^2$, and the trajectory of the market state is plotted in a $\|p\| - \|x\| - t$ cube with proper scaling. Every point is also color coded according to t. We can clearly see the separation of the trajectory into two branches that extend along the time axis. The projections of the trajectory onto p-t and x-t planes are also shown in grey scale, which show that the norm of the price vector converges after about 1000 iterations, while the norm of the demand vector gradually settles on a fluctuations between the values. Projection onto to the p-x plane closely resembles an aggregate demand curve in economics.

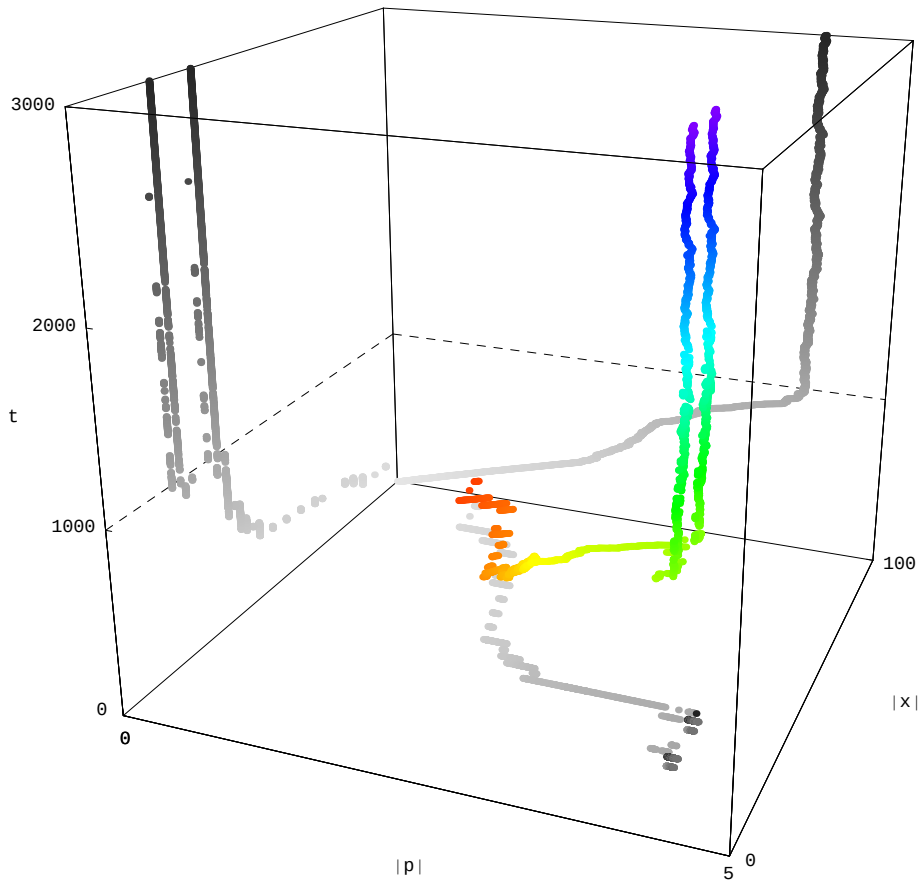


Figure 5.6: **Trajectory of the Market in Price-Demand-Iteration Space**.

5.4.3 Explanations

Here are some factors that may contribute to the pseudo-convergence of the market, in order of decreasing importance:

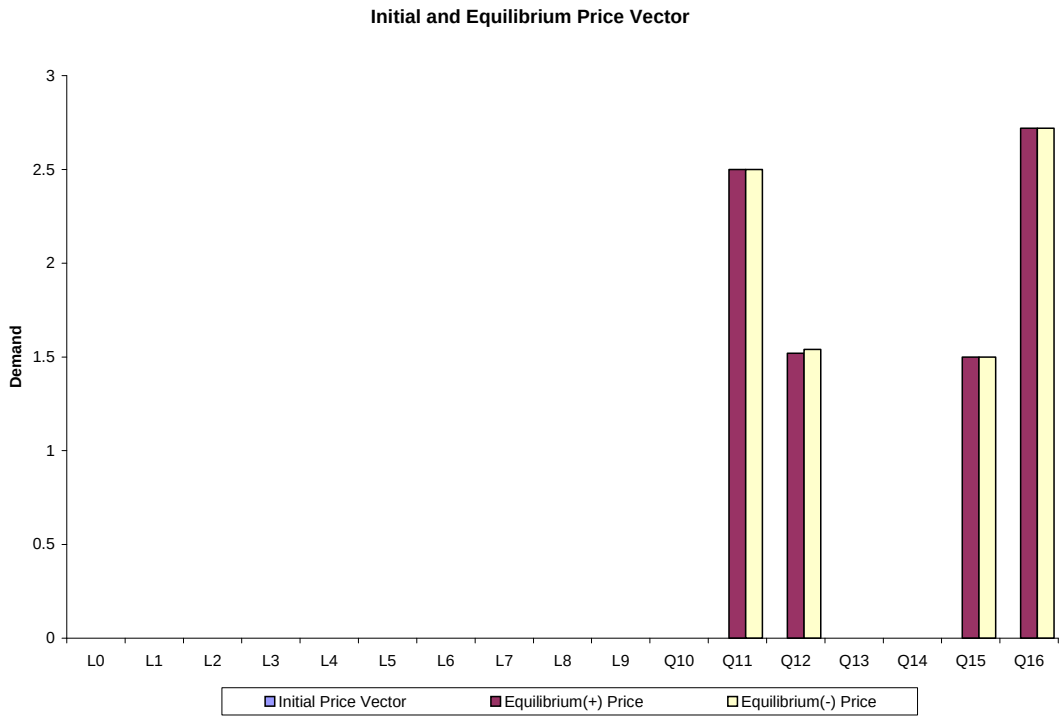
- **Complementarity of goods:** The general stability and convergence property of a competitive market equilibrium has been shown to reply upon certain strong assumptions [6, 9]. Of particular importance is the result demonstrated by Arrow and Hurwicz [4, 5] that *gross substitutability* is a sufficient condition for a tatonnement process to be globally stable. Gross substitutability means that when the price for a good increases, aggregate demands for *other* goods does not decrease. This condition does not hold when there is complementarity between goods, as is the case in our market. An increase in a good's price will lead to an increase of the unit cost of a path that requires that good, and will possibly force an application to choose a completely different path and thus the demand for goods along the original path may decrease. To illustrate this point, we have plotted the final two equilibrium demand vectors and two of their corresponding price vectors in figures 5.4.3. As we can see, a small change in price around the efficient prices will lead to a big change in the demand vector.

- **Discrete price and demand space:** Since our prices are discretized by the increment δ, it is also plausible that the true equilibrium price lies between two step values separated by δ. This effect can be mitigated by reducing the price increment δ, either globally or gradually over time. The discreteness of demands is inevitable in the model with discrete time span.

- **Identical application utility functions:** The fact that all applications share the same utility function in our example market may have exacerbated the problem of big demand changes around the equilibrium, because several applications constrained by one particular good may change their demand at exactly the same price level. In practice when the utility of applications are more diverse this effect is eliminated. We can also add some small random noise on top of an identical utility functions to

reduce this effect.

- **Specifics of network topology and interference characteristics:** It is plausible that some special characteristics of this particular network topology and interference graph is causing an ill-behaved aggregate demand function. However, the phenomenon of cycling equilibrium appears to be common over a wide range of setups with varying network topology, interference model and application profiles. In the experiments presented in section 6.2, 619 cases out of 1125 market instances end in pseudo-convergence (55%), while the others reaches an ideal equilibrium with strictly equal supply and demand for non-zero-priced goods. A general observation is that simpler markets are more likely to reach perfect convergence while more complex markets are more likely to reach pseudo-convergence.



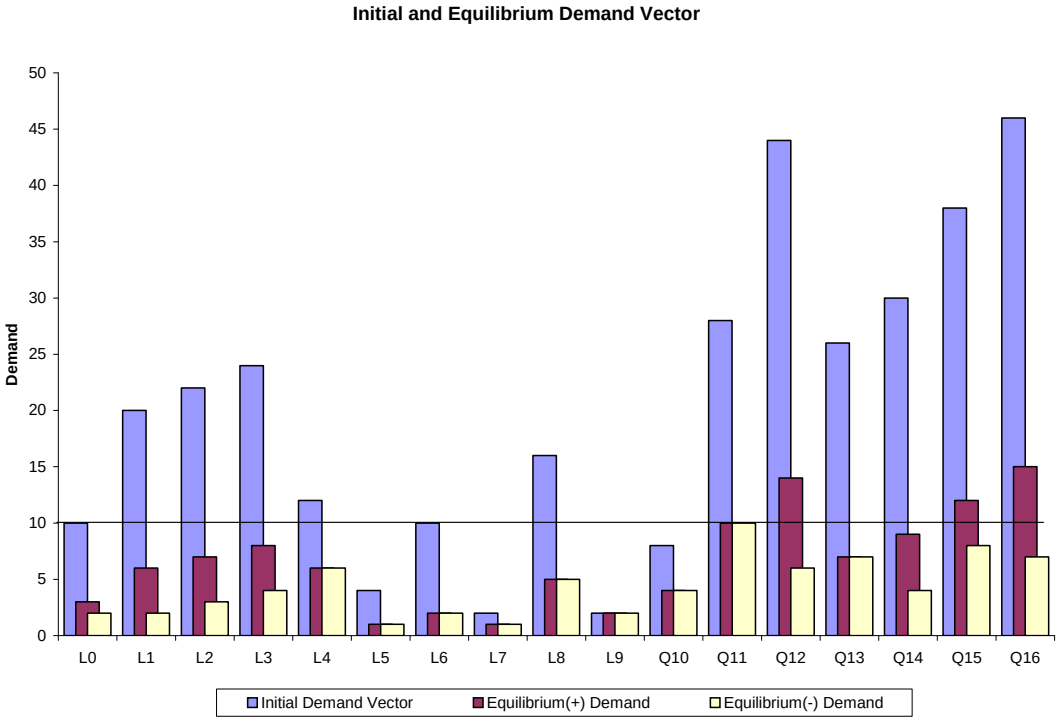


Figure 5.7: **Price and Demand Vector at initialization and equilibrium**. *Note that: a) The initial prices are all zero; b) the horizontal line in the demand graph represents a fixed supply of 10 units for each good; c) most demanded goods at the beginning are also goods with positive prices at equilibrium; d) A small increase in the price for Q12 in the equilibrium prices leads to a big decrease in the demands for Q12 and several other goods.*

5.4.4 Convergence Detection Algorithm

Although by looking at the empirical frequencies of all unique demand vectors we are able to tell that the market eventually fluctuates between several states, such an approach may not be useful programmatically because the number of unique demand vectors grows in the order of $O((ar)^g)$ where a is the number of applications, r is the number of time periods and g is the number of goods. Furthermore, more sophisticated clustering algorithms may be required to identify the fluctuation of prices, which may take on more possible values. Instead, we use the following simple algorithm to detect convergence of prices and demands.[15] The basic idea is that we track the magnitude of the first order difference of the price vector and demand vector. If it tends to zero, the system reaches ideal convergence;[16] if it stabilizes around a certain value, the system is fluctuating between a relatively stable set of states.

Formally, we define the following *Exponentially Weighted Moving Average* (EWA) filter with parameter α for any scalar or vector variable x:

$$E^*_\alpha[x](t) = \alpha E^*_\alpha[x](t-1) + (1-\alpha)x(t) \tag{5.6}$$

Using this, we define the *average price vector* at time t to be an exponentially weighted moving average of historical price vectors:

$$\tilde{\mathbf{p}}(t) = E^*_\alpha[\mathbf{p}](t) = \alpha\tilde{\mathbf{p}}(t-1) + (1-\alpha)\mathbf{p}(t) \tag{5.7}$$

The scalar *first order difference* of the price vector is defined as the L-2 norm of the difference between the current price vector and the average price vector:[17]

$$\Delta_{\mathbf{p}}(t) = \|\mathbf{p}(t) - \tilde{\mathbf{p}}(t)\| \tag{5.8}$$

---

[15] Since this algorithm is made up in an ad-hoc fashion, I may be re-inventing some well-known signal processing techniques but in a wrong way. Anyway, it seems to detect pseudo-convergence quite well in experiments.

[16] Since ideal convergence can be detected much easier when no auctioneer wants to change its price. The presented algorithm will be more useful in detecting pseudo-convergence.

[17] We did not use the natural choice of $\|\mathbf{p}(t) - \mathbf{p}(t-1)\|$ because it will always be δ because of the simple incremental price adjustment rule.

The value of $\Delta_{\mathbf{p}}(t)$ is plotted with blue dots in figure 5.8. Because this is still very volatile, we apply another EWA filter on it to obtain the *average price vector difference* (avgPVd, plotted in red line):

$$\tilde{\Delta}_{\mathbf{p}}(t) = E^*_{\beta}[\Delta_{\mathbf{p}}](t) \tag{5.9}$$

Finally the algorithm detects a pseudo-convergence in price vector when $\tilde{\Delta}_{\mathbf{p}}(t)$ stabilizes, i.e. its "moving standard deviation" is less than a certain percent of its moving average.

$$\frac{Std^*_{\gamma}[\tilde{\Delta}_{\mathbf{p}}](t)}{E^*_{\gamma}[\tilde{\Delta}_{\mathbf{p}}](t)} \leq \epsilon \tag{5.10}$$

where $Std*_{\gamma}$ is the traditional standard deviation with the expectation operators replaced by E^*_{γ}:

$$Std^*_{\gamma}[x](t) = \sqrt{E^*_{\gamma}[x^2](t) - (E^*_{\gamma}[x](t))^2} \tag{5.11}$$

Similar quantities are defined for the demand vector $\mathbf{x}(t)$ and the algorithm determines that the market reaches a pseudo-convergence when condition 5.10 and its counterpart for demand vector are both met.

The weights $(\alpha, \beta, \gamma, \epsilon)$ in the range (0,1) are the parameters of the detection algorithm and determine how sensitive it is to fluctuations and final stabilization. In general, the smaller they are, the long it takes for the algorithm to detect convergence.[18] They are chosen empirically to be (0.90, 0.95, 0.95, 0.05) in the final implementation, which is able to detect pseudo-convergence in all test cases we experiment on. Although the three rounds of EWA smoothing filters seem excessive, they are actually useful necessary due to the high fluctuation of the price and demand vectors. With these parameters, the algorithm detects pseudo-convergence in the example market at iteration 1102 (with $\epsilon = 0.03$ it detects convergence at iteration 3075).

In figure 5.8 we plotted the three quantities $\Delta_{\mathbf{p}}(t)$, $\tilde{\Delta}_{\mathbf{p}}(t)$ and $Std^*_{\gamma}\tilde{\Delta}_{\mathbf{p}}(t)$.[19] The corresponding versions of the demand vector are shown in figure 5.9. Notice the contrasting behavior of the price and demand vector difference at around iteration 1000: while the

[18]Some choices may not lead to detection of convergence at all because the criterion is too strict.

[19]Note that the initial high value of the blue and green line is due to the initialization of the moving average to some high value and is insignificant.

price vector difference drops significantly and stabilizes at a lower level, the demand vector difference rises and stabilizes at a higher level. These behaviors correspond to a phase shift from the all-price-increasing stage of the market to a dynamic-equilibrium stage of the market equilibrium prices (5.3). In the first stage, prices increase more or less uniformly, hence the price vector differences are at a higher and less volatile level, while the demand vector difference only spikes occasionally when prices reach some critical boundary. When the market falls into a dynamic equilibrium, the price vector fluctuates around a center of equilibrium and the price vector difference appear to be at lower level but more volatile. Because the responses of the demand vector around the efficient prices are extremely discontinuous, this leads to a higher level of demand vector difference in the equilibrium stage. Note that had we simply sought the first order convergence condition ($\Delta_{\mathbf{p}}(t) \to 0$ and $\Delta_{\mathbf{p}}(t) \to 0$), we would never find convergence in the market. Also had we used a version of condition 5.10 with *absolute* rather than *relative* magnitude, we would have discovered "convergence" prematurely at around iteration 900, at a single meta-stable state as shown in 5.5.

When MARKET algorithm detects pseudo-convergence, it simply outputs the last market state to the next stage CSMA scheduler. Because of the randomness in the market state, the distribution of the last market state is then roughly equal to the empirical distribution of component states in the dynamic equilibrium. It may be interesting to see if there is any consistent correlation between the choice of which state to output among the equilibrium states and the final utility metric. It may be the case that the lower demand states will lead to less congestion in the CSMA scheduler and thus a better outcome, or the opposite because the optimism in the higher demand states actually helps in CSMA. Due to time constraints, we did not investigate this problem fully, but simply report that in our example market Equilibrium(-) demand seems to perform better than Equilibrium(+) demand in terms of final achieved total utility after the CSMA stage (140 vs 130.5, or 1.08x).

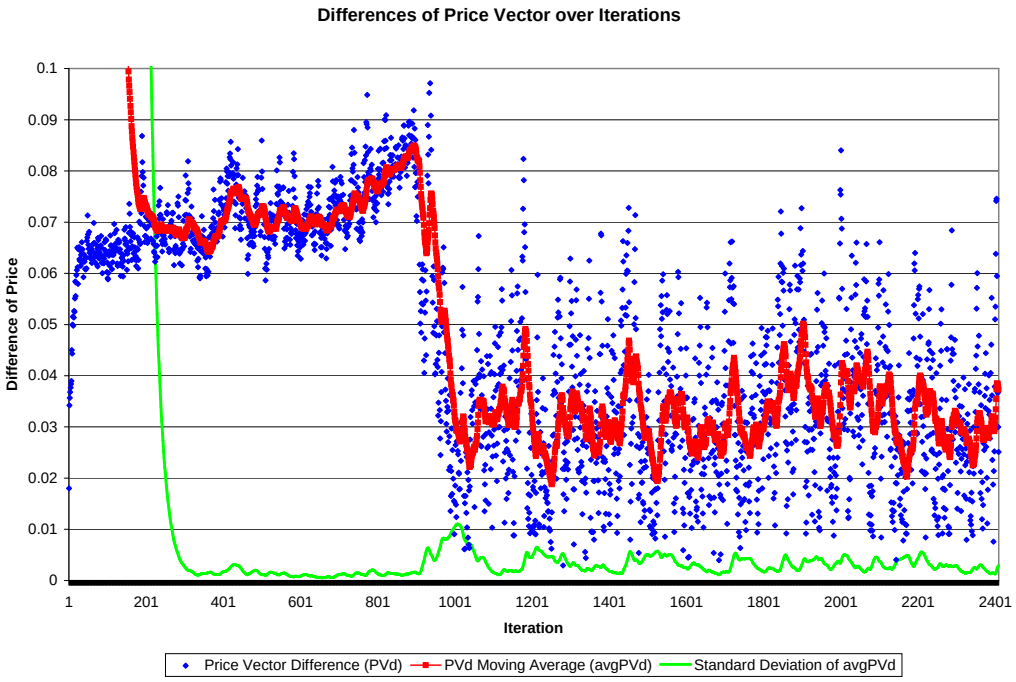


Figure 5.8: **Difference of Price Vector over Iterations**.

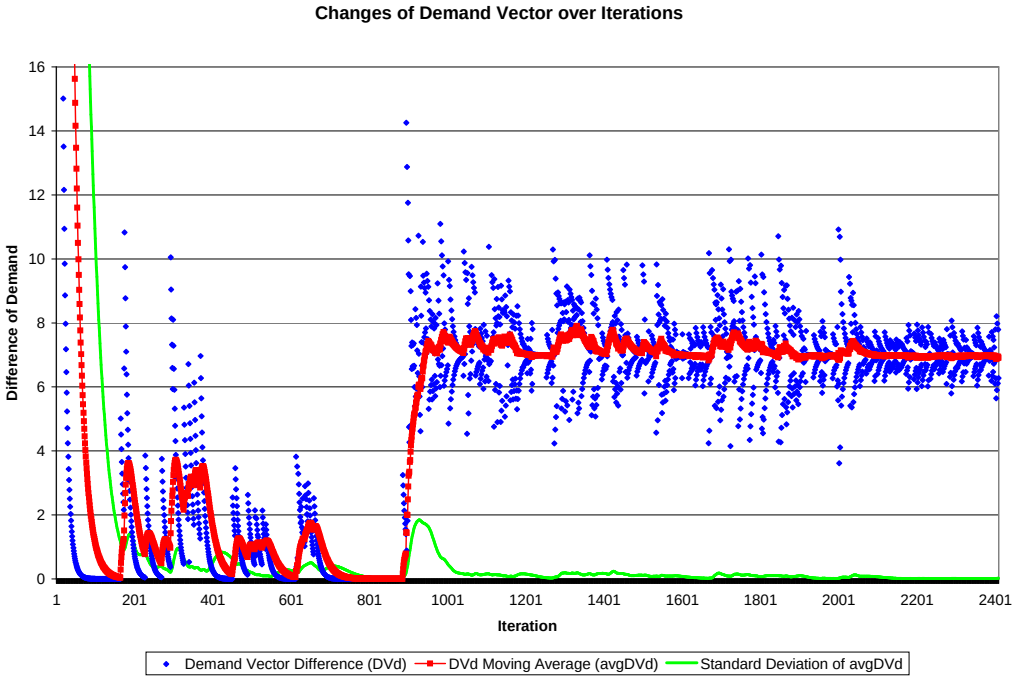


Figure 5.9: **Difference of Demand Vector over Iterations**.

5.5 Price Distribution

In this section, we investigate the distribution of prices for a given good over a varying demand profile. Again, we fix the network topology, interference model, the choice of goods, the number of simultaneous applications and application utility function (see 5.2) but sample over random source and destination nodes of the application. We have added additional 10 odd-hole goods (H17-H26), chosen probabilistically as described in section 4.2, to the set of goods. We sample over 1000 test cases (each with 20 application with random source and destination), and report the price distribution for each good, shown in figure 5.10. The list of 1000 prices for each good is sorted in increasing order and plotted with (percentile,price) pairs. Note that two points on two series with the same x coordinate do not have to be from the same test instance.

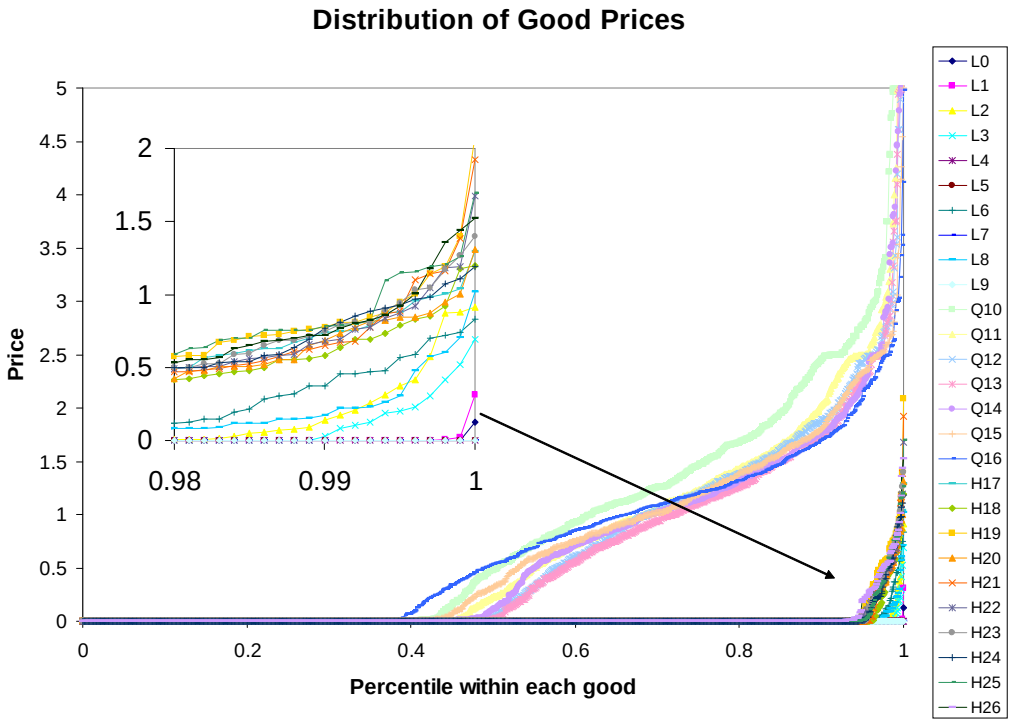


Figure 5.10: **Distribution of price for each good, sampled over random applications**.

It seems that only clique goods are priced consistently at non-negligible prices. The amplification of the lower right corner shows that holes are price slightly higher than links

though both of them are significantly less likely to be priced at a non-zero level compared to clique goods (holes are priced only 10% of the time, and links are priced only 5% of the time). One lesson we can draw from this is that cliques are the important goods that correspond to limited resources due to heavy interference in MARKET algorithm.

5.6 Choice of Goods Basis

Since it is infeasible to enumerate all cliques and all odd-holes in general, we will choose some set of cliques and odd-holes to form the goods basis at the beginning of the protocol. The generation process is probabilistic (see section 4.2) and we will now investigate the effect of goods basis choices on the convergence time and the output quality of the algorithm.

In the following experiment, we compare a class of MARKET algorithms parameterized by the number of cliques (C) and the number of holes (H) in the goods basis. For each particular parameter (C,H), we run the algorithm through 20 test cases and measure the average of the *normalized* final achieved utility and the average number of iterations to convergence. The normalized final achieved utility is the ratio of the final achieved utility of the MARKET algorithm to that of the ORL-CSMA algorithm on the same input. Results on example network in figure 6.1(a) are presented in figure 5.11. Some important observations are:

1. ($U < 1$) The normalized performance of MARKET is always than 1, which means MARKET performs worse than ORL-CSMA on average.

2. ($\Delta U/\Delta C > 0$ at $H = 0$) When no holes are present, the effect of adding cliques on output quality is positive and significant (compare different lines at H=0).

3. ($\Delta U/\Delta H > 0$ at $C = 0$) When no cliques are present, the effect of adding holes is positive but moderate (compare points on C=0 line).

4. ($\Delta U/\Delta H < 0$ when C is big) However, when all cliques are added (C=7), adding more holes seems to have a *negative* effect on the output quality, which is quite surprising.

5. The number of iterations to convergence does not grow significantly as the number of goods grows. This is consistent with the observation of "non-catastrophic scaling of equilibration process with the number of goods" by Wellman [26].

6. ($\Delta t/\Delta H < 0$) In the example network, increasing the number of holes actually *decreases* the number of iterations rather than increasing them.

7. ($\Delta t/\Delta C \approx 0$) The number of cliques has no apparent effect on the number of iterations to convergence.

The validity of some of these observations may depend on the specifics to this particular network topology and interference model. As a robustness test, we perform the same experiment on a different network topology[20] and level-2 interference model. Results are shown in figure 5.12. In this second experiment, observations 1, 2 and 5 are reinforced; observations 3, 4 and 6 are present but weaker; observation 7 is not true. Instead, adding cliques increases the number of iterations significantly, which is what we would expect.

From these two example, we draw the following lessons to guide the choice of the goods basis in our next experimental section:

- Adding cliques has a significantly positive impact on output quality, though the increase in convergent time is acceptable. Thus we will add as many cliques as possible to the market.

- Adding holes has a moderate and ambivalent effect on output quality and small effect on number of iterations. Thus we will simply not add any holes.

Although these lessens may be over-generalizing, they are the only available guidelines for fine-tuning the performance of MARKET algorithm in our experimental settings. Further investigation into the effect of goods basis may be carried out in future work.

[20] A new topology is generated by means of section 6.1 with longer link range (l=0.4).

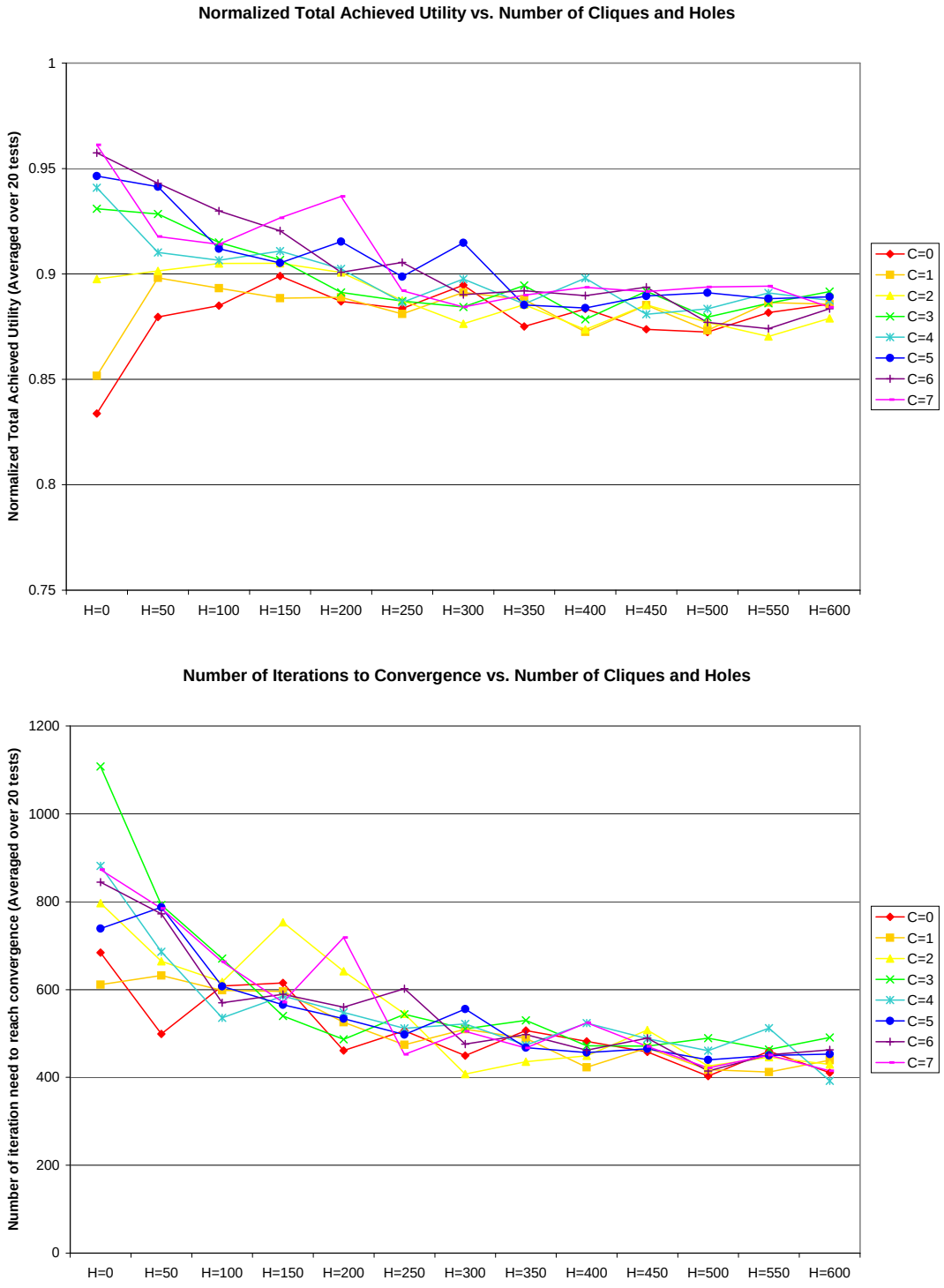


Figure 5.11: **Effect of Price Vector Choices on Output Quality and Convergence Time (1)**. *Top graph shows the normalized total achieved utility, which is the ratio of total final achieved utility of* MARKET *to that of* ORL-CSMA *on the same input. Each point in both graphs corresponds to an average over 20 test cases, each of which consists 10 applications with homogeneous utility and random sources and destinations. Bottom graph shows the average number required to reach convergence.*

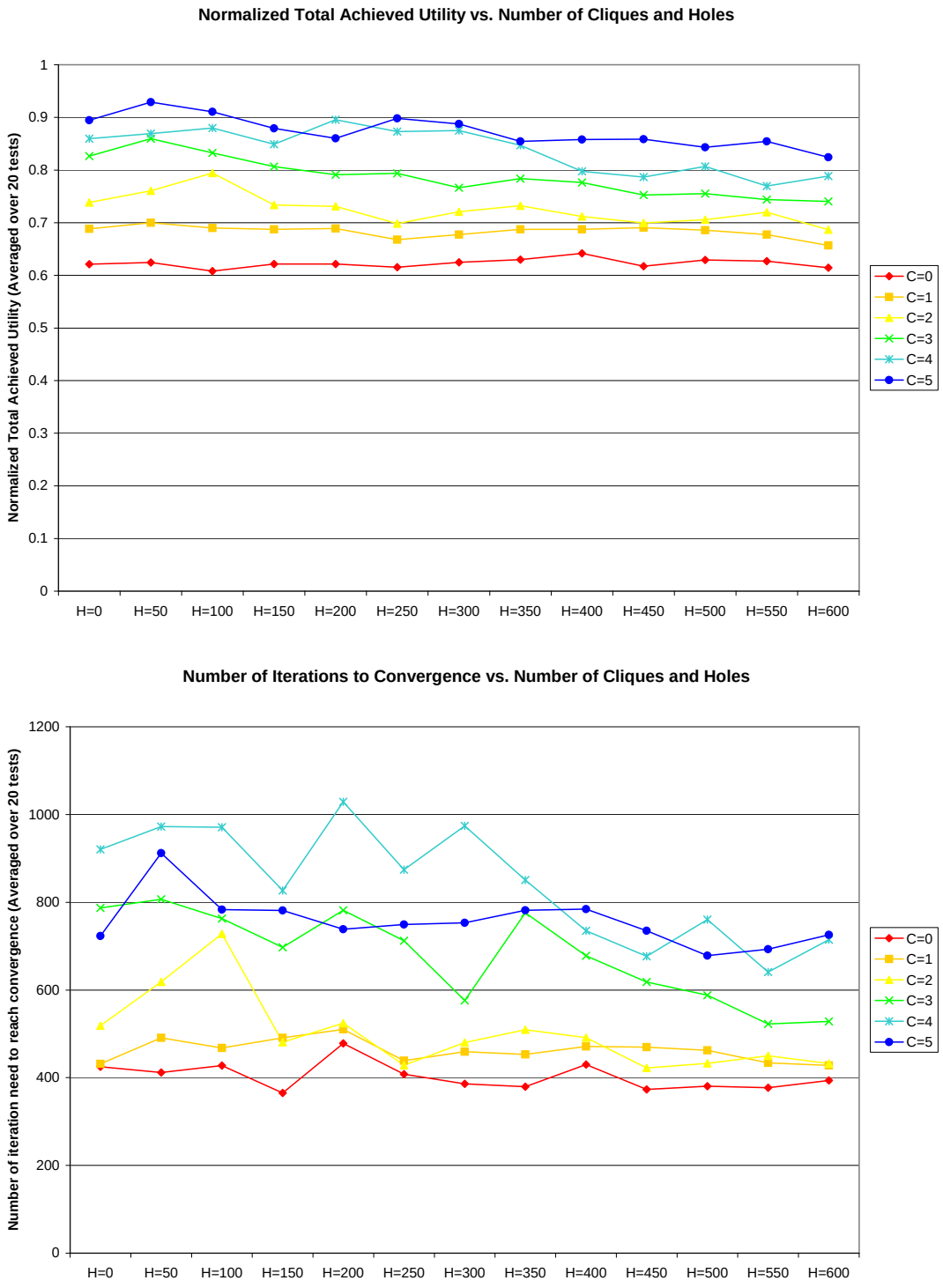


Figure 5.12: **Effect of Price Vector Choices on Output Quality and Convergence Time (2)**. *Same experiment as Figure (5.11) but with different network topology and interference model.*

5.7 Summary

We conclude this chapter by highlighting the main contributions and limitations of this chapter:

- We formulated and implemented a computational market algorithm using a tatonnement process with simple incremental price adjustments and greedy application demand functions.
- The usage of virtual goods (cliques and holes) to capture externality due to interference constraints in the network is an invention of this thesis compared to previous market-oriented approaches to the various resource allocation problems in [9, 27, 29].
- The convergence property of the market presents a challenge due to complementarity of goods and discrete prices and demand space. A simple yet effective algorithm using exponential moving averages is presented to detect pseudo-convergence.
- MARKET algorithm is able to adapt to network topology and traffic profile to effectively price bottleneck goods (cliques in particular) that correspond to congestions in the network, while ignoring non-limiting resources such as links.
- Empirical study of the effect of goods choices on output quality and convergence time suggests that including more cliques and less holes in the set of goods tends to lead to better results.
- MARKET algorithm with CSMA scheduler tends to approximate the result of ORL-CSMA at about 90% with further reduced complexity. It has the potential of becoming a real-time online scheduling algorithm. Detailed comparison of the performance metrics will be presented in the next chapter.
- Studies presented in this chapter are primarily based on a limited number of problem setups. Robustness of the results need to be further tested in future work.

6

Evaluation and Analysis

6.1 Demand Saturation - A Case Study

To characterize the performance of the four algorithms (OPT, ORL-CSMA, MARKET and NAIVE-CSMA) as demand increases, we perform a case study on a specific instance of a randomly generated network shown in Figure 6.1. We assume links have identical capacity 10 which is also equal to the number of time periods (so that numerically the bandwidth equals the number of time slots scheduled). We generate a list of homogeneous applications with random sources and destinations and our typical (0,0)-(1,10)-(2,15) utility function in Figure 2.2(a). We include them, in order, to form a sequence of test cases to be scheduled by each algorithm.[1] To ensure comparability, we impose the constraints that applications cannot operate at a bandwidth higher than the maximum bandwidth specified in the utility function.[2] We have chosen to fix the network and the list of applications in this experiment to ensure comparability across the application axis – we will allow their randomization in the next section.

In our initial experiment with the more appealing l=0.4 network and level-1 interference, it is demonstrated that the amount of computation required by OPT is extremely prohibitive. We are only able to collect data up to 20 applications with a time limit of 1 hour. Hence, in order to gather enough data points over a wider range of demand level,

[1] So that the 20 applications in test instance #20 are the same as the first 20 applications in test instance #100. They can be interpreted as arriving one after another over time.

[2] Otherwise NAIVE-CSMA may appear to devote all available bandwidth (up to 10) to an application that does not gain any additional utility beyond a bandwidth level 2. This may cause confusion in the bandwidth comparison graph.

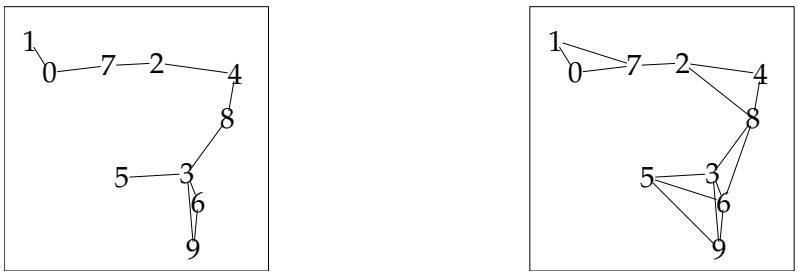


Figure 6.1: **A randomly generated network to carry out case study**. *10 nodes are placed uniformly at random in the unit square, with symmetric directional links connecting every pair of nodes within distance l. (l=0.3 for left and 0.4 for right.)*

we run the same experiment on the simpler l=0.3 network with level-0 interference. In Figures 6.2 to 6.4 we report the running time, average link usage,[3] total and average bandwidth and utility on all test runs. In addition, we also show the first stage upper-bound solution obtained by ORL-CSMA as an absolute upper-bound on the maximum extractable utility.[4] Since data points are taken more sparsely as the number of applications increases, the x-axis has been compressed by a factor of 4 in the right half of each graph for better visual presentation.[5] Some important observation from these results are:

- **All three algorithms outperform NAIVE-CSMA significantly in both bandwidth and utility.** All three algorithms that we develop seem to outperform NAIVE-CSMA by a significant margin in both bandwidth and utility level. Furthermore, the gap widens as the number of applications increases (from about 30% improvement at 10 applications to almost 100% improvement in 100 applications). After 20 applications NAIVE-CSMA is almost flat, while others continue to increase marginally. This shows that NAIVE-CSMA suffers from overwhelming contention in high demand and has degrading performance.[6] ORL-CSMA does not suffer from the same problem due to the effectiveness of the upper-bound rate-limit computed in the first stage.

[3]The link usage for each period is defined as the number of active links divided by half of the total number of (directed) links. A hypothetical link usage of 1.0 means one link out of each reverse link-pair is active. In practice this ratio will be much lower due to interference.

[4]Note that the bandwidth line of csma-ub is not necessarily a strict upper-bound because the first stage ORL-CSMA optimizes for utility rather than bandwidth.

[5]Note that lines in the right half of each graph appears to be 4x as steep.

[6]In practice this degradation will be even worse due to collision, which is not modelled in our simulator.

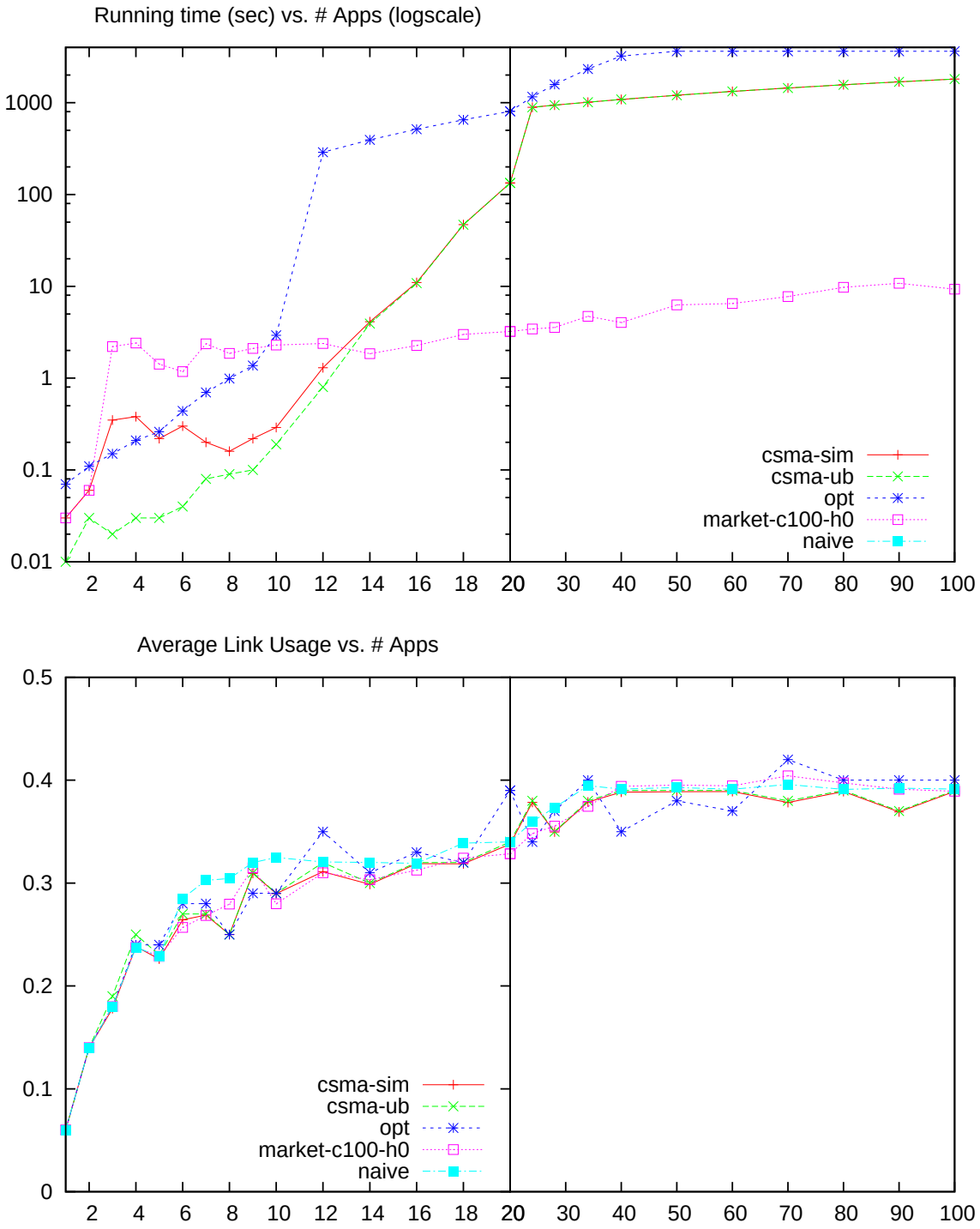


Figure 6.2: **Running Time and Link Usage of OPT, ORL-CSMA, MARKET and NAIVE-CSMA**. *Note that the right half of each graph is compressed by a factor of 4 along the x-axis.*

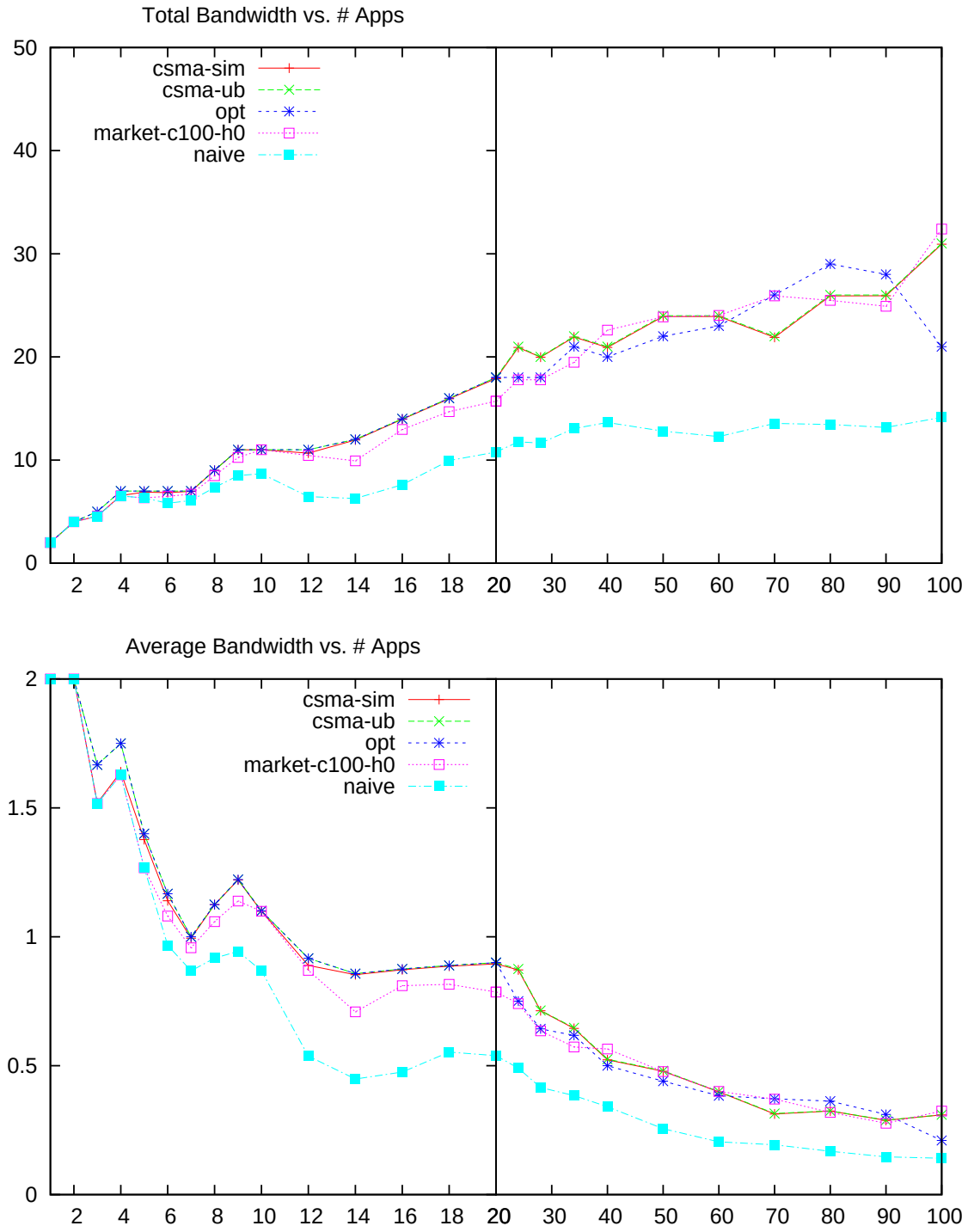


Figure 6.3: **Total and Average Bandwidth of OPT, ORL-CSMA, MARKET and NAIVE-CSMA**. *Note that the right half of each graph is compressed by a factor of 4 along the x-axis.*

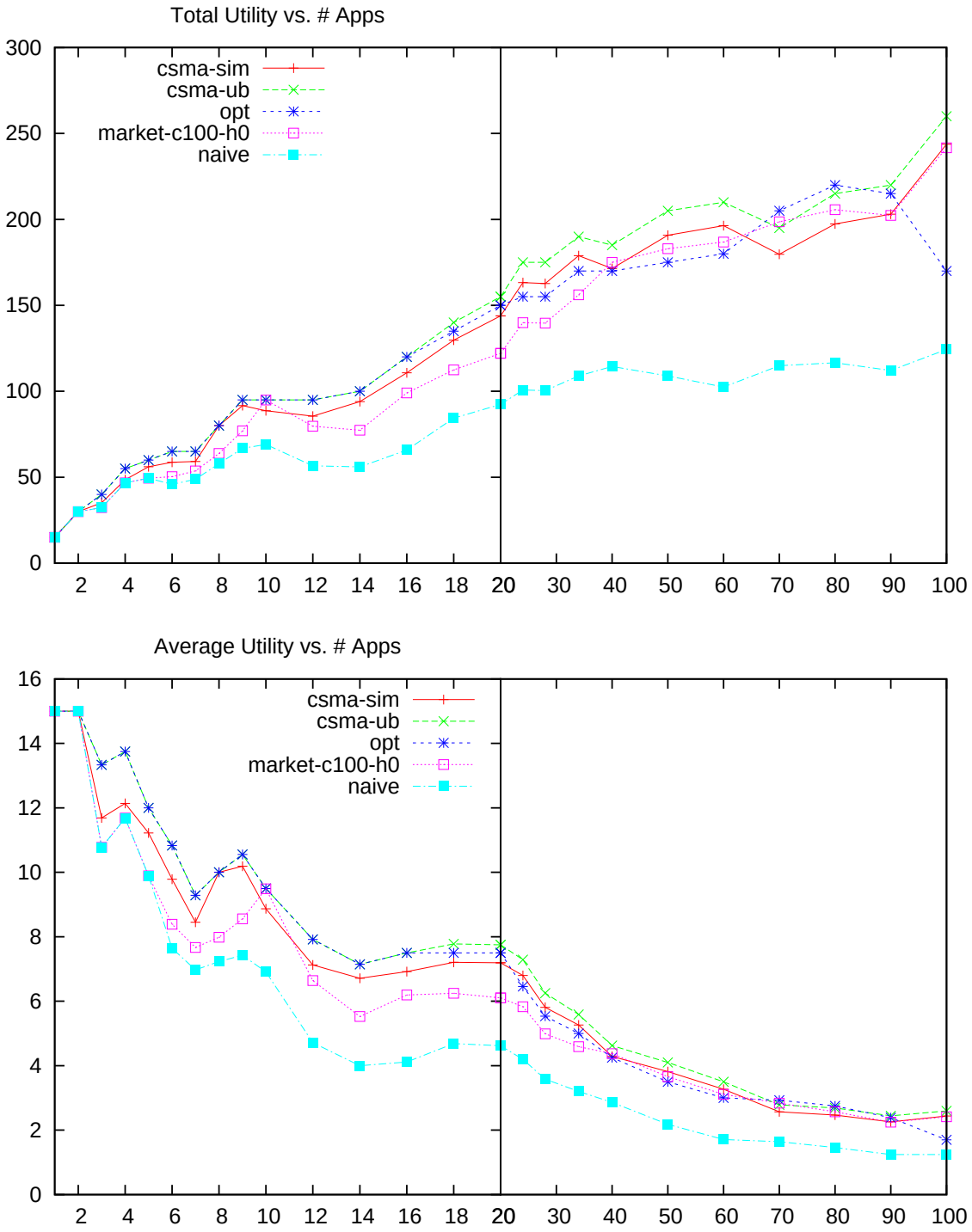


Figure 6.4: **Total and Average Utility of OPT, ORL-CSMA, MARKET and NAIVE-CSMA.** *Note that the right half of each graph is compressed by a factor of 4 along the x-axis.*

- **Run-time complexity OPT≫ORL-CSMA≫MARKET.** The running time of OPT grows worse than exponentially in the number of applications (note the time axis is in log-scale). ORL-CSMA grows approximately exponentially in the number of applications and also hits the upper time limit after 20 applications. The running time of MARKET does not seem to grow much as the number of application increases. There is an initial jump from 2 to 3 applications of MARKET because prior to 3 applications, the demand is so low even at all zero prices that the market terminates immediately. The "running time" of NAIVE-CSMA is not plotted because it does not require prior computation.[7] The regularity of OPT line after 12 applications and ORL-CSMA line after 20 applications is due to a manually-scaled timeout limit. Thus, OPT and ORL-CSMA are computationally constrained after that point and may not find the optimum solution in their linear programs.

- **Link Usage very similar, with NAIVE-CSMA slightly worse.** The percentage of active links is very similar in all four algorithms, with NAIVE-CSMA appearing to be the worst since it tends to waste a lot of bandwidth in delivering messages that will get dropped later. Note that the average link usage converges to about 0.4. In fact the absolute upper-bound on link usage is 0.5 because, as shown in figure 6.1, at most 5 links can be active simultaneously under the level-0 interference model (e.g. 0−>1, 7−>2, 8−>4, 3−>5, 6−>9). Consequently, an absolute upper-bound would be 10*10*0.5=50 for bandwidth and 500 for the utility in the best case scenario where there are a sufficient number of non-interfering one-hop demands.

- **Bandwidth OPT≈ORL-CSMA≈MARKET.** Although the three algorithms are not geared towards bandwidth maximization, it makes sense that bandwidth must be roughly maximized in order to achieve maximum utility. The Bandwidth characteristics of the three algorithms are virtually indistinguishable, with MARKET slightly lower than OPT and ORL-CSMA *before they hit computational limit*.

- **Utility OPT>ORL-CSMA>MARKET but close.** Despite similar usage of bandwidth

[7] Although the running time of the simulator grows roughly linearly because the number of inbound packages in the simulator queue scales linearly with the number of applications.

resources, the utility levels show real differences in performance. In low demand when computational constraint is not yet harsh for OPT, ORL-CSMA tracks OPT very closely (about 95%) while MARKET only tracks about 80%-90% of OPT's utility.

- **Computationally constrained OPT under-performs ORL-CSMA in high demand. Similarly for ORL-CSMA and MARKET.** After OPT hits the computational bound, ORL-CSMA begins to outperform OPT with demand higher than 20 applications. Similarly after ORL-CSMA hits the computational bound in extreme high demand (>70 apps) MARKET catches up. Because the time limit we set is extremely generous (30 minutes to 1 hour), we expect that MARKET would be the most scalable algorithm in practical settings.

- **ORL-CSMA upper-bound is not strict.** As shown in [13], the first stage ORL-CSMA upper-bound need not converge to the optimal solution even when all clique and hole constraints are added. We can see in the utility graph that the ORL-CSMA upper-bound (csma-ub) coincides with the optimal solution in low demands until 18 applications.[8] After that the upper-bound and the OPT solution diverge. At 70 and 80 applications the csma-ub line is actually lower than OPT line because both of them are computationally constrained and thus the order of the two is unpredictable.[9]

[8] Although it may also be the case that OPT would have found the optimal solution given sufficient time.

[9] In CPLEX the first stage ORL-CSMA returns a feasible solution when it runs out of time. We should probably have let it return the bound from the dual optimization problem.

6.2 Utility Optimization vs. Bandwidth Optimization

In order to highlight the difference between traditional throughput optimization and our utility optimization algorithms, we will compare OPT, ORL-CSMA, and their bandwidth-optimizing counterparts OPT-B and ORL-CSMA-B. The bandwidth-maximizing version of each algorithm can be trivially obtained by supplying the algorithm with an identity function as the utility function for all applications.

In this experiment, we randomize the network as well as the applications. In each test case, a network is generated in the fashion of previous sections with a random number of nodes (5-15) and a nominal range l chosen from $(0.3, 0.4)$. A random number (5-20) of applications with random sources, destinations and randomly generated utility functions will enter. A utility function is generated by first choosing the maximum bandwidth level m, then choosing m random utility values from {0,5,10,15,20} and finally assigning them to each bandwidth level sorted in ascending order.[10] We fix the number of rounds at 10 and use the level-1 interference model. Each test case will run for at most 10 minutes by each algorithm.

In Figure 6.5, the first two graphs show that on average OPT extracts about 1.083x of the utility of OPT-B although only consuming 0.89x of the bandwidth. This means OPT utilizes bandwidth resources more efficiently to achieve applications' utility. The third graph shows that ORL-CSMA approximates about 95.3% of OPT's optimal utility (when both of them finish in time). The data points in the opt-vs-csma graph exhibit interesting gathering in two groups: the majority lie around and below the diagonal, while a couple spread across the csma-axis at low value of opt. This is perhaps due to the fact that OPT tends to return a trivial and poor feasible solution when it runs out of time. ORL-CSMA-B and OPT-B are statistically indistinguishable in terms of their achieved bandwidth objective, which is an indication that utility optimization is a more difficult task than bandwidth optimization.

[10] This ensures that the utility function is non-decreasing. Concavity is not guaranteed (e.g. if the values chosen are 0,5,20).

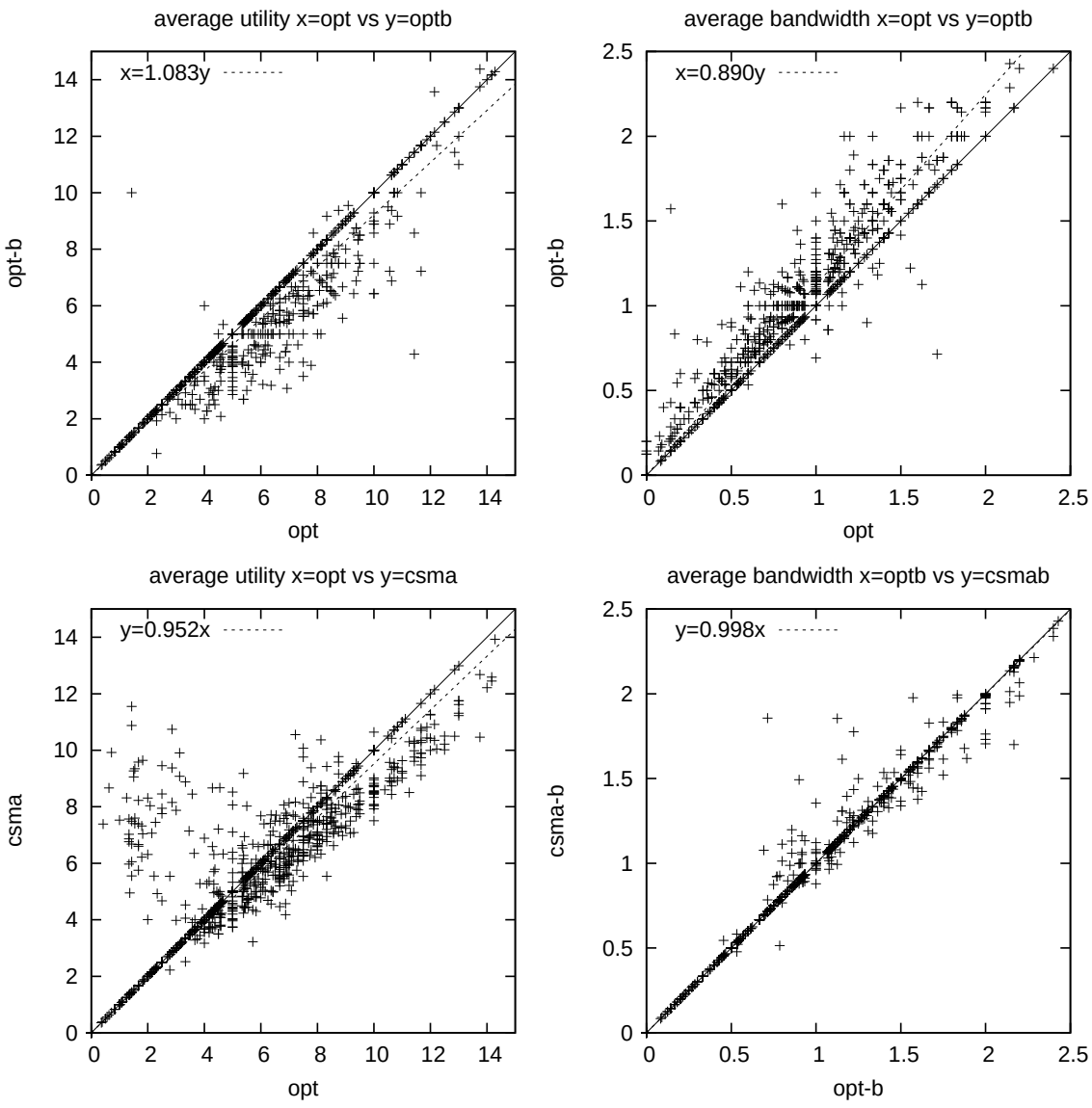


Figure 6.5: **Performance comparison in random sample.** *Each point represents a unique test instance to which both algorithms in question provide a solution within 600 seconds.*

6.3 Performance Comparisons of OPT, ORL-CSMA, MARKET and NAIVE-CSMA

Finally, we extend the experiment in the previous section to compare all four algorithms by looking at several performance metrics. In addition to running time, average bandwidth, average utility and link usage, we also look at additional fairness metrics using the Jain's Fairness Index [14] on bandwidth and utility vectors. The Jain's Fairness Index for a vector

$\mathbf{x}$ with dimension n is defined as:[11]

$$fairness(\mathbf{x}) = \frac{(\sum x_i)^2}{n \cdot \sum x_i^2} \tag{6.1}$$

We run MARKET and NAIVE-CSMA on the same test instances run by OPT and ORL-CSMA in the previous section, and report the pairwise comparison of each performance metric in figure 6.6 and 6.7.

Using NAIVE-CSMA as a baseline, we see that OPT, ORL-CSMA and MARKET achieves 1.20x, 1.22x and 1.13x the utility while only using 0.86x, 0.95x and 0.89x of the bandwidth, respectively.[12] In terms of bandwidth and utility fairness, OPT has much poorer fairness property compared to NAIVE-CSMA (as the fairness plots for opt-vs-naive mostly lie below the x-y line). Relatively speaking, ORL-CSMA and MARKET has slightly better bandwidth fairness and utility fairness than OPT. This is probably due to the fact that OPT computes and fixes a TDMA schedule once and for all, while the CSMA-brand algorithms have the second stage best-effort scheduler that contribute to a mixing and smoothing of bandwidth (and utility) across applications. In terms of utility fairness, ORL-CSMA and MARKET does not lose much compared to NAIVE-CSMA.

In the scatter plots between each pair of OPT, ORL-CSMA and MARKET, we observe the following: In most test cases OPT is better than ORL-CSMA and MARKET in terms of utility. However, the computational problem of OPT manifests itself as occasionally OPT achieves extremely low bandwidth and utility (see several outlier points in the utility opt-vs-orl-csma and opt-vs-market that have very low opt-axis values). As a result, on average OPT ties with ORL-CSMA in average utility metric and is only 5% better than MARKET in our sample average. On the contrary, ORL-CSMA and MARKET have more consistent performance (since points the csma-vs-market are less scattered). In terms of bandwidth and utility fairness, we can see clearly that the CSMA-brand algorithms achieve better fairness index than OPT.

[11] A value closer to 1 mean more fairness.

[12] Note that less bandwidth is viewed as a positive here because the objective is to maximize utility. If an algorithm uses less bandwidth to achieve the same or more utility of another algorithm, then it uses bandwidth resources more efficiently.

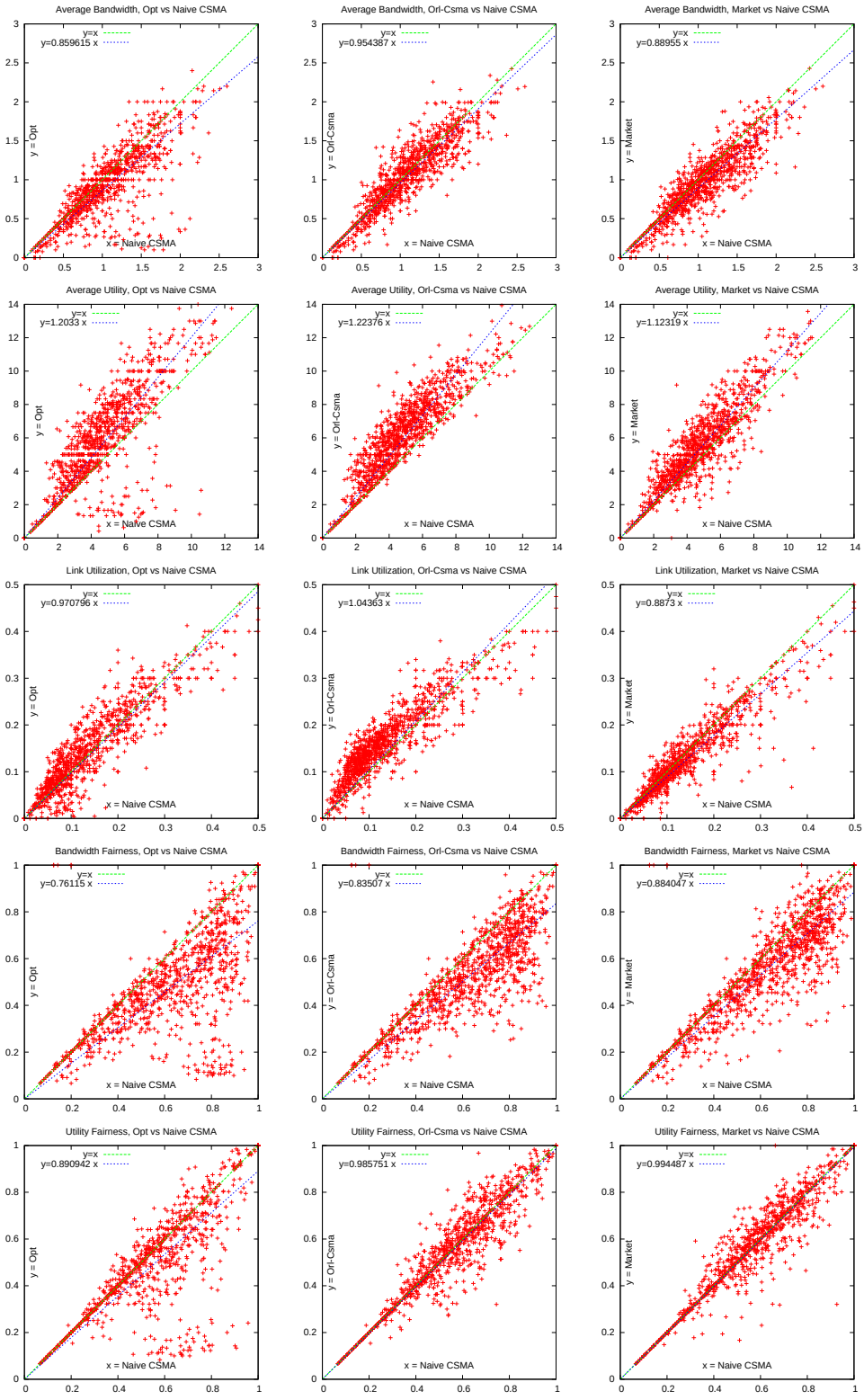


Figure 6.6: **Comparison of OPT, ORL-CSMA, MARKET vs. NAIVE-CSMA**. *The performance metrics include: average bandwidth, average utility, link usage, Jain's fairness index on bandwidth vector and utility vector.*

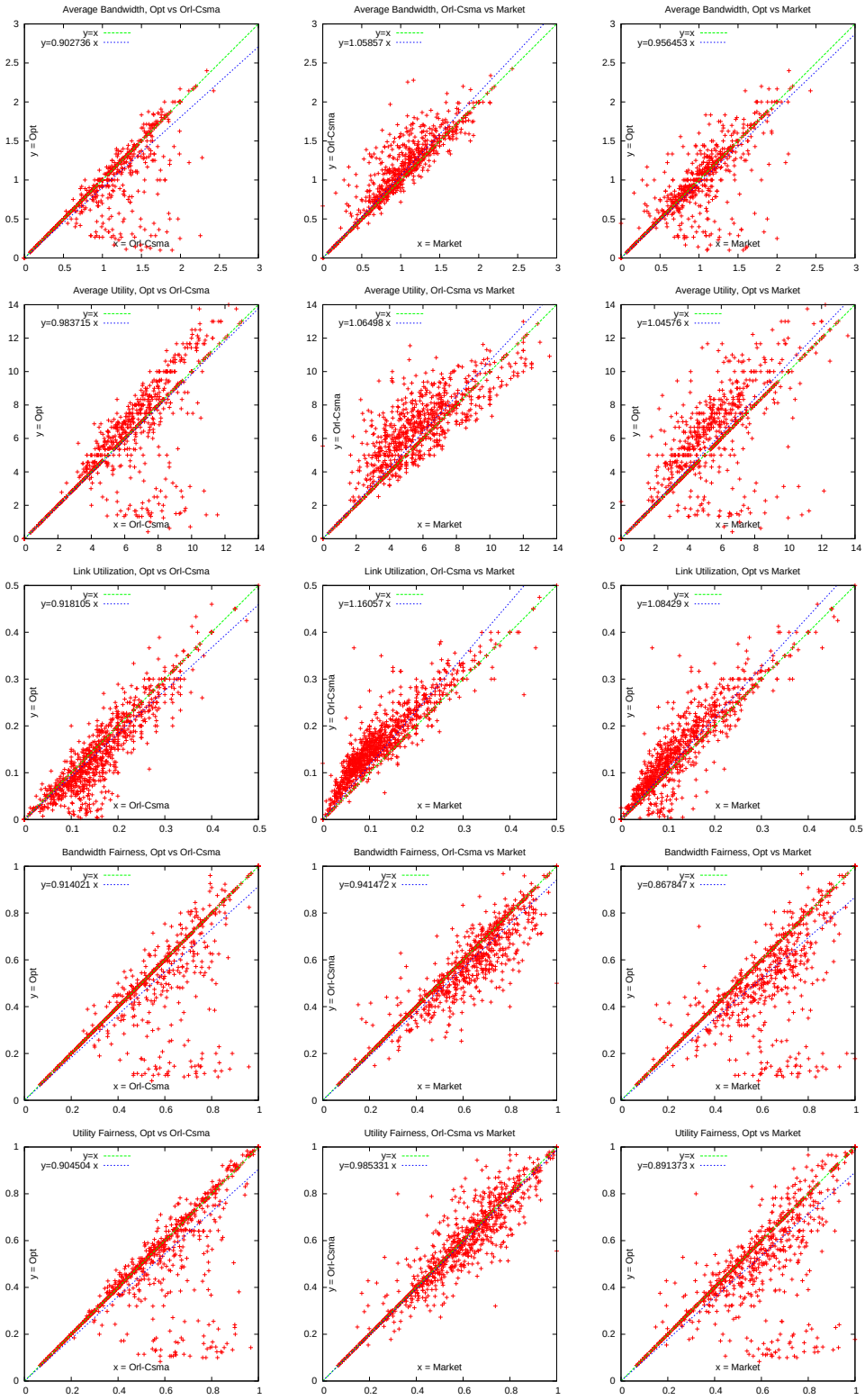
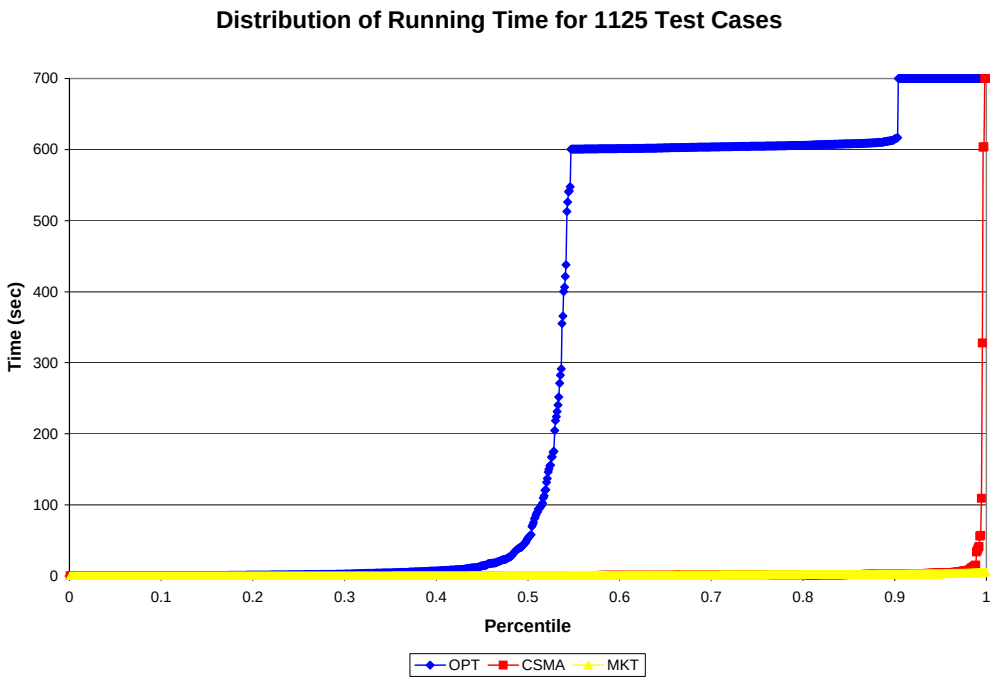


Figure 6.7: **Comparison between OPT, ORL-CSMA and MARKET**. *The performance metrics include: average bandwidth, average utility, link usage, Jain's fairness index on bandwidth vector and utility vector.*

Figure 6.8 shows the distribution of running time across these 1125 test instances for all three algorithms. Note that the timeout limit on the optimization steps for OPT and ORL-CSMA is 10 minutes. A value of 700 seconds denotes time-out. Out of the total 1125 test cases, OPT finishes optimization in 580 (51.5%) cases, runs out of time and returns a best feasible solution in 437 (38.8%) cases and fails to even find a feasible schedule for the rest 108 cases (9.6%), whereas ORL-CSMA finishes all in time. MARKET runs for under 5 seconds for all of them. Thus we can see the complexity reduction from OPT to ORL-CSMA is substantial, and even further in MARKET. Considering that the size of problems in our random sample is not too large, and that our time limit of 10 minutes is still quite generous for scheduling around 10-20 applications, we shall recommend using ORL-CSMA or MARKET in a practical setting with more stringent time constraints. In particular, if the time limits were set at a more realistic value of 5 seconds under conditions of online real-time scheduling, MARKET would be the recommended implementation to offer a good balance between output quality and running time.



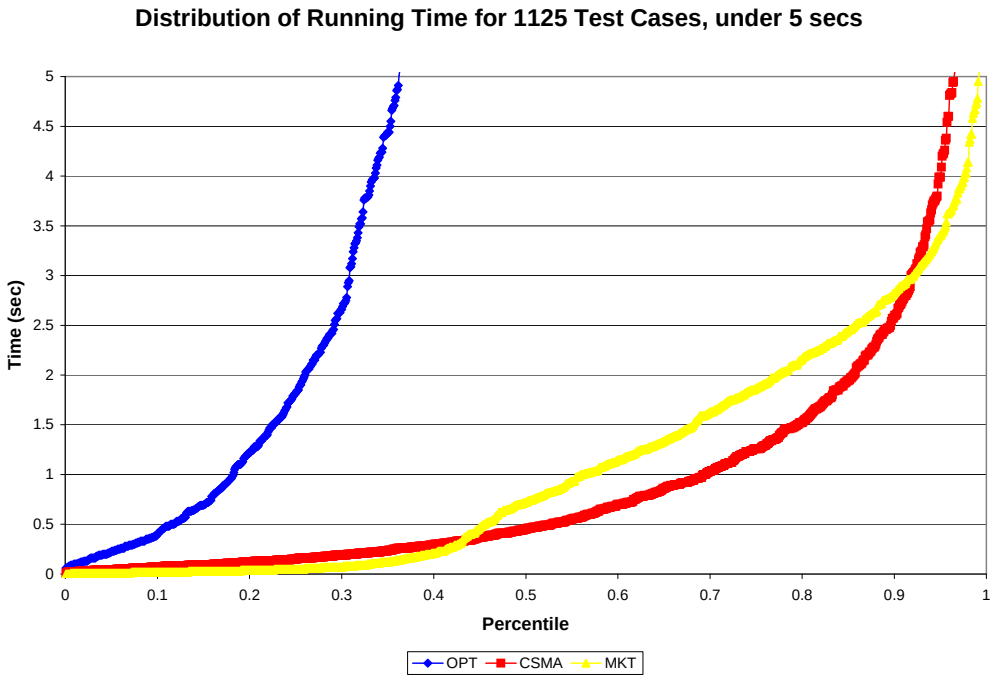


Figure 6.8: **Distribution of running time for OPT, ORL-CSMA and MARKET in random samples**. *The time limit for the optimization step is 600 seconds. A value of 700 in the top graph denotes a "timeout". The lower graph shows the portion of the top graph under 5 seconds.*

6.4 Summary

The main results from our experimental studies can be summarized as:

- All three OPT, ORL-CSMA and MARKET algorithms performs significantly better than NAIVE-CSMA(by 20% in normal demand and up to 100% in high demand).
- When computational constraints are not harsh, OPT extracts the maximum utility. ORL-CSMA approximates about 95% of the optimal utility. MARKET approximates about 80%-90%.
- In our random samples of problem instances, the utility performance of the three is roughly OPT: ORL-CSMA: MARKET= 1.00 : 1.02 : 0.96. OPT is lower than ORL-CSMA mostly due to returning very poor solutions under computational constraints on large problems.
- Runtime complexity of OPT grows catastrophically as demand increases and shall not be used in practice.
- In low to medium demand, ORL-CSMA offers a good balance of running time and approximation quality. However, complexity of ORL-CSMA also grows rapidly as demand further increases to high range.
- Although with a slightly worse approximation than ORL-CSMA, MARKET has much better scalability and is the best choice under high demand or stringent time constraints.

7

Conclusions and Future Work

We have presented a research project attempting to address the open problem of optimizing aggregate user utility in wireless ad-hoc networks under the constraints of wireless interference. We identified motivations for utility-based scheduling in several key usage scenarios, formulated a general model for the Optimal Utility Scheduling problem, then developed a brute-force OPT optimizer and a hybrid ORL-CSMA scheduling algorithm guided by the relaxed upper-bound solution. Moreover we developed a market-oriented approach MARKET with a tatonnement process and demonstrated its ability to effectively price bottleneck resource in the network and thereby approximate the optimal solution with substantially reduced complexity. We met and solved several interesting challenges in our market setup, including complementarity of goods, discontinuous demand functions, complex externality structures and pseudo-convergence. Our experimental results have shown that the computational intractability of the complete solver OPT renders it practically unusable; ORL-CSMA offers a compromise between reduced complexity and a very good approximation (about 95%) of the optimal solution. However, ORL-CSMA still requires solving an integer linear program in a centralized fashion, whereas MARKET provides further relaxation and approximation (at about 90% level of the optimal). MARKET's better scalability and possible decentralization makes it more appropriate for realistic online realtime scheduling settings. We conclude by sketching out an outline for further research directions:

- **Towards a decentralized online realtime scheduling Protocol:** It is conceivable that MARKET allocation coupled with CSMA scheduler could become an online realtime

scheduling protocol. The demand response in the market could be continuously fed to the second stage CSMA scheduler without waiting to settle down on a particular stable state (which may not exist). Instead the dynamic equilibrium of the market will translate to dynamic outcomes in the CSMA schedule. Moreover, our market protocol is already decentralized on the agent level and the auctioneer level. A distributed and decentralized implementation can be achieved by delegating each auctioneer onto some individual nodes, and propagating the price information and bid information by piggy-backing. The exact details of the implementation has to be figured out in future research.

- **Further testing of empirical results on a wider range of setups:** Most studies in this thesis are performed in a limited number of problem setups. Further robustness of the empirical results need to established by testing on a wider range of different network topology, interference model and demand profiles.

- **Practical implementation and deployment on real wireless networks:** Due to space and time constraints of this thesis, we did not have the opportunity to implement and deploy our algorithms on real-life wireless network test beds such as the MoteLab [17]. Further complication may arise due to realistic wireless behaviors such as collision and partial interference that's not captured by our model or simulator. When time allows, we shall implement our algorithms in TinyOS and measure relevant performance metrics in real-life experiments.

References

[1] Daniel Aguayo, John Bicket, Sanjit Biswas, Glenn Judd, and Robert Morris. Link-level measurements from an 802.11b mesh network. URL `citeseer.ist.psu.edu/aguayo04linklevel.html`.

[2] Matthew Andrews, Lijun Qian, and Alexander Stolyar. Optimal utility based multi-user throughput allocation subject to throughput constraints. URL `citeseer.ist.psu.edu/730790.html`.

[3] Erdal Arikan. Some complexity results about packet radio networks. *IEEE Transactions on Information Theory*, 30(4):681–, 1984.

[4] K. J. Arrow and L. Hurwicz. On the stability of the competitive equilibrium i. *Econometrica*, 26:no. 4, 522–552, 1958.

[5] K. J. Arrow and L. Hurwicz. On the stability of the competitive equilibrium ii. *Econometrica*, 27:82–109, 1959.

[6] Kenneth Joseph Arrow. *General Competitive Analysis*. Holden-Day, 1971., San Francisco.

[7] Yossi Azar, Edith Cohen, Amos Fiat, Haim Kaplan, and Harald Racke. Optimal oblivious routing in polynomial time. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 383–388, New York, NY, USA, 2003. ACM Press. ISBN 1-58113-674-9. doi: http://doi.acm.org/10.1145/780542.780599.

[8] Randeep Bhatia and Li (Erran) Li. Characterizing achievable multicast rates in multi-hop wireless networks. In *MobiHoc '05: Proceedings of the 6th ACM*

international symposium on Mobile ad hoc networking and computing, pages 133–144, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-004-3. doi: http://doi.acm.org/10.1145/1062689.1062708.

[9] J. Cheng and M. Wellman. The walras algorithm: A convergent distributed implementation of general-equilibrium outcomes, 1996. URL `citeseer.ist.psu.edu/cheng96walras.html`.

[10] S. Even, O. Goldreich, S. Moran, and P. Tong. On the NP-completeness of certain network testing problems. *Networks*, 14:1–24, 1984.

[11] Bruce E. Hajek and Galen H. Sasaki. Link scheduling in polynomial time. *IEEE Transactions on Information Theory*, 34(5):910–917, 1988.

[12] Ilog, Inc. CPLEX solver, 2006. URL `http://www.ilog.com/products/cplex/`. (accessed 10 March 2006).

[13] Kamal Jain, Jitendra Padhye, Venkata N. Padmanabhan, and Lili Qiu. Impact of interference on multi-hop wireless network performance. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 66–80, New York, NY, USA, 2003. ACM Press. ISBN 1-58113-753-2. doi: http://doi.acm.org/10.1145/938985.938993.

[14] R. Jain, A. Durresi, and G. Babic. Throughput fairness index: An explanation. In *ATM Forum*, pages 99–0045, 1999.

[15] V. S. Anil Kumar, Madhav V. Marathe, Srinivasan Parthasarathy, and Aravind Srinivasan. Algorithmic aspects of capacity in wireless networks. In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 133–144, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-022-1. doi: http://doi.acm.org/10.1145/1064212.1064228.

[16] Yuxi Li, Janelle Harms, and Robert Holte. Traffic-oblivious energy-aware routing for multihop wireless networks. In *TR05-24*, University of Alberta,

September 2005. Dept. of Computing Science, University of Alberta. URL http://www.cs.ualberta.ca/ yuxi/.

[17] motelab. Harvard sensor network testbed. web. http://motelab.eecs.harvard.edu.

[18] Jitendra Padhye, Sharad Agarwal, Venkata N. Padmanabhan, Lili Qiu, Ananth Rao, and Brian Zill. Estimation of link interference in static multi-hop wireless networks. In *IMC '05: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, New York, NY, USA, 2005. ACM Press.

[19] Bozidar Radunovic and Jean-Yves Le Boudec. Rate performance objectives of multi-hop wireless networks. *IEEE Trans. Mob. Comput.*, 3(4):334–349, 2004.

[20] S. Ramanathan. A unified framework and algorithm for channel assignment in wireless networks. *Wirel. Netw.*, 5(2):81–94, 1999. ISSN 1022-0038. doi: http://dx.doi.org/10.1023/A:1019126406181.

[21] S. Ramanathan and Errol L. Lloyd. Scheduling algorithms for multi-hop radio networks. In *SIGCOMM '92: Conference proceedings on Communications architectures & protocols*, pages 211–222, New York, NY, USA, 1992. ACM Press. ISBN 0-89791-525-9. doi: http://doi.acm.org/10.1145/144179.144283.

[22] E. Royer and C. Toh. A review of current routing protocols for ad-hoc mobile wireless networks, 1999. URL citeseer.ist.psu.edu/royer99review.html.

[23] Herbert E. Scarf. Some examples of global instability of the competitive equilibrium. Cowles Foundation Discussion Papers 79, Cowles Foundation, Yale University, 1959. available at http://ideas.repec.org/p/cwl/cwldpp/79.html.

[24] Victor Shnayder, Bor rong Chen, Konrad Lorincz, Thaddeus R. F. Fulford Jones, and Matt Welsh. Sensor networks for medical care. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 314–314, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-054-X. doi: http://doi.acm.org/10.1145/1098918.1098979.

[25] Godfrey Tan and John Guttag. Capacity Allocation in Wireless LANs. Number 973, Cambridge, MA, November 2004.

[26] Michael P. Wellman. Market-oriented programming: Some early lessons. In Scott Clearwater, editor, *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific, River Edge, New Jersey, 1996. URL citeseer.ist.psu.edu/wellman95marketoriented.html.

[27] Michael P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993. URL citeseer.ist.psu.edu/wellman93marketoriented.html.

[28] Shanchieh Jay Yang and Gustavo de Veciana. Enhancing both network and user performance for networks supporting best effort traffic. *IEEE/ACM Trans. Netw.*, 12(2): 349–360, 2004. ISSN 1063-6692. doi: http://dx.doi.org/10.1109/TNET.2004.826280.

[29] F. Ygge and H. Akkermans. Duality in multi-commodity market computations, 1997. URL citeseer.ist.psu.edu/article/ygge97duality.html.