ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΜΕΛΕΤΗ ΕΓΩΙΣΤΙΚΗΣ ΣΥΜΠΕΡΙΦΟΡΑΣ ΚΟΜΒΩΝ ΣΕ ΑΣΥΡΜΑΤΑ ΔΙΚΤΥΑ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΗΜΗΤΡΙΟΣ ΓΙΑΤΣΙΟΣ

Επιβλέπων Καθηγητής: ΛΕΑΝΔΡΟΣ ΤΑΣΙΟΥΛΑΣ

Βόλος, Ιούνιος 2011

Περίληψη

Τα τελευταία χρόνια, οι διαστρωματικοί αλγόριθμοι βελτιστοποίησης έχουν αποτελέσει αντικείμενο εντατικής έρευνας στο πεδίο των ασύρματων δικτύων. Ο στόχος αυτών των αλγορίθμων είναι η εξασφάλιση της σταθερότητας του δικτύου και η επίτευξη κάποιας μορφής δικαιοσύνης μεταξύ των χρηστών, όποτε η ζήτηση ξεπερνάει τη χωρητικότητα του δικτύου. Σε αυτήν την εργασία μελετάμε το πρόβλημα της εγωιστικής συμπεριφοράς κόμβων σε δίκτυα που λειτουργούν με τέτοιους αλγορίθμους. Συγκεκριμένα, επικεντρώνοντας σε διαστρωματικούς αλγορίθμους που βασίζονται σε πληροφορία σχετικά με τα μήκη των ουρών δικτύου στους κόμβους, εξετάζουμε την περίπτωση όπου ένας κόμβος δηλώνει ψευδή τιμή για το μήκος της ουράς του, με σκοπό να αυξήσει τη ρυθμαπόδοσή του. Η επίδραση μιας τέτοιας συμπεριφοράς σε ένα απλό δίκτυο αποτελούμενο από ένα σημείο πρόσβασης και πολλαπλούς ασύρματους κόμβους που ανταγωνίζονται για πρόσβαση στο κανάλι μελετάται με τη βοήθεια προσομοιώσεων. Στη συνέχεια προτείνονται τροποποιήσεις των σχετικών αλγορίθμων ώστε να επιτρέπουν την ανίχνευση εγωιστών κόμβων, υπό την προϋπόθεση ότι το σημείο πρόσβασης γνωρίζει τις συναρτήσεις οφέλους των κόμβων. Επίσης, επιβεβαιώνεται ότι με κατάλληλο μηχανισμό δυναμικής τιμολόγησης μπορούν να αποφευχθούν τέτοια φαινόμενα εγωιστικής συμπεριφοράς, στην περίπτωση που οι συναρτήσεις οφέλους δεν είναι γνωστές.

Abstract

During the last few years, cross-layer optimization algorithms have been the object of intensive research in the field of wireless networks. The purpose of these algorithms is to guarantee stable operation and to achieve some form of optimal fairness among users, whenever the traffic demand exceeds the network capacity. In this thesis we consider the problem of selfish misbehavior under such schemes. In particular, focusing on cross-layer algorithms utilizing queue backlog information, we examine the case where a node declares a false backlog value for its queue, in order to achieve throughput gain. The effect of such a behavior on an uplink with an access-point and contending stations is evaluated through simulations. Modifications of the algorithms are proposed, in order to detect misbehaving stations, under the assumption that the access-point is aware of the utility functions of the stations. It is also verified that, in the absence of knowledge of the utility functions, an appropriate pricing mechanism can be employed for the avoidance of such selfish behaviors.

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer, Telecommunications and Network Technology from the Department of Computer, Telecommunications and Network Engineering of the University of Thessaly.

Its subject is the study of selfish behavior in the context of cross-layer optimization algorithms designed for wireless networks. In particular, the effect of false backlog declaration in algorithms utilizing queue backlog information for their operation is examined, along with potential detection or alleviation strategies. This is a type of selfish misbehavior not explicitly studied before in the literature to the best of our knowledge.

The structure of this thesis is as follows:

In ***Chapter 1*** the notion of fairness in telecommunications networks is introduced and related work on selfishness is presented. In ***Chapter 2*** a class of throughput optimal cross-layer algorithms is presented which provably stabilize the network and provide some form of optimal network fairness. In ***Chapter 3*** the effect of selfish backlog declaration on the performance of these algorithms in a simple one-hop network is evaluated for the case of infinite traffic demand. Accurate detection schemes are proposed, under the assumption of knowledge of utility functions. An appropriate pricing scheme is shown to alleviate such behaviors. In ***Chapter 4*** the same analysis is performed for the more realistic case of finite traffic demand. Finally, in ***Chapter 5*** we summarize the most important conclusions and propose directions for future work.

Acknowledgements

I would like to thank my professors Leandros Tassiulas, Iordanis Koutsopoulos and Thanassis Korakis for giving me the opportunity to do my master´s thesis in an active research topic such as cross-layer optimization algorithms for wireless networks, as well as for their support throughout my efforts.

This thesis has been funded by a Center for Research and Technology Hellas (CERTH) scholarship.

Table of contents

1. Fairness and selfishness in telecommunication networks

1.1 Background

Modern data networks consist of a variety of heterogeneous components, and continue to grow as new applications are developed and new technologies are integrated into the existing communication infrastructure. While network resources are expanding, the demand for these resources is also expanding, and it is often the case that data links are loaded with more traffic than they were designed to handle. Concurrent flows compete for the finite resources of a network and the rate allocation to each flow must be regulated by some control mechanism to avoid congestion and reduce packet losses in the network.

A straightforward approach for rate allocation in competitive scenarios is to find a feasible rate allocation that maximizes the total throughput. However, total throughput maximization can be considered as an unfair allocation policy, as it usually neglects some users and offers as much resources as possible to others. On the other hand, offering the same throughput to every user might lead to poor overall performance. Usually a compromise between these two often conflicting goals is pursued, leading to some form of so-called network fairness.

1.2 Categorization of traffic

Traffic in data networks can be roughly divided into two categories: i) non-elastic traffic and ii) elastic traffic.

Non-elastic traffic is generated by services and applications that require strict quality of service guarantees for minimum available bandwidth, maximum allowed packet loss, delay, jitter, etc. This type of traffic is, for example, generated by streaming (real-time) services and applications such as traditional telephony and IP telephony (VoIP), video conferencing, video broadcasting over IP (IPTV), video on demand (VOD), etc.

The other type of traffic is elastic traffic, which constitutes the majority of the traffic in the Internet. Elastic traffic is generated by “greedy” services and applications, i.e. those that can

adapt their instantaneous packet rate to the available bandwidth in the network. Examples of applications and services generating this kind of traffic are web browsing, newsgroups, file downloads and peer-to-peer applications.

In the next section we discuss some types of network fairness designed for elastic traffic.

1.3 Types of fairness

The notion of fairness is a somewhat subjective concept and it doesn't have a unique definition. It may depend on several different session priorities and service requirements. Different fairness criteria favor or discriminate sources or traffic classes on a different basis.

In the effort to establish a quantitative measure of network fairness, the most typical approach is the use of utility functions. A utility function $g_i(x_i)$ is assigned to each session i in the network and represents the "satisfaction" received by the user of this session when sending data with a time average rate of x_i. In this way, fairness can be formulated as an optimization problem involving the maximization of the aggregate network utility.

Various types of fairness have been proposed in the networking literature. Below we summarize some of the most well-known.

Proportional fairness was proposed in [1]. In this type of fairness, deviation from the fair allocation causes a negative average proportional change, i.e. increasing the average throughput of one user from the optimal level by x% results in a cumulative percentage decrease by more than x% of the average throughput of other users [2]. A rate allocation $\{\hat{x}_i\}$ is proportionally fair if it satisfies

$$\sum_i \frac{x_i - \hat{x}_i}{\hat{x}_i} \leq 0 \tag{1.1}$$

where $\{x_i\}$ is any other possible rate allocation. It turns out that such a solution is achieved when the optimization objective is to maximize the sum of the logarithms of the expected rates (or equivalently the product of expected rates), subject to network stability.

Max-min fairness is a simple, well-recognized approach to define fairness in networks [3, 4]. It aims at allocating as much as possible to poor users and, at the same time, not unnecessarily wasting resources. An allocation $\{\hat{x}_i\}$ is called max-min fair, if it satisfies the following property:

$$\min_i \hat{x}_i \geq \min_i x_i \tag{1.2}$$

where $\{x_i\}$ is any other possible allocation. The property implies that under max-min fairness the least fortunate user gets a greater throughput share than in any other allocation.

Another form of fairness that has been discussed in the literature is called minimum potential delay fairness, introduced in [5]. Under this form of fairness, the goal is to minimize $\sum_i 1/x_i$. The term $1/x_i$ measures the delay (in particular it is the delay associated with the completion of the transfer of a unit size file).

All the above notions of fairness can be captured by using utility functions of the form

$$g_i(x_i) = \frac{x_i^{1-a}}{1-a} \tag{1.3}$$

for some $a > 0$. Resource allocation using the above utility function is called a-fair. Different values of a yield different ideas of fairness. For instance, $a = 2$ yields minimum potential delay fairness, $a \to 1$ yields proportional fairness and $a \to \infty$ yields max-min fairness. The concept of a-fairness as a common framework to study many notions of fairness was proposed in [6].

Furthermore, for each of the discussed types of fairness there is a weighted version, where the utility function of the user is multiplied by a weight w_i. In this way, schemes with explicit priority service classes can be designed, where some users are favored over others, according to their weights.

1.4 Selfishness in networks – Related work

Conflicting interests of users competing for a greater share of the available network resources create the circumstances for selfish misbehavior. Therefore resource allocation algorithms for networks should be designed in a way that either selfishness is discouraged through appropriate pricing mechanisms or a framework for detecting and punishing selfish users is incorporated into the algorithms.

Pricing mechanisms for elastic traffic in networks have received much attention during recent years, particularly in the context of cross-layer optimization algorithms. These algorithms

provably outperform traditional layered algorithms in terms of efficiency, but typically involve the cooperation of users for the achievement of some optimal system-wide performance objective. The goal of a pricing mechanism in this context is to either discourage selfishness completely or to bring the system into an equilibrium which achieves a certain fraction of the maximum possible aggregate utility. Below we refer to three examples of such mechanisms in the literature.

Consider the aggregate utility maximization problem which, as we mentioned in the previous section, corresponds to some optimal fairness criterion. The network planner can only maximize the aggregate utility if he knows the utility functions of the users or if there is an incentive for the users to reveal their utility functions truthfully. One potential candidate for such an incentive mechanism is the VCG (Vickrey-Clark-Groves) mechanism, developed in [7, 8]. The mechanism works as follows. The network planner calculates the reduction in the sum of the utilities obtained by other users in the network due to the presence of user i and sets this amount as the price to be paid by user i. Under this pricing mechanism, an optimal strategy for a rational user is to truthfully reveal its utility function. A proof of this property in a networking context can be found in [9]. The flaw of this mechanism is its high complexity, as it involves the solution of many, potentially computationally expensive optimization problems. Furthermore, it requires revealing entire functions through information exchange between the users and the network planner.

One possible method to tackle the latter problem, i.e. to reduce the communication burden required to exchange information between users and the network planner is to ask the users to submit bids which are amounts that the users are willing to pay for the resource. Then it can adapt the price per unit amount of resource at each timeslot according to the offered bids. This mechanism was introduced in [1]. It can be shown that this scheme achieves the desired utility maximization in the case where the users are "price-taking", i.e. they are unaware of the effect of their bids on the prices charged. In the case of price anticipating users, it is suboptimal, but it has been shown in [10] that the price of anarchy in this case is no greater than one fourth of the optimal solution.

In [11] a different problem is examined. In particular, the authors study the case where a user achieves extra throughput by misleading the scheduling component of the network regarding its link capacity. The authors prove that when this kind of misbehavior occurs the cheater increases its utility function by a factor k in the equivalent optimization problem being solved by the cross-layer algorithm, where k is the ratio of the cheater's link capacity as perceived by the scheduler to the actual capacity of the link. This leads to an increased throughput share for the cheater. The authors propose a pricing scheme for the alleviation of such selfish

behaviors. In particular, they propose a heuristic with a linear pricing function λx, with parameter λ initially equal to zero and increased by the controller up to the point where the network's resources are being underutilized. Underutilization of resources occurs when one or more of the network's capacity region constraints are satisfied with strict inequality, so the controller needs to make that check as it increases λ. The method does not require knowledge of the utility functions, but knowledge of the capacity region constraints is required, which might be impossible or particularly difficult even for moderately complex wireless topologies.

While pricing mechanisms, or else incentive engineering, is the usual approach for clean-slate cross-layer optimization algorithms, detection-oriented approaches have been proposed as modifications or additions to more practical and readily implementable algorithms, or even as enhancements to the protocols actually used in today's networks.

Routing algorithms for multi-hop ad-hoc networks are a typical example. A user participating in such a network might choose to drop packets he is supposed to forward, in order to save battery power or processing resources. Various approaches have been proposed for detecting such users. See for example [12] where the collusion of multiple misbehaving nodes is also taken into consideration and the solution proposed is the use of virtual graphs at the destination of a flow.

Another area where detection methods are applicable is that of 802.11 MAC misbehavior. In this case a node may deviate from the legitimate behavior (for example, it may alter its average contention window), in order to increase the amount of time it is scheduled to transmit. See for example [13] for an approach where such a behavior is tackled timely and efficiently through a robust minimax detection mechanism.

In this thesis, we will concentrate on a particular type of selfish misbehavior, which arises in the context of cross-layer stochastic optimization algorithms for wireless networks and has not been explicitly studied before, to the best of our knowledge. This is the false declaration of queue backlog values in cross-layer algorithms which involve exchange of queue backlog information between the nodes and the network controller.

As far as tackling selfish misbehavior is concerned, the primary focus of this thesis is on scenarios where pricing is not an option or it is not desirable. In this case the network controller can adopt a certain fairness criterion, announce its decision to the contending users and monitor their behavior in order to detect deviations from legitimate behavior. Alternatively, a negotiation phase between a user and the controller might take place upon the user's entrance into the network, resulting in the selection of a specific utility function on behalf of the user. In any case it is assumed that the network controller is aware of the utility

functions of the users, during network operation. Despite the fact that our focus will be on detection approaches, we will also refer to a pricing mechanism in chapter 3, which, as we will show, discourages users from adopting the misbehavior pattern we examine, without requiring knowledge of the utility functions.

2. Optimization-based cross-layer algorithms for wireless networks

It is clear nowadays that significant portions of the Internet are evolving towards wireless components. Examples include data access through cellular phones, wireless LANs and multi-hop wireless networks which are prevalent in military, law enforcement and emergency response networks, and which are expected to become common in the civilian section at least in certain niche markets such as viewing replays during football games, concerts, etc. The inherently limited capacity of these networks has motivated a lot of research on the problem of pushing these networks to their limits, naturally leading to optimization-based approaches. One of the fundamental results of this research area is the fact that traditional strictly layered approaches, such as the ones based on the OSI or the TCP/IP protocol stacks, are suboptimal and cannot exploit the full potential of the networks. For that reason, cross-layer algorithms have been developed, which achieve superior performance through some form of coupling among the lower protocol stack layers, usually implemented with message passing.

In this chapter we will present a particular class of dynamic cross-layer algorithms for wireless networks, whose basic characteristic is the key role of the network level queue backlogs of the nodes in the resource allocation, routing and flow control decisions. These algorithms will form the ground for the selfish strategies to be discussed in chapters 3 and 4.

2.1 The network model

In this section we describe a rather generic network model, in order to subsequently present a class of cross-layer optimization-based algorithms that have been designed for such networks. In the next chapters, where we examine selfish behavior patterns and possible detection approaches, we will restrict our analysis to a simple example, which can be considered as an important special case of the generic model we present here. We follow the notation of [14].

Consider a general network with a set $\mathcal{N}$ of N nodes and a set $\mathcal{L}$ of L links. Each link represents a communication channel for direct transmission from a given node a to a given node b and is labeled by its corresponding ordered pair (a, b). We assume that the network operates in timeslots normalized to integral units, where slot t refers to the interval $[t, t+1)$.

The wireless medium conditions are assumed to be changing over time according to some random process. Furthermore, the nodes might move in space according to some random pattern. We capture all these parameters with a single process, the topology state process $S(t)$, which represents the current attenuation coefficients between each node pair.

Due to interference constraints generally present in a wireless network, not all links in the network can be active (i.e. transmitting traffic) simultaneously. We denote by $I(t)$ the transmission control input for timeslot t, that is, the particular set of links chosen for activation during that timeslot. Every timeslot the network controller observes the current topology state $S(t)$ and chooses a transmission control input $I(t)$, according to some policy. $I(t)$ takes values in a general state space $\mathcal{I}_{S(t)}$ which represents all the possible resource allocation options under a given topology state $S(t)$.

The link transmission rates, which we denote by the matrix $\boldsymbol{\mu}(t) = (\mu_{ab}(t))$, depend on both $I(t)$ and $S(t)$, and are determined by a link transmission rate function $C(I, S)$, so that

$$\boldsymbol{\mu}(t) = \boldsymbol{C}(I(t), S(t)) \tag{2.1}$$

All data that enters the network is associated with a particular commodity, which minimally defines the destination of the data, but might also specify other information, such as the source node or its priority service class. Let K represent the number of distinct commodities in the network. Each node i maintains a set of internal network layer queues for storing data according to its commodity. Let $U_i^{(c)}(t)$ represent the current backlog (unfinished work) of commodity c data stored in a network layer queue at node i. Naturally, no node maintains a queue for data destined to itself (data leaves the network when it reaches its destination), therefore $U_i^{(i)}(t)$ is by default equal to zero for all t. The queue backlog $U_i^{(c)}(t)$ can contain both data that arrived exogenously from the transport layer at node i as well as data that arrived endogenously through network layer transmissions from other nodes.

Let $A_i^{(c)}(t)$ represent the amount of new commodity c data that exogenously arrives to source node i during slot t. This data first enters a storage reservoir at the transport layer of the node and awaits admission to the network layer. There is a separate storage reservoir for each commodity at each node, whose backlog we denote by $L_i^{(c)}(t)$. Every timeslot, each source node makes flow control decisions by choosing the amount of bits $R_i^{(c)}(t)$ to deliver to the network layer, subject to the constraint

$$R_i^{(c)}(t) \leq L_i^{(c)}(t) + A_i^{(c)}(t) \tag{2.2}$$

and possibly subject to an upper bound for the amount of data admitted at each timeslot, in order to prevent burstiness of arrivals.

Apart from the resource allocation decision $I(t)$, the network controller also needs to make a routing decision at each timeslot. In particular, it needs to determine the commodity of the data to be transmitted over each active link. In a more general case, where multiple commodities can be transmitted over a link during a single timeslot, it must determine the respective rates of transfer $\mu_{ab}^{(c)}(t)$ for each commodity c. It is often convenient to pose restrictions to the set of links which commodity c data is allowed to use. Denoting this set of links as $\mathcal{L}_c$, the routing variables $\mu_{ab}^{(c)}(t)$ are subject to the following constraints:

$$\sum_{c\in\mathcal{K}} \mu_{ab}^{(c)}(t) \leq \mu_{ab}(t) \tag{2.3}$$

$$\mu_{ab}^{(c)}(t) = 0, if\ (a,b) \notin \mathcal{L}_c \tag{2.4}$$

It is easy to see that the slot-to-slot dynamics of the queue backlog $U_i^{(c)}(t)$ satisfy the following inequality:

$$U_i^{(c)}(t+1) \leq max\left[U_i^{(c)}(t) - \sum_b \mu_{ib}^{(c)}(t), 0\right] + R_i^{(c)}(t) + \sum_a \mu_{ai}^{(c)}(t) \tag{2.5}$$

The above expression is an inequality rather than an equality because the actual amount of commodity c data arriving to node i during slot t may be less than $\sum_a \mu_{ai}^{(c)}(t)$ if the neighboring nodes have little or no commodity c data to transmit.

2.2 The notions of stability and network capacity region

An important class of problems in networking deals with the development of opportunistic scheduling schemes with the intention of accommodating the maximum possible offered load on the system without violating stability or other QoS constraints. Below we define the notion of stability of a network.

A queue is called strongly stable if:

$$\lim_{t\to\infty}\frac{1}{t}\sum_{\tau=0}^{t-1}E\{U(\tau)\}<\infty \tag{2.6}$$

That is, a queue is strongly stable if it has a bounded time average backlog.

A network is strongly stable if all individual queues of the network are strongly stable. Hereon, we will use the term stability to refer to strong stability of a queue or a network, without mentioning it explicitly.

We assume that the exogenous arrival processes $A_i^{(c)}(t)$ are admissible with rates $\lambda_i^{(c)}(t)$. This essentially means that their time average expected arrival rates are $\lambda_i^{(c)}(t)$ and they further meet some burstiness constraint requirements. For a formal definition of arrival process admissibility refer to [14].

The network layer capacity region Λ is the closure of the set of all arrival rate matrices $\lambda_i^{(c)}(t)$ that can be stably supported by the network, considering all possible strategies for choosing the control variables to affect routing, scheduling and resource allocation (including strategies that have perfect knowledge of future events).

In order to construct the capacity region Λ, we first examine the one-hop capacity region Γ of a wireless network, which is the region of all achievable long-term link rates in the network. For a given topology state space $S(t)$ there are in general many possible transmission control policies. As it is possible to randomize between policies and achieve long-term rates not achievable by a single policy, the region of achievable link rates for a given topology state is the convex hull of the link rates allocated by each policy. Furthermore, each topology state occurs with a certain probability, so we must average over all possible topology states. Thus, the one-hop capacity region Γ is defined as follows:

$$\Gamma \triangleq \sum_{s\in S}\pi_s Conv\{\boldsymbol{C}(I,s)|I\in\mathcal{I}_s\} \tag{2.7}$$

In the case of a single-hop network, the capacity region Λ coincides with the region Γ as defined above. In the more general case of a multi-hop network, Λ is defined with the help of multi-commodity flow variables $f_{ab}^{(c)}$, which represent the long term flow rates of data of different commodities over the various links of the network.

As we mentioned, due to routing constraints, some commodities do not traverse certain links and therefore might never visit certain nodes. These nodes do not maintain queues for these commodities. Let us denote by $\mathcal{D}$ the subset of node-commodity pairs associated with internal queues in the network. An arrival rate matrix $\left(\lambda_i^{(c)}\right)$ is in the capacity region Λ if there exists a link rate matrix $(G_{ab}) \in Cl\{\Gamma\}$ together with multi-commodity flow variables $f_{ab}^{(c)}$ satisfying:

$$f_{ab}^{(c)} \geq 0, f_{aa}^{(c)} = f_{dest(c),b}^{(c)} = 0, \ \forall a, b \in \mathcal{N}, \ c \in \mathcal{K}, \tag{2.8}$$

$$\lambda_i^{(c)} = 0, if \ (i,c) \notin \mathcal{D}, \tag{2.9}$$

$$\lambda_i^{(c)} \leq \sum_b f_{ib}^{(c)} - \sum_a f_{ai}^{(c)}, \forall (i,c) \in \mathcal{D} \text{ with i}\neq\text{dest(c)}, \tag{2.10}$$

$$f_{ab}^{(c)} = 0, if \ (a,b) \notin \mathcal{L}_c, \tag{2.11}$$

$$\sum_c f_{ab}^{(c)} \leq G_{ab}, \forall \ a, b \in \mathcal{N}. \tag{2.12}$$

The above relationships describe a traditional graph network. They can be used to define the capacity region of a wired network as well, with the only difference being that in the wired case the link rates (G_{ab}) are constant.

The set Λ is convex, closed, bounded and contains the all-zero matrix. It has been proved [15] that $(\lambda_i^{(c)}) \in \Lambda$ is a necessary condition for stability, while $(\lambda_i^{(c)})$ within the relative interior of Λ is sufficient.

In the sequel, we present a class of scheduling schemes which stabilize the network whenever any other feasible scheduler can. Schemes with this property are called throughput-optimal.

2.3 Scheduling in one-hop wireless networks: the max-weight principle

Let us first consider the case of a one-hop wireless network such as the one depicted in figure 2.1. It is clear that this kind of topology is also applicable to the case of cellular networks, as well as to ad-hoc one-hop wireless networks where all links interfere with each other and the transmission schedule at each timeslot is determined by a central controller aware of the current topology state and queue backlog values of the nodes. We examine the uplink direction of traffic here, but the downlink problem is exactly equivalent.

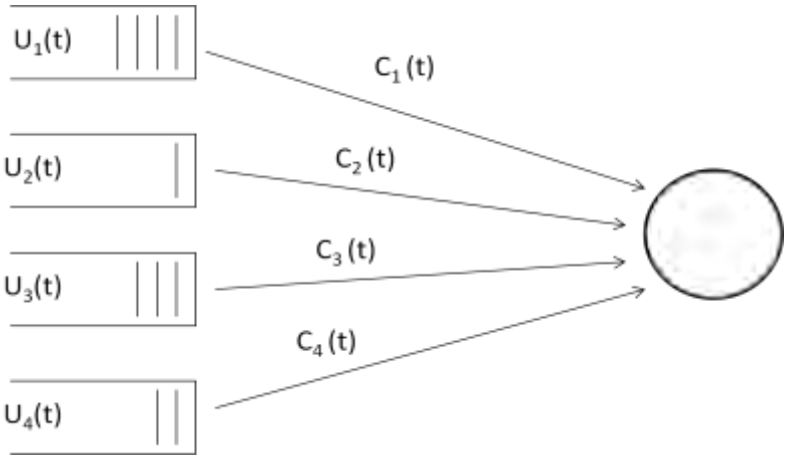

Figure 2.1: An access-point based one-hop wireless network

In this scenario, a transmission schedule coincides with the decision on the single user to transmit during each timeslot. Furthermore, since all the uplink traffic is headed to the access point, there is only a single commodity and each node i maintains a single network layer queue with backlog $U_i(t)$. Each link has a random instantaneous capacity $C_i(t)$.

Stochastic stability of access-point-based single-hop wireless networks is a well examined problem. Most of the work has been motivated by the seminal work on throughput-optimal scheduling in [16, 17]. This work shows that scheduling schemes that maximize the queue-weighted sum of physical rates are throughput optimal. The transmission control policy in this case is thus

$$I(t) = \underset{i=1,\ldots,N}{\operatorname{argmax}} U_i(t)\mu_i(t) \tag{2.13}$$

where by $\mu_i(t)$ we denote the actual transmission rate of node i towards the access-point, if node i is scheduled to transmit.

This resource allocation algorithm is widely known as the *max-weight* algorithm. It provides the important insight that queue-length information is critical in developing throughput optimal scheduling schemes. It is important to note that the algorithm doesn't require knowledge of the arrival rates or the topology state probabilities.

Based on this idea, similar scheduling policies have been developed. Some of these replace the queue-based weights with delay-based weights. Intuitively, this approach is equivalent to the queue-based scheme (as Little's law implies). Approaches utilizing the so-called head-of-line delay (HOL delay), which is the delay experienced by the head packet of each queue, have the additional desirable property of avoiding extensive delays for packets of small queues, while retaining throughput optimality [18].

While opportunistic scheduling schemes such as the max-weight algorithm expand the capacity region over their non-opportunistic counterparts, it may be difficult to utilize this gain if we are unable to operate the system close to the boundary of the capacity region. For example, if one were to make a conservative estimate of the boundary of the capacity region and traffic were injected into the system based on this conservative estimate, the opportunistic gains may never be realized. Thus it is imperative that one solves the problem of determining the rates to be injected into the network (i.e. congestion control) jointly with which user(s) to be scheduled for transmission.

In section 2.5, we will examine a class of flow control algorithms which are coupled with the scheduling algorithms, so that the resulting cross-layer algorithms are operating near the capacity region boundary, whenever the exogenous arrival rate matrix lies outside the capacity region of the network.

2.4 Joint routing and scheduling in multi-hop wireless networks: the back-pressure algorithm

For general multi-hop networks, stability depends on both the selection of an appropriate scheduling algorithm and efficient routing of the commodities from node to node. A throughput-optimal joint routing-scheduling scheme exists in this case too and is called the *dynamic back-pressure algorithm*, which operates as follows.

Every timeslot t, the network controller observes the queue backlog matrix $\boldsymbol{U}(t) = \left(U_i^{(c)}(t)\right)$ and the topology state variable $S(t)$ and performs the following actions for routing and resource allocation.

Resource Allocation: For each link (a, b), it defines the optimal commodity $c_{ab}^*(t)$ as the commodity that maximizes the differential backlog (ties broken arbitrarily):

$$c_{ab}^*(t) \triangleq \text{argmax}_{\{c|(a,b)\in\mathcal{L}_c\}}\left[U_a^{(c)}(t) - U_b^{(c)}(t)\right], \tag{2.14}$$

and defines $W_{ab}^*(t)$ as the corresponding optimal weight:

$$W_{ab}^*(t) \triangleq max\left[U_a^{c_{ab}^*(t)}(t) - U_b^{c_{ab}^*(t)}(t), 0\right] \tag{2.15}$$

It then chooses the control action $I(t)$ that solves the following optimization:

$$\text{Maximize: } \sum_{ab} W_{ab}^*(t)\, C_{ab}(I(t), S(t)), \tag{2.16}$$

$$\text{Subject to: } I(t) \in \mathcal{I}_{S(t)}. \tag{2.17}$$

Routing: For each link (a, b) such that $W_{ab}^*(t) > 0$, the controller offers a transmission rate $\mu_{ab}(t) = C_{ab}(I(t), S(t))$ to data of commodity $c_{ab}^*(t)$. If there is not enough data of this commodity in node a to transmit over all outgoing links requesting this commodity, idle fill bits are transmitted, with an arbitrary allocation of actual data and idle fill data over the corresponding outgoing links.

It is clear that the max-weight algorithm based on queue lengths we saw in the previous section can be viewed as the implementation of the back-pressure algorithm in single-hop networks.

The proof of the stability of a general multi-hop network under the dynamic back-pressure algorithm is based on the theory of Lyapunov functions and Lyapunov drifts and can be found in [14]. The first application of this theory to the design of dynamic control algorithms for radio networks appeared in [6].

The routing part of the algorithm can be implemented in a distributed way, given that each node is aware of the backlog level of its one-hop neighbors. The resource allocation part, however, involves the solution of a system-wide optimization problem and thus assumes the existence of a central network controller aware of the $W_{ab}^*(t)$ weights of each link in a

network and of the topology state $S(t)$. In most cases this is not practical and it is the bottleneck of the cross-layer solutions proposed, such as the ones we present in the next section.

The dynamic back-pressure algorithm has been designed for efficiency of the network under heavy traffic loads. In cases when the network is lightly loaded, the algorithm can lead to large end-to-end delays. For example, suppose that a given node wishes to send a single packet to a remote destination node. In this case, there is no back-pressure to suggest which is the most appropriate path to the destination. Therefore, the packet will take a random walk until, by chance, it reaches its destination. Such problems can be avoided if the back-pressure algorithm is combined with a shortest path or similar routing algorithm, in order to provide efficient routing of packets for both light and heavy traffic loads. See [15] for an example of such a variation.

2.5 Flow control for optimal network fairness

The max-weight algorithm for single-hop networks and the dynamic back-pressure algorithm for multi-hop networks guarantee stability whenever the traffic load is within the network capacity region. However, typically the capacity region is not known in advance, as the topology state probabilities are usually unknown. Therefore, we cannot operate the network near its boundaries, unless we have a systematic way of doing so.

Another important point is that these algorithms alone do not account for fairness. Thus, if the traffic demand matrix lies outside the capacity region, there should be an additional mechanism regulating the admission rates $R_i^{(c)}(t)$ of each network flow in a way that a system-wide performance metric is optimized. This metric can be designed in a manner that some type of optimal fairness is achieved, as we discussed in chapter 1.

These considerations lead to the conclusion that there should be some form of coupling between the flow control mechanism at each node and the network's resource allocation and routing mechanism, possibly through message-passing. In other words, optimal network fairness is achievable through a cross-layer strategy.

Next, we present a class of cross-layer dynamic network control algorithms which achieve these goals for a general network model.

2.5.1 Infinite traffic demand

We first consider the case where all the sessions have an infinite backlog in their transport layer reservoirs, so that they can choose the values of their flow control variables $R_i^{(c)}(t)$ without first establishing that this much data is available for admission. In order to limit congestion in the network, it is important to restrict flow control decisions at every node so that $\sum_{c=1}^{K} R_i^{(c)}(t) \leq R_i^{max}$, where R_i^{max} is defined to be the largest possible transmission rate out of node i.

A cross-layer control algorithm for the infinite demand case was developed in [19], called CLC1. The CLC1 algorithm uses the dynamic back-pressure algorithm we presented in section 2.4 for routing, scheduling and resource allocation. For flow control, it uses a decentralized algorithm called FLOW1, which operates as follows.

FLOW1: Every timeslot, the flow controller at each node i observes the current level of queue backlogs $U_i^{(c)}(t)$ for each commodity c. It then sets $R_i^{(c)}(t) = r_i^{(c)}$, where the $r_i^{(c)}$values are solutions to the following optimization:

$$\max_{r_i^{(c)} \geq 0} \sum_{c=1}^{K} \left[V g_i^{(c)}\left(r_i^{(c)}\right) - r_i^{(c)} U_i^{(c)}(t) \right], \tag{2.18}$$

$$\sum_{c=1}^{K} r_i^{(c)} \leq R_i^{max} \tag{2.19}$$

where $V > 0$ is a given constant that affects the performance of the algorithm.

The V parameter determines the extent to which utility optimization is emphasized, with a related tradeoff in network congestion. In particular, by increasing V we can decrease the distance of the aggregate utility's value from the optimal value of $\sum_{i,c} g_i^{(c)}(r_{i,c}^*)$ arbitrarily, but at the cost of a linear increase in the network congestion (or equivalently, end-to-end delay).

The FLOW1 algorithm can also be viewed as a dynamic pricing scheme. If the price for injecting data into the network is defined as

$$PRICE_{i,c}(t) = \frac{U_i^{(c)}(t)}{V} \tag{2.20}$$

then a rational user trying to maximize its payoff (utility – price) at each timeslot will naturally admit data according to FLOW1.

Observe that the FLOW1 algorithm assumes that data can be admitted as fractional unities (packets or bits). A simple solution in order to overcome this problem is to append an additional stage to the flow control reservoir that only sends actual packets to the network when the accumulated admitted but undelivered data exceeds the packet length. In fact, we use this solution in our simulations of CLC1 in chapter 3.

2.5.2 Finite traffic demand

Let us now consider the case of finite traffic demand, where the transport layer reservoirs are assumed to be finitely backlogged and thus flow control decisions are subject to the additional constraint $R_i^{(c)} \leq L_i^{(c)}(t) + A_i^{(c)}(t)$.

The goal is to support a fraction of the traffic demand matrix $\left(\lambda_i^{(c)}\right)$ to achieve throughputs $\left(r_i^{(c)}\right)$ that maximize the sum of user utilities. Thus the following optimization problem must be solved.

$$Maximize: \sum_{n,c} g_i^{(c)}\left(r_i^{(c)}\right) \tag{2.21}$$

$$subject\ to: \left(r_i^{(c)}\right) \in \Lambda\ , \tag{2.22}$$

$$0 \leq \left(r_i^{(c)}\right) \leq \left(\lambda_i^{(c)}\right) \tag{2.23}$$

Inequality (2.22) is the stability constraint and ensures that the admitted rates are stabilizable by the network. Inequality (2.23) is the demand constraint and ensures that the rate provided to a session is no more than the incoming traffic rate of that session.

Let r_{nc}^* represent the solution of the above optimization. Assuming non-decreasing utility functions, it is clear that $r_{nc}^* = \lambda_n^{(c)}$ whenever $\lambda_n^{(c)} \in \Lambda$. So the challenge is to compute the optimal rate allocation whenever the arrival rate matrix lies outside the capacity region.

The optimization problem (2.21-2.23) could in principle be solved if both the arrival rates and the capacity region were known in advance, and all users could coordinate by sending data according to the optimal solution. However, the capacity region depends on the topology state probabilities, which are unknown to the network controllers and to the individual users. Furthermore, the individual users do not know the data rates or utility functions of the other users.

Below we present a cross-layer algorithm designed for the finite demand case. It is named CLC2b. It was first presented in [19] with a slightly different form (hence the 'b' in the newer version) and appeared in its current form in [14]. The dynamic back-pressure algorithm (same as in CLC1) is used for routing and resource allocation decisions in CLC2b. The flow control algorithm used is called FLOW2 and utilizes a virtual queue $Y_i^{(c)}(t)$ for every active input stream. The virtual queues are used in order to efficiently handle the demand constraint (2.23). For practical purposes, the virtual queues play a role in determining the transient time or "learning time" required for the system to approach optimal performance. The FLOW2 algorithm operates as follows.

FLOW2: Every timeslot, the queue values $\left(U_i^{(c)}(t)\right)$, $\left(Y_i^{(c)}(t)\right)$ are observed and the flow controller chooses

$$R_i^{(C)}(t) = \begin{cases} min\left[L_i^{(c)}(t) + A_i^{(c)}(t), \hat{R}_i^{(c)}\right], if\ \ \eta Y_i^{(c)}(t) > U_i^{(c)}(t) \\ 0 \qquad\qquad\qquad\qquad\quad ,otherwise \end{cases} \tag{2.24}$$

Furthermore, for each (i,c), it chooses $\gamma_i^{(c)}(t) = \gamma_i^{(c)}$, where $\gamma_i^{(c)}$ solves

$$\max\ (\ V g_i^{(c)}(\gamma) - \eta Y_i^{(c)}(t)\gamma\ ),\ \ subject\ to\ \ 0 \le \gamma \le \hat{R}_i^{(c)} \tag{2.25}$$

The virtual queues $Y_i^{(c)}(t)$ are then updated according to

$$Y_i^{(c)}(t+1) = max\left[Y_i^{(c)}(t) - R_i^{(c)}(t), 0\right] + \gamma_i^{(c)}(t) \tag{2.26}$$

The parameter η satisfies

$$0 < \eta \le 1 \tag{2.27}$$

and determines the relative weights of the virtual and the actual queues in the control problem. Choosing a small η decreases the time average backlog in the actual queues, but increases the average "learning time" required for the system to approach optimal performance.

$\hat{R}_i^{(c)}$ are suitably large constants, satisfying

$$A_i^{(c)}(t) \leq \hat{R}_i^{(c)}, \forall t \tag{2.28}$$

The proof the stability and optimality of the CLC2b algorithm is based on a Lyapunov drift technique enabling stability and performance optimization to be achieved simultaneously. It can be found in [14, 19]. The virtual queues are also stable under the CLC2b algorithm.

3 Selfish backlog declaration in the case of infinite demand

3.1 Selfish motive and misbehavior pattern

Consider the access-point-based one-hop wireless network topology we saw earlier (see Figure 2.1). Let's assume that the nodes have an infinitely backlogged transport layer storage reservoir and that the network is assumed to operate under the max-weight principle for resource allocation and under the FLOW1 algorithm for flow control at each node.

If all nodes declare their backlogs to the access point truthfully, then, as we saw in chapter 2, the network will stabilize to an allocation of physical rates which maximizes the aggregate utility and thereby some fairness criterion is satisfied.

Since the scheduling algorithm allocates the channel for use at each timeslot to the station with the highest backlog-rate product, there is a clear motive for each of the stations to declare backlog values higher than the true values. A selfish user will attempt in this way to increase its throughput by gaining more access to the channel.

Let's look at an example. We consider the single-hop access-point based network of figure 2.1. Let's assume that the channel capacities are ergodic Gaussian processes with mean values $E[C_1] = 2, E[C_2] = 1, E[C_3] = 2, E[C_4] = 3$ and equal standard deviations, equal to 0.67. Each sample of each process is rounded towards the nearest non-negative integer value, resulting in the discrete distributions we can see in Figure 3.1. The specific channel state probabilities are used in all of our simulations in this and the next chapter, so that the results are comparable to each other. The network operates under the CLC1 algorithm and all the nodes have identical utility functions $g_i(r_i) = \log(r_i), i = 1,2,3,4$. The parameter V is set equal to 30.

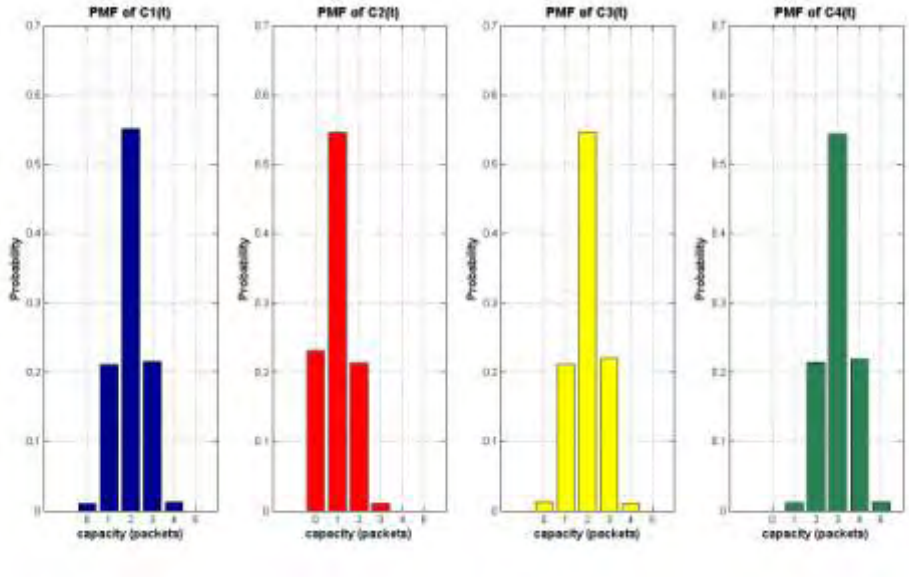

Figure 3.1: Probability mass functions of the capacities of the 4 links in the example uplink network

In figure 3.2 we can see the backlog values and the allocated throughputs for the four contending stations over a period of 9000 timeslots, when all nodes behave honestly. The imbalance in throughputs (despite the equal utility functions) is due to the differences in the time average capacities of the links. The capacity region is not symmetric so a uniform distribution of throughputs would be suboptimal from a resource allocation point of view. We also observe that a station with higher throughput than some other due to better average channel quality has a smaller average queue backlog and consequently less average delay.

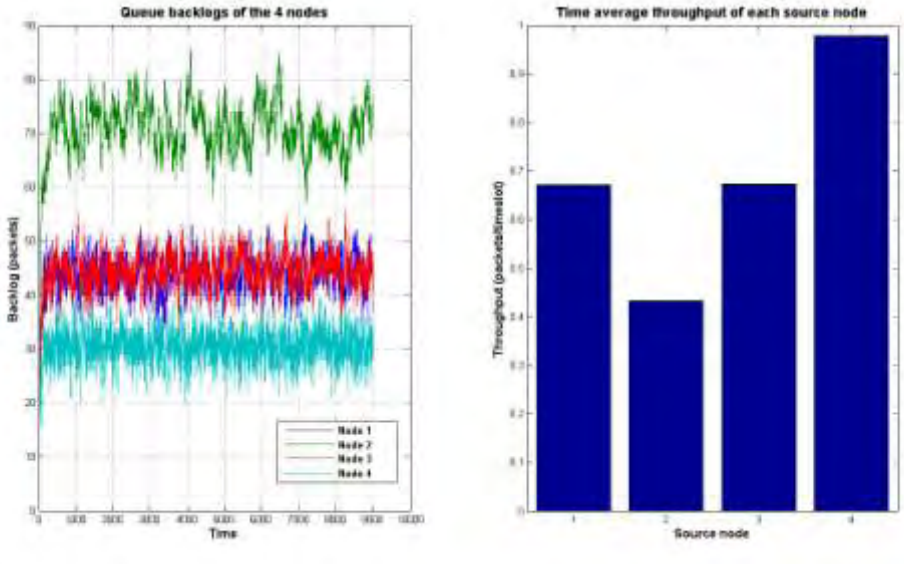

Figure 3.2: Backlog values and throughput of 4 honest stations in a single-hop network operating under the CLC1 algorithm

Now, let's assume that node 1 is selfish and declares a higher backlog value than its true backlog. For instance, we assume that it constantly declares a value greater by 10 packets than its true value. The node locally runs the FLOW1 algorithm using its real backlog value, it just announces the false value to the access-point. The results can be viewed in figure 3.3. As we see, the selfish node achieves an increase of its throughput through this strategy. Furthermore, its real backlog value is decreased and therefore it experiences less delay. The performance of the rest of the nodes, which act honestly, is deteriorated. They experience a decrease in throughput and an increase in average congestion.

It is evident that a node can increase its throughput arbitrarily in this way, limited only by its own channel's capacity. Therefore, it is crucial that detection and/or alleviation mechanisms exist to tackle this kind of misbehavior. In the next section we see how, given the utility functions of the stations, the access-point can easily detect a misbehaving user. We also study a suitable pricing mechanism in section 3.4.

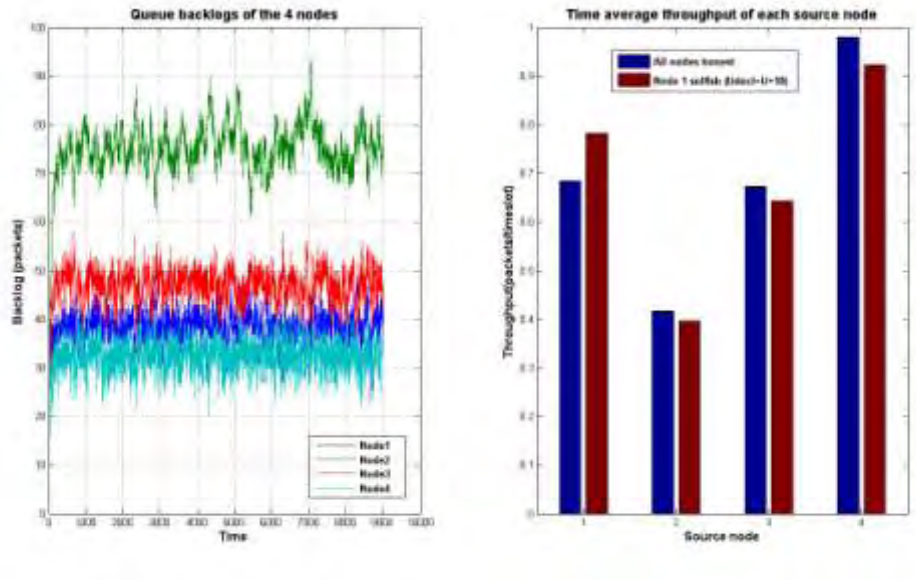

Figure 3.3: Network operating under CLC1. Real backlog values and throughputs of nodes in the case where node 1 constantly declares a backlog greater than its true backlog (declared=real+10)

3.2 Detection of selfish users

How can one go about the detection of such a selfish behavior? In the scenario we examine, the access-point is the controller which makes the resource allocation decisions. The information it has in hand is the declared backlogs of the stations at each timeslot and the

channel state at each timeslot. It can also keep track of its resource allocation decisions, thereby monitoring the actual throughput of each of the stations.

If we assume that the access point is also aware of the utility functions of the stations, it is relatively straightforward for it to detect a misbehaving selfish node. The assumption of knowledge of the utility functions is valid in cases where the network controller (in our case, the access-point) defines the utility functions itself or in cases where the stations negotiate and choose a utility function from a restricted set of functions offered.

Recall that the flow control algorithm of CLC1 is:

$$\max_{r_n^{(c)} \geq 0} \sum_{c=1}^{K} \left[V g_n^{(c)}\left(r_n^{(c)}\right) - r_n^{(c)} U_n^{(c)}(t) \right], \qquad subject\ to \sum_{c=1}^{K} r_n^{(c)} \leq R_n^{max} \tag{3.1}$$

Suppose for simplicity that we have logarithmic utility functions. In the single-hop case where there is only one commodity and one queue at each station, this leads to a simple $'1/U'$ flow control rule. For utility functions of the form $g_i(r) = w_i \log(r)$, this rule is

$$R_i(t) = min\left[\frac{Vw_i}{U_i(t)}, R_i^{max}\right] \tag{3.2}$$

For utility functions of the form $g_i(r) = w_i \log(1 + \beta r)$, the rule is

$$R_i(t) = min\left[max\left[\frac{Vw_i}{U_i(t)} - \frac{1}{\beta}, 0\right], R_i^{max}\right] \tag{3.3}$$

Once the queues build up and the network reaches steady state, the queues oscillate within a limited range of values. For large enough V this range is such that $Vw_i/U_i(t)$ is always below R_i^{max} and over $1/\beta$. Thus the above relationships can be rewritten for the steady state as

$$R_i(t) = \frac{Vw_i}{U_i(t)} - \frac{1}{\beta} \tag{3.4}$$

where the proportional fair case can be derived for $\beta \to \infty$. Taking the time average of the above equality for an interval of T timeslots during steady state yields

$$\frac{1}{T}\sum_{t=1}^{T} R_i(t) = Vw_i \frac{1}{T}\sum_{t=1}^{T} \frac{1}{U_i(t)} - \frac{1}{\beta} \tag{3.5}$$

The right hand-side of the above equality can be precisely calculated by the access-point as the stations declare their backlogs at each timeslot. It involves time averaging of a function of the backlog. The left-hand side can be implicitly estimated with great precision, because at steady state the rate of admission of packets to the network layer queue is equal to the service rate of the station (we're discussing the infinite demand case where there are always enough packets to admit). The precision depends on the length of the observation interval, but it is very high even for moderate interval sizes. In our example, for an interval of 500 timeslots, the precision was better than 1%. The access-point can keep track of the service rate of each station, based on the allocation decisions it takes and on the current capacity of the scheduled link at each timeslot.

It is clear that the above detection algorithm can be used with any type of continuously differentiable utility function, by replacing (3.4) with an equivalent rule, dependent on the derivative of the respective utility function.

We saw that given perfect knowledge of the stations' utility functions, it is straightforward for the access-point to detect a misbehaving user. Note, however, that, if the access-point is aware of the utility functions, it possesses all the information needed to run the CLC1 algorithm by itself, given that infinite demand is assumed. Therefore, there is no need for distributed flow control.

This gives rise to an alternative, totally centralized approach, where the access-point runs a "virtual" flow control algorithm for each of its associated stations. In particular, two floating point numbers must be maintained for each station, one corresponding to virtual admitted packets and one to virtual queue backlog. The numbers are allowed to be fractional, since they don't represent real packets, so there is no need for an additional intermediate stage, as in the distributed case. The virtual backlog values are used as weights in the max-weight scheduling algorithm. The virtual admitted packets quantity is updated according to (3.3) and the virtual backlog according to (3.6) below:

$$U_i(t+1) = U_i(t) - \mu_i(t) + R_i(t) \tag{3.6}$$

Since in this scenario the access-point guarantees fairness and operation near the boundary of the capacity region, the stations can implement a trivial local flow control algorithm. For instance, this could be the following:

$$R_i(t) = \begin{cases} 0, if\ U_i(t) = C_i^{max} \\ C_i^{max} - U_i(t), if\ U_i(t) < C_i^{max} \end{cases} \quad (3.7)$$

With this simple algorithm, each node guarantees that there are enough packets at its real queue to be transmitted should it be scheduled by the access-point. Network level congestion is minimized in this way and no longer depends on the proximity to the optimal aggregate utility value. Therefore the access-point is free to choose a large value for the V parameter of its virtual flow control algorithms, in order to maximize the aggregate utility.

3.3 Architectural considerations

As we mentioned, the assumption of infinite demand is generally not valid and is typically utilized in order to provide insights for the finite demand case. However, there are scenarios where the assumption of an infinitely backlogged transport layer storage reservoir can be considered realistic, such as in the case of the uploading of large files. Therefore, it is worth looking at the possible architectural implications of the proposed approaches.

The time-window observation and detection approach involves the maintenance of a detector for each station associated to the access-point. Given that short duration time windows are sufficient for the detection of a selfish station, the access-point could only run a single detection mechanism at a given time, monitoring the behavior of a single station. The station being monitored could change randomly time window after time window. In this way the processing burden for the access-point becomes $O(1)$ instead of $O(N)$. Even further, the detection algorithm could run only for a fraction of the network operation time.

The centralized implementation of virtual flow controllers for each node associated to the access-point moves a processing burden from the stations to the access-point. This burden scales with the number of stations as $O(N)$. This is not a particularly heavy load. With this approach each wireless station is now limited to the task of controlling its admission decisions so that the admission rate is not greater than its service rate, which it can no longer influence.

Comparing the two approaches, we note that the time window approach does not remove the flow control processing burden from the stations, but adds a smaller extra processing burden at the access-point compared to the virtual flow control approach. Furthermore, the time window detection approach can be extended to the case of finite demand, as we will see in the next chapter. An advantage of the virtual flow control approach is that the aggregate utility value can be arbitrarily close to the optimal value, without a congestion penalty.

3.4 A dynamic pricing approach for selfishness avoidance

In this thesis we mainly concentrate on scenaria where pricing is not an option or is not desirable by the network administrator. For the sake of completeness, however, we will briefly explain in this section how a dynamic pricing mechanism can be used to completely avoid selfish behaviors such as the ones described.

In subsection 2.5.1 we explained that the FLOW1 algorithm can be viewed as the rational admission decision of a user when it knows it has to pay a price of $U_i(t)/V$ per data unit (packet or bit) that it admits to its network layer. The payoff for an honest user , which is the utility minus the price is

$$payoff_{honest}(t) = g_i\big(R_i(t)\big) - \frac{U_{i,honest}(t)}{V} R_i(t) \tag{3.8}$$

so a rational user makes the decision

$$R_i(t) = r_i^* \tag{3.9}$$

where r_i^* satisfies

$$g_i'(r_i^*) = \frac{U_{i,honest}(t)}{V} \tag{3.10}$$

Now suppose that a user follows the strategy of declaring a greater queue backlog than the true value. The network controller, in this case the access-point, which is unaware of the true backlog value $U_{i,selfish}^{real}(t)$ will charge the user according to its declared backlog $U_{i,selfish}^{decl}(t) > U_{i,selfish}^{real}(t)$. The access-point is also unaware of the internal admission

decisions of a node, but it can charge the user per served packet, which is exactly equivalent, as the admission rate is the same with the service rate of a session (or else the respective queue would not be stable).

In this case, the payoff becomes

$$payoff_{selfish}(t) = g_i\big(R_i(t)\big) - \frac{U_{i,selfish}^{decl}(t)}{V} R_i(t)$$

This payoff is maximized for the value

$$R_i(t) = r_{i,selfish}^{*}$$

which satisfies

$$g_i'\big(r_{i,selfish}^{*}\big) = \frac{U_{i,selfish}^{decl}(t)}{V}$$

In the selfish misbehavior pattern we examined earlier, the selfish user utilizes its true backlog value $U_{i,selfish}^{real}(t)$ when running the flow control algorithm. Therefore the resulting admission decision at each timeslot is different than the optimal $r_{i,selfish}^{*}$ value. Consequently, we deduce that it is a suboptimal and dominated strategy from a payoff point of view.

Now let's see what happens in the network if a user adopts the false backlog declaration strategy, but makes flow control decisions based on the network price $U_{i,selfish}^{decl}(t)$, which is the price for which it is charged. Let's call this strategy SELFISH2 to distinguish it from the original which we call SELFISH1. We simulated this scenario for our example and the results are depicted in figure 3.4 below.

We observe that the allocation of throughputs is the same with the case where node 1 declares its backlog honestly. The only difference is in the real backlog of node 1, which is smaller than its respective value for honest declaration by 10 packets. Thus, in the case of SELFISH2 the false backlog value corresponds exactly to the true backlog value of the honest scenario. This is a normal result since the CLC1 algorithm (both the flow control and the scheduling part) only involves the value $U_{i,selfish}^{decl}(t)$, while $U_{i,selfish}^{real}(t)$ is invisible as far as the algorithm is concerned.

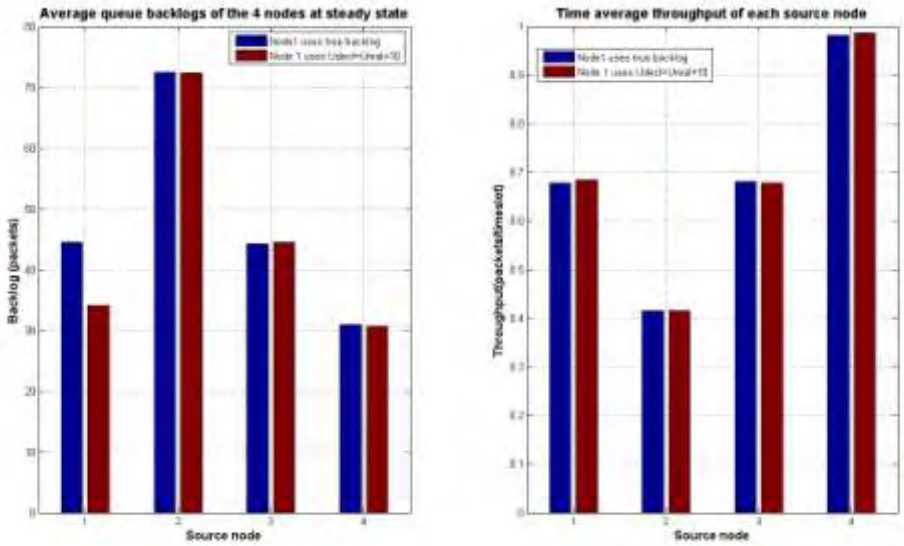

Figure 3.4: Comparison of average queue backlogs and throughputs in the cases where i) node 1 is honest and ii) node 1 declares Udecl=U+10 and also uses this value for flow control

Two are the points that must be made clear here. First, we see that, using this strategy, the selfish node gets the same throughput as before paying the same price as before, so it gains no benefit in terms of payoff. The original SELFISH1 strategy where the node uses its real backlog for flow control was proved to be dominated by SELFISH2, so it is also dominated by the case where the node acts honestly. Second, it is worth noting that with SELFISH2 the average congestion at the selfish node decreases without any throughput or payoff penalty. This is an interesting finding, suggesting that the flow control part of CLC1 could be modified in order to limit congestion, through the use of local virtual backlogs. We plan to investigate this behavior as part of our future work.

Finally, we note that the dynamic pricing strategy proposed charges a price for a packet whenever it is served, although the source node admitted this packet earlier, with possibly a different price. This is an inherent flaw of the strategy, but is not important in the case we examine, where at steady state the network prices oscillate within a limited range of values.

4. Selfish backlog declaration in the case of finite demand

4.1 Selfish motive and misbehavior pattern

Consider the same simple network as before, but drop the infinite demand assumption. This essentially means that the transport layer storage reservoirs are finitely backlogged. Assume that the network operates under the CLC2b algorithm we saw in chapter 2. Let the arrival process at each sender node $A_i(t)$ be a Poisson process with rate λ_i. We assume that the arrival rate vector lies outside the capacity region of the network. If the rate vector is inside the capacity region, no selfish motive exists, as the admission rates of all the nodes would be the same as their arrival rates.

The difference with the case of infinite demand is that now some of the nodes may demand less throughput than their "fair" admission rate under the infinite demand assumption. This extra available throughput is shared among the nodes whose admission rate is less than their arrival rate. We will refer to these nodes as nodes "starving" for throughput. We will use this term throughout this chapter.

Let's look at an example. Considering the same probability distributions for the attenuation coefficient processes, the same utility functions and the same V parameter as in our example in the infinite demand case, we now assume Poisson arrival processes with a rate vector $\lambda = (2, 2, 2, 0.6)$. The η parameter of CLC2b is taken equal to 0.5. Comparing these arrival rates with figure 3.2, showing the "fair" throughputs for the infinite demand scenario, we deduce that node 4 requests less throughput than its "fair" share, while the other three nodes are starving for throughput.

In figure 4.1 we can see the backlogs and the throughputs of the four stations in a scenario where all of them are honest. We observe that, as expected, the throughput of node 4 is equal to its arrival rate, while the extra throughput not claimed by node 4 is shared among the three other nodes in a fair way.

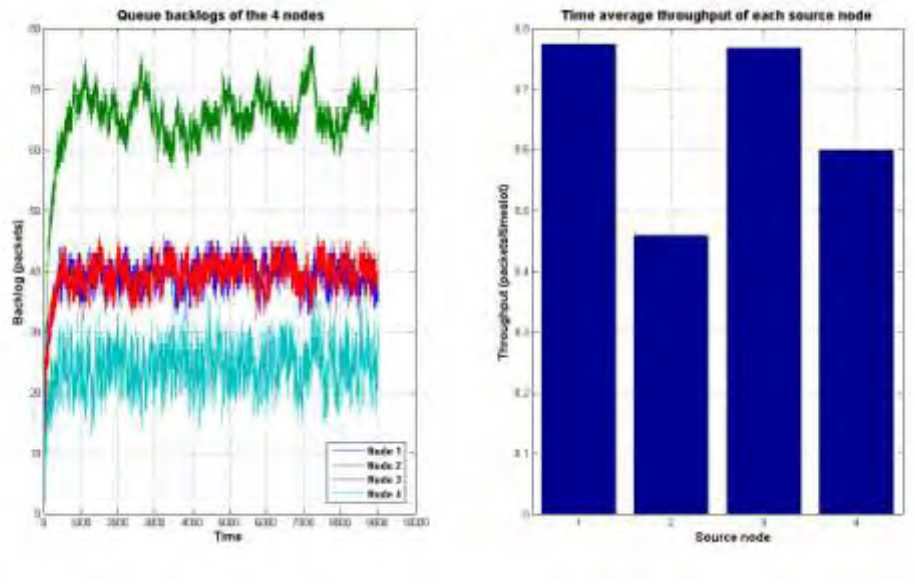

Figure 4.1: Backlog values and throughput of 4 honest stations in a single-hop network operating under the CLC2 algorithm. All nodes are starving for bandwidth except for node 4.

In figure 4.2 we can see the scenario in which node 1 is selfish and declares a backlog greater by 10 than its true value. It is clear that the selfish node achieves a throughput gain, along with a decrease in its average congestion, as in the CLC1 case we saw in chapter 3.

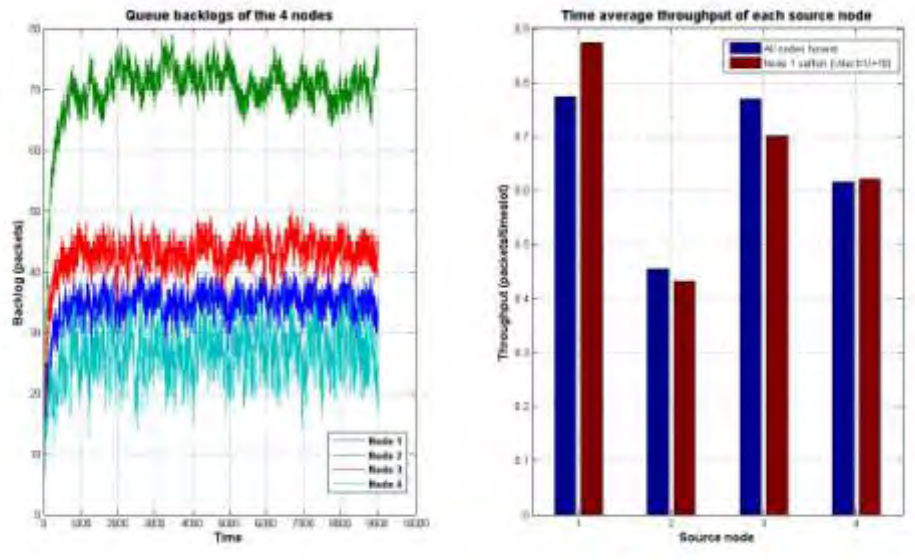

Figure 4.2: Network operating under CLC2b. Real backlog values and throughputs of nodes in the case where node 1 constantly declares a backlog greater than its true backlog (declared=real+10)

We observe from figure 4.2 that node 4 didn´t suffer a throughput decrease, unlike nodes 2 and 3. Through the simulations it was verified that honest non-starving nodes are indeed less

susceptible to throughput degradation than honest starving nodes. However, if the selfish node further increases its declared backlog, these nodes become starving too. For example, consider the case where the arrival rate vector is $\lambda = (2,0.3,0.6,0.8)$, that is, none of the honest nodes is normally starving for throughput. In figure 4.3 the throughput gain of selfish node 1 is depicted for various values of the additive parameter k, where $U_{1,decl}(t) = U_{1,real}(t) + k$. The throughput approaches (and ultimately reaches) the average capacity of the selfish user's link to the access-point, as the parameter k increases, as expected.

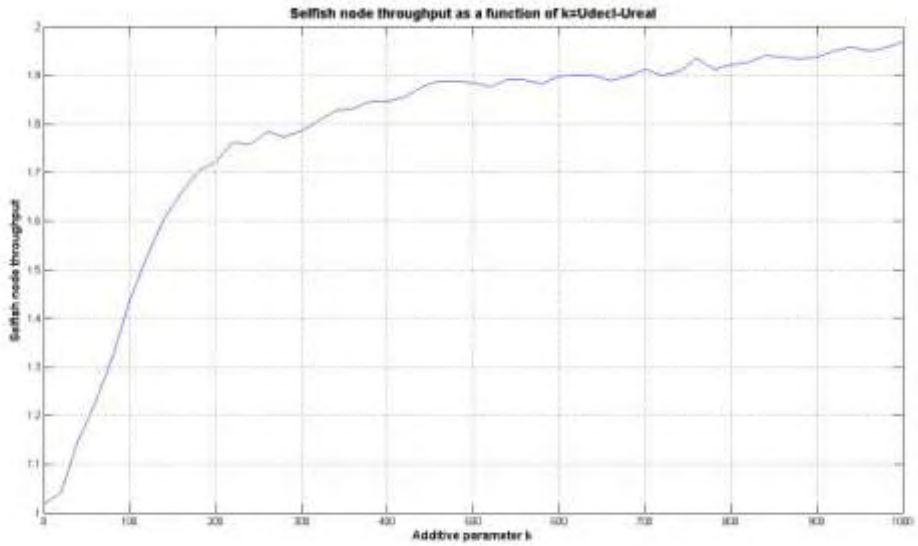

Figure 4.3: Throughput gain of selfish node as a function of the parameter k=Udecl-Ureal for a scenario with initially non-starving honest nodes

We observe, however, that for an approximately 10% increase of throughput, which in the example of figure 4.2 was achieved with $k = 10$, now a value of $k \cong 40$ is required. For k values approximately up to 20, the throughput gain of the selfish node is negligible. In order to understand this behavior, we examine the throughputs, the average congestion levels and the average transport layer reservoir backlogs of the honest nodes 2,3,4 for various values of k. We see this information in figure 4.4 below.

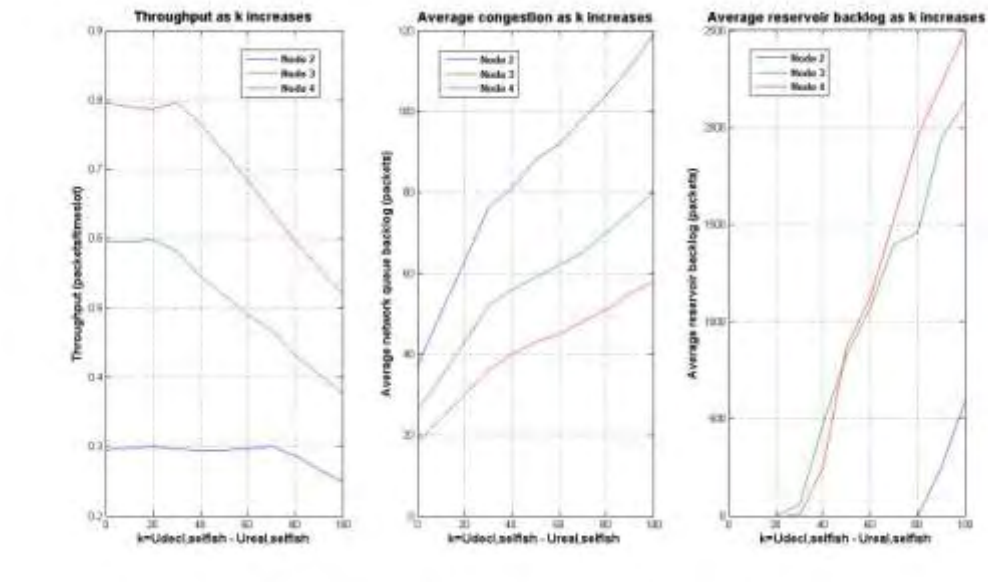

Figure 4.4: Evolution of performance of honest and normally non-throughput-starving nodes, as the parameter k=Udecl,selfish – Ureal,selfish increases

We observe that for small k values the only effect experienced by each of these nodes is an increase in average congestion. As the k parameter continues to increase, at some point each one of these nodes becomes starving, something we deduce from the increase in the backlog of the initially empty transport layer storage reservoirs of the nodes. At this point, which is different for every node, the throughput of the nodes starts decreasing. This throughput loss is translated into throughput gain for the selfish node. If there are nodes starving for throughput under normal operation, as was the case in figure 4.2, the selfish node claims a throughput portion from these nodes even for relatively small values of k.

4.2 Detection of selfish users

The approach of running a virtual flow control algorithm for each node, considered in chapter 3 for the case of infinite reservoir backlogs, cannot be applied in this case. The problem is that if the access-point is not aware of the arrival rates, it will not be able to run a virtual CLC2b algorithm for each of the associated users. If the access-point makes an infinite demand assumption for all of the users and runs a virtual CLC1 algorithm for each user, as before, the resulting rate allocation will be suboptimal in the sense that it allocates the channel to some users more than they need it and deprives it from other starving users. In this way, it deviates from the desired objective of maximizing the aggregate utility.

However, under the assumption that the access-point is aware of the utility functions of the users, we will see that it is relatively straightforward to design a detection algorithm for selfish users declaring false queue backlogs, analogous to the time window detection approach we saw in the previous chapter.

For this purpose, let's revisit the flow control part of the CLC2b cross-layer algorithm. In the single-hop one-commodity context we examine, it can be formulated as

$$R_i(t) = \begin{cases} min[L_i(t) + A_i(t), R_i^{max}], if \ \ \eta Y_i(t) > U_i(t) \\ 0 \qquad\qquad\qquad\qquad\quad , otherwise \end{cases} \tag{4.1}$$

where the virtual queue backlogs evolve according to

$$Y_i(t+1) = max[Y_i(t) - R_i(t), 0] + \gamma_i(t) \tag{4.2}$$

and

$$\gamma_i(t) = \gamma, \tag{4.3}$$

where γ satisfies

$$\max\ (\ Vg_i(\gamma) - \eta Y_i(t)\gamma\ ),\ \ subject\ to\ \ 0 \le \gamma \le R_i^{max} \tag{4.4}$$

Our goal is to examine if the strategy of comparing a time average of a function of the declared network queue backlog of a node with the service rate of this node can be applied in this case too, assuming once again that the access-point is aware of the utility functions of the nodes.

From (4.2) we deduce that at steady state

$$\frac{1}{T}\sum_{t=1}^{T}\gamma_i(t) = \frac{1}{T}\sum_{t=1}^{T}R_i(t) \tag{4.5}$$

for a not too small interval of T timeslots. As the time average value of the admission rate is the same as the time average of the service rate, which can in turn be observed by the access-point, we see that we can equivalently examine the relation of $\gamma_i(t)$ with the backlog value $U_i(t)$. At steady state the constraint in (4.4) can be dropped, so that $\gamma_i(t)$ is the solution to a non-constrained optimization problem. For a logarithmic utility function of the form $g_i(r) = w_i \log\,(1 + \beta r)$, the solution is given by

$$\gamma_i(t) = \frac{Vw_i}{\eta Y_i(t)} - \frac{1}{\beta} \tag{4.6}$$

The difference with the corresponding formula we encountered in the infinite backlog case is that we have $\eta Y_i(t)$ instead of $U_i(t)$.

In the case of a starving node, (4.1) can be written as

$$R_i(t) = \begin{cases} R_i^{max}, & if\ \eta Y_i(t) > U_i(t) \\ 0 & ,otherwise \end{cases} \tag{4.7}$$

In this case, the relation between $\eta Y_i(t)$ and $U_i(t)$ depends on the channel state probabilities, the arrival rates of the other users and on the choice of R_i^{max}. In figure 4.5 below we show the evolution of the quantity $U_i(t) - \eta Y_i(t)$ for the starving node 1 and the non-starving node 4, given the arrival rate distribution $\lambda = (2,0.3,0.6,0.8)$ we saw earlier. The constants R_i^{max} are set equal to the maximum capacities of the respective links.

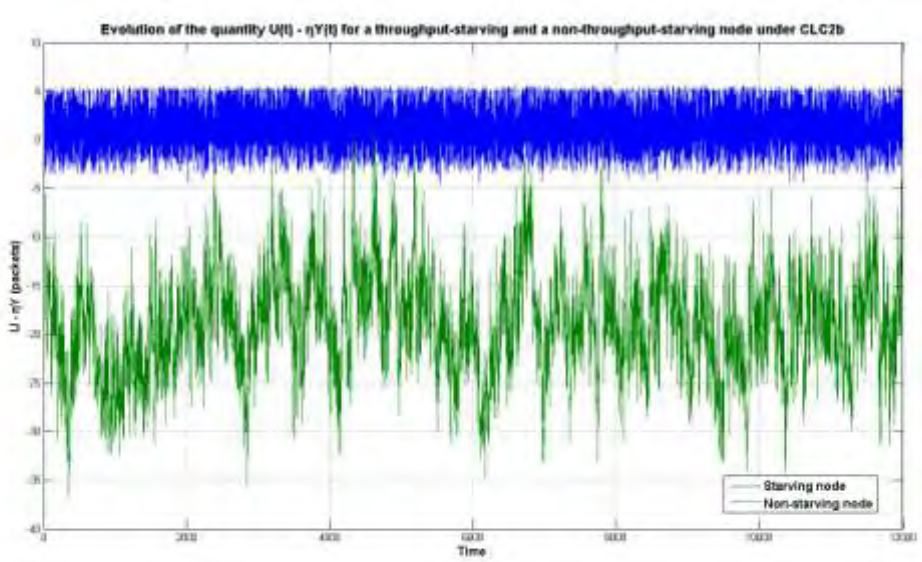

Figure 4.5: Evolution of the quantity U(t) – ηY(t) for a throughput-starving and a non-throughput-starving node under the CLC2b algorithm

We observe that for the non-starving node this quantity is constantly negative, therefore the condition $\eta Y_i(t) > U_i(t)$ is always valid for this node. From this inequality and from (4.6) we have

$$\gamma_i^{non-starving}(t) < \frac{Vw_i}{U_i^{non-starving}(t)} - \frac{1}{\beta} \tag{4.8}$$

Relationship (4.8) provides the access-point with a mechanism in order to distinguish starving from non-starving users.

Mechanism for the distinction between starving and non-starving users:

- Keep track of the quantities $\frac{Vw_i}{U_i^{non-starving}(t)} - \frac{1}{\beta}$ for each node and average over a (not too small) time window
- Calculate the actual throughput $\overline{\mu_i}$ of each node for the same time window
- If for some i, $\overline{\mu_i} < \overline{\frac{Vw_i}{U_i^{non-starving}(t)}} - \frac{1}{\beta}$, then the respective node is non-starving

From a throughput perspective, only a starving node can be potentially selfish. Therefore the above process excludes potential "suspects" as far as selfish misbehavior is concerned.

For a starving node the sign of the quantity $U_i(t) - \eta Y_i(t)$ changes with time, but it is positive on the average, i.e. for a starving node we have $E[U_i(t)] > E[\eta Y_i(t)]$. Thus from (4.6) we have

$$\overline{\mu_i} > \overline{\frac{Vw_i}{U_i^{starving}(t)}} - \frac{1}{\beta} \tag{4.9}$$

Now we need to see what happens with a selfish starving node. Since by definition we have

$$U_{i,decl}^{selfish} > U_{i,real}^{selfish} \tag{4.10}$$

and also

$$E\left[U_{i,real}^{selfish}(t)\right] > E[\eta Y_i(t)], \tag{4.11}$$

we deduce from (4.6) that the observed throughput will be greater than the throughput estimate, which will be based on the false declared backlog value:

$$\overline{\mu_i} > \overline{\frac{Vw_i}{U_{i,decl}^{selfish}(t)}} - \frac{1}{\beta} \tag{4.12}$$

Comparing (4.9) and (4.12), we note that the direction of the inequality itself is not a sufficient criterion for selfishness. The distance of the estimated value with the observed

value of the throughput – measured either in absolute value or in percentage – should be the criterion.

The distance between the estimate and the observed value in the case of an honest starving node depends on the channel state probability distribution among other parameters. This means that a given distance might be due to the normal deviation between $U_i(t)$ and $\eta Y_i(t)$ for some channel state and some arrival rate distribution, but the same distance might be interpreted as an effort of a selfish node to increase its throughput, in a different channel state and arrival rate distribution context. Thus the selection of a fixed global detection threshold over which a node is marked as selfish would inevitably incur a small percentage of false alarms. A strict formulation of this detection problem and its dependence on the various parameters will form part of our planned future work.

5. Conclusions and directions for future work

5.1 Concluding remarks

We examined a class of cross-layer network optimization algorithms which are based on information about the queue backlogs at each node. In these algorithms, queue backlogs are used for both centralized resource allocation decisions and distributed flow control decisions. Focusing on the simple scenario of an access-point-based single-hop network, we showed that a node has a clear selfish motive to declare false backlog values, misleading the network controller in order to increase its throughput.

We first examined the problem in the context of the infinite demand case, i.e. when the assumption can be made that there are always enough data to admit from the transport to the network layer of each node. For this scenario, we proposed two solutions, one involving accurate detection based on observations during a time window and another one which removes the flow control task from the stations and runs virtual flow controllers at the access-point. Both solutions require the knowledge of the users' utility functions and an increase of the access-point's processing burden. The latter solution has the additional advantage of approaching the optimal aggregate utility arbitrarily with no incurred congestion penalty for the network.

For the case where the utility functions are not known, the optimality of a simple pricing mechanism charging each node a price proportional to its declared backlog was verified. It was shown that an optimal strategy (in terms of payoff) for each node is to declare its backlog truthfully.

For the case of finite demand, we examined the effect of selfish misbehavior on two classes of honest nodes, nodes starving for throughput and nodes whose traffic demand is fully serviced by the network. It was found that, as the selfish node gradually increases the value of its declared backlog, up to a certain point the second class of nodes only experience an increase in average congestion, but no throughput degradation. Beyond that point, which is in general different for each node, they additionally suffer throughput degradation (i.e. become starving). On the contrary, nodes which are starving for throughput under normal network operation will suffer a throughput degradation even for a relatively small increase of a selfish user's declared backlog.

As far as detection in the case of finite demand is concerned, the choice of virtual flow-control algorithms at the access-point is not applicable, due to assumed absence of knowledge of the arrival rates. A detection scheme based on observations during a time window exists in this case too, but it is not as accurate as in the case of infinite demand and possibly a small amount of selfish throughput gain should be allowed in order to avoid false alarms. A more in depth inspection of the parameters affecting this tradeoff will form part of our future work.

5.2 Directions for future work

The extension of the proposed algorithms in a more general multi-hop context is the natural continuation of the results of this thesis. It is interesting to examine how false backlog declaration affects practical back-pressure inspired routing and scheduling algorithms, developed for multi-hop wireless ad-hoc networks.

The idea of using exclusively virtual backlogs in the cross-layer control algorithms with the significant increase in network congestion it involves is an interesting problem worth investigating further.

Finally, an interesting task would be to consider analogous selfish misbehavior strategies for delay-based back-pressure schemes and examine their effect on the network performance and if detection or pricing approaches for the queue-based case can be readily transferred to this case too.

References

[1] F. Kelly, “Charging and rate control for elastic traffic”, European Transactions on Telecommunications, vol. 8, pp. 33-37, 1997.

[2] A. Jalali, R. Padovani and R. Pankaj, “Data throughput of CDMA-HDR, a high-efficiency high-data-rate personal communications wireless system”, in Proc. IEEE Veh. Technol. Conf., 2000, pp. 1854-1858.

[3] L. Tassiulas and S. Sarkar, “Max-Min fair scheduling in wireless networks”, in Proceedings of Infocom, 2002.

[4] D. Bertsekas and R. Gallager, “Data Networks”, Prentice Hall, 1987.

[5] L. Massoulie and J. Roberts, “Bandwidth sharing: Objectives and algorithms”, in Proceedings of Infocom, New York, NY, March 1999.

[6] J. Mo and J. Walrand, “Fair end-to-end window-based congestion control”, in SPIE International Symposium, Boston, MA, 1998.

[7] E. H. Clarke, “Multipart pricing of public goods”, Public Choice, 1971.

[8] T. Groves, “Incentives in teams”, Econometrica, pp. 617-631, 1973.

[9] Srinivas Shakkottai and R. Srikant, "Network Optimization and Control," Foundations and Trends in Networking, NOW Publishers, 2007.

[10] R. Johari and J. N. Tsitsiklis, “Efficiency loss in a network resource allocation game”, Mathematics of Operations Research, vol. 29, pp. 407-435, March 2004.

[11] Ghazale Hosseinabadi and Nitin Vaidya, “Selfish Misbehavior in the Optimal Cross-Layered Rate Control of Wireless Networks”, Technical Report, March 2010.

[12] K. Graffi, P. Mogre, M. Hollick and R. Steinmetz, “Detection of colluding misbehaving nodes in mobile ad hoc and wireless mesh networks”, in Proceedings of GLOBECOM 2007, pp. 5097-5101.

[13] S. Radosavac, J. S. Baras, and I. Koutsopoulos, “A framework for MAC layer misbehavior detection in wireless networks”, Proceedings of ACM Workshop on Wireless Security (WiSe) 2005, Cologne, Germany.

[14] L. Georgiadis, M. J. Neely and L. Tassiulas, "Resource Allocation and Cross-Layer Control in Wireless Networks", Monograph, NOW publishers, 2006.

[15] M. J. Neely, E. Modiano and C. E. Rohrs, “Dynamic power allocation and routing for time varying wireless networks”, IEEE Journal on Selected Areas in Communications, Special Issue on Wireless Ad-Hoc Networks, vol. 23, No. 1, pp. 89-103, January 2005.

[16] L. Tassiulas and A. Ephremides, “Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks, “IEEE Trans. Autom. Control, vol. 37, no. 12, pp. 1936-1948, Dec. 1992.

[17] L. Tassiulas and A. Ephremides, “Dynamic server allocation to parallel queues with randomly varying connectivity”, IEEE Transactions on Information Theory, vol. 39, No. 2, pp. 466-478, 1993.

[18] M. Andrews, K. Kumaran, K. Ramanan, A. Stolyar, P. Whiting and R. Vijayakumar, “Providing quality of service over a shared wireless link”, IEEE Commun. Mag., vol. 39, pp. 150-153, Feb. 2001.

[19] M. J. Neely, E. Modiano and C. P. Li, “Fairness and optimal stochastic control for heterogeneous networks”, in Proceedings of IEEE Infocom, March 2005.