POLITECHNIKA WARSZAWSKA
WYDZIAŁ ELEKTRONIKI I TECHNIK INFORMACYJNYCH
INSTYTUT INFORMATYKI

Rok akademicki
2004/2005

PRACA DYPLOMOWA MAGISTERSKA

Szymon Jakubczak

Joint rate control and stochastic routing in packet networks

Zintegrowana kontrola przepustowości i rutowanie stochastyczne w sieciach pakietowych

Opiekun pracy
dr inż. Sławomir Kukliński

Ocena .....

.....
Podpis Przewodniczącego
Komisji Egzaminu Dyplomowego

Specjalność: Informatyka — Inżynieria Oprogramowania i Systemy Informacyjne
Data urodzenia: 9 marca 1983 r.
Rozpoczęcie studiów: 1 października 2000 r.

Życiorys

Urodziłem się 9 marca 1983 roku w Warszawie. Naukę w warszawskiej Szkole Podstawowej nr 82 rozpocząłem w wieku 6 lat i, ominąwszy jeszcze klasę trzecią, ukończyłem w roku 1996. Wykształcenie średnie zdobyłem w XIV LO im. Stanisława Staszica w Warszawie, w klasie o profilu matematycznym-experymentalnym. W tym czasie uzyskałem m.in. tytuły finalisty Olimpiady Fizycznej i Matematycznej. Przy wręczeniu świadectw maturalnych otrzymałem Nagrodę Burmistrza dla Najlepszego Absolwenta.

Sukcesy ze szkoły średniej umożliwiły mi rozpoczęcie studiów bez egzaminów wstępnych. Na Wydziale Elektroniki i Technik Informacyjnych wybrałem specjalność Inżynieria Oprogramowania i Systemy Informacyjne. Co semestr otrzymywałem stypendium naukowe, a za ostatnie dwa lata – Stypendium Ministra Edukacji Narodowej i Sportu za osiągnięcia w nauce. Cały czwarty rok studiów spędziłem na wymianie zagranicznej w University of Waterloo w Kanadzie, gdzie byłem także zatrudniony jako *research assistant* w Shoshin Lab.

.....
podpis studenta

EGZAMIN DYPLOMOWY

Złożył egzamin dyplomowy w dniu .....2005 r.
z wynikiem .....
Ogólny wynik studiów .....
Dodatkowe wnioski i uwagi Komisji .....
.....
.....

Zintegrowana kontrola przepustowości i rutowanie stochastyczne w sieciach pakietowych
Streszczenie

Pożądany przydział zasobów do przepływów w sieciach IP cechuje się zarówno równością wśród użytkowników jak i rozkładaniem obciążenia pomiędzy łączami. Alokacja jest typowo kontrolowana za pomocą dynamicznego rutowania i ograniczenia obciążenia w źródłach. Te dwa mechanizmy są tradycyjnie podzielone pomiędzy dwie warstwy stosu protokołów, i zazwyczaj nie mają konretnie określonej globalnej funkcji celu. W tej pracy proponujemy nowe podejście, które łączy w sobie rutowanie stochastyczne i kontrolę przepustowości. Podejście to ma dobrze określoną i dostosowywalną funkcję celu, która bierze pod uwagę zarówno użyteczność użytkowników związaną z przepustowością jak i koszty zasobów wynikające z obciążenia. Pokazujemy, że ten problem optymalizacyjny jest wypukły i może zostać zdekomponowany na dwa dobrze znane podproblemy, które pasują do podziału stosu protokołów na warstwy, ale używają wspólnej metryki kosztu przyrostowego. Dokonujemy przeglądu uprzednio zaproponowanych rozwiązań dla obu podproblemów i omawiamy problemy implementacyjne algorytmu łączonego. Podajemy także warunki wystarczające dla zbieżności w wyidealizowanym modelu. Opisujemy ze szczegółami jeden z możliwych wariantów i poddajemy go ewaluacji poprzez symulację numeryczną.
Słowa kluczowe: zintegrowana kontrola przepustowości i rutowanie, rutowanie stochastyczne, równość użyteczności, rozkładanie obciążenia oparte na kosztach

---

Joint rate control and stochastic routing in packet networks
Abstract

In IP networks, a desirable allocation of resources to user flows exhibits both fairness among users and load balancing across links. The allocation is typically controlled by means of dynamic routing and load limiting at sources. Traditionally, the two mechanisms are split between two layers of the protocol stack, and mostly lack any explicitly–defined global objective. We propose a new scheme that combines stochastic routing and rate control and attains a well-defined and tunable goal that accounts for both rate–related user utility and load–induced resource cost. We show that this optimization problem is convex and can be decomposed into two well–known problems that conform to the layer–separation, but employ a common marginal cost metric. We survey multiple solutions previously proposed for each of the two sub-problems. Further, we discuss the implementation issues of a joint algorithm and state sufficient conditions for convergence in an idealized model. We describe in detail one possible variant and evaluate it through numerical simulations.
Keywords: joint routing and rate control, stochastic routing, utility fairness, cost–based load balancing, global performance optimization

Contents

Chapter 1

Introduction

The IP networks that connect our computers are highly dynamic environments. There is a persistent struggle to balance the varied needs of many users with the limitations of resources. The dynamics are already present at both sides. The user demand is bursty at a wide spectrum of time scales, and the capacity of resources (e.g. data links) has gotten variable with the rise of lossy wireless media. As the user needs are ever-increasing, and just over-dimensioning the network capacity does not suffice, we shall need dynamic, adaptive algorithms to resolve the users' competition for resources.

With the evolution of IP networks, the primary concern switched from rough connectivity to the specifics of the needs of users. A typical user is elastic, i.e. able to adjust its load in response to resource availability. Fairness or utility are some proposed measures of satisfaction of such users from *quantity* of service. At the same time, an important network-centric issue is how the load is balanced across resources. Load balancing can improve *quality* of service and/or reduce costs. The focus of this work is how to improve the satisfaction of users through adaptive routing and rate control in dynamic networks.

1.1 Motivation

While the history of both adaptive routing and load control is not much younger than that of the pre-Internet ARPANET project, the solutions proposed to-date conform to the layer separation paradigm as described below.

The traditional reference model for data networks is a stack of layers as shown in Fig. 1.1. The Internet Protocol stack roughly follows this model [SK91]. Two adjoining layers are the focus of this work:

network layer - responsible for routing, i.e. establishing routes and forwarding packets along them.

transport layer - responsible for splitting streams into packets, flow control (buffer protection),

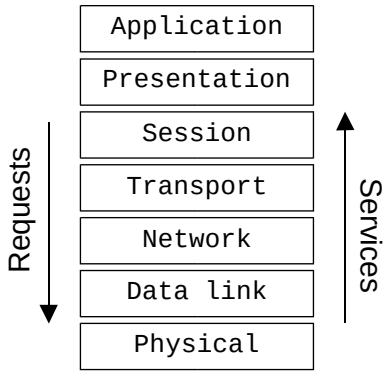

Figure 1.1: The ISO Open Systems Interconnection (OSI) reference model. Each layer performs services in response to requests incoming from the layer directly above it. The inner workings of a layer are concealed within and unavailable to others.

and also reliable delivery.

Adaptive mechanisms installed in the network layer, are automatically rerouting the traffic based on analysis of current network conditions (traffic patterns, resource availability), as to reduce costs or improve quality of service (e.g. end-to-end latency). But a packet network is also a queuing system and when the demand surpasses its capacity, a congestion occurs, i.e. the packet delay rises, and packet delivery ratio may suffer due to limits on queue space. In the extreme case, a congestion collapse (severe inefficiency) and undesired unfairness may occur. In order to avoid the collapse, mechanisms for congestion control or avoidance are incorporated into the protocol stack. Since they are generally of the same nature as flow control, they are built into the transport layer, e.g. TCP. Therefore the network layer is responsible for the allocation of resources (links) to users by selecting routes, while the transport layer is responsible for the sharing of allocated resources by limiting the load of users. As such, the mechanisms that control the load in response to congestion (resource availability) are not allowed to change the routes that were established in the network layer, but only regulate traffic along the preselected paths. The network layer does not control the users, and the transport layer does not control the routes.

Commonly, to deal with dynamics present in networks, the current solutions comply with this layer separation offering adaptive routing and congestion control or avoidance algorithms. While this decomposition is, generally, advantageous, it lacks a well-defined global objective. Also, the interactions between the two mechanisms were not well studied until very recently [AA03, Voi04, KV05]. A different approach is to let the user control its allocation of resources in a *selfish* manner. This however does not fit the IP datagram routing paradigm, and is typically realized in overlay layers [Col98, ABKM01].

The focus of this thesis is resource sharing in unstructured packet networks aiming for both user utility and resource cost, using both aspects of share allocation: routing and load control. We are looking for a tunable, adaptive mechanism that deals with the dynamics of both users' needs and available resources. Most of the inspiration comes from the works of Kelly and Voice [KV05, Voi04].

1.2 Contribution

In this work, we introduce a new scheme for packet data networks: a joint rate control and stochastic routing algorithm. The scheme is a combination of well-studied Wardrop hop-by-hop stochastic routing (e.g. [Gal77, SDC97, GK97, VGLA99, CD98, BK03, XQYZ04, RK04]) with source rate control (e.g. [KMT98, KST01, LA02, LBS03]). Both mechanisms have been shown to be beneficial for the performance from both the viewpoints of users (throughput and delay) and resources (utilization and load balancing). However, in our solution, the two mechanisms operate jointly by sharing a common cost metric based on virtual resource price as discussed by Kelly and Voice in [KV05] and Voice in [Voi04]. We show that the equilibrium of this algorithm is globally optimal according to an intuitive and tunable goal function that is defined in terms of both user utility and resource cost.

1.3 Outline

The rest of this thesis is organized as follows.

In Chapter 2, we provide the relevant background information to facilitate the reading of the thesis. In particular, we introduce the terminology, overview the model of network services, and discuss the objectives of resource sharing from both user and network perspective. We have given increased attention to the notion fairness driven with concave utility functions and the paradigm of selfish flow routing and equilibria. Also in Chapter 2, we briefly introduce the mechanisms employed in the proposed scheme: dynamic proportional routing and rate control.

Chapter 3 contains the definition of the system model and objective, together with interpretation and theoretical analysis. Our system model basically assumes rate control at flow sources and flow–conserving routing. We consider per–flow and proportional routing policies and show their equivalency with respect to the chosen objective. Our assumptions on routing are in contrast to the most relevant work on joint routing and rate control scheme proposed by Kelly and Voice [KV05] that uses preselected routes. In our analysis, we show that the objective is a convex optimization problem, how it can be decomposed into two well-known sub-problems of minimum cost proportional routing and maximum profit rate allocation, and how a joint scheme could attain it. This decomposition conforms to the traditional layer–separation principle.

In Chapter 4, we describe several solutions to each of the two sub-problems that have been proposed by other researchers. Only a small portion is surveyed. We focus on such methods that could be employed in our algorithm and are fairly representative to various approaches.

The implementation details of the joint algorithm are discussed in Chapter 5. We propose that its two components would use a common marginal cost estimate, and consider possible ways to obtain and distribute it among nodes. For the theoretical analysis we introduce a continuous model that neglects both propagation and update delays and assumes that the adjustments are performed continuously. We state the sufficient conditions for such adjustment algorithms that

yield improvement in the overall performance at every instant. One variant that shall be used in our evaluation is described in detail and proven to satisfy these requirements. Finally, we discuss the issues that arise in a more realistic model that includes propagation and update delays.

We present the results of our evaluation through numerical simulation in Chapter 6. In the simulation we neglect propagation delays to match the model used in Chapter 5. The focus of the evaluation is to illustrate how the algorithm converges using either *phased* or *concurrent* interleaving, and the properties of the resulting allocation. Specifically, we demonstrate how routing adapts to increasing load, how fairness and load balancing is achieved and how the network can be driven at desired saturation level using global utility weighing. We also discuss convergence issues that arise in the discrete-time simulation.

Chapter 7 summarizes and concludes this work. We also outline open problems and future research directions.

Chapter 2

Background

In this chapter, we provide the preliminaries. It is intended as a less formal introduction to the concepts discussed later. It describes basic mathematical models of a network service, the principles of resource sharing and introduces the mechanisms employed to attain them.

2.1 Modelling

2.1.1 Network model

A network comprises hosts, routers and communication links. Hence, it is commonly described using directed graphs, where vertices represent the nodes and directed edges represent the links, as illustrated in Fig. 2.1. Typically, there can be at most one link for an ordered pair of nodes, but this restriction is usually easily relaxed. The practical difference between hosts and routers is that the hosts host users that inject their load into the network, while the routers are solely responsible for directing the load through links to the target host. Thus, a simplified model contains only routers and links, and places the users in the routers that their hosts are connected to.

Each user sends data from its designated *source* to its *destination* host. Different users may have different sources and destinations. Therefore the network is a *multicommodity* resource, as opposed to a *single commodity* resource to which all users have exactly the same access through common source and destination nodes.

In order to transmit data over the network, it needs to be split into small chunks, called *packets*. The packets traverse the network according to the *store and forward* paradigm: a packet needs to be fully received by the router and then transmitted (forwarded) further. The links are characterized by propagation delay or latency (i.e. time for a single bit to traverse it), and bandwidth (i.e. number of bits that can be transmitted in unit time) which is the measure of its capacity. Therefore, the routers that forward the packets need to manage a queue, in which they store the packets before the specific outgoing link is available.

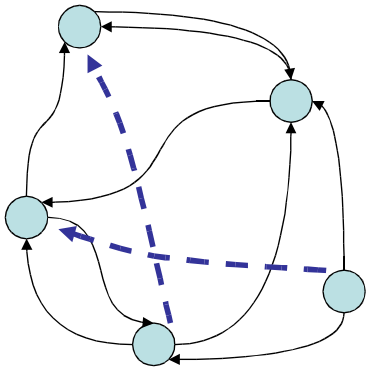

Figure 2.1: Modelling network as a graph. Solid arrows represent links. Dashed arrows represent users sending data across the network between their specific source and destination.

In this *queuing* model, when a router persistently receives more packets than it can forward, a congestion collapse occurs: its queue grows until it overflows, and then the packets are being dropped. In this work, we will assume that the network is never overloaded, though we do account for the congestion, i.e. increase in queuing delay when the load increases.

2.1.2 Service model

The network is providing the users with service, i.e. shipping their packets over from source to destination hosts. Naturally, it is a discrete process consisting of such events as packet creation (at source), packet relay (between routers), and packet consumption (at destination). However, as the number of packets is large, such a detailed view is not usually necessary. Instead, more usable aggregate descriptions of packet *flows* rather than the individual packets have been proposed.

A *flow* is comprised of the packets that belong to a specific user, and are transmitted over the network from its designated source to destination node. A flow is approximated as a continuous fluid and can be described by the following statistical characteristics:

rate - the average number of packets (or amount of data) that is created within unit of time

delivery ratio - the fraction of the packets created at source that get delivered to the destination. Sometimes replaced with *loss ratio* = 1 - *delivery ratio*

throughput - the rate of packets that get successfully delivered. In stationary systems, throughput equals the product of rate and delivery ratio

delay - (or latency) the time for a packet to reach the destination

delay or **throughput jitter** - the variation or range of delay or throughput

Our assumption that the network is not overloaded implies a flow-conservation law: *the total rate of flows coming out equals the total rate of flows coming in plus/minus the flows injected/consumed at the node.*

2.2 Resource sharing

By design, the service in IP networks is "'best-effort"' only, i.e. all packets are treated alike, and no guarantees about the service (e.g. reliability) are made. *Elastic* users, defined later in this section, are well-prepared for this environment, and will adjust their load to match the resources available when necessary. Still, the objective is to maximize their satisfaction, minimize the incurred cost, and yet remain fair in the allocation. In this section, we describe the utility and cost objectives as well as the notions of fairness and selfishness of a resource allocation.

2.2.1 Objectives

There are several aspects of resource sharing in data networks. We focus on the user perspective and treat any system-oriented objectives (e.g. load balancing) as purposed to improve the provided services (e.g. reliability, delivery ratio, etc.). The user is interested in a number of characteristics of the service it receives:

delivery ratio - Ideally, the network would be lossless, and the delivery ratio would be 100%. Losing packets is clearly undesirable for users that require reliable data transfer. When packets get lost, transport-layer protocols need to take counter-actions and re-issue them.

throughput - Throughput determines the time needed to transfer finite amounts of data, e.g. a document, and also the bandwidth available for online streaming applications.

delay - In the Internet setting, this characteristic is crucial for interactive applications.

jitter - For many applications the unpredictability of delay is more undesirable than the delay itself. Streaming applications are a particular case, in which fixed delay would allow uninterrupted consumption of the incoming data, to the extent of bufferless reception.

Throughput is distinguished as the *quantity* opposed to the *quality* of service. In this work, throughput will be given precedence followed by delay. In the wireless domain, we should most likely focus also on delivery ratio that firmly depends on the quality of wireless links.

With respect to these characteristics, the requirements may be more or less strict. In their research, Roberts and Massoulie make a distinction between *elastic* and *non-elastic* applications [RM98]. An elastic user is able to utilize surplus bandwidth and, at the same time, can benefit from even the smallest amounts of service. The majority of Internet traffic comprises of elastic documents (web requests, file transfers). But the state-of-the-art streaming applications are rarely elastic - their satisfaction is a step function of throughput, i.e. a specific threshold

exists. Later in this chapter we describe the elasticity of user with a *utility* function which describes its satisfaction regarding the throughput it receives.

In human world, to deal with the possible variety of user needs and resource costs, we have developed the notions of money and prices. Inspired directly by theory of economy, a concept of virtual service pricing is gaining increased interest amongst Internet researchers [GK99,KMBL99, DHK04]. We distinguish two variants of the concept:

- each resource is characterized by a unit price. The price and load determine the cost. The goal is to minimize the total cost of service. Note that this goal alone could stabilize at the trivial state of zero service. Hence, it is rather used in the pure routing problem, where the user demand is fixed.

- each user offers to "'pay"' a unit price per service. The goal is to maximize the total income. Note that this price may be virtual, or in another scenario, used to determine the optimal rate of the user. We later relate to such a scheme that was proposed by Kelly *et al.* [KMT98].

In our work we combine both aspects of pricing and aim to maximize the profit as the difference between income and costs. However, as researchers before us [Kel97, KMT98, LA02] we use pricing only as a facility, no payments are actually made. We are especially interested in the particular way of pricing the resources in proportion to the latency incurred to packets that use them. Thus, the profit of a user could be defined as the difference between its income due to achieved throughput and its costs due to the perceived latency.

We have purposefully omitted the notion Quality of Service, because we focus our work around elastic applications for which detailed QoS specifications make little sense. Further, QoS mechanisms are typically founded on resource reservation and call admission control, which rejects new requests, when they could violate the QoS guarantees already made. But the inherent property of elastic applications is that they are ready to adjust in both directions, also to sacrifice some amount of their service, when the incurred costs are too high.

2.2.2 Fairness

Suppose that fairness amongst users is our primary concern. What is it that constitutes fairness? Intuitively, an allocation of resources is fair, if it does not "'hurt"' any user. Some of the users' access to the network may be hindered, and such users may be starved out of service by others if global performance (e.g. resource utilization) was the only factor that counted. Specifically, when we introduce a unit resource-price of unit service, then to maximize the total amount of service, we would definitely prefer those users that are cheaper to serve, i.e. have smaller prices per unit.

Fairness criteria has recently gained increased interest with the evolution of single- and multi-hop wireless networks. Wireless links are unlike wired in that they are subject to both external

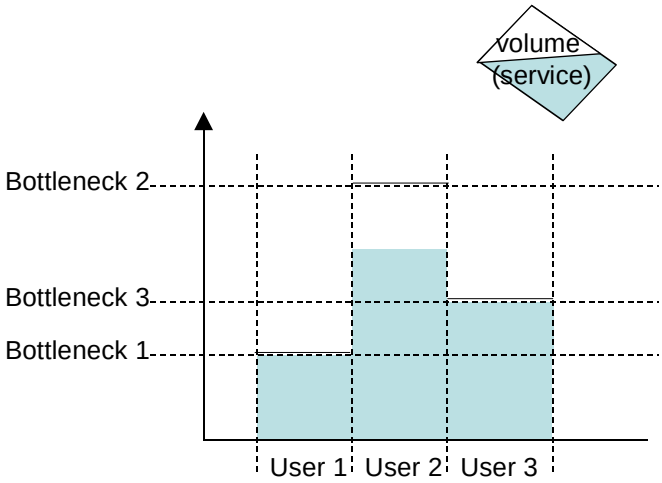

Figure 2.2: Water-filling algorithm for max-min fairness. Bottleneck i limits the volume allowed for user i. It is impossible (laws of hydrodynamics) for a user to receive less than its limit when the water-line has exceeded it.

and internal interference. The effects are such that flows virtually share common resources even if they do not traverse common nodes. Moreover, a flow suffers from self-interference, since the transmissions over subsequent hops on a path interfere with each other. See [LBB04, HJB04, GSK04] for reference.

Max-min fairness

One common approach to prevent user starvation is *max-min fairness* which in its principle aims to maximize the service of the least serviced user. In its widely used definition, the criterion is stated as: *the gain of no user should come at the expense of a less serviced user.* More formally, an allocation of services to users $\mathbf{x}$ is max-min fair if and only if for any other allocation $\mathbf{x}^*$, an improvement in one user's service $x_s^* > x_s$ implies there exists a less serviced user $x_t < x_s$ which is hurt by the improvement, i.e. $x_t^* < x_t$. An equivalent definition states that, if users are constrained by resource capacities, then for each user there exists a bottleneck (the actually constraining resource) in which it is the best serviced user. Max-min fairness has been well-studied in the areas of both wired and wireless networks [KVR95, Mar03, RB03, RB02, TS02]. The common approach to finding max-min fair allocations is a water-filling algorithm that increases the service of all users until each hits its bottleneck, as illustrated in Fig. 2.2.

Proportional fairness

Kelly in [Kel97] argues that max-min fairness is not truly appropriate for the bandwidth sharing problem. Instead, he advocates for *proportional fairness*, which has its roots in economy and game theory and is directly connected to the maximum aggregate utility problem.

In contrast to max-min fairness, proportional fairness allows to hurt a less serviced user's share provided that the improvement is at least proportionally better. More formally, an allo-

cation of services to users $\mathbf{x}$ is proportionally fair if and only if for any other allocation $\mathbf{x}^*$:

$$\sum_s \frac{x_s^* - x_s}{x_s} \leq 0$$

Kelly notes that this criterion restated as: $\sum_s dx_s/x_s \leq 0$ is equivalent to the optimality condition for:

$$\max \sum_s \log x_s$$

Utility fairness

Inspired by the aforementioned results by Kelly on proportional fairness, researchers turned to the notion of utility, which allows to define collective objectives of elastic users and still accounts for fairness. In general, the criterion of utility fairness is to maximize the sum of individual utilities:

$$\max \sum_s U_s(x_s)$$

A sensible utility function for a user is a strictly increasing and strictly *concave* function. The smaller the current share, the more worth is its increase, and the better is the improvement in user satisfaction. Doubly differentiable concave functions have the first derivative increasing and the second derivative positive. When the concavity is *strict*, then the derivatives are, respectively, strictly increasing and strictly positive. The strict concavity is essential for fairness. In Fig. 2.3, we illustrate how it improves the balance of the allocation of user shares on a very simple example. In general, for a single-commodity resource and multiple users of same concave utility functions, the optimal allocation is absolutely fair (i.e. all equal shares). In the very same manner, if we use strictly *convex* resource cost functions (of total resource load) to evaluate the performance, then a minimization problem of total cost would balance the load over resources.

Although the user might be interested in various aspects of service as outlined in the previous section, the characteristic that is typically subject to utility evaluation is throughput or flow rate. A selection of utility functions has been proposed:

$U(x) = \log x, \quad U'(x) = 1/x$ The logarithmic function attains the proportional fairness as proposed by Kelly in [Kel97], and is also shown to be the result of the popular *additive-increase-multiplicative-decrease* scheme also by Kelly in [KMT98]. We describe this scheme in more detail in section 2.3.2.

$U(x) = -1/x, \quad U'(x) = 1/x^2$ Massoulie and Roberts propose in [MR99] to use the reciprocal of throughput interpreted as *potential delay*. If all packets were equal sized, then this utility is proportional to the packet delay.

$U(x) = x^{1-\alpha}/(1-\alpha), \quad U'(x) = x^{-\alpha}$ A generalization for a class of utility functions was proposed by Kelly in [KMT98]. Referred to as α-utility, the class encompasses both pro-

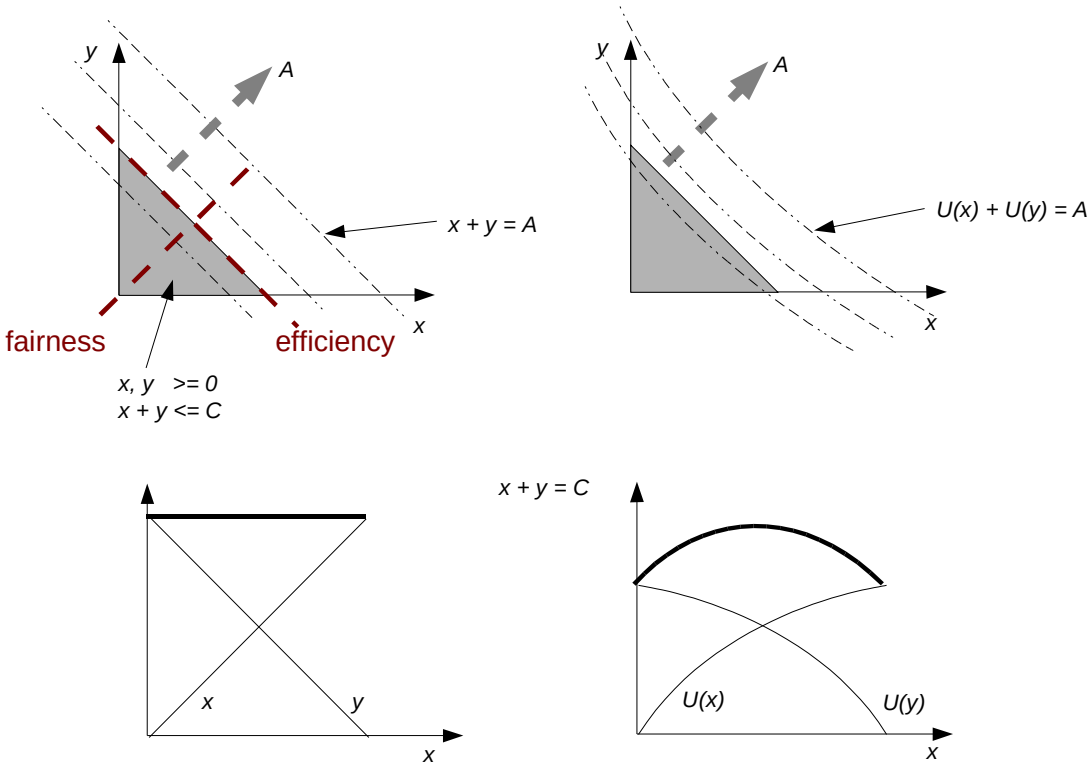

Figure 2.3: A comparison of total utilization and *strictly* concave utilities as performance measures. User shares x and y are limited by resource capacity C. The charts on the top indicate the gray region of feasible allocations. The lines of perfect efficiency (resource utilization) and perfect fairness (equality) are plotted. The family of dashed lines across the region are equal-performance lines. The charts below show how the performance varies along the perfect efficiency line $x + y = C$. Total utilization does not "care" about fairness.

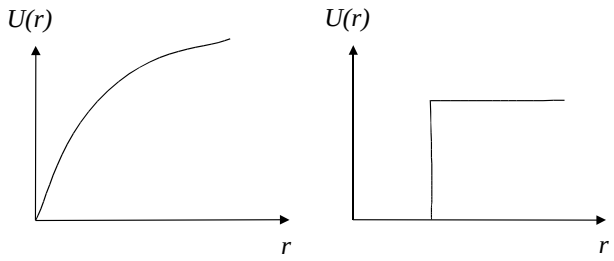

Figure 2.4: A sample comparison of utility functions of elastic (left) and non-elastic (right) users. Non-elastic users may have a strict threshold of service below which the level of satisfaction is none, but above which the satisfaction is full.

portional ($\alpha \to 1$) and potential delay ($\alpha = 2$) fairness as well as the max-min fairness ($\alpha \to \infty$).

Each of the schemes allows for additional differentiation between users by per-user weighing. We point to the derivative of the utility function, as it is more informative. It is not the value of global optimum that is of interest to us, but rather the optimal allocation of resources to users. Note, that none of the classes allows an allocation to "'starve"' any user, because $\lim_{x \to 0} U(x) = -\infty$. It should be emphasized that non-elastic applications do not respect any concave utility. Figure 2.4 illustrates the contrast between the utility functions of elastic and non-elastic users. Also, many applications are sensitive to more than just the throughput received.

2.2.3 Selfishness and social welfare

Let us suppose that we can measure each user's *profit* which increases with the quantity of service received and decreases with the costs incurred by the service on resources. The two poles on the scale of possible ways to approach such system are:

social way - when the users cooperate to maximize the total or average (*per-capita*) profit, also called *social welfare*

selfish way - when each user seeks to maximize its individual profit

Users' behaviour is *selfish* or *non-cooperative* when they are not interested in the system-wide criteria but only their own performance. The discrepancy between "system-optimality" and "user-optimality" was evidently first noticed by Pigou [Pig20] and we describe it later in this subsection.

The notion of selfishness originates from game theory [Mye91] and has been studied thoroughly in the past. Recently, it has also received increased attention in network research. The classic hop-by-hop routing paradigm (in which the node only decides on the outgoing link, not on the whole path) does not support selfishness *per se*, but it has been shown to be inefficient from the user's perspective [Col98, TGSE01]. In response, new end-to-end routing paradigms (e.g.

source routing [CCS96, ELR+96, JMB01], overlay routing [Col98, ABKM01, CFSK04]) emerged allowing the user (or host) to pick the path for its flow.

In this thesis, we do not resort to selfish routing directly, but we do base on the notion of *equilibrium*, i.e. the stable state of a selfishly driven system.

Equilibrium

Consider a sharing problem from a different, yet quite related domain of road traffic. Which route, given origin and destination, would one usually take? Reasonably, the one that gives shortest trip time. Note that even if the decisions of other road users are known, the strategy remains to optimize own goal unilaterally, i.e. assuming that others will not adjust their decisions to our favor. Naturally, the trip time of a route increases with the amount of traffic using it. Obviously, this system reaches its equilibrium (i.e. stable state) when no user is willing to change his route, because it would increase his trip time.

Let us formalize this notion in a more general system. A *flow game* is a multicommodity network shared by a set of users. Each user selects a route for its flow. The cost of the route is the sum of the costs of the links it uses (traverses over), and the cost of a link is an increasing function of load, i.e. number of flows using the link. Two definitions of equilibria have been defined for non-cooperative flow games:

Nash equilibrium in which no flow (seen as an agent playing the game) has the incentive to unilaterally adjust its route. In other words, the flow evaluates its reduction in cost assuming that no other flow would be rerouted.

Wardrop equilibrium in which routes used by all flows having common source and destination have equal cost, and there exists no route of smaller cost that could be used by the flows.

The Nash definition is used for small number of atomic users. When the number of users increases and the impact of each one's decision becomes negligible (zero-weight), the users form an atomless continuum and the Nash equilibrium converges to Wardrop equilibrium as shown by Haurie [HM85]. There is a clear intuition behind Wardrop's definition: if, for a given source-destination pair, there existed a route of smallest cost, the envious users would reroute their flows until the costs of all routes in use get equalized.

In a continuous variant of the model there is a fixed flow demand per each source-destination pair which can be split in any fraction over all routes available. In such a case, the social welfare is the average cost per flow which is the average of costs of routes weighed by the amount (i.e. rate) of flow using each route.

Consequences

Selfish behaviour comes at a price, i.e. decrease in global performance goal (or *social welfare*). Consider the common example by Pigou [Pig20] of a single commodity (one source-destination

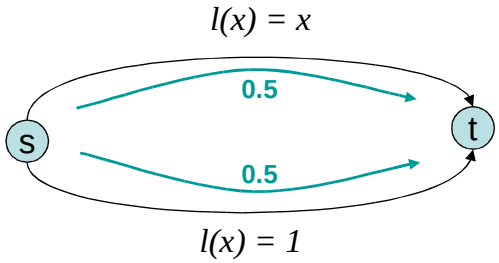

Figure 2.5: Pigou's example of loss in social welfare due to selfish routing. Total flow is 1 and the optimal allocation is shown. Under selfish routing, the flows from s to t that take the bottom link are envious as long as the load on the top link is less than 1, because the latency over there is smaller. Effectively, all flows take the top link. The loss in social welfare is the ratio of average latency of selfish routing to optimal, $(1 * 1)/(0.5 * 0.5 + 0.5 * 1) = 4/3$.

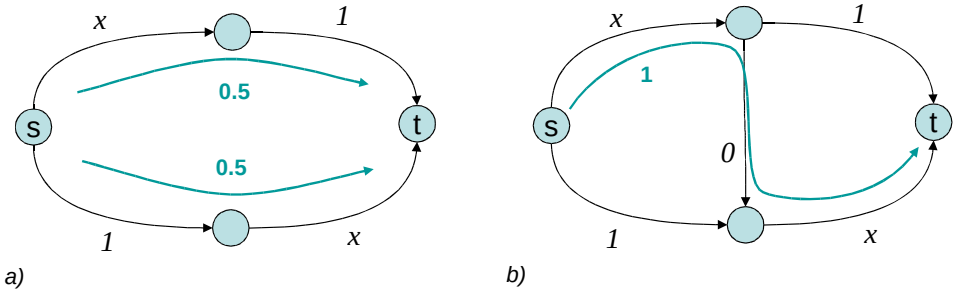

Figure 2.6: Braess paradox. The social welfare of the equilibrium in the original network is 3/2 (a). After adding a new zero-cost link, the cheapest route traverses the new link and all flows switch to it. The social welfare of the *augmented* network is 2 (b). The price of anarchy is 4/3, the worst-case for linear latency functions.

pair) network with two links characterized by different load-latency functions with flows willing to minimize their latency, as depicted on Fig. 2.5.

This *price of anarchy* in general networks have been shown by Roughgarden and Tardos in [RT00] to be unbounded. Yet, in the same work it was also shown that, if the load-latency functions are linear, the price of anarchy is no more than 4/3. In other work, Qiu *et al.* [QYZS03] substantially reduces this gap with simulations in a more realistic, restricted overlay setting.

The Pareto-inefficiency of the equilibria of non-cooperative flow games also manifests itself as a potential degradation of social welfare when available resources are increased, commonly referred to as the Braess paradox [Bra68]. An illustration taken from [Rou02] is given in Fig. 2.6.

A property of non-cooperative flow games that is crucial to this thesis was shown by Wardrop in [War52] for road traffic, generalized by Beckmann *et al.* in [BMW56], and later applied to selfish routing by Roughgarden in [Rou02]. It states that the globally-optimal flow (i.e. maximizing social welfare) is a flow at Nash/Wardrop equilibrium with respect to a different set of edge latency functions. Specifically, given a network with latency functions l that are

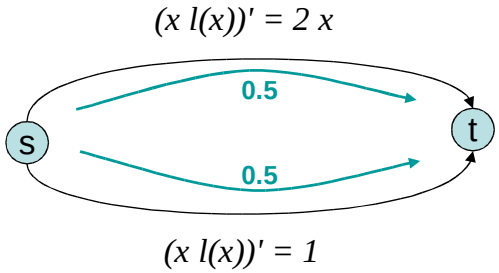

Figure 2.7: Marginal latencies used in the example by Pigou on Fig. 2.5. The allocation shown is at Wardop equilibrium, all paths have the same latency. It matches the socially optimal allocation of the original flow game.

differentiable and such that $x\,l_e(x)$ is convex [1] for each edge e, the optimal flow is a Nash flow for the same network with marginal latency functions $l_e^*(x) = (x\,l_e(x))'$. Hence, a switch from user-optimality to network-optimality can be made by a simple replacement of cost function. Fig. 2.7 illustrates how this property works in Pigou's example from Fig. 2.5. The globally-optimal equilibrium that results from selfish routing using marginal latency is often referred to as Wardrop's second equilibrium. In Chapter 4 on related work we refer to past research on globally optimal routing that makes use of this fact.

2.3 Mechanisms

In this section, we introduce the mechanisms that shall be employed in our algorithm. These are adaptive methods for routing and load control.

2.3.1 Adaptive routing

In the earliest computer networks, the primary task of routing was to ensure at least crude connectivity. With their evolution, the secondary concern on quality of service increased. The task is not only to provide *a* path but also a *good one*. This shift of focus was due to the fact that the dynamics of network topology (i.e. links and nodes going up and down) went down with the development of more reliable communication systems. A routing algorithm is adaptive if it responds to the dynamics of load and capacity. The class of networks they are designed to operate in is often called *quasi-static*.

While most of the routing protocols implemented for IP networks (e.g. shortest hop-count) are dynamic in the sense that they do react to the changes in network topology (i.e. connectivity), the dynamics we are concerned with are less severe. Yet, the varying load of users can lead to congestion even in otherwise static networks. There still exist, however, such network settings as wireless mobile *ad hoc* networks (MANET), in which it is difficult to satisfy the primary task, i.e. provide *any* path.

[1] Note, that if $l(x)$ is increasing, then $x\,l(x)$ is convex.

Broadly, traffic-aware adaptive routing algorithms perform the following steps:

1. measure current state of the network, such us traffic load or residual link capacity
2. compute routes
3. operate the network using the new routes

Basically, the cycle is repeated ad infinitum. In-between measurement and computation necessary information needs to be communicated between the participants. In this work, we resort to a particular class of load-distributing algorithms: hop-by-hop stochastic routing. The two essential components are dynamic link and route metrics and the stochastic routing scheme which we describe below. We also give a brief overview of means used by routers to distribute the information on dynamic link state between them.

Dynamic metrics

Metrics are used to evaluate routes in order to differentiate between them and select the one that best matches needs. The metric of a route is typically composed of the metrics of all the links it traverses. Usually, it is a plain sum, but not necessarily, as we discuss later.

The purpose of a link metric is to evaluate the quality of link, i.e. its value for routing. As outlined in Section 2.1, the (typically bidirectional) links are characterized by latency, and bandwidth. It should also be noted that for most communications media the transmission of a packet may fail. The lower layer (i.e. data link) could attempt to hide this with automatic retry (e.g. ARQ), but, in general, the lossiness of a link could be significant. For example, this issue is a definite concern in modern wireless networks based on IEEE 802.11. Which of these characteristics should the metric be based on? That depends on what objective the routing algorithm is designed for. For example, if the goal is to minimize end-to-end latency, then the route metric would be the sum of the latencies of its links. Alternatively, the routes could be evaluated on end-to-end basis, i.e. time for the probing packet to traverse the whole route. If the links are lossy and we are interested in the probability of packet loss over the whole route, then the link metric is multiplicative, i.e. probability of delivery is the product of delivery probabilities over all links (assuming that the losses on different links are independent) [1]. On the other hand, if the goal is to maximize throughput, then under the fluid model, the throughput of a route is determined by the capacity of its bottleneck (i.e. least bandwidth) link. Yet another scheme that combines the sum and minimum operators was recently proposed for multi-channel wireless mesh networks [DPZ04]. In other work, exponential weighted moving average was advocated [Wan03], in which the closer the link to the source the more important its metric. But usually, the metric of a route is a simple sum over its links. The rationale behind that is the ease of computability of *cheapest paths* with costs defined by additive metrics using Bellman-Ford (distance vector, e.g. RIP [Moy89]) or Dijkstra (link state, e.g. OSPF [Hed88]) algorithms.

[1] Actually, if the loss probability is low, then it can be approximated by an *additive* metric.

Whichever metric is chosen, the fundamental problem is how to obtain it, i.e. measure the interesting characteristic. As for the latency, it could be measured with timestamped probing packets, but the there is no global clock in a distributed system. A round-trip time (RTT) of a packet can be measured instead. Assuming that the delays in both directions are equal, RTT divided by two gives the one-way latency. However, this assumption is frequently unfounded. Gupta and Kumar [GK97] point out that in order to discern between routes the absolute value of the latency is not required, but rather the differences in latencies, which in fact can be measured using timestamps. Estimates on maximum link throughput are often obtained indirectly from the queuing delay. If, on average, a packet remains in the queue for the specific link for time δ, then the packet throughput is $1/\delta$. A different approach to measure the available bandwidth ignoring the queuing delays (useful for multiple rate wireless channels) is to send a pair of back-to-back packets of differing sizes and infer it from the difference in time needed to transmit them [Kes93]. As for the delivery probability, it could be estimated by observation of a sequence of probing packets, as proposed by De Couto [CABM03]. A different view on the purpose of link metrics bases on the concept of resource pricing. A virtual additive cost may be assigned to links, so that savvy users would adjust their resource allocation in order to minimize their costs or optimize their profit. Such cost could be based on the aforementioned link characteristics, but it can also incorporate the total load on the link, defined as the sum of rates of all flows that use the it. In such case the load needs to be measured, either as the total observed flow rate, or by gathering the demand from the flows explicitly, e.g. the packets of the flow could contain explicit information on the flow rate at source. Some metrics, though, are quite difficult to obtain. Consider the *marginal latency* metric as defined in Section 2.2.3, which is the value of a derivative rather than a directly observed characteristic. We discuss several methods for this purpose in Chapter 4.

Another issue is the metric stability. On one hand, the metric needs to be sensitive enough so that the system as a whole is reactive and actually adapts to the current inputs. On the other, the metric should be insensitive to transient spikes and noise. Otherwise, the system may fail to converge and remain in sub-optimal region of state-space for most of the time. A common way to achieve this is through smoothening the metric with a moving exponentially-weighted average. If $m(t)$ is the current reading, then the long time-scale average $\widehat{m}$ over all observations is computed as:

$$\widehat{m}(t+1) \leftarrow (1-\gamma)\,\widehat{m}(t) + \gamma\, m(t)$$

The scalar $\gamma < 1$ is sometimes called the *damping* or *forgetting factor*. The older a sample is, the least it counts as it is exponentially attenuated:

$$\widehat{m}(t) = \widehat{m}_0 + \sum_i \gamma^i m(t-i)$$

Table 2.1 summarizes the main metrics and their properties.

Table 2.1: A summary of dynamic metrics

metric	route-link composition	measurement
latency	sum	round-trip, timestamps
throughput	minimum	queuing delay, packet-pair
delivery probability	product	sequence observation
load-dependent cost	sum	based on other metrics

Stochastic routing

The classic hop-by-hop forwarding paradigm is simplistic: the router needs only to lookup the destination address of the processed packet in its routing table and select the proper outgoing link. Therefore, the table actually contains the next hop information for all addresses. Note, that the decision is made basing purely on the destination address. Other information on the flow that the packet belongs to, such as source address or flow id, is not considered. As a result, all packets destined for the same node will be pushed through the single designated outgoing link. But redundancy is one of the design principles of the Internet, and so there are usually multiple routes to destination available. Lorenz *et al.* in [LOR+01] give quantitative comparison of such non-proportional destination-oriented and per-flow splittable routing, and indicate that the benefits in throughput of the latter one are unbounded in general network topologies. Therefore, a firm association of the next hop to the destination addresses could be inefficient, in the sense of both bandwidth utilization and reliability of delivery.

Several methods for distributing the load addressed to a single destination over a set of routes have been proposed. One way of extending the hop-by-hop scheme is to select the next hop using more information, including the source address or even more precise flow identification. A different approach, *multi-path datagram routing*, is to distribute all packets addressed to the same node over a set of multiple routes either in round-robin fashion (if equally good, e.g. equal-cost-multi-path (ECMP) employed in OSPF [Moy89]) or in proportion to the preference.

The main difference between the two schemes is that multi-path datagram routing is still making decisions basing on the destination address alone. Hence, it is *source-invariant* and insensitive to the path that the packet already travelled before reaching the routers. Yet, in some cases it could be desirable to account for the past of the packet as illustrated on Fig. 2.8. Note, however, that flow-sensitivity serves the flows' individual goals and bears a mark of selfishness.

Another difference between the two concepts is that per-flow routing protects the nature of flow, i.e. all packets of a flow follow the same route and thus out-of-order delivery is unlikely. On the other hand, multi-path datagram routing selects the routes per-packet, so the packets can follow routes of substantially different latency, which in turn leads to loss of order. However, proportional routing can be combined with per-flow routing for flow protection, e.g. by

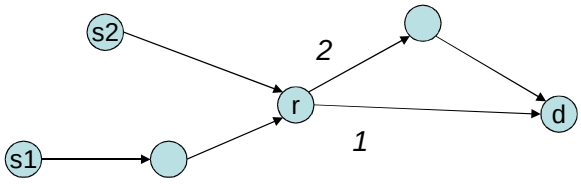

Figure 2.8: Sources $s1$ and $s2$ send packets to d through router r. If the hop count is to be minimized, both flows will be routed over link 1. But if the capacity limit is reached, and both links should be used, then a *fair* router would select link 1 for $s1$ and 2 for $s2$.

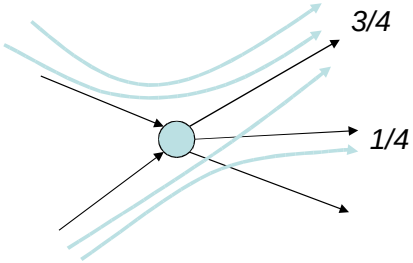

Figure 2.9: Incorporating flow protection into proportional routing. The fraction of unsplittable flows routed along a specific link corresponds to the desired proportion for that link.

distributing the flows (rather than packets) over the selection of outgoing links. See Fig. 2.9 for illustration.

We should point out, that of the two multi-path datagram variants, a routing paradigm which allows the routing proportions to vary is substantially more general than such that uses the multiple paths in equal fractions. Actually, it is relatively simple to give a two-link flow game as in Fig. 2.5 that would require such capability to achieve optimal social welfare.

An interesting argument for multi-path routing is its increased robustness in dynamic environments. An analysis by Wang and Crowcroft [WC92] shown that single-path routing algorithms using dynamic metrics may fail to converge and display undesirable oscillations. If multiple paths are used, then a change in the metric of a single link could affect just one of the paths. Even better, if the routing proportions can be varied, then the load can be tuned more finely.

A wide-spread implementation of proportional hop-by-hop routing is *stochastic routing*. Instead of a single next hop, the stochastic router at each node maintains *for each destination* a probability distribution over all its neighbours. When a packet for a specific destination arrives, the router uses this distribution to draw a random neighbour and forwards the packet there. Effectively, this distribution marks the proportions of incoming flow that are routed through each outgoing link. This paradigm is still destination-oriented, so we can discuss the case for each destination separately. We can estimate the rate of flow forwarded over to each neighbour to be the associated fraction of the incoming flow. This is illustrated on Fig. 2.10 and later

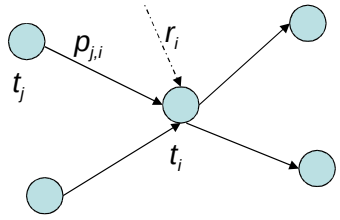

Figure 2.10: Flow conservation rule for proportional routing. For each destination we have the following: If t_i denotes the total rate of flows routed by node i, $p_{i,j}$ is the fraction of flow that is routed from i to j and r_i is the total rate of flows sourced at i, then $t_i = \sum_j t_j\, p_{j,i} + r_i$.

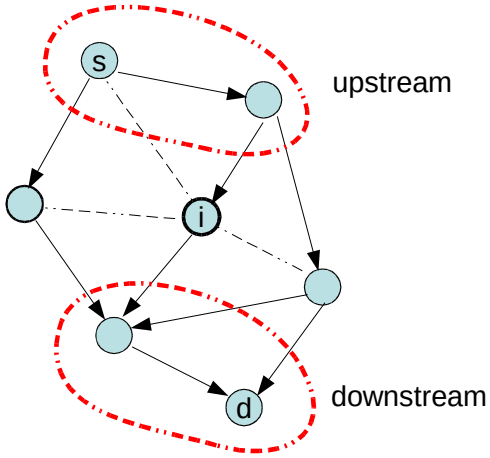

Figure 2.11: The dashed lines represent unused links. The solid arrows are the directed links that are used for routing, i.e. $p_{\cdot,\cdot} > 0$. A packet is sent from s to d. All nodes that can be traversed by the packet before it reaches the node i are *upstream* from it. All nodes that can be traversed by the packet after it is forwarded by i are *downstream* from it.

formalized in the core chapter 5 of the thesis.

One inherent problem of stochastic routing is the possibility of loops. Suppose there exists a cycle in the network graph $(i_1, i_2, \ldots, i_n = i_1)$. If $p_{i,j}$ is the probability of routing a packet from i to j, then, with probability $\prod_k p_{i_k,i_{k+1}}$, the packet will be forwarded over the whole loop. Although, if this product is less than 1, the loop is not *persistent* and no packet is looped forever, the packet could be trapped for too long and eventually dropped. Another way to state the condition of loop-freedom is using the terms *downstream* and *upstream* illustrated on Fig. 2.11. A loop is formed whenever a node is upstream and downstream of another at the same time. We describe two methods to avoid forwarding loops:

- One that works in single-path routing as well, is to tag the packet with the information on the nodes it already visited. Then a forwarding loop will be immediately discovered when the packet reenters a node. This, however, is incompatible with the classic datagram IP routing, in which no modifications to the relayed packet shall be made.

- The second solution is to ensure *loop-freedom* by restricting the set of neighbours that can be used for routing to a particular destination. Loop-free routing has the following property for each destination: *there exists an assignment of values c_i to nodes, such that* $p_{i,j} > 0 \Leftrightarrow c_i > c_j$. Conversely, if the routing algorithm is designed to enforce such assignment, then it is loop-free. The straightforward interpretation of the value c_i is the "'distance"' or "'cost"' of routing packet through node i. The work by Vutukury and Garcia-Lunes-Aceves [VGLA99] explicitly uses this property to preclude transient loops in minimum-delay routing. We elaborate on their solution in Chapter 4.

Managing metric information

In order to react to dynamically changing state of the network, the routers need to gather and share the dynamic metrics of their links. We describe several basic variants here. Chapter 4 contains reference to other concepts, e.g. reinforcement learning.

A *centralized* routing algorithm would only require the routers to communicate their link status to a central entity (the coordinator), whose work is to manage the routing and balance the load. The coordinator selects the routes and instructs the routers to conform to the selection. Like all centralized solutions, such algorithm would be prone to reliability issues (when the coordinator fails), as well as efficiency and speed of convergence. The routes will not be updated until the whole round-trip communication between the routers and the coordinator is complete.

In contrast, a *distributed* algorithm allows the routers to make quick adjustments in their immediate neighbourhood and enables a kind of softer convergence. In his early work, Gallager argues that distributed algorithms have important advantages in robustness, since the central entity would need means to communicate with the routers and for that it needs another set of pre-set routes [Gal77].

A centralized algorithm can be transformed into a distributed *link-state* algorithm using a simple technique. Instead of having a single coordinator node compute the routes, we can spread the burden over all routers. In such scheme, the routers share their link state (local topology) information with all others, and use this global information to compute the routers. The information is spread through *flooding*: as soon as it is received at a node, it is relayed over to all neighbours, but only at the first reception, so that no node transmits the message more than once. Fig. 2.12 illustrates this procedure.

Afterwards, all the participants essentially perform the same computation. The router is only interested in the routes that traverse it, and therefore the amount of necessary computation can be reduced. Note, if the hop-by-hop routing paradigm is employed as usual, then the routers only decide on the next hop, and therefore they must use a common path selection algorithm so that they fully agree on the computed routes. If, for example, ties were not properly broken, forwarding loops could be formed. A widely-used routing algorithm that operates in this link-state fashion and computes shortest paths using Dijkstra algorithm is OSPF [Moy89]. One of

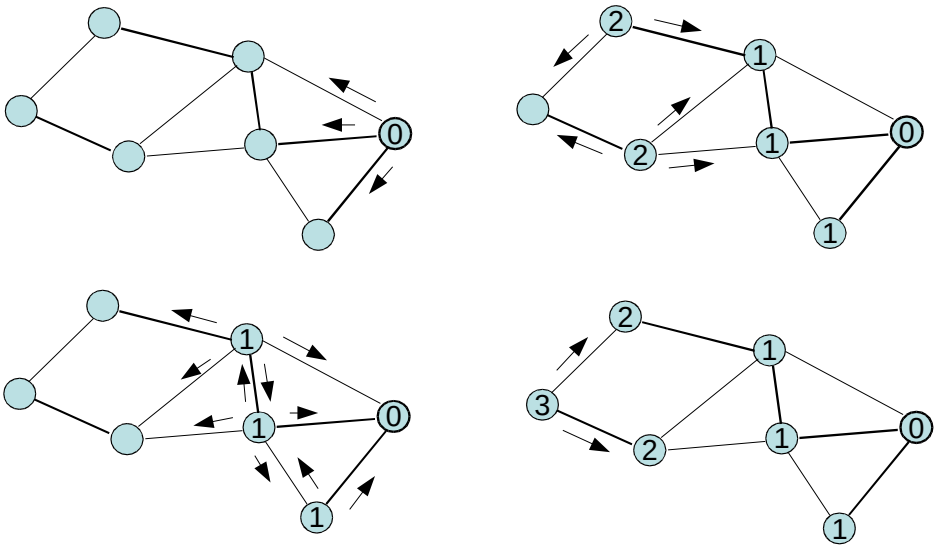

Figure 2.12: Message flooding. The arrows represent message transmissions, and the numbers on nodes denote how many transmissions were needed to obtain the message. In reality, the communication is not synchronous and hazards occur, though usually harmless.

the frequently praised features of link-state algorithms is their inherent loop-freedom provided the information flooding is complete [1].

Yet, the first dynamic routing algorithms were constructed quite differently. Truly distributed algorithms use distributed computation, i.e. the nodes do not exchange full input data, but rather partial or intermediate results. For example, the neighbours can exchange information about the routes they use, and next decide which of the routes – own or the neighbour's – is better. This procedure is equivalent to edge relaxation for shortest (or cheapest) paths in graphs: if the known path to d from node i has cost c_i, and the cost of edge (i, j) is $c_{i,j}$ then after relaxation of edge (i, j) the known path cost is updated to $\min(c_i, c_j + c_{i,j})$. Bellman and Ford have independently proven that, if every edge is relaxed and the whole cycle is repeated a number of times equal to the number of nodes, then the resulting costs are optimal, i.e. of cheapest paths. The crucial property of this algorithm is that it is easily distributable: edge (or link) relaxation can be performed asynchronously and its scope spans just two participants. This yields the Distributed Bellman-Ford (DBF), and a class of *distance-vector* routing algorithms, of which the most widely used is RIP [Hed88, Mal98]. Some advantage of distance-vector algorithms is that the routers do not have to use the very same globally-known routing algorithm, as long as the reported costs are accurate. Furthermore, the routers need not inform other participants how they route the packets. This might slow convergence down (as we discuss in Chapter 5), but is advantageous for stochastic routing, where the routing tables are substantially larger.

Up to now, we have silently assumed that the routes are computed *proactively* for all destina-

[1] Loop-freedom of most link-state algorithms referrs to *permanent* loops, as transient loops, while the information is disseminated, are possible.

tions. That is, the set of destinations that routes are maintained to is not dependent on current need. A different class of routing algorithm is based on *reactive* or *on-demand* operation. A reactive routing algorithm builds routes in response to requests and maintains them as long as they are needed. On-demand algorithms receive increased interest in networks of highly variable topology, e.g. MANET, in which the overhead induced by proactive routing is unacceptable. *Hybrid* solutions that combine the features of both schemes have also been proposed. As an end note, we point out that Feamster *et al.* have recently proposed to actually *separate the routing from routers* at the inter-domain level, i.e. remove the burden of finding routes from the nodes that take care of forwarding along them [FBR+04]. Still, in this thesis we will follow the traditional distributed approach.

2.3.2 Congestion avoidance

Admission control and resource reservation do not lie in the nature of "'best-effort"' service, and so the IP networks when overloaded can be driven into a congestion collapse. But before congestion collapse occurs, the queuing, connection-less nature of the datagram networks reveals itself in the rise of end-to-end delay and packet-loss. The idea behind congestion control is to protect the network's resources from being flooded with the users' requests.

Figure 2.13 illustrates the difference between the two notions of congestion control and avoidance. Congestion avoidance mechanisms allows the network to remain in the state area of high throughput and low delays and keep the packetloss low. A good survey on the fundamentals of congestion control and avoidance is given by Jain in [Jai90]. By considering the popular myths about possible ultimate solutions to congestion (such as increasing resources), Jain makes an important statement:

> (...) congestion is a dynamic problem. It cannot be solved with static solutions alone. We need protocol designs that protect networks in the event of congestion. The explosion of high-speed networks has led to more unbalanced networks that are causing congestion. In particular, packet loss due to buffer shortage is a *symptom* not a cause of congestion.

A typical way to deal with congestion is through demand reduction. It should be noted that, although the primary goal of congestion regulation is to avoid uncontrolled service degradation, it is usually also burdened with a fairness requirement. So, if the users' load is restricted to maintain high overall service quality, this allocation of load should also be fair (see section 2.2.2 for overview).

The three main components of a congestion avoidance scheme are:

detection - The network needs to monitor the load and remaining capacity of resources. Otherwise, congestion-related events such as losing packets and overflowing queues can work as

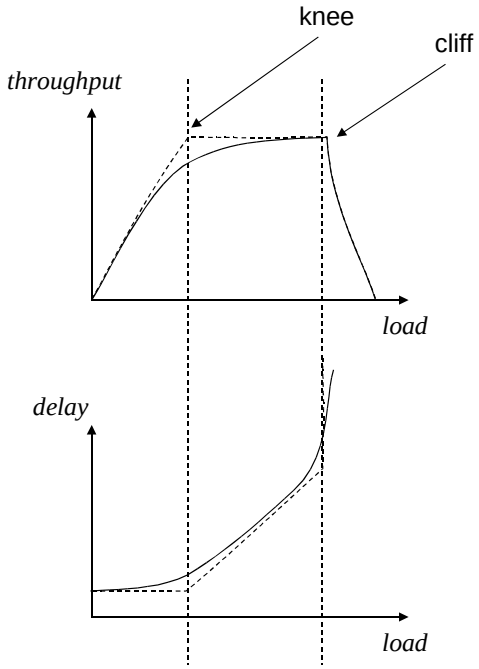

Figure 2.13: The characteristics of service as the load increases. The purpose of congestion avoidance is to keep the dynamic state near the *knee* rather than the *cliff* as in the case of congestion control.

implicit evidence of congestion. However, remedial action should be taken well in advance of the service *cliff*.

signalling - After congestion is detected, the entities responsible for actions need to be informed. A number of feedback schemes is discussed below.

control - Using the feedback as a source of information the restraining mechanism reduces the load. This mechanisms can be placed in a number of locations along the IP stack, as discussed below.

Feedback mechanisms

Typically, the resources are monitored on-location by adjacent routers. For example, a router can monitor its outgoing links by observing the persistent size of its queue, the average queuing delay, the remaining capacity (based on the current load estimate), etc. If this is the case, then the routers will be the source of congestion feedback signals. Such signal can be conveyed by sending out a special control packet towards the restraining entities (most likely the sources that use the congested resource). Alternative scheme, that potentially reduces additional overhead, is to use designated fields in the data packets that flow in the desired direction. The disadvantage of the second method is the dependence on existing data flows. Hybrid solutions are possible.

However, it is also possible for the user to perform the monitoring of the resources that its flow uses. This is achieved with *probing*, i.e. sending probe packets across the network and measuring

its delay or delivery probability. A particular case is when probing is done implicitly by observing the backward flow of acknowledgement (ACK) packets, as used in reliable transport protocols. Using ACKs the round-trip-time (RTT) can be estimated, and when no ACK is received within some the timeout period, the packet is presumed lost.

The great advantage of implicit feedback in the form of ACK loss is its minimum additional overhead (the ACKs need to be sent anyway). However, the inherent problems of this scheme are its slow signal propagation (several RTTs needed), susceptibility to delay spikes (robust timeout threshold is needed), and the fact that, by default, it is triggered at the congestion cliff (packets are actually lost).

An interesting method to match the implicit feedback scheme with the requirements of congestion avoidance is *random early discard* (RED) queuing discipline. A RED queue starts to randomly drop packets when the persistent queue size exceed a specified threshold. That way, the implicit feedback signal is sent out before the *real* congestion occurs.

Load control

When congestion is detected, the competing sources need to reduce their load in order to protect the resources. The load restraining mechanism can be implemented at several alternative locations.

The *transport layer* seems to be best fit for load control, because it is directly below the source user (at the end node). Note that the same amount of control should be applied only to the flows that share bottleneck links. If the amount of control is determined per-flow, then one load-control agent is needed for each flow. Also, as some flow control mechanism might be already implemented (with purpose to protect the destination), it is natural for the transport layer to also take on congestion control. A common way to implement both functions is with ack-paced moving window, as illustrated in Fig. 2.14. A different way to implement load control is by enforcing flow rate using a *paced* queue, i.e. such that releases packets at specified intervals.

The responsibility of load control can be pushed down into the *network layer*. One concept is to apply load limits at each hop rather than just at the source node. Such limits could be applied per flow using queuing schemes that alleviate congestion by isolating traffic load of different users, e.g. as in fair queuing [DKS89]. In a coarser variant, the limit would be applied to all incoming traffic aggregately per destination. This latter scheme has recently gained popularity in the area of sensor networks in which there are frequent floods of data towards a common destination, e.g. [WEC03, HJB04].

Control loop

A typical congestion avoidance scheme is a control feedback loop: the load causes congestion which causes load control. Various adaptation algorithms to determine the appropriate amount of control have been developed.

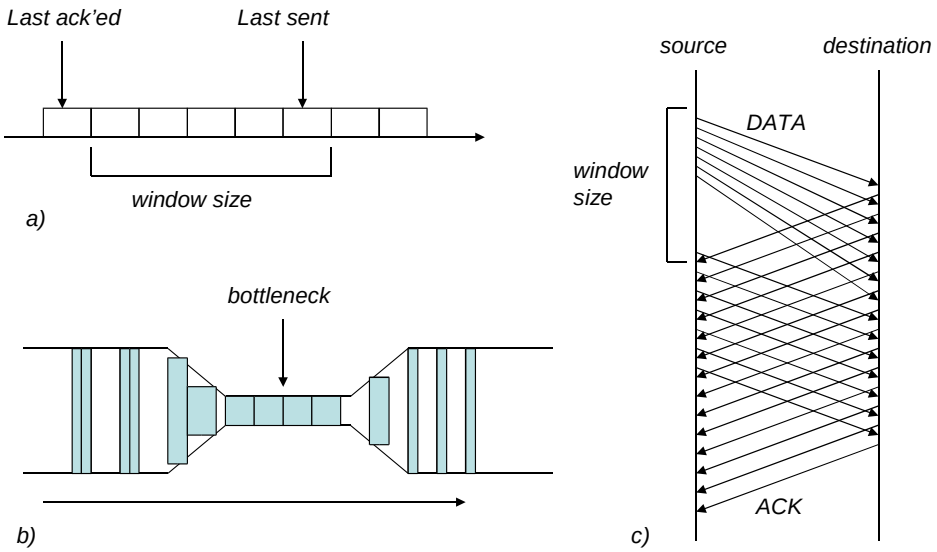

Figure 2.14: The sender keeps a buffer of packets sent but not yet acknowledged (a). When an ACK is received, the window is moved to the right and more packets can be sent out. Since the destination acks every packet, the acks return to the source at a steady rate determined by the bottleneck resource (b). Thus, the system is self-regulatory (c). The size of the window depends on the available resources and is to be maximized to the extent when ACKs are lost.

One deeply-studied and widely-implemented scheme is the already mentioned in Section 2.2.2 *additive-increase-multiplicative-decrease* (AIMD) scheme. AIMD proceeds as follows: normally the rate increases at a steady rate α (additively), but whenever congestion is detected, the rate is multiplied by $\beta < 1$ (multiplicatively). This scheme could be approximated with a differential equation:

$$\frac{d}{dt}x(t) = \alpha - \beta\, p(t)$$

where $p(t)$ is the rate of congestion feedback

Chiu and Jain show in [CJ89] how this scheme attains both efficiency (i.e. high utilization) and absolute fairness in single-commodity environments. Moreover, Kelly *et al.* prove in [KMT98] that this scheme actually attains *proportional fairness* as defined in section 2.2.2. Recently, however, the fact that this simple regulator manages both utilization and fairness has been argued by some researchers as disadvantageous. Instead, Katabi *et al.* propose in [KHR02] to decompose the control loop into two regulators, one to control utilization by adjusting the aggregate allocation, and the other to manage fairness by balancing the individual rates within the allocation. In Chapter 4 we discuss other schemes.

Chapter 3

System Model and Objective

In this chapter, we state the system model and the global objective that we design our algorithm for. The objective is inspired by the work of Kelly and Voice [KV05], but our system model uses a different routing paradigm. W discuss the details and interpretation of the objective and its components, and provide a brief theoretical analysis. Specifically, we state the conditions on optimal solution and propose a decomposition of the problem.

3.1 System model

First, we define how we model the system network and users. Essential to the model is the routing paradigm employed by the nodes. The model incorporates the assumptions we make about the system.

3.1.1 Entities

As described in Chapter 2, the system comprises of network resources and users. We model the network with a graph and the users with flows. Therefore, the formalization of the system model contains the following:

- *network graph* $G = (V, E)$, of which vertices (denoted by i, j, k) represent network nodes and directed edges (denoted by e or (i, j)) represent links.
- *user set* S. Each user $s \in S$ is described by its source node, $src(s)$, and destination node, $dst(s)$. We use terms *user*, *source* and *flow* interchangeably.

We also assume that the network is not overloaded, thus no packets are lost and the flows have a single rate along the whole route. In other words, the flow allocation is *feasible*. Define:

- the flow rate of user s denoted by y_s
- the total load (aggregate flow rate) of link $e = (i, j)$ denoted by z_e or $z_{i,j}$

To simplify notation we shall omit the containing set in most enumerations and let $i, j, k \in V$, $e, (i,j) \in E$, $s \in S$, and so on. We assume that the users enforce rate control at source (see Section 2.3.2). The next section covers the relationship between y_s and z_e determined by the routing policy.

3.1.2 Routing

We first define a generalized per–flow routing paradigm and subsequently restrict it to match the stochastic routing paradigm. In the absence of packet–loss, the policy is traffic–conservative, i.e. at a stable state the total rate flowing in equals the total rate flowing out (see Section 2.3.1). For both systems, we state the constraints that are implied by the assumption on flow conservation.

Per–flow routing

Let us first define the generalized per–flow routing paradigm. Each flow is routed independently and can be split at any fraction at any node it passes on its route. The routing variables are of the form $x_{s,e}$ which denotes the flow rate of the part of flow s that flows through e. Then the relationship between y_s and z_e is imposed by flow conservation requirements. For each flow, the total rate of the part coming out of a node (in any direction) equals the total rate of the part coming in, unless it is the source or the destination node. In the former case the total rate coming out is increased by the flow created y_s, and in the latter it is null. The total flow over a link is the sum over all flows that use it. We can write:

$$\begin{aligned} x_{s,e} &\geq 0 \\ \sum_s x_{s,e} &= z_e \\ \sum_j x_{s,(i,j)} &= \begin{cases} 0 & \text{if } i = dst(s) \\ \sum_k x_{s,(k,i)} + y_s & \text{if } i = src(s) \\ \sum_k x_{s,(k,i)} & \text{otherwise} \end{cases} \end{aligned} \tag{3.1}$$

Note that the total flow coming out of its source may be larger than y_s in the pathological case of forward looping.

Since it allows the users to choose the routes for their flows, this paradigm is very flexible and supports selfish routing in particular. However, it is not compatible with the datagram forwarding nature of IP networks. That is why we propose to use stochastic routing instead. Later in this chapter, we show that they are equally powerful for the purpose of our objective.

Stochastic proportional routing

The stochastic hop-by-hop routing policy is destination–oriented, i.e. the routes for different destinations are defined independently (see Section 2.3.1). Therefore, we maintain the set $D = \{dst(s) : s \in S\}$ of all possible destinations, and instantiate the routing policy for each $d \in D$.

Let $Nb(i)$ denote the set of i's neighbours, $Nb(i) = \{j : (i,j) \in E\}$. Each node i maintains a probability distribution over its neighbours, $j \in Nb(i)$. A packet addressed to d is forwarded by i to j with probability $p^d_{i,j}$. For easier manipulation, we extend the vector over all $j \in V$ and require $p^d_{i,j} = 0$ when $(i,j) \notin E$. Therefore:

$$p^d_{i,j} \geq 0$$
$$\sum_j p^d_{i,j} = 1$$

If $i = d$, the flow is not forwarded but consumed, so we make an exception to the above and require that:

$$\forall_j \, p^d_{d,j} = 0$$

If we take the flow–conservation requirements of (3.1) then under the stochastic routing policy, we also add that:

$$x_{s,(i,j)} = p^{dst(s)}_{i,j} \sum_k x_{s,(i,k)} \tag{3.2}$$

That is, the amount of flow that is routed over a particular outgoing link is given as a specific fraction of the total outgoing flow. This fraction depends not on the flow, but solely on its destination. Thus, this policy does not support selfishness.

Let us introduce an aggregate description of this system in a fashion similar to other researchers, e.g. [Mas85,Gal77,Seg77,BGG84]. Denote by t^d_i the total amount of traffic (flow rate) that node i needs to route to destination d. Further denote by r^d_i the total amount of traffic that is injected at node i by the users that connect to it. Thus we require $\forall_{i \in V, d \in D}$:

$$\begin{aligned} r^d_i &= \sum_{s:src(s)=i,dst(s)=d} y_s \\ t^d_i &= \sum_j p^d_{j,i} \, t^d_j + r^d_i \end{aligned} \tag{3.3}$$

Finally, for each link $e = (i,j) \in E$, the aggregate flow rate that flows through it, i.e. its load $z_e = z_{i,j}$ is given by:

$$z_{i,j} = \sum_d p^d_{i,j} \, t^d_i \tag{3.4}$$

Fig. 3.1 illustrates the notation and relationships between values on a simple example.

3.2 Objective

We shall now define the global objective that will be the point of optimization and drive our algorithm. We also discuss possible interpretation and how the objective can be tailored to match desired goal.

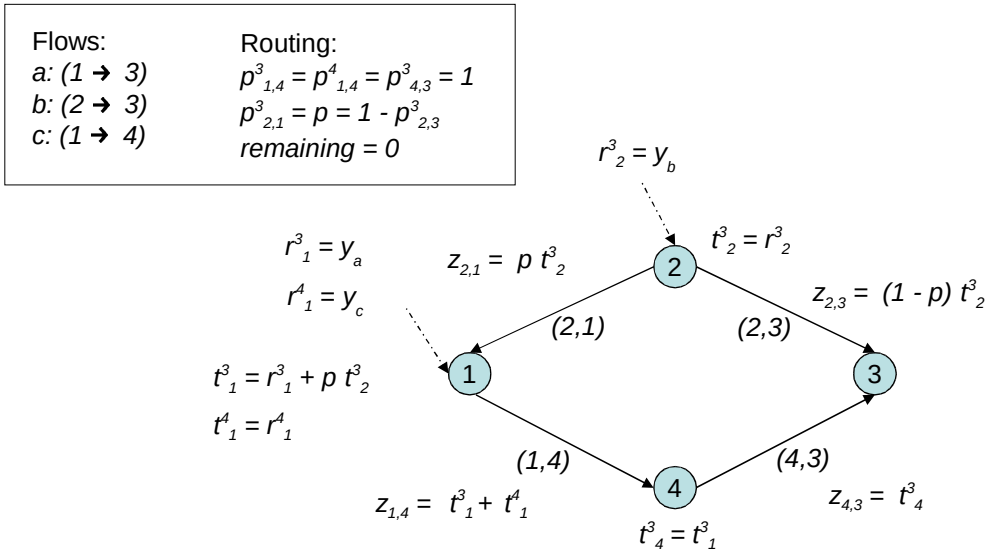

Figure 3.1: Example network and set of flows. The routing probabilities are given and so is the resulting allocation of r, t and z.

3.2.1 Definition

The global optimization objective is the profit defined as the difference between total utility of users and total cost of resources. The utility of user s is given by its strictly increasing utility function of its rate $U_s(y_s)$. The resources of the system are network links, and the cost of link e is given by its strictly increasing cost function of its load $C_e(z_e)$. Hence, the system objective is:

$$\max \sum_{s \in S} U_s(y_s) - \sum_{e \in E} C_e(z_e) \tag{3.5}$$

with z_e defined as in the previous section. Note that the variables of this system are source rates y_s and either $x_{s,e}$ under the per–flow routing policy, or routing probabilities $p^d_{i,j}$ under the stochastic routing policy. We will also use $U_T = \sum_s U_s(y_s)$ and $C_T = \sum_e C_e(z_e)$.

The history of this optimization problem is quite long both in the context of road transport networks [War52, BMW56], and communication networks [Gol80, BG92]. In this choice, we are directly inspired by the work of Kelly and Voice [KV05]. Many researchers state a different objective, called *Kelly's system problem*, that is only concerned with the aggregate utility subject to feasibility constraints due to limited resource capacities [Kel97, KMT98, LA02, Chi04], i.e.:

$$\max \sum_{s} U_s(y_s)$$
$$\text{subject to } z_e < Cap(e)$$

However, such a system problem accounts only for network feasibility requirement and satisfaction of received throughput. In contrast, using a cost function we can account for link utilization and end-to-end delays, considerably extending the scope of optimization. There has also been substantial research done in pure cost minimization problem, in which only the aggregate cost

C_T is considered, e.g. [Gal77, BGG84, GK97, BK03, XQYZ04, RK04].

$$\min \sum_e C_e(z_e)$$
$$\text{subject to given } y_s$$

Still, this problem does not account for user satisfaction of received throughput, but rather assumes that the user demand (i.e. load) can be fully satisfied. Thus, the proposed objective combines the features of *both* problems as we discuss in the next section. Also note that the Kelly's problem can be transformed into (3.5) by means of Lagrange multipliers, $C_e(z_e) = \lambda_e(Cap(e) - z_e)$.

In a fashion similar to that of Kelly and Voice [KV05], we restrict the classes of the utility and cost functions. $U_s(y)$ are differentiable, strictly increasing and strictly concave over $y > 0$. Furthermore, assume that $U_s(y) \xrightarrow[y \to 0^+]{} -\infty$. $C_e(z)$ are differentiable and $C_e(z)/z$ is strictly increasing over $z > 0$, which implies that $C_e(z)$ is strictly convex. In addition, assume that there exists a capacity equivalent $Cap(e) > 0$, such that $C_e(z) \xrightarrow[z \to Cap(e)^-]{} \infty$. Finally, we shall assume that there exists at least one feasible allocation, i.e. one that yields a finite value of the goal. Note that this implies that the network is connected as needed, i.e. there exists a directed path between each user source and destination. Similar assumptions have also been made by other researchers, e.g. [Gal77, BGG84, KV05, Rou02, RK04].

3.2.2 Interpretation

The proposed objective function has the advantage of comprising both user utility and explicit resource costs. A function of flow rate, $U_s(y)$ represents user's satisfaction from the received throughput y. On the other hand, $C_e(z)$ reflects the penalty for using the capacity of the resource. In particular, C_T may well represent the total aggregate latency of all users. In fact, since we require that $C_e(z)/z$ be increasing, we may define $C_e(z) = z\, l_e(z)$, with $l_e(z)$ being the packet latency on edge e. Then the global objective can be rewritten as:

$$\max \sum_{s \in S} U_s(y_s) - \sum_{e \in E} z_e\, l_e(z_e)$$

and is equivalent to:

$$\max \sum_{s \in S} \left(U_s(y_s) - \sum_{e \in E} x_{s,e}\, l_e(z_e) \right)$$

where $x_{s,e}$ is the flow rate of the part of flow s that flows through e. Note, $\sum_e x_{s,e}\, l_e(z_e) = y_s \hat{l}_s$, where $\hat{l}_s$ is the average end-to-end latency of flow s. Therefore, the objective can be interpreted as a maximization problem of aggregate of users' *profits*, defined as the difference between each user's utility and cost.

Another interpretation inspired by Kelly [KMT98] is possible if $U_s(y_s) = w_s \lg y_s$, for some weight w_s. Informally, $U_s(y_s) - y_s\, c_s$ is at an optimum with respect to y_s, when $y_s = w_s/c_s$, i.e.

the received rate matches the price paid w_s divided by the incurred average cost per unit of flow s, that is c_s.

Further, since $U_s(y) \xrightarrow[y\to 0^+]{} -\infty$ and $C_e(z) \xrightarrow[z\to Cap(e)^-]{} \infty$ for some $Cap(.)$, the optimal allocation must be *within* the interior of $y_s > 0$ and $z_e < Cap(e)$, i.e. no user is starved and no edge is overloaded.

We discuss a number of cost functions previously used by researchers in Chapter 4. However, the reader should bear in mind, that the specifics of the utility and cost functions beyond that assumed in the previous section are *immaterial*, provided that the their derivatives can be measured for the purpose shown in the next section.

3.3 Optimal allocation

In this section, we discuss the character of the optima of the objective under both routing paradigms and show their equivalence. Further, we show that the composite profit problem can be decomposed into two well-known problems.

3.3.1 Convexity and equivalence

Consider the generalized per–flow routing policy (3.1). In other words, the users are fully controlling the routes of their flows, rather than subject to aggregate routing probabilities as in (3.2).

Lemma 3.1. *The local optima of the per–flow system (3.1) are also global and the optimal y_s and z_e are unique.*

Proof. Since $U_s(.)$ are concave, $C_e(.)$ are convex, and the solution space of $x_{s,e}$ is also convex, therefore the maximization problem (3.5) is convex. Thus all local optima are also global [BNO03]. While $x_{s,e}$ at optimum do not need to be unique, due to the strict concavity of $U(.)$ and strict convexity of $C(.)$ the optimal values of y_s and z_e are unique. □

Let us point out that the requirements for strict concavity/convexity could be relaxed to allow linear functions. For example, if latency of a link was constant (i.e. not strictly increasing), then its cost function would be linear (i.e. not strictly convex). Although such scheme would still be feasible, it would not support load balancing (See Section 2.2.2).

Lemma 3.2. *Any allocation of y_s and z_e feasible in the per–flow system can be achieved in the the stochastic system (3.3)-(3.4).*

The proof of this lemma is given in Appendix A. The implications of this lemma are far-reaching. Although the per–flow system allows for selfish routing (see Section 2.2.3), the significant values of y_s and z_e can be achieved using stochastic hop-by-hop routing, which does not

explicitly support selfishness. The global objective is the aggregate of user's profits, thus the individual user costs depending on $x_{s,e}$ need not be optimized.

The two lemmas give us the main result of this section:

Theorem 3.1. *The local optima of the system (3.3)-(3.4) are globally optimal and the optimal allocation of y_s and z_e is unique.*

3.3.2 Problem decomposition

Theorem 3.1 implies that to obtain the global optimum it is sufficient to satisfy conditions of local optimality. Therefore, we can decompose the joint routing and rate control problem along the typical layer splitting line into two separate problems:

1. optimal routing under fixed user load
2. optimal user rates under fixed routing

It is quite clear how the composite problem can be solved using procedures solving these sub-problems. The two algorithms need just to be run iteratively interleaved. But actually, a single run of each procedure does not need to go all the way and reach the local optimum, but only improve on the global objective.

We finish this section by stating the conditions of local optimality in the two sub-problems. However, before we proceed, we define marginal cost which is a common notion to both problems.

Marginal cost

Recall from section 2.2.3, selfish mechanisms can be used towards social goals provided that marginal cost is used instead of the full cost.

Define the *marginal cost* of using node i to route packets to destination d:

$$\lambda_i^d = \frac{\partial C_T}{\partial r_i^d} \tag{3.6}$$

and the marginal cost for each link e:

$$\mu_e = C_e'(z_e) \tag{3.7}$$

As stated above, conditions for both optimal routing and optimal rate control are defined with respect to λ_i^d. Let us show how the marginal cost can be computed. For this purpose, each node i maintains the estimated marginal cost c_i^d to reach destination d. Given the costs of all neighbours and the routing probability distribution, we can give the estimated cost at the node:

$$\begin{aligned} c_d^d &= 0 \\ c_i^d &= \sum_j p_{i,j}^d (c_j^d + \mu_{i,j}) \end{aligned} \tag{3.8}$$

This formula is based on the intuition that that the cost is the outcome of a probabilistic try, and thus its estimated value is an average of the possible outcome costs weighed by outcome probability. Same formula was used for hop-by-hop (in contrast to end-to-end methods) cost estimation in [Gal77, Mas85, CAT90, VGLA99, XQYZ04]. In order to show that this estimate is accurate, we obtain closed-form (fix-point) solutions to the recursive equations for routing load (3.3)-(3.4) and cost estimate (3.8). This procedure is similar to that in [Gal77] and [Mas85].

Define a family for $d \in D$ of routing matrices $\mathbf{P}^d$ containing $p^d_{i,j}$, vertical source flow vectors $\mathbf{r}^d$ containing r^d_i, and rate load vectors $\mathbf{t}^d$ containing t^d_i. Also denote $\mathbf{A} * \mathbf{B}$ the element-by-element product of same order matrices, and $\mathbf{A}^{\mathrm{T}}$ be the transpose matrix of $\mathbf{A}$.

Then the stable state of the routing paradigm (3.3)-(3.4) can be restated as:

$$\mathbf{t}^d = \mathbf{P}^{d\mathbf{T}}\mathbf{t}^d + \mathbf{r}^d$$

This yields a fix-point solution:

$$\begin{aligned} \mathbf{t}^d &= \mathbf{A}^{d\mathrm{T}}\mathbf{r}^d, \\ \text{where } \mathbf{A}^d &= (\mathbf{I} - \mathbf{P}^d)^{-1} \end{aligned} \tag{3.9}$$

Therefore, the matrix $\mathbf{A}$ contains the rate transfer functions of the network:

$$a^d_{i,j} = \frac{dt^d_j}{dr^d_i}$$

In particular, the diagonal values can be used to determine if the routing is loop-free. If p is the probability that a packet injected at i addressed to d returns to i, then:

$$a^d_{i,i} = \frac{1}{1-p} \begin{cases} = 1 & \text{if there are no loops from } i \text{ addressing } d \\ > 1 & \text{otherwise} \end{cases}$$

Note that, if persistent loops (of cycling probability equal 1) are present, then the matrix $\mathbf{A}$ cannot be computed, as $\mathbf{I} - \mathbf{P}$ is singular.

Similarly, if we use cost vectors $\mathbf{c}^d$ containing c^d_i and link cost matrix $\mathbf{M}$ containing $\mu_{i,j}$, then we obtain a new fix-point form for cost estimation (3.8):

$$\begin{aligned} \mathbf{c}^d &= \mathbf{P}^d\mathbf{c}^d + (\mathbf{P}^d * \mathbf{M})\mathbf{1} \\ \mathbf{c}^d &= \mathbf{A}^d(\mathbf{P}^d * \mathbf{M})\mathbf{1} \end{aligned} \tag{3.10}$$

where $\mathbf{1}$ is a vector of 1's of appropriate height. Note that $p^d_{d,i} = 0$ ensures that $c^d_d = 0$.

Theorem 3.2. *At a stable state, the marginal cost estimate obtained by (3.8) is accurate. For all $s \in S$:*

$$c^d_i = \lambda^d_i \tag{3.11}$$

A proof of this theorem using the fact that the matrix **A** is common to both formulae is given in Appendix A.

Remark: It is tempting to approach this fact using mathematical induction. Unfortunately, as the cost estimation formula recursively bases on the estimates of neighbours and cycles are not explicitly forbidden, there is no direct method to build atop of a previous inductive step.

Optimal routing

When the allocation is feasible (as assumed) and flow rates are conserved, the routing variables p do not affect the utility component U_T which depends on the fixed y_s. Therefore, the optimal routing given fixed user rates y_s is such that minimizes the total cost $C_T = \sum_e C_e(z_e)$. We will show that the optimal routing is at a Wardrop equilibrium equivalent for proportional routing.

Lemma 3.3. *Given fixed user rates y_s, a Wardrop equilibrium of the* generalized *system (3.1) driven selfishly has the property that the cost of routing flow s from node i to destination d does not depend on s nor whether the flow begins at or is relayed by i. Moreover, if this cost is given by λ_i^d, and the cost of using link (i,j) is $\mu_{i,j}$, then*

$$\lambda_i^d = \min_j(\mu_{i,j} + \lambda_j^d)$$

Proof. At a Wardrop equilibrium, no flow has the incentive to adjust its routing unilaterally. It follows that all routes in use have the same cost, which is less than that of all unused routes. Otherwise, envious flows would unilaterally adjust their routing to follow cheaper routes and reduce their cost. □

Theorem 3.3. *Given fixed user rates y_s, the optimal* proportional *routing is at Wardrop equilibrium of marginal costs. If $t_i^d > 0$, then:*

$$\lambda_i^d = \min_j(\mu_{i,j} + \lambda_j^d) \tag{3.12}$$

where λ_i^d and $\mu_{i,j}$ are defined in section 3.3.2. or equivalently:

$$\mu_{i,j} + \lambda_j^d \begin{cases} = \lambda_i^d & \text{if } p_{i,j}^d > 0 \\ \geq \lambda_i^d & \text{if } p_{i,j}^d = 0 \end{cases} \tag{3.13}$$

Proof. As shown in section 3.2.2, we can define $C_e(z) = z\, l_e(z)$. Using the result first shown by Beckmann *et al.* [BMW56], described in Section 2.2.3, we know that the optimal flow of the generalized minimum cost problem:

$$\min \sum_{s,e} x_{s,e}\, l_e(z_e)$$

is given by the Wardrop equilibrium of marginal costs μ_e. By lemma 3.3 we have the conditions of Wardrop equilibrium of selfish per–flow routing. We just observe that, since the cost does not

depend on s, the conditions can be satisfied under proportional routing, which routes the flows in an aggregate manner. We obtain the above conditions since we know that:

$$\lambda_i^d = \sum_j p_{i,j}^d (\lambda_j^d + \mu_{i,j})$$

Note this is only necessary, when some routes are actually used, i.e. $t_i^d > 0$. □

Remark: Gallager offers a slightly different but equivalent proof for minimum delay routing in [Gal77]. The author points out that:

$$\frac{\partial C_T}{\partial p_{i,k}^d} = t_i^d (\mu_{i,j} + \lambda_i^d)$$

and states the optimality condition as follows:

$$\frac{\partial C_T}{\partial p_{i,j}^d} \begin{cases} = \min_k \frac{\partial C_T}{\partial p_{i,k}^d} & \text{if } p_{i,j}^d > 0 \\ > \min_k \frac{\partial C_T}{\partial p_{i,k}^d} & \text{if } p_{i,j}^d = 0 \end{cases}$$

Yet another sketch proof, though for user–optimal (i.e. selfish) Wardrop routing, is given by Gupta and Kumar in [GK97] [1]. They first show that at equilibrium all sub-paths (i.e. from intermediate nodes) have the Wardrop property. Next, they map the problem to the domain of electrical circuits.

Corollary 3.1. *The optimal routing is loop-free.*

This observation stems straight from Theorem 3.3, (3.13) and the assumption that the cost functions $C_e(.)$ are increasing, which yields $\mu_e > 0$. It follows that $p_{i,j}^d > 0$ only if $\lambda_i^d > \lambda_j^d$. Thus, λ_i^d form an ordering of nodes such that the packets traverse the routes in the direction of decreasing cost, i.e. downstream nodes have lower cost. Therefore, routing loops are impossible.

Corollary 3.2. *The optimal routing under linear cost uses cheapest paths.*

In the previous section, we noted that the class of cost functions could be relaxed to accept linear functions. In such case, the marginal cost forms a load–independent metric and the Wardrop condition yields cheapest–paths routing. If there are ties between paths (i.e. multiple of the same minimal cost), all of them are used for routing in *any* proportions. Note that in ECMP (see Section 2.3.1) the proportions would be equal.

[1] Although Gupta and Kumar do not refer to Wardrop's definition, their optimality conditions are equivalent.

Optimal rates

To obtain optimal rate conditions, we make use of the fact that the problem is convex and consider the local optima of the goal (3.5) with respect to each user's rate.

Theorem 3.4. *Given a fixed routing of p, the user flow rates y_s are optimal if and only if:*

$$U'(y_s) = \lambda_{src(s)}^{dst(s)} \tag{3.14}$$

A detailed proof of this theorem is given in Appendix A.

Chapter 4

Related work

We have shown in Chapter 3 that our joint control problem can be decomposed into two separate sub-problems of routing and rate control. Multiple algorithms solving each of the two parts have been proposed. In this chapter, we refer to suitable solutions and results developed by other researchers for the purpose of adaptive rate control and routing. In the next chapter, we will discuss how these algorithms could be combined to attain the global objective defined in the previous chapter.

4.1 Scope

We need to emphasize that we attempt to provide an exhaustive survey of *neither* of the two domains of adaptive routing and rate control. We consider only a small part of the research done and leave much out, regarding it out of this thesis' scope. The routing algorithms we seek for are:

- *optimal*, or sub-optimal, i.e. attempt to optimize or approximate a well defined cost-based goal, either selfishly or socially
- *adaptive*, i.e. react to dynamic network state (traffic pattern, resource availability), not just topology
- *hop-by-hop*, i.e. comply to the datagram routing paradigm where router only decides on the next-hop neighbour

The rate control algorithms we consider are:

- *optimal*, i.e. attempt to optimize a goal defined in terms of throughput-related utility functions
- *feedback-driven*, i.e. base their actions on implicit or explicit, quantitative cost-related signals

Most importantly, we discuss only *online* algorithms, without full knowledge of the system (utility and cost functions), but only its current state. Still, the review presented here is far from complete.

Let us point out that since the problem is convex, many general optimization methods are available. We refer the interested reader to [BNO03]. However, such methods are generally designed to work *offline*, having perfect knowledge about the system.

We shall begin with adaptive routing, subsequently discussing a few methods for marginal cost estimation, and proceed to rate control and joint algorithms that directly inspired this work. To avoid confusion, we will translate the concepts of cited authors to our notation whenever possible. We mostly classify the algorithms by the influential work that they are based on.

4.2 Adaptive routing

In section 2.3.1, we have pointed out the difference between routing for dynamic topology and routing for dynamic load and capacity. In this section, we shall relate to distributed algorithms for stochastic or proportional routing, since they are compatible with the routing paradigm specified in Chapter 3. Segall proposes several different models for adaptive routing in [Seg77].

We indicated in section 3.2.1 that there has been extensive research done targetting the problem of minimizing the aggregate cost or individual user costs. We discussed the difference between "system-optimality" (the *social* approach) and "user-optimality" (the *selfish* approach) in section 2.2.3. Such models assume that the user demand (load) is given as input, and cannot be regulated.

The cost function addressed by multiple researchers is the (expected) end-to-end message delay. The benefits of using delay as the overall performance indicator is that it intrinsically covers multiple aspects of network operation, e.g. providing best service, load balancing (avoiding congestion). Some of the algorithms discussed here address the *selfish* optimization problem, but as shown in section 2.2.3 could be adopted for *social* (i.e. global) optimization.

We begin with the seminal work by Gallager on minimum-delay social routing and its derivatives. Next, we discuss a different approach to minimum-delay selfish routing proposed by Gupta and Kumar. A substantial part of this section is dedicated to routing algorithms based on reinforcement learning including ant colony optimization. We finish this section with a short discussion on how marginal cost essential to social routing objective can be estimated.

Since most of the algorithm described below perform adaptation steps locally, we will assume for later convenience that the adaptation is performed at node i for destination d and drop these

indices in our considerations. Thus:

$$\mu_j = \mu_{i,j}$$
$$p_j = p^d_{i,j}$$
$$t = t^d_i$$
$$\lambda = \lambda^d_i$$
$$\text{but } \lambda_j = \lambda^d_j$$

4.2.1 Gallager's minimum delay

As a starting point, we present Gallager's work on minimum-delay routing using distributed computation [Gal77]. Although his algorithm is similar in its workings to the one proposed earlier by Agnew in [Agn76], Gallager deals with more general multi-commodity networks. The goal is to find routing probabilities that minimize *the total expected delay of all messages per unit time*. Gallager's algorithm solves the problem of *optimal routes*, minimizing C_T over routing probabilities p as defined in Section 3.3.2, provided that $\mu_{i,j}$ is the marginal *delay* of the link (i, j). The optimality conditions developed by Gallager match the Wardrop's second equilibrium definition. Using our notation:

$$\frac{\partial C_T}{\partial p_j} = t\left(\mu_j + \frac{\partial C_T}{\partial r_j}\right) =$$
$$= t(\mu_j + \lambda_j) \quad \begin{cases} = \delta & \text{if } p_j > 0 \\ \geq \delta & \text{if } p_j = 0 \end{cases}$$

for some δ (defined for the particular node and destination under consideration). Therefore, this algorithm constitutes one candidate that could be employed to solve the routing part of the joint objective.

Gallager's algorithm is a subgradient method and proceeds by making small re-routing steps at all nodes and for all destinations at a time. Thanks to a special blocking technique, the routing is loop-free at every instant. Loop-freedom allows for easy distributed marginal cost computation: the node can compute its cost as soon as all nodes that are *downstream* (i.e. are used for further forwarding) to the specific destination reported their costs.

Both the original algorithm and its derivatives adapt the routes by promoting the best routes. In a single re-routing step each node boosts the routing probability of the best of its neighbours. The boost in the routing probability to the best route neighbour is comprised of cuts among remaining neighbours. Hence, no re-normalization to ensure that p remain in the probability simplex [1] is necessary. Fur future convenience, let us define k to be the best neighbour of node i for destination d:

$$k = \text{argmin}_j\{\mu_j + \lambda_j\}$$

[1] positive and summing up to 1

Also, we will denote by Δ the cuts in probabilities, so that

$$\Delta_j \leq p_j$$
$$p_j \leftarrow p_j - \Delta_j$$
$$p_k \leftarrow p_k + \sum_j \Delta_j$$

In the Gallager's algorithm, the cuts among remaining neighbours are proportional to the excess in marginal cost above the best (minimal cost) neighbour, and inversely proportional to the flow rate, so that the change in load after a single step is bounded:

$$\Delta_j \quad = \min\left[p_j\,,\,\eta\frac{a_j}{t}\right]$$
$$\text{where } a_j = (\mu_j + \lambda_j) - (\mu_k + \lambda_k)$$

To maintain loop-freedom, the algorithm maintains in a distributed fashion a blocked set of routing probabilities that are forbidden from being raised above 0, as it would create a loop. Gallager provides a proof that the algorithm always converges to the optimal equilibrium if one exists, provided that the convergence speed factor η is sufficiently small.

Vutukury and Garcia-Lunes-Aceves point this dependence on small fine-tuned adaptive steps as a shortcoming of Gallager's algorithm. Instead, they propose in [VGLA99] a sub-optimal algorithm for the same problem that offers a much improved speed of convergence. Their solution is two-phased: first, a set of multiple loop-free paths for each source and destination is created using distributed computation; afterwards, adaptive adjustment of routing proportions is performed. The path-setup part does not consider traffic load, and hence it is run only when the network topology changes. Since this preselection of paths might preclude a path that is used in the optimal routing, the second part works in a restricted environment, and the whole algorithm is sub-optimal. However, the authors claim that the overall performance in terms of goal is not much hindered, while the speed of convergence and robustness (e.g. no parameters depending on network topology) is greatly improved.

The original definition of marginal cost at node i (λ) is modified into marginal *distance* so that:

$$\lambda = \min_j \{\mu_j + \lambda_j\}$$

Then the forwarding set (for that node and destination) $S = \{j|p_j > 0\}$ is restricted to subsets of $\{j|\lambda > \lambda_j\}$. The λ-ordering ensures loop-freedom. Furthermore, this new definition of λ implies that μ is an additive metric and any shortest-path algorithm can be used to compute λ given μ, e.g. Dijkstra, Bellman-Ford. Their path-selection algorithm is a link-state routing algorithm, specifically tailored to minimize the amount of exchanged information and maintain multiple paths. The second part of the framework dynamically distributes traffic over the preselected paths using heuristics for both initial allocation and incremental adjustments. The initial allocation prefers the neighbours of smaller costs:

$$p_j \leftarrow \left(1 - \frac{\mu_j + \lambda_j}{\sum_{l\in S}(\mu_l + \lambda_l)}\right)\frac{1}{|S| - 1}$$

Like in the original Gallager's algorithm, the incremental adjustment procedure is boosting the probability of routing to best among neighbours and matching the increase with cuts over remaining neighbours. Each cut is effectively computed as:

$$\Delta_j = \frac{1}{2}\, a_j \, \min_l \left\{ \frac{p_l}{a_l} \right\}$$

so that $\Delta_j \leq p_j/2$ for all $j \neq k$.

A different approach to improve Gallager's algorithm was proposed by Bertsekas *et al.* in [BG92]. They address the same deficiency of Gallager's method: its reliance on small steps driven by a scaling parameter η, which needs to be tuned for each network and user demand (also called *input*). Bertsekas *et al.* first generalize the original algorithm. If at each step the probabilities are adjusted $p_j \leftarrow p_j + \Delta_j$, then the (vertical) vectors $\mathbf{\Delta}$ are any solution to the problem:

$$\begin{aligned} &\text{minimize} \;\; \delta^{\mathrm{T}}\, \mathbf{\Delta} + \alpha\, t \mathbf{\Delta}^{\mathrm{T}} \mathbf{M} \mathbf{\Delta} \\ &\text{subject to} \;\; \mathbf{p} + \mathbf{\Delta} \geq 0 \qquad \sum_j \Delta_j = 0 \qquad \forall_{j \in B} \Delta_j = 0 \\ &\text{where} \quad \delta_j = \mu_j + \lambda_j \end{aligned}$$

$\mathbf{M}$ is some symmetric matrix, scalar α is a positive parameter, and B is the *set of blocked nodes*. The definition of blocked set is slightly different from Gallager's. The matrices $\mathbf{M}$ determine how the adjustments in probabilities Δ_j depend on route-through-neighbour costs δ_j. The conditions basically restrict the performed changes to such that preserve the routing probabilities in the probability simplex and prevent looping.

The choice of $\mathbf{M}$ defines a whole class of algorithms, and a specific selection yields the original algorithm by Gallager with the small redefinition of B. The main convergence result of their work is that if $\mathbf{M}$ satisfy a simple condition, namely for some positive scalars λ and Λ:

$$\begin{aligned} &\mathbf{M} \geq \Lambda \\ &\lambda |\mathbf{v}|^2 \leq \mathbf{v}^{\mathrm{T}} \mathbf{M} \mathbf{v} \qquad \text{for all } \mathbf{v} \text{ such that } \sum_l v_l = 0 \end{aligned}$$

then there exist a range of α such that the algorithm converges preserving its initial loop-freedom, regardless of the network topology and input. The authors propose to use diagonal $\mathbf{M}$ of $\overline{p}_j/(t)^2$ where $\overline{p}$ is an upper bound obtained using second cost derivatives. The resulting Δ vectors are computed iteratively using Lagrange's method. Although this method substantially alleviates the η-stepsize problem of Gallager's original algorithm, it does not guarantee convergence. Hence, the authors try to correct this with approximate Newton's optimization method. Recall the Newton's method in optimization is driven by both first and second derivatives:

$$\begin{aligned} &\text{for single dimension} \qquad x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)} \\ &\text{for multiple dimensions} \qquad x_{n+1} = x_n - \gamma [Hf(x_n)]^{-1} \nabla f(x_n) \end{aligned}$$

where H is the Hessian matrix of second partial derivatives. However their method resembles that of Newton, it has been tailored for real-time, distributed and loop-free operation.

4.2.2 Gupta and Kumar's STARA

A simple approach to Wardrop routing was conceived by Gupta and Kumar for wireless networks [GK97]. Although their algorithm (STARA) is designed for user–optimal (i.e. selfish) routing, the adaptation procedure could be adopted for system–optimal routing with a switch of link metrics.

Algorithms based on STARA are driven by the differences between the cost of regarded route and the weighted average cost among routes in use. For convenience, let us denote the difference between the total cost of all routes in use by i for destination d and the cost of the route offered by neighbour j as:

$$\theta_j = \lambda - (\mu_j + \lambda_j) \text{“‘“} (\mu_j + \lambda_j)$$

All of the algorithms presented here require normalization of p. Therefore, let use denote by $[.]^+$ the projection onto the simplex of probability vectors. The projection $p^* = [p]^+$ is the solution to the problem:

$$\begin{aligned} &\min \sum_j (p_j^* - p_j)^2 \\ &\text{subject to } \sum_j p_j^* = 1 \\ &p_j^* \geq 0 \end{aligned}$$

In the original STARA, the adjustment in routing probabilities is simply:

$$p_j \leftarrow p_j + \alpha\, \theta_j$$

Although this fact is not mentioned by the authors, this scheme can yield p out of the probability simplex, and therefore normalization is required. The authors did not discuss how to choose the appropriate value for α. A valuable contribution by Gupta and Kumar is their method to obtain delay estimates which is unidirectional and based on timestamp offsets, assuming only that the discrepancy in individual clocks' speed is negligible. Unfortunately, this scheme cannot be directly applied to obtain marginal delay estimates.

Recently, Raghunathan and Kumar have built a practical implementation directly on STARA in [RK04]. The original scheme is modified in two ways. Firstly, the adaptation step uses two sets of routing probabilities, of which only one is actually used for routing. Using our notation (p are used for routing and q are virtual):

$$\begin{aligned} q_j &\leftarrow [q_j + \beta\, p_j\, \theta_j]^+ \\ p_j &\leftarrow (1 - \epsilon)\, q_j + \epsilon\, \frac{1}{|Nb|} \end{aligned}$$

Thus the adjustment step of virtual probability is sized proportionally to the respective routing probability, and the routing probability distribution always spreads ϵ over all neighbours that

can be used for routing to the specified destination (the set Nb is defined for each node i and destination d). This is to ensure the minimum amount of probing of unutilized routes. The two-plane scheme is based on work of Borkar and Kumar [BK03]. The concept to keep some miniscule load on all routes for the purpose of probing is commonly used in algorithms based on reinforcement learning discussed in the next subsection. Secondly, Raghunatan and Kumar address multiple practical issues such as speeding up convergence and maintaining loop-freedom while converging. The convergence is sped up restricting the forwarding neighbour sets (i.e. the subset of neighbours that can be used for forwarding to a particular destination) in a fashion much similar to the one of Vutukury and Garcia-Lunes-Aceves described above [VGLA99]. The only allowed forwarding neighbours are the ones with not-greater hop-count distance to the destination. Note, that this renders the resulting algorithm sub-optimal. However, as the domain of their work is wireless meshes, this sub-optimality might be negligible. Their algorithm ensures loop-freedom by introducting two packet states, "odd" and "even", which are alternated as the packet traverses the network. When the packet is "even" it can be routed to any neighbour in the forwarding set (i.e. of no farther in hops from the destination), but when it is "odd", it can only be routed to a neighbour of smaller hop-count distance to the destination.

Another work based on [BK03] is that by Xie *et al.* [XQYZ04]. They evaluate the scheme for both user– and system–optimal routing through simulations. It is comprised of two components. Firstly, the delay or marginal delay is "learned" using from the current sample μ^* the exponentially-weighted average similarly to [GK97,RK04]:

$$\mu_j = (1-\alpha)\,\mu_j + \alpha\,\mu_j^*$$

The cost λ is computed as usual. Next, the virtual probabilities are "learned":

$$q_j \leftarrow [q_j + \beta\,(q_j\,\theta_j + \xi_j)]^+$$

where ξ_j are i.i.d. [1] random vectors distributed on an appropriate unit ball, which purpose is to add disturbance so that non-Wardrop equilibria are avoided. The actual routing probabilities p are computed from q as in [BK03, RK04] and described above. It should be emphasized that the algorithm developed by Borkar and Kumar [BK03] requires the learning factors α and β are positive, non-increasing, of infinite sum, and finite power, and also that at learning step n they are such that:

$$\sum_n ((\alpha(n) - \alpha(n+1))/\alpha(n))^r < \infty$$

$$\sum_n (\beta(n)/\alpha(n))^s < \infty$$

$$\text{for some } s, r \geq 1$$

This yields $\beta \to 0$ as $n \to \infty$. Thus, at least β needs to be reset whenever the input (i.e. user demand) or network capacity (i.e. topology) changes.

[1] independent, of identical distribution

4.2.3 Reinforcement learning

Machine learning is an area of artificial intelligence concerned with the development of techniques which allow computers to "learn", in other words, improve automatically through experience. This experience comes from the analysis of data sets. *Reinforcement learning* is a class of problems in machine learning which postulate that an *agent* is exploring a dynamic *environment* through trial-and-error interactions with it. In return of the agent's actions the environment returns a reward (either positive or negative). Reinforcement learning differs from the supervised learning problem in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected. Further, there is a focus on on-line performance, which involves finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge). A good reference is available from Sutton and Barto [SB98], and survey of reinforcement learning is available from Kaelbing *et al.* [KLM96].

The influential work we begin with is Q-routing by Boyan and Littman [BL94]. The task of the packet routing policy is to answer the question: *to which neighbour should the packet be forwarded so that it gets to its eventual destination as quickly as possible?* Thus, the problem they address is that of selfish minimization of end-to-end delays. The authors point out that, although the policy is rewarded according to its performance, i.e. the time it took to deliver the packet, which can only be measured when the packet reaches its destination, the policy can be updated more quickly using local reinforcement. Suppose node x is given the task to forward a packet P addressed to destination d. The total delay of routing is comprised of queuing delay at each node and propagation delay over links. Let $Q_x(d, y)$ be the time that x estimates it takes to deliver P if its forwarded to x's neighbour y. Upon sending a packet to y, x immediately receives y's current estimate for the remaining trip:

$$t = \min_{z \in Nb(y)} Q_y(d, z)$$

If P spent time q at x (queuing), time s on the link (x, y) (propagation), then the new estimate for $Q_x d, y$ is $q+s+t$ which is used as the reinforcement signal in a scheme derived from Q-learning (see [SB98]):

$$\Delta Q_x(d, y) = \eta(q + s + t - Q_x(d, y))$$

where η is the "learning rate". This resembles the relaxation step of Distributed Bellman-Ford algorithm (see Section 2.3.1).

Q-routing as a reinforcement learning problem suffers from the conflict between exploration and exploitation. If the routers always use the neighbour y that is charted with smallest $Q_x(d, y)$, then they fully exploit the knowledge learned to-date, but fail to properly explore the action-space. Specifically, since only the neighbours and links used for routing report their updated states (delays), the node fails to notice the appearance of better choices among unused neighbours. To support exploration, reinforcement learning algorithms often involve random actions or some minimum probing. Random actions, however, can spoil the overall performance of the

system. Q-routing uses a scheme called "full-echo" which requires the nodes to ask all neighbours for t before the routing decision is made. This scheme resembles minimum probing by the actively routing nodes.

An improvement over Q-routing was proposed by Kumar and Miikkulainen in [KM97]. Dual Reinforcement Q-routing (DRQ) makes use of the observation that when packet P from source s to destination d is received at y from neighbour x, then y can get up-to-date information about the time it took the packet to get there from s. Therefore, they employ both forward and backward exploration. When forwarded to y, packet P also carries the x's estimate for the trip to s:

$$t = \min_{z \in Nb(x)} Q_x(s, z)$$

Node y uses this to update $Q_y(s, x)$:

$$\Delta Q_y(s, x) = \eta(q + s + t - Q_x(d, y))$$

where q, s are y's estimates for queuing and propagation delays. Experimental results show that DRQ is twice as good as Q-routing in terms of adaptation speed and average delay.

If we were to adopted Q-routing for our purpose, we would use $\mu_{i,j}$ in place of $q(i)+s(i, j)$ and $\mu_{i,j} + \lambda_j^d$ in place of $Q_i(d, j)$. The probability distribution over neighbours favors the neighbour k of smallest $Q_i(d, k)$.

4.2.4 Ant colonies

Many reinforcement learning algorithms employ metaheuristics such as simulated annealling, tabu search, evolutionary computation, and so on. Among nature-inspired techniques, there is *swarm intelligence* based around the study of collective behaviour of distributed, self-organized systems. Examples of systems like this can be found in nature, including ant colonies, bird flocking, animal herding, bacteria molding and fish schooling. Although such systems are typically made up of simple agents interacting only locally (with one another and the direct environment), these interactions lead to the *emergence* of global behaviour. As such they are particularly suited for managing a "herd" of network routers that can only interact with neighbours. Ant Colony Optimization (ACO) is the most successful swarm intelligence technique employed for routing in packet networks. See the book by Dorigo and Stuetzle [DS04] for reference on both theory and applications. ACO works on problems that can be reduced to finding best paths in graphs. Artificial ants build solutions by moving around on the problem graph and leaving trails of attractive *pheromones*. The ants that visit the same place again, are more likely to follow the paths of stronger reinforcement. Hence, the ant agents use *stigmergy* to communicate, and require very limited memory. Eventually, all ants follow the globally best path. The advantage of ACO over many other metaheuristics is that it works well in dynamically changing graphs, adapting to the changes on–line. This is particularly desirable in network routing.

A simple ant-driven algorithm was proposed for adaptive routing by Subramanian *et al.* in [SDC97]. It uses small, fixed-size ants to probe the current state of the network, i.e. link costs. As the ant is routed through the network, it accumulates the reverse cost of its path, i.e. when it is sent over a link (i, j), the cost information it carries is increased at j by the cost of reverse link (j, i). Thus, an ant injected at d and seen at j carries the cost of a path from j to d. The ant causes adjustment in routing probabilities of the routers it visits, i.e. leaves a trail. An ant of origin d received at node i from it neighbour k carrying cost c makes an update:

$$p_{i,j}^d = \begin{cases} \frac{p_{i,j}^d + \Delta p}{1+\Delta p} & \text{if } j = k \\ \frac{p_{i,j}^d}{1+\Delta p} & \text{otherwise} \end{cases}$$

where $\Delta p = k/f(c)$. Factor $k > 0$ is the learning rate, and $f(c)$ is a non-decreasing function of c. Thus, the link just used receives a reinforcement in the routing probability. Note that the updated p remain in the probability simplex. Subramanian *et al.* distinguish two types of ants by how they are forwarded. *Regular ants* are forwarded according to the routing probability distribution, i.e. better paths get more ants. *Uniform ants* are forwarded using a uniform distribution, i.e. all neighbours have equal chances. They show that regular ants converge to the same result as Q-routing, but lose exploratory properties afterwards, because "good news" (i.e. improvement in unused paths) are not propagated at all. In contrast, the uniform ants occupy all possible routes in proportion to the available capacity. The authors propose a hybrid scheme which allows the regular ants to be routed uniformly $f\%$ of the time, for some small f. This noise affects only ants and not the data packets. One advantage over classic shortest-path algorithms indicated by authors is that the routing traffic (or control overhead) is independent of network dynamics.

A more sophisticated AntNet was conceived by Caro and Dorito in [CD97, CD98]. AntNet uses both forward and backward ants. Forward ants memorize the path they traverse using a memory stack, on which each node pushes its identifier, the delay and congestion information of the used link. Forward ants are routed same as data packets, using the current routing probability distribution over the neighbours that have not yet been visited (not on its stack). If no such neighbours are left, the ant is routed randomly, and the loop that is formed that way is popped from the stack. When the ant reaches its destination, it changes into a backward ant. The backward ant uses the node identifiers popped from the stack to follow the route of the original ant in reverse. During the backward travel the ant updates the local models of the network state and the routing probabilities of the visited nodes. The ants are destroyed, once they reach their initial source. Besides the routing probability distribution, each node maintains model of the current network state $M(\mu_d, \delta_d)$, containing the estimated mean and variance of the times to reach each destination d. The forward ants are initiated at s to random destination d chosen according to the current demand of flows going from s to d. The backward ants update M with the new time sample o using reinforcement learning: $\mu_d \leftarrow \mu_d + \eta(o - \mu_d)$ and $\delta_d \leftarrow \delta_d + \eta((o - \mu_d)^2 - \delta_d)$ The backward ant coming back from destination d through node

k update the routing tables at node i. For all destination nodes traversed by the forward ant, i.e. for all $d' \in i \rightarrow d$:

$$p_{i,j}^{d'} \leftarrow \begin{cases} p_{i,j}^{d'} + r\,(1 - p_{i,j}^{d'}) & \text{if } j = k \\ p_{i,j}^{d'} - r\,p_{i,j}^{d'} & \text{otherwise} \end{cases}$$

If the reinforcement signal (the goodness measure) r is constant then only the implicit signal (ant rate) is used. The authors propose a more elaborate scheme that scores r using the values of M. For example, if the ant carries new time sample o and the currently known mean is μ, then μ/o could be used to evaluate how out-of-scale the new sample is. Their simulation results indicate that AntNet offers a much better reactivity and adaptivity over classic shortest-paths algorithms (distance-vector and link-state) at the price of increased routing overhead. We refer the reader to Kassabalidis *et al.* for a survey on other routing algorithms based on swarm intelligence [KESI+01].

4.2.5 Marginal cost estimation

As shown in section 2.2.3 on the consequences of non-cooperative routing, essential to the socially-oriented routing objective is the marginal resource cost. In section 2.3.1, we indicated that marginal cost metrics are particularly troublesome to use, as they rarely can be measured directly.

In his original work [Gal77], Gallager suggests a function derived from an analytical model could be used. For his off-line algorithm in [Kle64], Kleinrock proposed a delay function based on an M/M/1 queuing model. The M/M/1 is the analytical model for a single server with unbounded queue in which both request inter-arrival and service times are exponentially distributed (i.e. a Poisson process). When the average rate [1] of requests is λ and the average service rate is μ then the average queuing delay is $\frac{1}{\mu-\lambda}$. Thus, the cost function for the links takes the form: $C_e(z) = \frac{z}{Cap(e)-z}$. Unfortunately, the assumptions for M/M/1 are far from being fulfilled in the IP packet networks, in which the traffic is self-similar rather than Poisson.

We need to point out that, if the chosen cost function is given by a *closed-form formula* of link load, then obtaining marginal estimate is only as difficult as it is to measure the load. The total incoming flow rate can be estimated using, for example, observation windows. The estimate is the amount of inbound data recorded over that fixed time divided by window length, and possibly averaged exponentially [2] to smooth out any noise.

However, if such function form is unavailable, then the direct and indirect effects of the cost need to be measured so that the marginal cost can be deducted. For example, if the cost is the *queuing* delay, then it could be estimated by timestamping the packets when they enter the queue and observing them as they depart. But to measure the marginal delay directly, one would need to observe how the average delay changes with load.

[1] The average rate is the reciprocal of average time between consecutive events.

[2] See Section 2.3.1.

Segall considers in [Seg77] methods to estimate the marginal queuing delay $dD(f)/df$ of a link that would not require actual deviation in the load, but rather base on the record of arrival and departure times over the interval between routing updates, and recursively update the current estimate, so that its memory-requirement is reduced. His method requires no assumptions whatsoever about the nature of the statistics or other parameters of the traffic, just that the packets are processed on a first-come-first-served basis (FCFS). The procedure estimates the effect of a hypothetical decrease of δf in the current incoming flow rate. The particular algorithm by Segall is sometimes called "Customer Rejection". Suppose that incoming packets are rejected with probability ϵ, then the decrease in flow rate is $\delta f = \epsilon f$ [1]. If c_m^n is the amount of time that the mth packet would save if the nth packet were to be removed, then a simple recursive formula to compute c_m^n from the record of arrival and departure times a_n and d_n respectively can be formulated:

$$\begin{aligned} c_m^n &= 0 \qquad \text{for } m < n \\ c_n^n &= d_n - a_n \\ c_{n+1}^n &= d_n - \max(a_{n+1}, d_{n-1}) \\ c_m^n &= \min(c_{m-1}^n, d_{m-1} - a_m) \qquad \text{for } m > n+1 \end{aligned}$$

A summation of c_m^n for all m, n within the observation period gives $\delta D(f)$, which divided by δf estimates the marginal delay.

Cassandras *et al.* propose a different marginal delay estimation procedure based on the same set of simple assumptions in [CAT90]. They use a technique known as *perturbation analysis* (PA), originally developed to efficiently estimate the performance sensitivities of complex discrete systems. The main improvement is noticed in the convergence speed to a stable estimate. Moreover, the authors compare through simulations their PA algorithm with analytical approximations with queuing theory models. Cassandras *et al.* also study the possible application of this technique to improve the convergence of the algorithms by Gallager [Gal77] and Bertsekas *et al.* [BGG84], described earlier in this section, by selecting an appropriate step size η.

The drawback of both methods described above is that they only measure the queuing delay, while there are other factors that contribute to the total end-to-end delay as perceived by user packets. In particular, if the link propagation delay is load-dependent, we need other ways to determine what the marginal cost is. Guven *et al.* propose in [TKL+04] a method that measures the marginal costs by actually introducing flow rate deviation (stochastic noise). Their technique is based on Monte Carlo estimation.

[1] since the chances of removing more than one packet are negligible

4.3 Rate control

In this section, we relate to rate control solutions proposed by other researchers for the purpose of utility optimization. Note that there are many rate control algorithms intended for congestion control, but those utility-driven are of more interest to us, as they target an objective similar to (3.5). The gross majority of the work discussed here was either part or inspired by Kelly's work on utility-based rate control. Also, we relate here to the work that directly inspired this thesis by Kelly and Voice.

4.3.1 Kelly's decomposition

The seminal work was done by Kelly *et al.* [Kel97, KMT98]. The *Kelly's system problem* assumes single–path routing and can expressed using our terminology is:

$$\max \sum_s U_s(y_s)$$
$$\text{subject to } z_e \leq Cap(e)$$

It can be decomposed into two sub-problems of which one is solved individually by all users and the other is solved by the network. Each user is responsible for establishing the price it is willing to pay if its unit price for service (throughput) is λ_s. The user's problem is:

$$\max U_s(w_s/\lambda_s) - w_s$$
$$\text{over } w_z \geq 0$$

The network does not need to know each user's utility functions but just their declared prices. The network's goal is that of weighted proportional fairness:

$$\max \sum_s w_s \lg y_s$$
$$\text{subject to } z_e \leq Cap(e)$$

They show that there exist unit prices λ_s such that the solution to the decomposed problem optimizes the original problem. Their rate control algorithm operates as follows:

$$\frac{d}{dt} y_s = k \left(w_s - y_s \sum_{e \text{ used by } s} p_e(z_e) \right)$$

The factor k is a positive gain, and the function $p_e(.)$ gives the virtual price of link e. Kelly *et al.* show that this algorithm converges under a variety of pricing schemes, e.g. explicit notification, packet loss, delay. If $p_e(.)$ is the rate of feedback signals from link e then the algorithm is an AIMD scheme. La *et al.* propose different algorithms for the Kelly's problem in [LA02]. Also, a simple algorithm driven by binary feedback has been proposed by Kar *et al.* in [KST01].

4.3.2 Joint routing and rate control

Also in [KMT98], Kelly *et al.* extend the single-path system model to multi-path and propose a tailored algorithm. If $x_{s,r}$ is the amount of flow s routed over a route r, then:

$$\frac{d}{dt}x_{s,r} = k\left(w_s - y_s \sum_{e \text{ used by } r} p_e(z_e)\right)$$

Kelly and Voice study the scheme of joint rate control and routing in [KV05] and show that, if designed properly, both algorithms can operate at the same time-scale of round-trip times. Their objective is exactly the same as (3.5) addressed in this thesis, except that they assume that the routing is done over a set of preselected routes, while our objective uses stochastic routing. Therefore, their algorithm adjust $x_{s,r}$ - the amount of flow s routed over route r as follows:

$$\frac{d}{dt}x_{s,r}(t) = k_{s,r}x_{s,r}\left(1 - \frac{\lambda_r(t)}{U_s'(y_s(t-T_r))}\right)$$
$$\lambda_r(t) = \sum_{e \text{ used by } r} p_e(z_e(t-T_r))$$

where T_r is the round-trip time of route r. They define:

$$C_e(z_e) = \int_0^{z_e} p_e(z)dz$$

which implies that $p_e(.)$ is the marginal cost of link e. The rationale behind using T_r-delayed values of marginal utility stems from the fact, that when implemented, the marginal cost feedback that is obtained at time t was actually generated after the load injected at source reached the link and the back-signal reached the source. The authors assume that the signal actually needs to reach the destination and travels back in ACK packets. One viable way to implement such scheme, as discussed by the authors, is to put the current y_s in the data packets and move them to the ACK packets at the destination. A simpler scheme would simply monitor the rate of incoming ACK packets to determine what was the rate of the packets that those packets acknowledge, assuming that each data packet is acknowledged.

For the utility function they use α-fairness:

$$U_s(y_s) = \frac{w_s {y_s}^{1-\alpha}}{1-\alpha}$$

For the marginal cost function they use:

$$p_e(z_e) = \left(\frac{z_e}{C_e}\right)^{\beta}$$

which can be interpreted as the probability of marking a packet that arrives at a M/M/1 queue serviced at the average rate of C_e to find β packets already in the queue. Kelly and Voice

conclude that such scheme not only converges to the optimum but is also locally stable at the optimum if:

$$k_{s,r}T_r(\alpha_s + \max_{e \text{ used by } r} \beta_e) < \frac{\pi}{2}$$

Therefore, the gain parameter k_r should be adjusted to the round trip time and steepness of utility and cost functions. See also work by Johari and Tan [JT00] for a study on stability of joint routing and rate control.

Voice further studies the composition of rate control and routing in [Voi04]. The author considers the algorithm:

$$\frac{d}{dt}x_{s,r} = f_r(x_{s,r})\left(U'_s(y_s) - \sum_{e \text{ used by } r} \mu_e\right)$$
$$\frac{d}{dt}\mu_e = g_e(\mu_e)\left(z_e - c_e(\mu_e)\right)$$

properly bounded at 0, where f_r, g_e are positive and:

$$U'_s(y_s) \to 0 \text{ as } y_s \to \infty \text{ or } \int_0^\infty \frac{w}{f_r(w)}dw = \infty$$
$$c_e(z_e) \to \infty \text{ as } z_e \to \infty \text{ or } \int_0^\infty \frac{w}{g_e(w)}dw = \infty$$

Voice proves that such algorithm always converges regardless of the initial state, assuming that there are no propagation delays.

Chapter 5

Joint Algorithm

Having stated the global optimization problem and how it can be decomposed into two sub-problems in Chapter 3, and how other researchers approached them in Chapter 4, we now proceed to describe the joint algorithm targeting the whole problem. In this chapter, we discuss the composite structure of the proposed algorithm. We describe the requirements and design concepts of its components. A brief theoretical analysis is also provided. The main focus of the analysis is convergence and stability in the presence of delays. We also discuss the arising implementation issues focusing mainly on information propagation mechanisms.

5.1 Composition

Given the results of Chapter 3 we propose to compose the algorithm of two sub-procedures each achieving the optimal solution for its sub-problem:

1. find optimal routing probabilities assuming that load will not change
2. find optimal user rates assuming routing that will not change

We already mentioned in section 3.3.2 that for such composition to work it is sufficient that every run of each of the two procedures either achieves its optimum or simply improves in the global performance goal. The two algorithms need just to be run interleaved. Although such method may be slowly converging, in a dynamic environment, it is reasonable to accept solutions that are near-optimal but quick.

As shown in Chapter 4, both problems have been addressed previously, and we could use the algorithms proposed to-date as components. The crucial point of our composition is based on the observation that both algorithms will need an estimate for the marginal cost λ_i^d [1]. We propose the two components to *share* the common cost estimate c_i^d. So, provided that the adaptive routing

[1] Segall points out that marginal cost is needed for all minimum-cost routing algorithms [Seg77].

algorithm already gathers this information, the rate control algorithm needs no further effort, e.g. a backstream of ACK packets, in order to estimate it. This is advantageous for applications that do not require reliability (e.g. like that provided by TCP), but still should be subject to congestion control. Currently, streams that are not rate-controlled in any way are usually not TCP-friendly, and as such give rise to many issues, e.g. how to protect TCP-controlled flows from preemption and starvation.

Another possible scheme of application is to apply the rate control to the aggregate flow of all users hosted by a node (i.e. node is an aggregate user). In such case, the node manages the whole traffic it sources using a single utility function. Such scheme might be useful for wireless mesh networks, where routers are not operated by a single institution, but rather form a self-regulated *community*.

It should be pointed out that this algorithm is not intended to fully substitute for TCP-like end-to-end control. Specifically, it will not take care of flow control (i.e. end buffer protection) nor of necessary retransmissions if reliable delivery is required. Instead, it will only regulate the load on the resources of the network trying to balance between utility and cost, and most importantly, protect the resources which should result in reduced packetloss.

Last but not least, as shown by Kelly and Voice [KV05], a properly driven, joint algorithm operating at the same time-scale is viable and stable. This suggests that both components can reasonably use the same information updated no quicker than it takes messages to cross the network. We discuss this issue in greater detail in this chapter.

5.2 Distributed cost estimation

First, we discuss design issues of the distributed algorithm for (marginal) cost estimation. We consider several variants of the algorithm following the popular paradigms of distance-vector and link-state among others (see Section 2.3.1). Afterwards, we will discuss how this estimate can be used by the two component adaptation algorithms.

5.2.1 Distance-vector algorithm

Recall from Chapter 3 the cost estimation formula (3.8):

$$\begin{aligned} c_d^d &= 0 \\ c_i^d &= \sum_j p_{i,j}^d (c_j^d + \mu_{i,j}) \\ \mu_e &= C_e'(z_e) \end{aligned}$$

This suggests a straightforward algorithm following the distance-vector paradigm:

1. node i monitors its outgoing links and computes $\mu_{i,\cdot}$
2. i periodically advertises its c_i^d estimate for each destination d to its neighbours

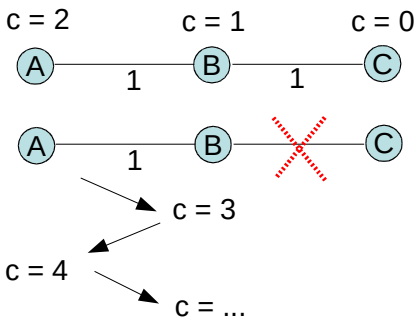

Figure 5.1: A trivial example of *counting to infinity*. The task is to obtain the correct cost to node C. When the link (B, C) fails, node B turns to its currently best neighbour A, which advertises cost 2. A loop is formed between A and B and the cost fails to converge.

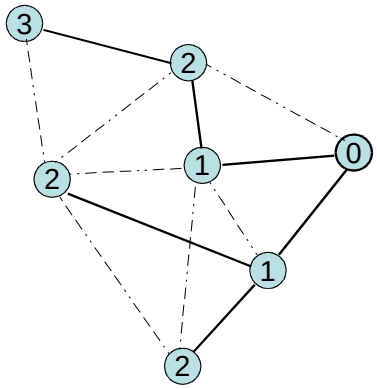

Figure 5.2: If the routing (thick lines) is loop-free (acyclic), then the routes to a single destination (thick border) form a spanning tree. The numbers on the nodes count the number of messages (or update periods) necessary to obtain up-to-date information about the route cost to the destination node.

3. using the advertisements from its neighbours, i updates c_i^d

While this algorithm is bound to converge to accurate values of λ_i^d, it suffers from the same deficiency as the Distributed Bellman-Ford (used in e.g. RIP). It turns out, the convergence time is infinite if temporary routing loops may exist. See Fig. 5.1 for illustration of such behaviour. However, if only finite accuracy is needed, then the algorithm converges within finite time, if there are no loops of probability 1 (see Section 2.3.1).

On the other hand, if the routing is loop-free then the number of advertisement messages necessary for the estimate to converge is equal to the hop-distance (over existing route) to the farthest node from the source of update, i.e. the node that noticed a change in locally observed μ. Supposing the advertisements are sent periodically, the sufficient number of periods equals the same bound, as illustrated on Fig. 5.2.

The route adaptation algorithm proposed by Gallager in [Gal77] preserves loop-freedom and ensures that the cost (in this case, delay) estimation procedure is complete before making re-

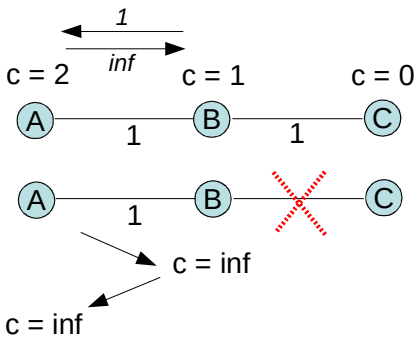

Figure 5.3: With *split horizon* the nodes do not advertise their cost to nodes that they use for routing. With *poison reverse* the nodes advertise their cost as infinity. Thus, A will advertise inf to B, because it was its next hop to C. The routing loop is avoided.

routing decisions. Thus, each node waits until all its downstream (i.e. towards destination) nodes report their costs on a recent update. A similar technique was used by Vutukury and Garcia-Lunes-Aceves in [VGLA99], although they proposed a different method to maintain loop-freedom. Also, see their work in [VGLA01] for a distance-vector algorithm that maintains loop-freedom at every instant and avoids counting to infinity.

Split horizon

To alleviate the convergence problem we propose a technique similar to *split horizon* or *poison reverse* developed for the RIP distance-vector protocol [Mal98], and used in EIGRP. The split horizon rule states: *Never advertise a route to a neighbour from which you learned it.* The poison reverse rule states: *Once you learn of a route through an neighbour, advertise it as unreachable back through that same neighbour.* The rules are illustrated in Fig. 5.3.

This technique can be tailored to stochastic routing. Since the node may have routes available through its other neighbours, it does not report the destination unreachable but only excludes it from its advertisement. The cost that node i advertises to node j is computed as follows:

$$c^d_{i,j} = \frac{\sum_{k \neq j} p^d_{i,k}(c^d_{k,i} + \mu_{i,k})}{\sum_{k \neq j} p^d_{i,k}} \tag{5.1}$$

Note, that, in the stable state of Wardrop equilibrium, this technique either does not affect the marginal cost estimation (if multiple neighbours are used), or is equivalent to poison reverse in the single-path case. Hence, the rate adaptation algorithm will not be skewed.

5.2.2 Link-state algorithm

Compared to the distance-vector approach, link-state algorithms are burdened with less severe convergence issues. The relatively simple flooding algorithm does not use routes and hence is loop-proof. As soon as the update is complete, the routing is guaranteed to be loop-free, provided that the routing logic is proper.

Most improvements to the link-state approach aim to reduce the control overhead, i.e. the amount of traffic due to flooded updates. For example, link-vector algorithm (LVA [GLAB95]) reduces the scope of advertised topology to "preferred" links. When link-state information needs to be updated, it is not flooded across the network, but instead the nodes only update their neighbours if the link is a part of their routing *source-tree*, i.e. the node intends to use it. Another approach, least-overhead routing reduces the number of updates by allowing stale link-state, thus sacrificing optimality, but favouring small routing overhead.

Although link-state algorithms, due to their generally better convergence, have been more successful than distance-vector algorithms in the area of intra-domain routing, some of their features make them unsuitable for adaptive routing. Given the knowledge of full topology and link metrics, the router may compute the lightest paths and decide unilaterally to use them for routing. Yet, the hop-by-hop paradigm allows them to choose the first hop only.

The cost estimate is accurate only if all participants agree on the routes. The problem arises from the load-dependence of the metric, which make such unilateral, fixed decisions undesirable, however quick. Worst-case scenarios include indefinite oscillations. Ideally, the routes would be adjusted in such steps that each slightly improved the overall performance. But this would require the routers to maintain knowledge on the transient state and render the computed lightest paths not immediately usable. Unfortunately, the transient state also comprises the currently used routes, i.e. routing probabilities, which makes flooding it all impractical.

To sum up, link-state algorithms are poorly suited for adaptive stochastic routing. Distance-vector approach gains its advantage from the fact that the nodes share pre-computed information and do not need to share the details of their routing decisions. Further, when performing small adaptation steps, distance-vector routers may reasonably assume that the impact of their local decisions is as if they were taken unilaterally. We refer the reader to section 2.2.3 for reference on non-cooperative routing.

5.2.3 Information propagation

The marginal link cost is load-dependent, and therefore highly dynamic when rerouting or rate adjustment is performed. It is crucial for the adaptive algorithm to maintain up-to-date information. In section 2.3.1, we discussed the differences between proactive and reactive update schemes, which are distinguished by how they determine which routes are needed. In this subsection, we discuss several update schemes defined by how the cost update is triggered.

Periodical updates

A purely proactive update algorithm is driven with periodical updates. The incurred overhead of routing updates is stable and does not depend on the traffic and capacity dynamics. However, the choice of the update period is essential and impacts the balance between stability and optimality. If the period is too short, the overhead may be unacceptable, but the information is

very up-to-date and the adaptation algorithm may drive the system close to optimal. However, if the updates are too frequent, and measurement is not sufficiently smoothened, the control algorithm may have stability issues. Conversely, if the period is too long, then the updates are cheap, but the adaptivity is hindered by out-of-date information. The inherent problem of periodical updates is the appropriate choice of the update interval.

Triggered flooding

A different method defines thresholds for change of the monitored link cost, and initiates a flood of updates when the threshold is passed. The advantage over the periodical scheme is that the information is updated as frequently as needed to perform necessary adjustments. However, such updates may be dangerous to stability unless dampened, i.e. slowed-down. Also, this scheme is more prone to state corruption, if a node losses correct cost information, it will not be updated until the triggering event occurs. Therefore, hybrid solutions driven with both timed and triggered updates are most promising of the two schemes.

Packet-driven updates

Reinforcement learning methods described in section 4.2.3 uses a different approach. The updates performed locally between neighbours are triggered by destination-oriented packets. In the Q-routing algorithm, the node updates its neighbour whenever a packet for the particular destination is received. In ant algorithms, the nodes are updated when visited by ants. Note that the update-triggerring packets travel between the source and destination (or back as in the case of backward ants). Therefore, such schemes are purely reactive. If there are no packets for a particular destination, then the cost is not updated, and the costs to the destination are not propagated to the areas of the network that host no sources addressing it. However, all reinforcement learning algorithms require a minimal level of probing to maintain their exploratory properties.

Conceptually, the overhead of packet-driven updates is constant with respect to the amount of user traffic, and does not depend on neither selected update period, nor the dynamics of the cost. The design difficulty of such schemes is that the cost information needs to travel in the *opposite* direction than the user flow.

Rate-implied feedback

An interesting approach to cost propagation was proposed by Kelly *et al.* [KMT98, Kel03]. The scheme is based on the current TCP behaviour. The TCP source either receives an ACK or recognizes a loss. The frequency of the binary signal determines how fast the source increases or decreases its rate. In case of TCP, the frequency strongly depends on the flow rate at source. But in general such simple binary signal of variable rate could be used to drive the adaptation procedure performing miniscule steps at each "tick".

Consider an idealization of such algorithm:

- each resource (link) maintains its current cost estimate μ
- μ determines the rate of periodical ticks that are flooded across the network
- each user observes the ticks and whenever it spots one from a resource it uses, it performs an appropriate, tiny adaptation step

This rough concept is expensive and requires the sources to know the resources their flows use and to what extent they use them (i.e. the flow part $x_{s,e}$, see Section 3.2.2). An optimized variant would use the knowledge of routing variables to disseminate the ticks only in the regions that contain interested sources.

Recall the cost estimation formula (3.8):

$$c_i^d = \sum_j p_{i,j}^d (\mu_{i,j} + c_j^d)$$

One idea the tick dissemination algorithm aims to exactly mimic that equation. For each link (i,j) and for each destination d, ticks marked with d are generated with rate $\mu_{i,j}$ and sent to i [1]. Furthermore, each node i on receiving a tick marked d from its neighbour j or link (i,j), drops it with probability $p_{i,j}^d$. The ticks that remain are broadcast to *all* neighbours. Then the rate of ticks marked with d observed at node i equals c_i^d as desired.

This algorithm is simple, but it is not quite intuitive, since the ticks are replicated (broadcast) by each node. A more sophisticated scheme basing on the observed rates would route the ticks in the manner of stochastic routing, i.e. under flow rate conservation. For each link (i,j) and for each destination d, ticks marked with d are generated at rate $z_{i,j}\,\mu_{i,j}$. Subsequently, each node that receives the ticks can distribute the ticks among neighbours in accordance to known composition of t_i^d. The routing probabilities for the ticks would be:

$$q_{i,j}^d = \frac{p_{j,i}^d\, t_j^d}{t_i^d}$$

$$q_{i,i}^d = \frac{r_i^d}{t_i^d}$$

Probability $q_{i,i}^d$ denotes the proportion of ticks that is directed to the sources hosted at i addressing d. Note that $\sum_j q_{i,j}^d = 1$. The tick rates from all resources add up at the sources and represent accurate cost information. The main drawback of this algorithm is that it requires knowledge (through monitoring) of t_i^d. However, this could be alleviated with a tick-for-packet feedback scheme resembling the packet-driven update method described previously.

The general disadvantage of such tick-based methods is that the routing overhead depends on the chosen base rate of ticks-to-cost. The impact of this choice is same as for update period or change threshold discussed above. Note we mention these methods here as a possible implementation concept but do not study them any further in this thesis.

[1] Actually, the node i would monitor the link (i,j) and generate the ticks.

5.3 Continuous adaptation algorithm

Multiple algorithms have been proposed for both sub-problems as we have indicated in section 5.1. We have discussed some of them in Chapter 4. In this thesis, we do not intend to improve on these algorithms. Instead, we generalize and theoretically study the desired behaviour under idealized conditions and propose one possible variant. That variant shall be used for experimental evaluation in Chapter 6.

5.3.1 Continuous model

In the description below, we assume that there are no information update delays in the system whatsoever. The adjustments are made continuously and are driven with information that is *always up-to-date*, i.e. the estimate c_i^d always represents the current marginal cost λ_i^d. Conversely, the effects of the adjustments are immediate.

This model seems highly unrealistic, since in a practical implementation we would need to deal with inherent information update delays. These delays are comprised of:

- load propagation - the time needed for the effects of the adjustment decisions to be felt at the resources
- cost measurement - the time needed to observe and estimate the current marginal cost value
- metric propagation - the time needed to propagate the new metric information to the decision-making nodes (see section 5.2.3)
- inter-adjustment intervals - the shortest time between subsequent discrete adjustment steps

However, this model approximates an implementation where the individual adjustments have negligibly small effect and are performed at a time-scale much larger than propagation delays. Such model basically matches our simulation environment that shall be used in Chapter 6, and also the model used for rate control algorithms described in section 4.3. Still, before we proceed to the evaluation we shall discuss how the algorithm could be tailored to deal with update delays.

5.3.2 Rate control

General considerations

In the continuous model, the rate control algorithm is stated as the time-derivative of the vector of flow rates defined as a function of current rates and cost estimates:

$$\frac{d\mathbf{y}}{dt}(t) = f(\mathbf{y}(t), \mathbf{c}(t))$$

or better yet, using only local information:

$$\frac{dy_s}{dt} = f_s\left(y_s, c_{src(s)}^{dst(s)}\right)$$

Note that the domain of rates is restricted, and therefore when $y_s = 0$, we should also require that $dy_s/dt = f_s(0,.) \geq 0$. However, recall from Section 3.2.2 that thanks to the required behaviour of $U_s(y_s)$, the local optimum of rate control must be within $y_s > 0$.

Theorem 5.1. *A sufficient condition for the rate control algorithm to converge to local optimum is:*

$$f_s(y, c)\ \left(U_s'(y) - c\right) \geq 0$$
$$f_s(y, c) = 0 \quad \Leftrightarrow \quad \left(U_s'(y) - c\right) = 0$$

Proof. Let us denote the global profit by $\mathcal{U} = U_T - C_T$. We can observe that

$$\begin{aligned}\frac{d\mathcal{U}}{dt} &= \sum_s \frac{\partial \mathcal{U}}{\partial y_s}\frac{dy_s}{dt} = \\ &= \sum_s \left(U_s'(y_s) - \lambda_{src(s)}^{dst(s)}\right)\ f_s\left(y_s, c_{src(s)}^{dst(s)}\right) = \\ &= \sum_s \left(U_s'(y_s) - c_{src(s)}^{dst(s)}\right)\ f_s\left(y_s, c_{src(s)}^{dst(s)}\right)\end{aligned}$$

since $dU_T/dy_s = U_s'(y_s)$ and $dC_T/dy_s = \lambda_{src(s)}^{dst(s)}$, and also as the model assumes that $c_i^d(t) = \lambda_i^d(t)$. Using the condition, we obtain that all elements in the sum are non-negative, and thus:

$$\frac{d\mathcal{U}}{dt} \geq 0$$
$$\frac{d\mathcal{U}}{dt} = 0 \quad \Leftrightarrow \quad \forall_s \frac{\partial \mathcal{U}}{\partial y_s} = 0$$

so that the profit is always increasing with the adjustments, unless the condition for local optimum given by Theorem 3.4 is reached. □

Therefore, the algorithm should aim to equalize the marginal utility and marginal cost by increasing the rate whenever the marginal profit (i.e. the difference between marginal utility and marginal cost) is positive, and decreasing when otherwise.

Algorithm

Let us now proceed to our algorithm. Inspired by Kelly and Voice [KV05], we propose that the sources adapt their flow rates using the marginal cost estimate in the following manner:

$$\frac{dy_s}{dt} = \alpha\ \left(U_s'(y_s) - c_{src(s)}^{dst(s)}\right) \tag{5.2}$$

for some $\alpha > 0$. As desired, they aim to equalize marginal utility and marginal cost estimate. Note that we do not need to consider the time derivative at $y_s = 0$, due to the assumption that

$U_s(y_s) \to -\infty$ as $y_s \to 0^+$, which implies that $U_s'(y_s) \to \infty$. Further, the marginal cost function is positive and increasing with y_s. This yields that $dy_s/dt \to \infty$, and hence y_s is bounded away from 0. Furthermore, observe that the algorithm:

$$f_s(y, c) = \alpha \left(U_s'(y) - c\right)$$

satisfies the conditions of Theorem 5.1.

Corollary 5.1. *The rate control algorithm (5.2) is converging to local optimum.*

We should emphasize that the choice of gain factor α in the idealized continuous model is of little significance beyond that the larger its value the faster the algorithm converges. However, the factor becomes crucial for stability when propagation delays or discrete time-scale need to be dealt with. Note that the idealistic model approximates the case when the changes of single step are extremely small, and the adjustments are extremely rare. The time of convergence of such scheme is likely impractical. We shall discuss these issues later in this section.

Before we proceed to the routing algorithm, let us point out that the algorithm (5.2) like all gradient–driven optimization procedures suffers from a rapid reduction in speed of convergence as it approaches the optimum. One way to alleviate this drawback is to scale the marginal profit by current rate:

$$\frac{dy_s}{dt} = \alpha \left(y_s + y_{min}\right) \left(U_s'(y_s) - c_{src(s)}^{dst(s)}\right) \tag{5.3}$$

The value $y_{min} > 0$ is used to prevent the algorithm from stopping at $y_s = 0$. This formula will be actually used in our evaluation in Chapter 6. Alternatively, we could forcibly bound y_s by introducing minimum allowed rate.

5.3.3 Routing

General considerations

Theorem 3.3 shows that the routing is optimal, if it is at the Wardrop equilibrium. Multiple algorithms have been proposed for Wardrop routing, e.g. [Gal77, GK97, VGLA99, XQYZ04]. Their design is mostly similar. The task is to increase the routing probabilities of the routes of smaller marginal cost and decrease the probabilities of those with larger marginal cost. Suppose the probabilities are updated continuously as in the idealized model described above. As in section 4.2 we consider the adaptation at node i towards destination d and drop these indices in our considerations, i.e. use $p_j = p_{i,j}^d$, etc. Also, let us define:

$$\delta_j = \mu_j + \lambda_j$$
$$\Delta_j = \frac{dp_j}{dt}$$

In order to keep the probabilities in the simplex, any feasible algorithm must satisfy:

$$\sum_j \Delta_j = 0$$
$$p_j = 0 \Rightarrow \Delta_j \geq 0$$

Theorem 5.2. *The sufficient condition for the route adaptation algorithm to converge to the optimum is:*

$$\sum_j \delta_j \, \Delta_j \leq 0$$
$$\sum_j \delta_j \, \Delta_j = 0 \quad \Leftrightarrow \quad \delta_j \begin{cases} = \lambda & \text{if } p_j > 0 \\ > \lambda & \text{if } p_j = 0 \end{cases} \quad \Leftrightarrow \quad \delta_j \geq \lambda$$

Proof. As in the proof of Theorem 5.1, we observe how the global profit $\mathcal{U}$ changes over time as the routes are adjusted. We have that:

$$\begin{aligned} \frac{d\mathcal{U}}{dt} &= -\frac{dC_T}{dt} = \\ &= -\sum_{d,i,j} \frac{\partial C_T}{\partial p_{i,j}^d} \frac{dp_{i,j}^d}{dt} = \\ &= -\sum_{d,i} t_i^d \sum_j (\mu_{i,j} + \lambda_i^d) \frac{dp_{i,j}^d}{dt} \end{aligned}$$

If the condition of the theorem is satisfied, then we easily obtain that $d\mathcal{U}/dt \geq 0$ and $d\mathcal{U}/dt = 0$ if and only if the routing is at Wardrop equilibrium defined by (3.13) which is the necessary and sufficient condition for optimum. □

Remark: We can observe that:

$$\exists_\varphi \, \forall_j \, (\delta_j - \varphi)\Delta_j \leq 0 \quad \Rightarrow \quad \sum_j \delta_j \, \Delta_j \leq 0$$

The value φ acts as the boundary between "good" and "bad" neighbours – those of cost higher than φ should have their probabilities reduced and vice versa. Algorithms that boost only the best neighbour can be seen as using such φ that cuts off only the single smallest δ_j.

Example: Consider a simple algorithm:

$$\Delta_j = p_j(\lambda - \delta_j)$$

Since $\sum_j p_j \delta_j = \lambda$ and $\sum_j p_j = 1$, we have $\sum_j \Delta_j = 0$. Also $p_j = 0$ implies $\Delta_j = 0$. Furthermore, λ works as the boundary φ introduced in the remark above, and $(\delta_j - \lambda)\Delta_j \leq 0$. This implies that $\sum_j \delta_j \, \Delta_j \leq 0$.

This would suggest that this simple algorithm has all desirable properties. Unfortunately, the algorithm stops whenever:

$$\delta_j \begin{cases} = \lambda & \text{if } p_j > 0 \\ \neq \lambda & \text{if } p_j = 0 \end{cases}$$

even if $\lambda > \delta_j$ for some j. Therefore, it stops whenever all used routes are equally good even if better unused routes exist. In fact, any algorithm that separates the neighbours into the "good" and "bad" sets along λ can stop at false equilibria, if proper care is not taken of such neighbours that $\delta_j = \lambda$. However, we can preclude such spurious equilibria by ensuring that if $\lambda > \delta_j$, or more generally when the system is not at Wardrop equilibrium (3.13), then $\Delta_j > 0$ for *at least one* neighbour j. The algorithms based on the Gallager's seminal paper assure this by always boosting the best neighbour. The algorithms derived from Gupta and Kumar's STARA assure this by bounding the routing probability from 0 (for the purpose of exploratory probing) or by using random noise that prevents getting stuck in spurious equilibria.

Algorithm

Let us now proceed to our algorithm. The main idea of the algorithm is to calculate the adjustments Δ_j in some proportion to $(\lambda - \delta_j)$. For simplicity, we do not strive to define the algorithm by proper Δ_j, but rather resort to simple re-normalization procedure in order to keep the routing variables within the probability simplex. Also, we scale the adjustments by the current cost estimates to keep them in a steady range.

Although this is undesirable, we shall ignore temporary loops. Corollary 3.1 implies and the results of our numerical evaluation confirm that all loops are eventually broken.

The continuous-time algorithm first computes:

$$\hat{\Delta}_j = \beta \frac{\lambda - \delta_j}{\lambda + \delta_j} \tag{5.4}$$

for some $\beta > 0$. Subsequently, simple re-normalization is performed. In the discrete domain, it would be done as:

$$p_j = \frac{\max(0, \hat{p}_j)}{\sum_k \max(0, \hat{p}_k)} \tag{5.5}$$

In the continuous domain, the algorithm computes:

$$\Delta_j = \frac{[\hat{\Delta}_j]_+ - p_j \sum_k [\hat{\Delta}_k]_+}{1 + \sum_k [\hat{\Delta}_k]_+}$$

$$\text{where} \quad [\hat{\Delta}_j]_+ = \begin{cases} \hat{\Delta}_j & \text{if } p_j > 0 \text{ or } \hat{\Delta}_j > 0 \\ 0 & \text{otherwise} \end{cases}$$

Note that, the requirements on Δ_j are satisfied. Also note that the normalization does not change the order of adjustments, i.e. if $\hat{\Delta}_j \leq \hat{\Delta}_k$ then $\Delta_j \leq \Delta_k$.

Theorem 5.3. *The route adaptation algorithm (5.4) is converging to a Wardrop equilibrium.*

Proof. We can observe that $\delta_j < \delta_k$ implies $\hat{\Delta}_j \geq \hat{\Delta}_k$, which in turn results in $\Delta_j \geq \Delta_k$. Therefore, $\sum_j \delta_j \Delta_j < 0$ unless $\Delta_j = 0$ for all j.

The algorithm stops only when $p_j \sum_k [\hat{\Delta}_k]_+ = [\hat{\Delta}_j]_+$ for all j. Let us show that this is only possible if $\delta_j \geq \lambda$. Now, $\delta_j < \lambda$ implies $[\hat{\Delta}_j]_+ > 0$, and would yield $p_j > 0$ in such a case. But this is only possible if there existed some neighbour k, such that $\delta_k > \lambda$ and $p_k > 0$. However, $\delta_j > \lambda$ implies $\hat{\Delta}_j < 0$, and for $\Delta_j = 0$ we would need $p_j = 0$. We obtain a contradiction.

Therefore, the algorithm stops only at Wardop equilibrium, i.e. when $\delta_j \geq \lambda$. Consequently, the algorithm satisfies the sufficient condition stated in Theorem 5.2. □

Example: Suppose that the node under consideration has a choice between the routes of its four neighbours of marginal costs 1, 2, 3, 4, and current probabilities 0.2, 0.4, 0.4, 0. Assuming that $\beta = 1$ the algorithm computes as follows:

j	1	2	3	4
δ_j	1.0	2.0	3.0	4.0
p_j	0.2	0.4	0.4	0.0
λ		2.0		
$\hat{\Delta}_j$	0.5	0.0	-0.25	-0.4
$[\hat{\Delta}_j]_+$	0.5	0.0	-0.25	0.0
$\sum_j [\hat{\Delta}_j]_+$		0.25		
Δ_j	0.34	-0.08	-0.26	0.0
$\sum_j \Delta_j$		0.0		
$\sum_j \delta_j \Delta_j$		-0.6		

5.3.4 Joint operation

With the description of the adaptation components, we have shown that both algorithms converge to local optima with respect to the two subproblems of the global objective (3.5). We can now consider the two algorithms run in phases, that is let one converge before running the other.

Corollary 5.2. *The joint* phased *algorithm is converging to the global optimum of (3.5).*

This is a direct consequence of theorems 5.1 and 5.3. The joint algorithm converges to a local optimum, and according to Theorem 3.1, the objective (3.5) is a convex optimization problem and therefore local optima are global.

However, we have also shown that the algorithms improve on the goal at every instant unless at the optimum. Therefore, we consider them run concurrently, in the same time frame.

Corollary 5.3. *The joint* concurrent *algorithm is improving on the goal at every instant and thus converging to the global optimum of (3.5).*

Obviously, this second scheme is much more practical for a distributed deployment, since it does not require any synchronization of phases. Further, it yields a more steady improvement in total profit, and faster convergence to a near-optimal state.

5.4 Delayed adaptation

In the previous section, we assumed a continuous model. The algorithms given there are only viable in an idealized environment, where the marginal cost estimate is accurate *at every instant*, i.e. updated continuously and without propagation delays. In this section, we discuss what the impact of non-infinitesimal adjustments and propagation delays is.

5.4.1 Non-infinitesimal adjustments

The algorithm defined by a time-derivative is impractical and can be employed as an approximation at most. In practical implementation, any adaptive algorithm performs adjustment steps periodically or in response to update events. Therefore, there is a non-zero time interval between the subsequent adjustments and the effect of a single step is not negligible. The rate control algorithm (5.2) becomes:

$$y_s[n+1] = \max\left(0,\, y_s[n] + \alpha\left(y_s[n] + y_{min}\right)\left(U_s'(y_s[n]) - c_{src(s)}^{dst(s)}[n]\right)\right) \tag{5.6}$$

and the route control algorithm (5.4) becomes:

$$\begin{aligned} p_j[n+1] &= p_j[n] + \Delta_j[n] \\ [\hat{\Delta}_j]_+ &= \max(-p_j,\, \hat{\Delta}_j) \end{aligned} \tag{5.7}$$

As a result, the choice of *gain* factors α, β that are used to scale the adjustments becomes crucial for convergence. If the gain factor is too large, then the optimal value may be easily *overshot*.

Gallager's algorithm uses $\beta \sim 1/\lambda$ in order to maintain steady rate of adjustments [Gal77]. The approach taken by Bertsekas *et al.* in [BGG84] is to approximate the Newton's optimization method and adjust the gain factors in accordance to the current derivative of higher order. We refer the reader to that work for an excellent analysis of discrete-step route adaptation algorithms.

5.4.2 Update delays

Even if employing continuous-time adjustment (i.e. time derivatives) is acceptable one should account for update delays. More to it, the effect of discrete-time procedure can be modelled with update delays. An algorithm:

$$x(t+T) = x(t) + f\left(x(t),\, c(t)\right)$$

where T is the update period, can be approximated with:

$$\frac{dx}{dt}(t) = \frac{1}{T} f\left(x(t-T),\, c(t-T)\right)$$

In their work, Kelly and Voice regarded a joint routing and rate control scheme that accounted for the propagation delays [KV05]. If we employed this concept in our rate control algorithm, we would obtain:

$$\begin{aligned} c_i^d(t) &= \lambda_i^d(t - T_r) \\ \frac{dy_s}{dt}(t) &= \alpha \left(U_s'(y_s(t-T_r)) - c_{src(s)}^{dst(s)}(t) \right) \\ \alpha &\sim \frac{1}{T_r} \end{aligned}$$

T_r is the "age" of the cost estimate, i.e. the time delay it takes any change in y_s to affect the cost estimate at the node that hosts user s, i.e. $src(s)$. Such algorithm is not only bound to converge, but is also locally stable at the optimum, provided that the gain parameter α is tailored in accordance to the steepness of utility and cost functions and T_r. The original idea assumed that the cost is obtained at the time-scale of round-trip time (RTT) (see Section 4.3.2). In section 5.2.3, we discussed several schemes to compute the cost estimate can in a distributed fashion. If the information is updated periodically, then the estimate's worst-case age is determined by the diameter of the network. If the information is flooded between neighbours, then the age is no greater than the delay-diameter [1] from the last flood. If the updates are packet-driven, then the age of the cost estimates for the routes in use equals delay-diameter times the hop-distance. This is due to the fact, that each packet moves the actual cost estimate one hop closer to the source. We should point out, that under stochastic routing multiple routes are used, hence the packets are subject to multiple RTTs. However, note that, if the cost function was chosen so that routing was *selfishly* minimizing the latency, then at the optimum all routes in use would have equal RTT.

Furthermore, we pointed out in section 5.1 that if the rate control algorithm is driven with cost estimates used for routing, then no backstream of ACK packets is needed. Therefore, the idea by Kelly and Voice to use the backstream as memory for $y_s(t - T_r)$ is no longer applicable. Also, the backstream cannot be used to obtain RTT estimates. Instead, end-to-end packet-driven updates could be employed. Whenever, a packet reaches destination, it induces a backward flood of cost updates. This update could be timestamped in order to estimate the "age" of the estimate.

In the end, in our simulation environment used in Chapter 6 to evaluate the presented concepts update delays are ignored, but the issue remains open and deeper study is needed, especially in the context of adaptive routing.

[1] Delay-diameter is the maximum end-to-end delay between any source and destination pair.

Chapter 6

Algorithm Evaluation

In this chapter, the algorithm proposed in Chapter 5 is subject to experimental evaluation. The purpose of the experiments is to show how the joint algorithm performs under idealized conditions. Therefore, we shall not perform any comparison between different algorithms, but rather focus on the properties of the objective and the resulting optimum allocation.

6.1 Simulation methodology

In this evaluation, we simulate idealized conditions of negligible propagation delays. Alternatively, this is equivalent to slow adaptation, in which the intervals between adjustment steps are longer than the time needed for the network to stabilize.

In such environment, we can use fixpoint solutions obtained in section 3.3.2. Initially, the routing probabilities are spread uniformly over all neighbours, and the source rates are set to a small value. The simulation performs the following steps indefinitely:

1. compute stable state of link loads and marginal costs:
   (a) compute rates r from y
   (b) compute fixpoint node rates t using r and probabilities p
   (c) compute link load z and marginal cost μ
   (d) compute fixpoint cost estimates c
2. perform adjustment step of routes and rates

The fixpoints are computed using the linear formulas (3.9) and (3.10). The adjustment procedures are those given in section 5.3 by formulas (5.3) and (5.4) tailored for discrete-time as proposed in section 5.4. The gain factors α and β are chosen independently for each experiment so that specific issues can be illustrated.

We monitor the marginal and total costs, utilities and profits for each flow. Recall, the marginal profit of user s is defined as:

$$U'_s(y_s) - c^{dst(s)}_{src(s)}$$

and the total profit of user s is:

$$U_s(y_s) - y_s\, C^{dst(s)}_{src(s)}$$

where C^d_i is the average cost induced by flows going from i to d per unit flow. This value can be estimated using the same fixpoint procedure as c^d_i with the substitution of $C_e(z_e)/z_e$ for μ_e. We also monitor the global (aggregate) cost, utility and profit, which are equivalent to C_T, U_T, and $U_T - C_T$, respectively.

We need to emphasize that using this methodology we cannot simulate any packet-level events, thus packet-driven algorithms, e.g. Q-routing could only be approximated. But as we indicated, our intention is not to perform a comparative evaluation, but instead demonstrate the feasibility and study the optimal state.

6.2 Scenario

A simulation scenario is defined by the network topology and user configuration. The topology comprises the set of nodes and links, each described with a cost function C_e. We use a common cost function for all links, parametrized by the link's bandwidth and latency. The user configuration is just a set of source, destination, and utility function triples. We use a common utility function for all users, which can be optionally weighted.

The network topologies for simulation are generated using Tiers [Doa96]. It is a structure-based generator that is capable of generating 3-tier hierarchical networks (WAN/MAN/LAN). However, for the purpose of this evaluation we generate flat networks (WAN-only) and choose arbitrarily to use common bandwidth of 10kbps ($1e4$ bps) and latency of 10ms for all links. The links are bidirectional. The topology used in the experiments is shown in Fig. 6.1.

For our evaluation, we choose the cost function to reflect experienced delays incurred by the load. The cost function in use is based on M/M/1 model [Kle76]:

$$C_e(z_e) = z_e \left(\frac{1}{Cap(e) - z_e} + Lat(e) \right)$$

where $Cap(e)$ is the capacity of link e interpreted as maximum allowed flow rate (i.e. available bandwidth), and $Lat(e)$ is the fixed latency component due to propagation delay. A plot of the resulting cost function is shown on Fig. 6.2. Clearly, under low load stress, the cost is virtually linear. Therefore, it is expected that, when user rates are low, the routing converges to cheapest paths, which in our setting is equivalent to shortest hop–count.

As for the utility, logarithm function is employed in favour of proportional fairness. The utility function in use is thus:

$$U_s(y_s) = w \lg y_s$$

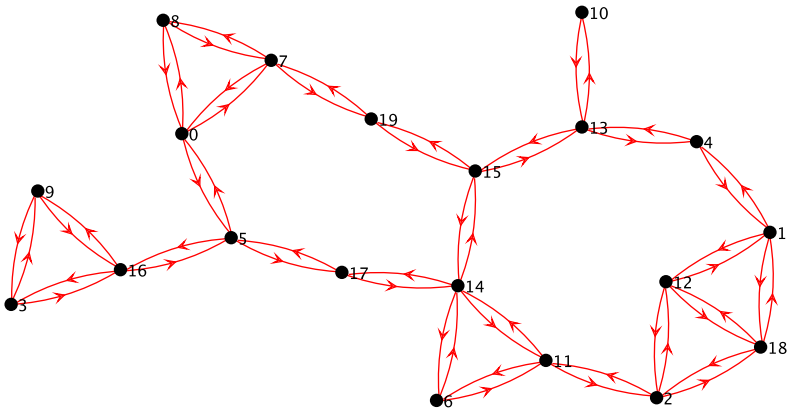

Figure 6.1: The flat network topology, generated using Tiers, used in the simulation. All links are bidirectional of capacity 10kbps and propagation latency 10ms.

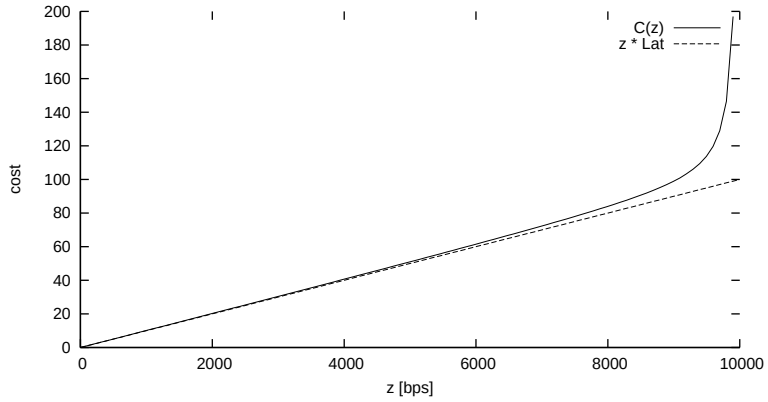

Figure 6.2: The plot of the link cost vs. load. Also plotted is the cost component incurred by load–independent propagation latency, which is dominant until the load approaches capacity.

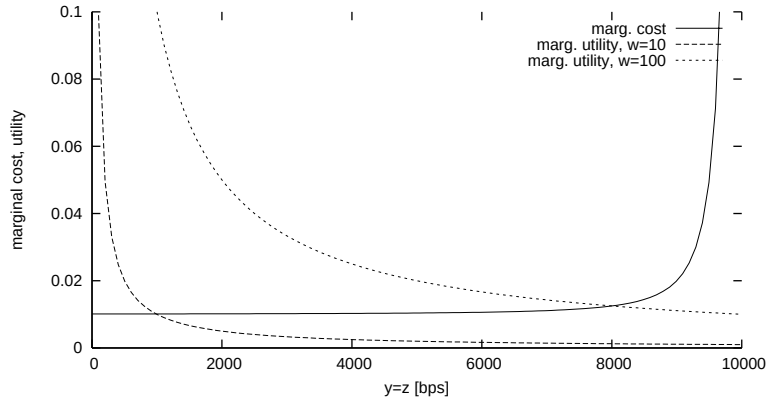

Figure 6.3: The marginal utility and cost in a single-user-single-resource ($z = y$) configuration. The crossing point marks the optimal rate and zero marginal profit $U'(y) - C'(z)$. The weight w controls how rewarding it is to increase the rate. Increasing w drives the system towards higher rate.

The positive weight w is used to control the interest in high throughput vs. small cost. The higher w is, the more rewarding it is to increase the rates. Figure 6.3 illustrates the impact of w in a single-user-single-resource configuration. Hence, using w we can drive the system into higher rates and toward *saturation*, which could be desiarble if capacity utilization was one of the primary goals.

6.3 Experiments

The main purpose of the experiments is to illustrate the operation of the joint algorithm, and the properties of the resulting allocation. The network topology used in simulations is illustrated in Fig. 6.1. A configuration of streams (sources, destinations and weights) is selected to expose the discussed concepts.

6.3.1 Phased vs. concurrent operation

First, we show how the algorithm converges to a global stable state when the two sub-procedures are run jointly. In section 5.3.4, we consider two variants of joint operation:

- *phased* - let one procedure converge before running the other
- *concurrent* - run the procedures together, i.e. both perform adjustment steps in every iteration

Since the algorithms are basically gradient–driven, it is reasonable to consider them as converged for practical purposes when the individual adjustments yield unnoticeable improvement in the global performance function. Recall that the flow rates are initialized to small values and the routing is initialized to uniformly random. We start the phased algorithm with rate adaptation,

although this seems less reasonable than to optimize the routing first, to enhance illustrational value.

The utility/cost weight and adaptation gain factors are set to:

w	1.0
α	20.0
β	1.0

and the configuration of streams for this particular experiment is:

src	→	*dst*
3		10
6		4
8		12
9		18

The results are plotted in Fig. 6.4. Certainly, under phased operation, when rates are adjusted and increased, then the total cost, utility and profit increase as well. But when the routes are adjusted, then the total cost drops, the utility remains unchanged (as the rates are fixed), and the overall profit improves. This marks the phases on the plots. The phased algorithm evidently needs to visit several local optima until it reaches the goal. In the next subsection, we will explain the reasons behind that.

Under concurrent operation, all performance characteristics increase at a steady rate. Clearly, both algorithms are converging to the same stable performance state. In fact, the resulting rate allocations are also nearby. Obviously, the concurrent operation is both more practical and advantageous than the phased method. These convergence results practically confirm our theoretical considerations in Chapter 5.

6.3.2 Optimal routing vs. demand

The phased algorithm visits several local optima before it can reach the goal. The reason behind this is the fact that the optimal routing varies with the user demand. We first demonstrate this behaviour and subsequently explain it.

Using the same topology as before, we set up a single user $19 \rightarrow 18$ which rate we control manually. We set the rate to $1e2$ and $1e4$ and compute the optimal routing. Recall that the capacity of each link is $1e4$, thus it would be impossible to satisfy the high demand using a single path within a finite cost. The results are shown in Fig. 6.5. Under the small load, the routing is same as shortest path using equal-cost paths in equal proportions. Under the high load, the flow is forked over a variety of paths including up to 5-hop detours.

In general, the higher the demand, the more likely the routing will span multiple paths. At a small load, the cost function is basically linear as shown in Fig. 6.2. As expected, the optimal routing collapses to cheapest paths with *Lat* as the link metric. Since *Lat* is equal for all links, this actually gives shortest paths, but such that breaks ties in equal proportions. As the load increases, it is more likely that a longer (in terms of total *Lat* or hop–count) but less loaded path will offer smaller cost than a shorter one but congested.

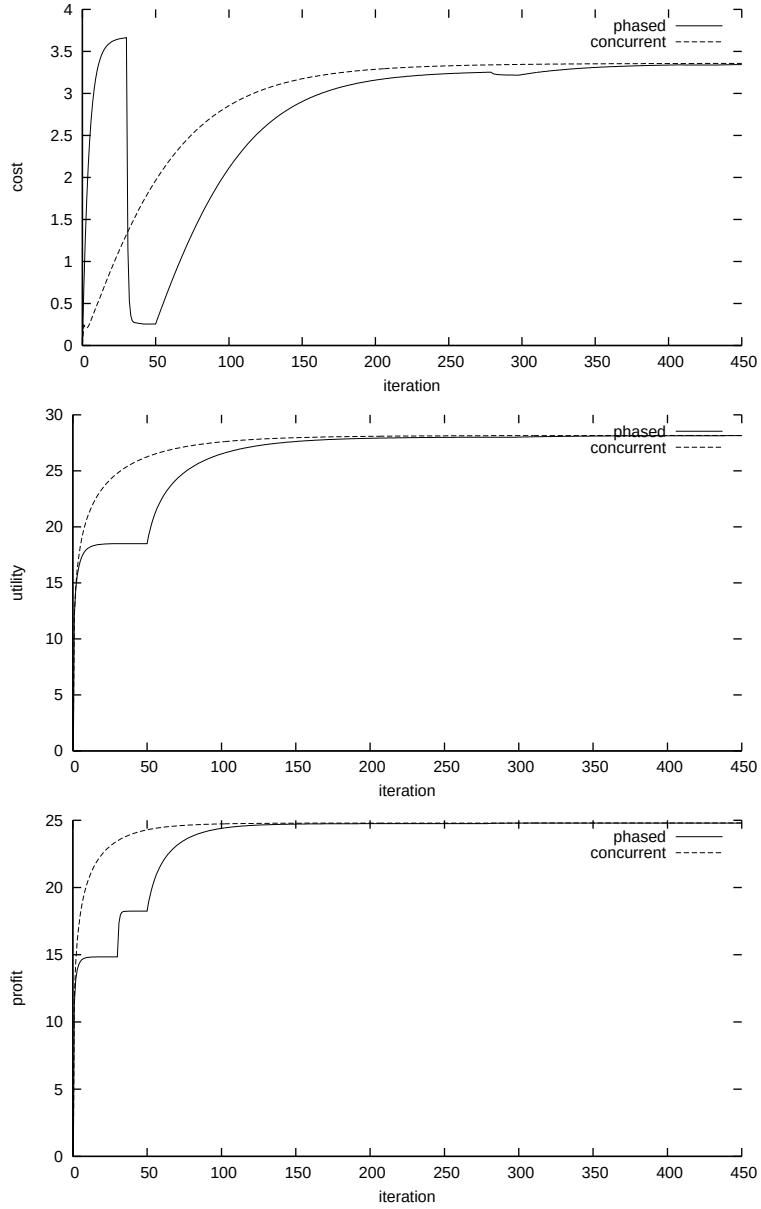

Figure 6.4: A comparison of *phased* and *concurrent* methods. From the top: total cost, total utility and total profit. Distinct phases of the *phased* method are visible: when rates are constant, only routes are adjusted. The two methods converge to the same stable state.

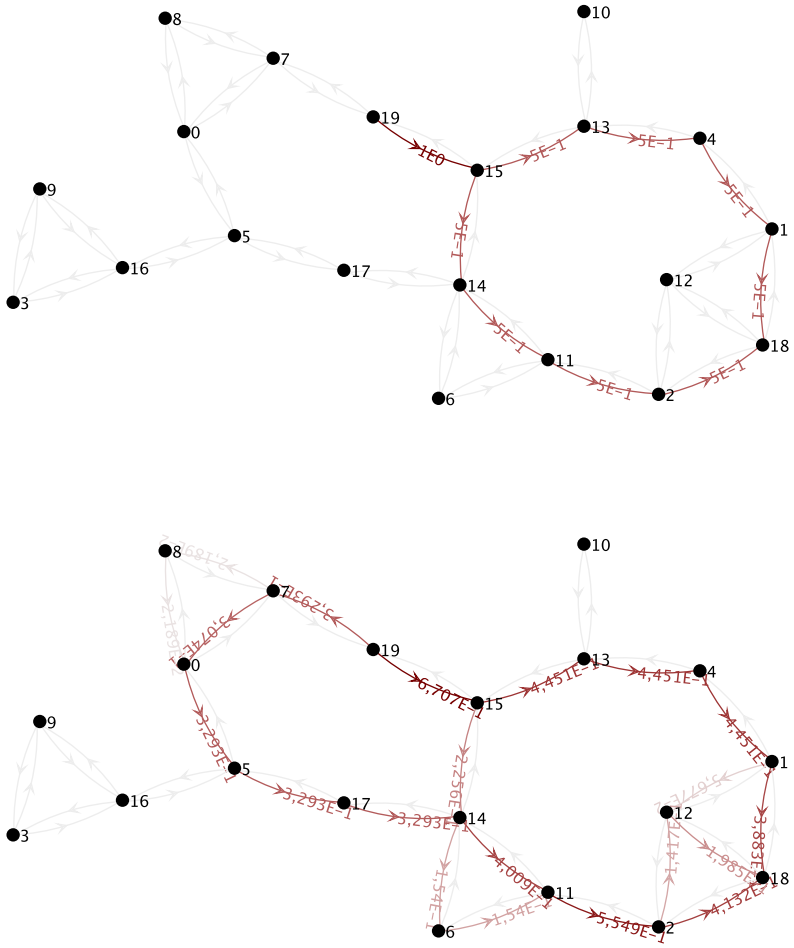

Figure 6.5: The optimal routing of flow $19 \rightarrow 18$ at rate $1e2$ (top) and $1e4$ (bottom). The labels and colour intensity show what fraction of the flow is routed across each link. Unlabeled gray links are unused.

This equal tie–breaking behaviour is due to the fact, that the cost function is strictly convex though approximately linear. In section 2.2.2, we have discussed how strictly convex functions are advantageous over linear functions for performance evaluation. In section 3.3.2, we have shown that if the cost functions were linear, then ties would be broken in any proportion.

6.3.3 Fairness and load balancing

Fairness and load balancing are the principles that led us to the formulation of the objective stated in Chapter 3. The utility and cost functions in our scenario are strictly concave/convex which helps enforce the principles. We have already shown in the previous subsection how load balancing is achieved. Now, we shall illustrate the quasi-proportional fairness that results from the system objective.

The user setup consists of two flows: $8 \rightarrow 2$ and $16 \rightarrow 1$. The resulting optimal allocation is shown in Fig. 6.6. Both flows have at least two equal-length sub-paths available, e.g. $8 \rightarrow 14$ through 17 or 15. However, due to the convexity of the cost function, the flows tend to separate their paths.

Note that the flow rates are close but not the same. This is due to the fact that flow $16 \rightarrow 1$ has a little bit more resources available in the form of two path segments $2 \rightarrow 1$. Therefore, its marginal cost is less and it can "afford" higher throughput. The two subject flows, when run individually, achieve the rates of $1.37e3$ and $1.2e3$ respectively. Thus, the relative proportion between them is somewhat preserved.

Introduction of another flow, $10 \rightarrow 11$, forces $16 \rightarrow 1$ into a single path via node 15, and yields rates of $1.2e3$, $1.09e3$, $1.57e3$ for $8 \rightarrow 2$, $16 \rightarrow 1$ and $10 \rightarrow 11$, respectively. This new flow can use fewer links and therefore is priced less. This, however, costs the old flows some of their allocation. In this way, the allocation that results from *profit–driven* performance optimization is sensitive to the costs incurred by users (unlike absolute or max-min fairness), but at the same time respects the concave utility functions that give revenue in reverse proportion to the benefits already gained.

6.3.4 Utility vs. cost

Up to now, we used $w = 1$. In Fig. 6.2, it is illustrated how w can be used to drive a single-user-single-resource system from underutilization to saturation. Now, we shall study it in a multi-commodity network.

Using the same set of flows as in the previous section we set w to 1, 10, and 100, and observe the load on the links. The results are shown in Fig. 6.7. As expected, the loads approach the capacity as w is increased. Hence, manipulating the global weight w is a relatively simple way to control the balance of preference of high throughput against low utilization. The main drawback is that this weight is common for all users and resources, which reduces its potential, but we use it here mainly for illustational purposes.

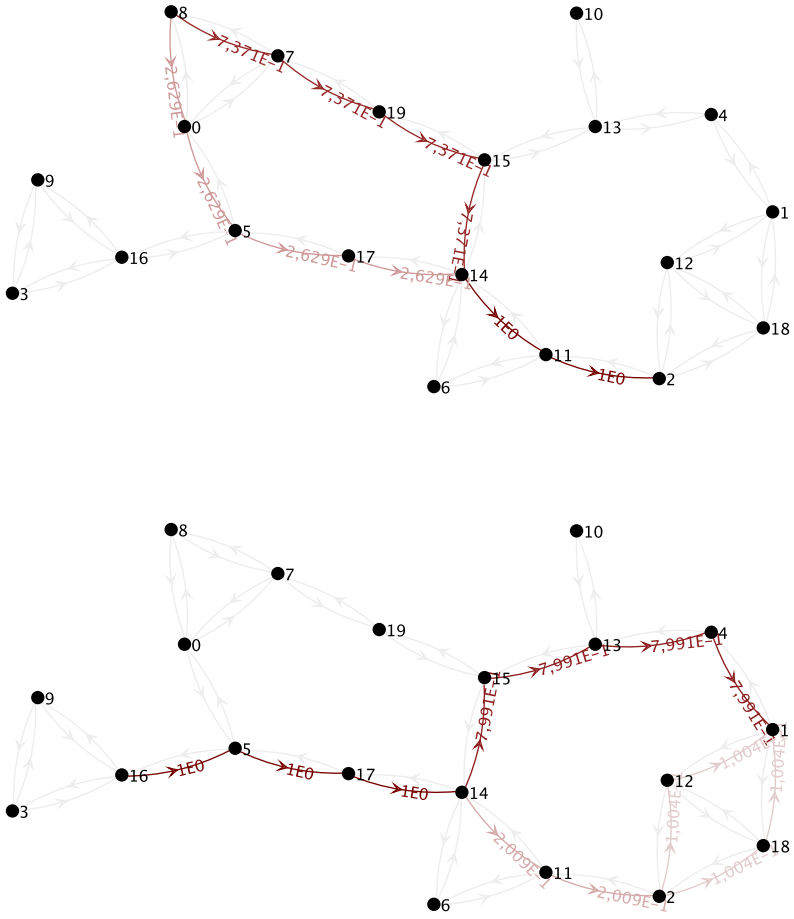

Figure 6.6: The optimal routing of flow $8 \rightarrow 2$ at the rate of $1.3e3$ (top) and flow $16 \rightarrow 1$ at the rate of $1.13e3$. The labels and colour intensity show what fraction of the flow is routed across each link. Unlabeled gray links are unused. Both fairness and load balancing is exposed.

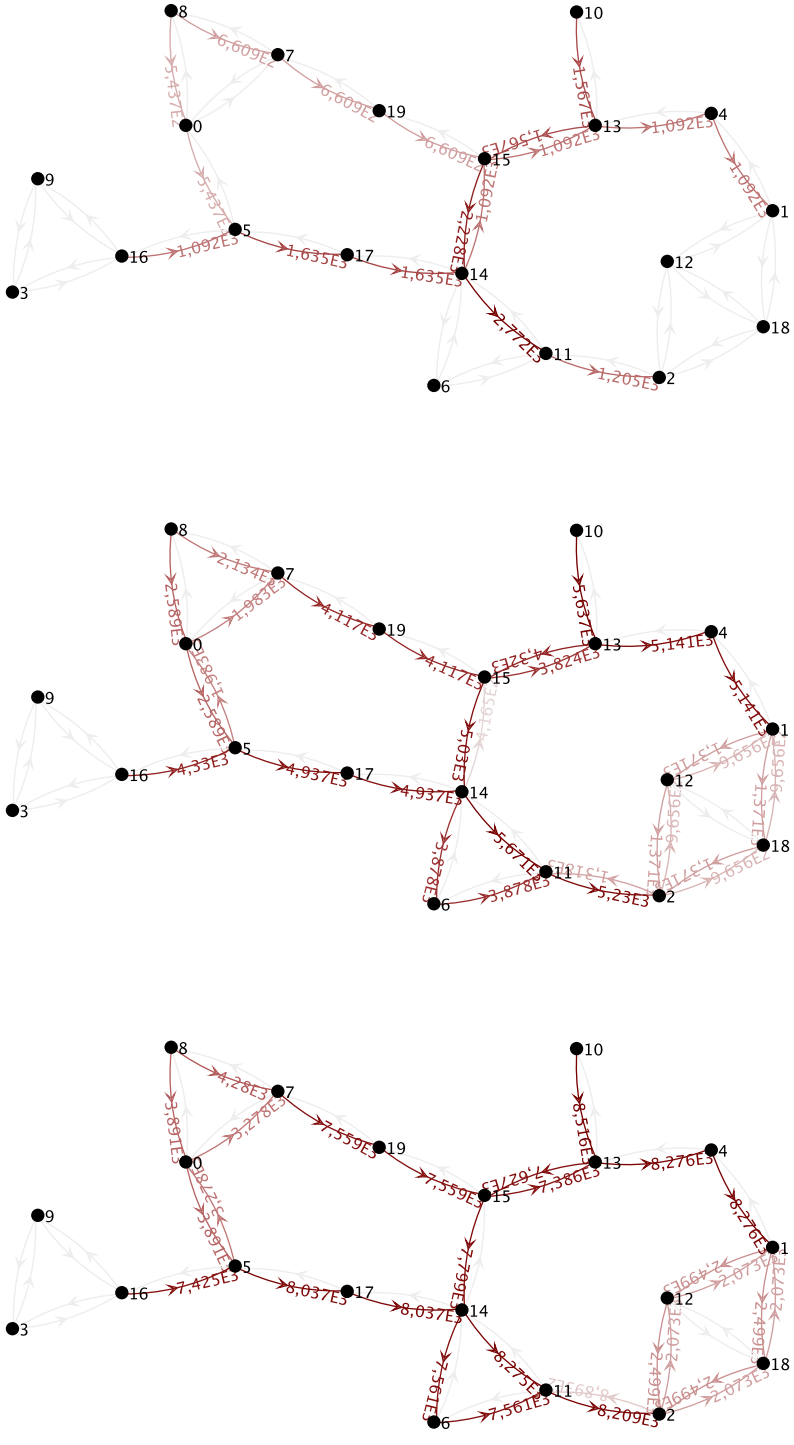

Figure 6.7: Link loads under optimal routing of flows $8 \rightarrow 2$, $16 \rightarrow 1$ and $10 \rightarrow 11$, using $w = 1$ (top), $w = 10$ (middle), and $w = 100$ (bottom). The link capacity is $1e4$. The labels and colour intensity show the link load (in bps). The maximum link load in the three cases is $2.72e3$, $5.67e3$, and $8.27e3$, respectively.

6.3.5 Selfishness

Interestingly, in the experiment described above we observed that, under high demand the routes have more in common than under low demand. As mentioned earlier, at $w = 1$ we had two of the flows use single paths only, while at $w = 100$ we observe a large overlap between the routes. See Fig. 6.8 for illustration. Let us point out, that since these flows have different destinations, it would be possible to rearrange the routing to reduce the overlap yet maintain the same incurred costs for each flow. However, this would be quite an unexpected behaviour from non–cooperative routers. Recall from section 2.2.3 that the Wardrop equilibrium that is aimed for, actually assumes, that no one else changes his decision in our favour.

6.3.6 Convergence issues

For the purpose of evaluation at $w = 100$ we had to reduce both gain factors, but mostly β that controls route adaptation. Weighing the utility high directly effects the marginal utility and the adjustment step size. If this size is larger than the remaining bandwidth, then the algorithm may accidentally leave the allowed space of $z_e < Cap(e)$. Further, large gains may catch the procedures in endless oscillations around the locally optimal value. This is certainly undesirable. We discuss these issues here. Using the same topology and stream configuration as before, and $w = 10$, we experiment with various settings of gain factors α and β.

First, we set $\alpha = 100$ and $\beta = 1$, which we subsequently reduce to achieve convergence. The resulting history of rates, costs, and total profit is shown in Fig. 6.9. Oscillations, especially in costs, are visible, although the total profit exhibits changes of less than 2%. We eventually achieve convergence for $\beta = 0.3$, but actually when we used $\beta = 0.3$ from the beginning, the algorithm converged after the first 30 iterations. Also, reducing α would not prevent the oscillations, as we observed them even at $\alpha = 1$. Furthermore, when we used $w = 5$ the algorithm easily converged for $\beta = 1$, but as we pointed out earlier we change the resulting optimal allocation when change w. We conclude that these oscillations are due to routing adjustment procedure. However, when we applied $\beta = 0.01$, the algorithm fell into a different kind of oscillations, illustrated in Fig. 6.10. The high variations occur initially, but later fade out. These oscillations are due to the rate adjustment procedure driven with high gain. We can observe them for low β, because this slows the routing adaptation down. As the marginal costs increase with iterations, the rate control algorithm uses smaller steps and eventually converges.

To sum up, the experiments reveal how setting the gain factors too high leads to two kinds of oscillations, one due to zig–zagging routing algorithm and the other due to zig-zagging rate control algorithm. To prevent these oscillations, we need to use moderate gain factors, bearing in mind, that this slows convergence down. Reasoning from the research done by others, we expect that the upper bound of values that guarantee convergence will be decreasing as update delay increase [JT00, KV05]. The longer the delays, the larger the mis–estimation of a single step can be.

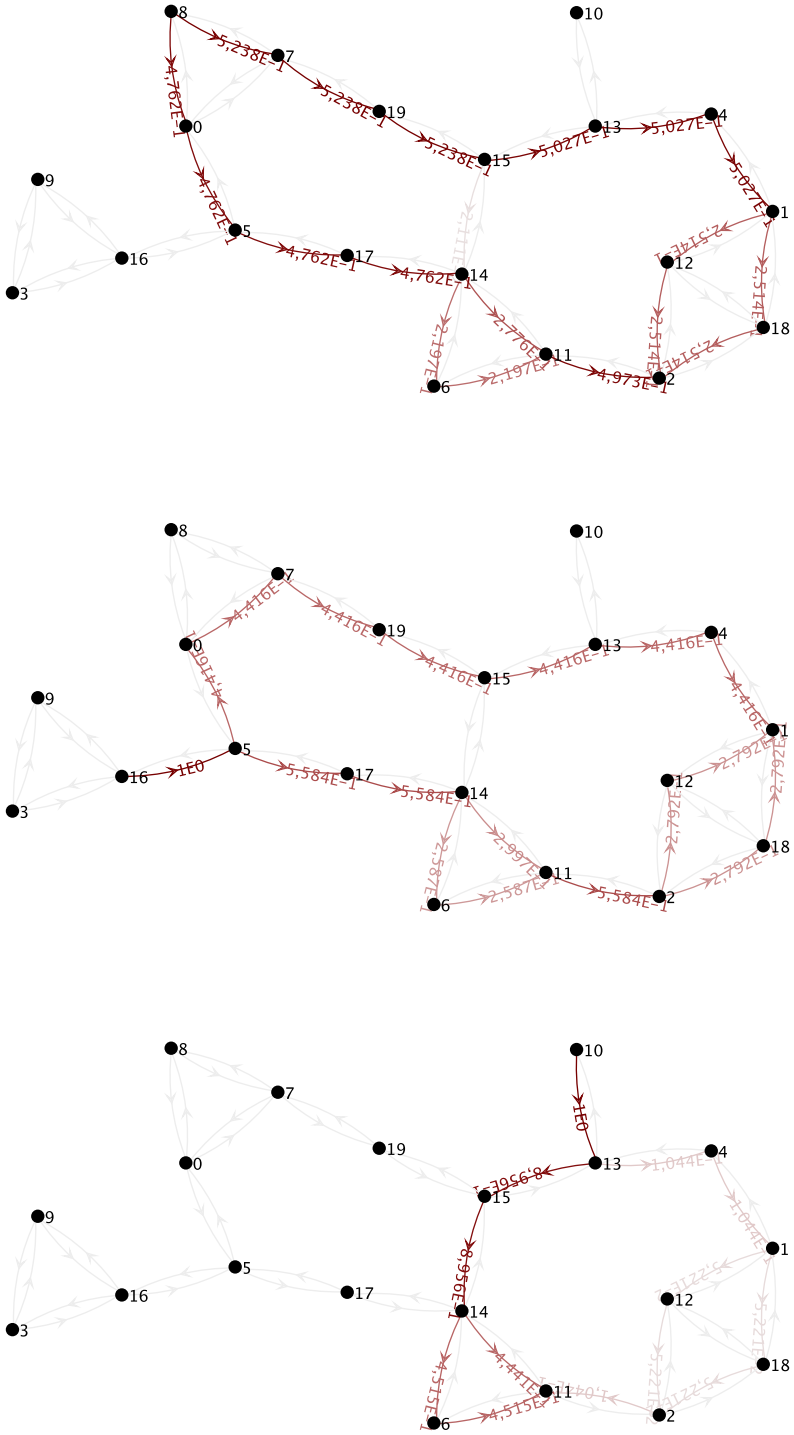

Figure 6.8: The optimal routing of flows $8 \to 2$, $16 \to 1$ and $10 \to 11$, using $w = 100$. The flow rates are $8.2e3$, $7.4e3$, and $8.5e3$ respectively The labels and colour intensity show what fraction of the flow is routed across each link. Substantial overlap between the routes of different flows can be observed. The flows have no strong incentive to separate their routes when they are given the same choice of routes as others.

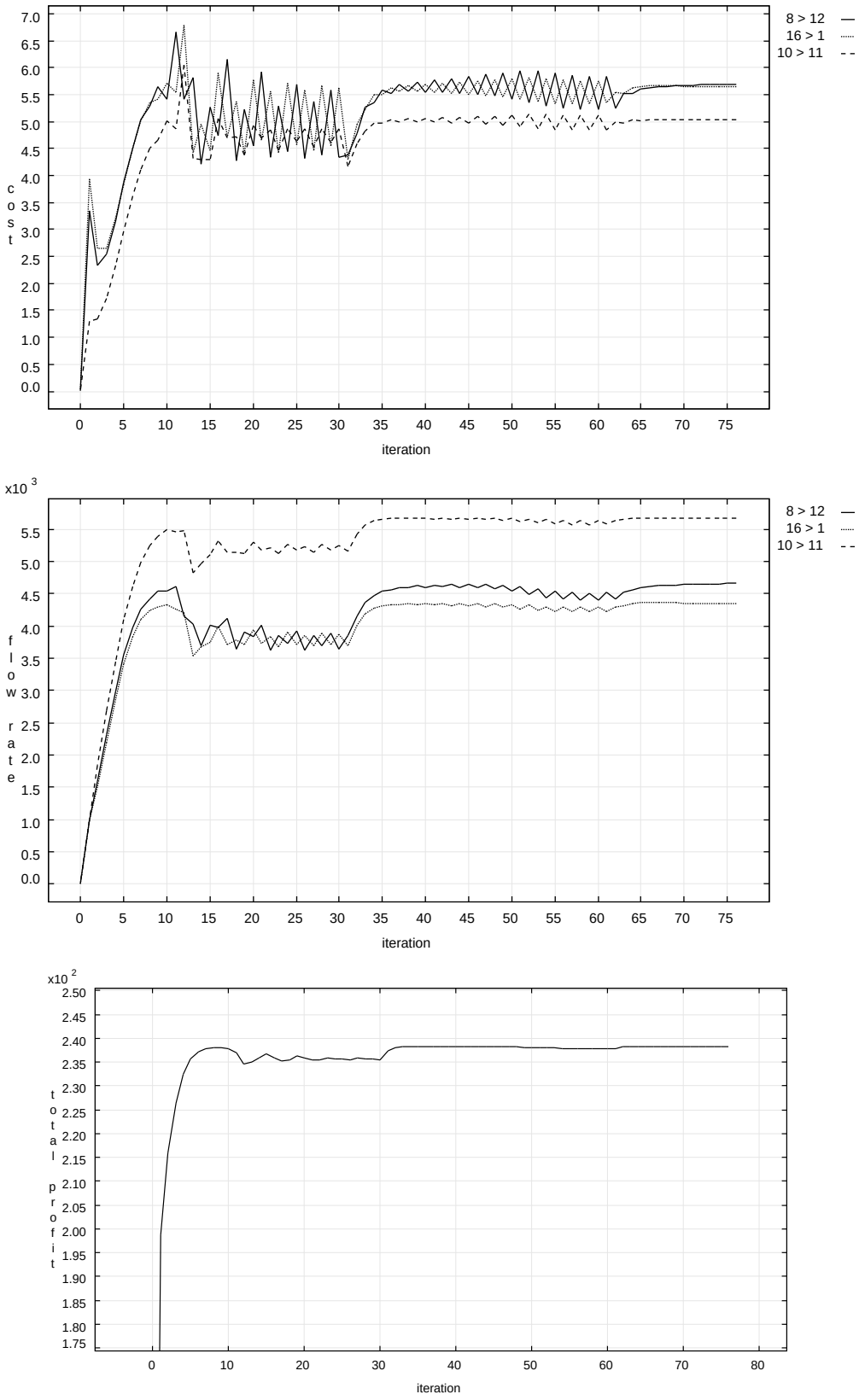

Figure 6.9: Convergence for $w = 10$, $\alpha = 100$. Initially, $\beta = 1$, after 30 iterations, we set $\beta = 0.5$, and after another 30 iterations, we set $\beta = 0.3$. The smaller the oscillations, the higher the rates and profit.

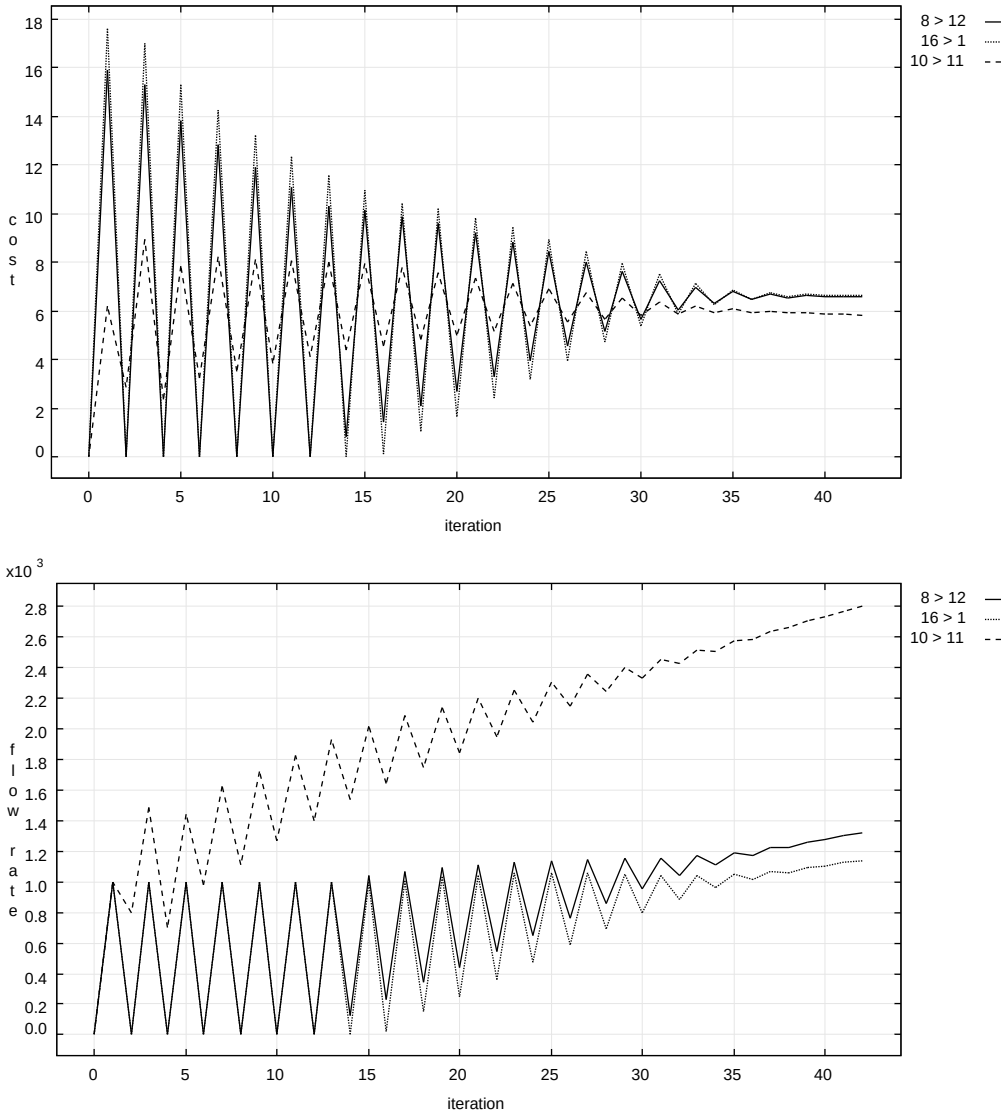

Figure 6.10: Convergence of individual costs and flow rates for $w = 10$, $\alpha = 100$, $\beta = 0.01$. The initial oscillations fade out.

Chapter 7

Conclusion

This thesis is focused around the problem of resource sharing in packet networks. Our task is to define a tunable, adaptive mechanism that deals with the dynamics of both user demand and resource capacity, and improves the satisfaction of users from both quantity and quality of service. The aim is to account for both fairness and load balancing, and employ both aspects of resource sharing: dynamic routing and load control. It is postulated to overcome traditional layer separation and define a global performance goal.

In the thesis we fulfil the tasks. We propose to use a global objective function, well–studied in the context of both road traffic and network communications, that combines the features of utility maximization and cost minimization. The benefits of using concave utility functions and convex cost functions are two–fold:

- such functions improve the balance of user rates and resource loads
- the optimization problem is convex and can be solved with simple means including a composition of well-known techniques

We show that such objective can be attained by a joint algorithm composed of stochastic routing and rate control. The two sub-procedures are glued together using a common marginal cost metric. This is an important result, because it suggests that a well–defined global performance objective can be formulated, if the adaptive mechanisms already commonly implemented in the IP stack shared the cost metric. Further, we show that these mechanisms, if properly designed, will converge when run concurrently.

Aside from the theoretical analysis of the goals, we discuss several existing implementations of the two sub-procedures, and also discuss implementation issues of a joint algorithm. Many problems are briefly outlined, and some are left open. At this stage we focus on the pure concept as a promising research direction.

Although run under idealized conditions, the numerical evaluation confirms our theoretical analysis with respect to joint convergence and demonstrates that the chosen global objective

is a versatile and tunable tool to ensure both fairness among users and load balancing across resources.

Many issues connected to the proposed scheme remain open. Clearly, the proposed algorithms need to be tailored for practical implementation. The chosen cost update method will be essential to the performance of the algorithm in real-world deployment. Specifically, if the algorithm converged slowly, then continuous loop-freedom would be desired. A comparative performance evaluation of several schemes would aid the choice of the best fit methods.

Also, the joint algorithm needs to be enhanced to deal with propagation and update delays omitted in the continuous model, but inherent to any practical update scheme. Theoretical analysis and a more accurate evaluation using a discrete simulator like *ns-2* [NS] is needed.

Further, concrete methods to obtain the marginal cost estimates from link load observation need to be developed or selected from the existent solutions. However, these are very domain–specific, and it is reasonable that they are left out of the scope of this thesis. For example, in the quickly evolving area of *stationary* wireless mesh networks, where it might be beneficial to employ our joint scheme, the cost due to load is associated with a *collision domain* that spans multiple links. The links that are located within a single contention area, interfere with each other and actually share the available bandwidth. We ought to bear in mind, however, that the choice of the appropriate resource cost function is up to the network designer, and constitutes a tool to regulate the flow behaviour.

Appendix A

Proofs for Chapter 3

Proof of Lemma 3.2. We show how to translate any feasible allocation $x_{s,e}$ to one that yields the same y_s and z_e but also conforms to (3.2) for some $p^d_{i,j}$. The intuition for the translation is that it does not really matter what route each user flow takes, but rather how the total load incoming at a node is split among its outgoing links. In fact, the link load incurred by flows addressed to a particular destination will also be preserved. Let us denote by z^d_e the part of z_e incurred by flows addressed to d, and state that:

$$z^d_{i,j} = \sum_{s:dst(s)=d} x_{s,(i,j)}$$
$$z_{i,j} = \sum_d z^d_{i,j}$$

We can apply to the per–flow allocation the notation introduced for stochastic routing (3.3)-(3.4). Let us compute the total load that enters node i and that node i needs to route to a particular destination d:

$$r^d_i = \sum_{s:src(s)=i,dst(s)=d} y_s$$
$$t^d_i = \sum_{s:dst(s)=d} \sum_j x_{s,(i,j)} = \sum_j z^d_{i,j}$$

Since we want to preserve the total load per–destination (r^d, t^d, z^d), for the new allocation we use the same y_s for source rates and match the implied routing proportions by selecting:

$$p^d_{i,j} = \frac{z^d_{i,j}}{t^d_i} = \frac{\sum_{s:dst(s)=d} x_{s,(i,j)}}{\sum_{s:dst(s)=d} \sum_k x_{s,(i,k)}}$$

The normalization conditions, $p^d_{i,j} \geq 0$, $p^d_{d,j} = 0$ and $\sum_j p^d_{i,j} = 1$ hold. Using (3.1), we can write:

$$z_{i,j} = \sum_d p^d_{i,j}\, t^d_i$$
$$t^d_i = \sum_j z^d_{j,i} + r^d_i = \sum_j p^d_{j,i}\, t^d_j + r^d_i$$

It follows that the loads resulting from the stochastic allocation using p are given by the very same equations and therefore they are the same. □

Proof of Theorem 3.2.

$$\begin{aligned}
\lambda_i^d &= \frac{\partial C_T}{\partial r_i^d} = \\
&= \frac{\partial}{\partial r_i^d} \sum_e C_e(z_e) = \\
&= \sum_e C'_e(z_e) \frac{\partial z_e}{\partial r_i^d} = \\
&= \sum_{j,k} C'_{j,k}(z_{j,k}) \frac{\partial}{\partial r_i^d} \sum_f p_{j,k}^f t_j^f = \\
&= \sum_{j,k} \mu_{j,k} p_{j,k}^d \frac{\partial t_j^d}{\partial r_i^d} = \\
&= \sum_j \frac{\partial t_j^d}{\partial r_i^d} \sum_k p_{j,k}^d \mu_{(j,k)}
\end{aligned}$$

And we have obtained by (3.9):

$$\frac{\partial t_j^d}{\partial r_i^d} = {a^d}^{\mathrm{T}}{}_{j,i} = a_{i,j}^d$$

and by (3.10):

$$\begin{aligned}
c_i^d &= \sum_j a_{i,j}^d \sum_k p_{j,k}^d \mu_{j,k} = \\
&= \sum_j a_{i,j}^d \sum_k p_{j,k}^d \mu_{j,k} = \\
&= \sum_j \frac{\partial t_j^d}{\partial r_i^d} \sum_k p_{j,k}^d \mu_{j,k}
\end{aligned}$$

□

Proof of Theorem 3.4. Since the problem is convex, and the optimum cannot occur at $y_s = 0$, the global objective is at optimum when

$$\begin{aligned}
0 &= \frac{\partial}{\partial y_s} \left(\sum_{s \in S} U_s(y_s) - \sum_{e \in E} C_e(z_e) \right) = \\
&= U'_s(y_s) - \frac{\partial C_T}{\partial y_s} = \\
&= U'_s(y_s) - \frac{\partial C_T}{\partial r_{src(s)}^{dst(s)}} = \\
&= U'_s(y_s) - \lambda_{src(s)}^{dst(s)}
\end{aligned}$$

□

Bibliography

[AA03] E. Anderson and T. Anderson. On the stability of adaptive routing in the presence of congestion control. In *Proceedings of INFOCOM*, San Fransisco, April 2003.

[ABKM01] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *Symposium on Operating Systems Principles*, pages 131–145, 2001.

[Agn76] C. Agnew. On quadratic adaptive routing algorithms. *Communications of ACM*, 19(1):18–22, 1976.

[BG92] D. P. Bertsekas and R. Gallager. *Data Networks.* Prentice-Hall, 2nd edition, 1992.

[BGG84] D.P. Bertsekas, E.M. Gafni, and R.G. Gallager. Second derivative algorithms for minimum delay distributed routing in networks. *IEEE Transactions on Communications*, 8:911–919, August 1984.

[BK03] V. Borkar and P. Kumar. Dynamic cesaro-wardrop equilibration in networks. *IEEE Transactions on Automatic Control*, 3(48):382–396, March 2003.

[BL94] Justin A. Boyan and Michael L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 671–678. Morgan Kaufmann Publishers, Inc., 1994.

[BMW56] M. Beckmann, C.B. McGuire, and C.B. Winsten. Studies in the economics of transportation. In *Cowles Commission Monograph.* Yale University Press, 1956.

[BNO03] Dimitri P. Bertsekas, Angelia Nedic, and Asuman E. Ozdaglar. *Convex Analysis and Optimization.* Athena Scientific, April 2003.

[Bra68] D. Braess. Uber ein paradoxen der verkehrsplannung. *Unternehmens-forschung*, 12:258–268, 1968.

[CABM03] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing. In *Proceedings of MobiCom*, pages 134–146, New York, NY, USA, 2003. ACM Press.

[CAT90] C.G. Cassandras, M. V. Abidi, and D. Towsley. Distributed routing with on–line marginal delay estimation. *IEEE Transactions on Communications*, 38(3), March 1990.

[CCS96] I. Castineyra, N. Chiappa, and M. Steenstrup. The nimrod routing architecture. RFC1992, August 1996.

[CD97] G. Di Caro and M. Dorigo. AntNet: a mobile agents approach to adaptive routing. Technical Report IRIDIA/97-12, Université Libre de Bruxelles, Belgium, 1997.

[CD98] Gianni Di Caro and Marco Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.

[CFSK04] Byung-Gon Chun, Rodrigo Fonseca, Ion Stoica, and John Kubiatowicz. Characterizing selfishly constructed overlay networks. In *Proceedings of (INFOCOM)*, Hong Kong, 2004.

[Chi04] M. Chiang. To layer or not to layer: balancing transport and physical layers in wireless multihop networks. In *Proceedings of INFOCOM*, Hong Kong, China, March 2004.

[CJ89] D.M. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17(1):1–14, June 1989.

[Col98] A. Collins. The detour framework for packet rerouting. Master's thesis, University of Washington, October 1998.

[DHK04] Gareth Davies, Michael Hardt, and Frank Kelly. Network dimensioning, service costing and pricing in a packet switched environment. *Telecommunications Policy*, 28:391–412, 2004.

[DKS89] A. Demers, S. Keshav, and S. Shenker. Analysis and similation of a fair queueing algorithm. In *Proceedings of SIGCOMM*, volume 19, pages 1–12, September 1989.

[Doa96] B. Doar. A better model for generating test networks. In *Proceedings of Global Internet*, November 1996.

[DPZ04] R. Draves, J. Padhye, and B. Zill. Routing in multi-radio, multi-hop wireless mesh network. In *Proceedings of MOBICOM*, 2004.

[DS04] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. MIT Press, 2004.

[ELR+96] D. Estrin, T. Li, Y. Rekhter, K. Varadhan, and D. Zappala. Source demand routing. RFC1940, May 1996.

[FBR+04] Nick Feamster, Hari Balakrishnan, Jennifer Rexford, Aman Shaikh, and Kobus van der Merwe. The Case for Separating Routing from Routers. In *ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA)*, Portland, OR, September 2004.

[Gal77] R.G. Gallager. A minimum delay routing algorithm using distributed computation. *IEEE Transactions on Communications*, 1:73–85, October 1977.

[GK97] P. Gupta and P. R. Kumar. A system and traffic dependent adaptive routing algorithm for ad hoc networks. In *Proceedings of IEEE Decision and Control*, 1997.

[GK99] R.J. Gibbens and F. Kelly. Resource pricing and the evolution of congestion control. *Automatica*, 35:1969–1985, 1999.

[GLAB95] J. Garcia-Luna-Aceves and J. Behrens. Distributed, scalable routing based on vectors of link states. *IEEE Journal on Selected Areas in Communications*, 13(8):1383–1395, October 1995.

[Gol80] S.J. Golestani. *A unified theory of flow control and routing in data communication networks*. PhD thesis, MIT, Cambridge, MA, 1980.

[GSK04] Violeta Gambiroza, Bahareh Sadeghi, and Edward W. Knightly. End-to-end performance and fairness in multihop wireless backhaul networks. In *Proceedings of MobiCom*, pages 287–301, New York, NY, USA, 2004. ACM Press.

[Hed88] C.L. Hedrick. Routing Information Protocol. RFC 1058 (Historic), June 1988. Updated by RFCs 1388, 1723.

[HJB04] Bret Hull, Kyle Jamieson, and Hari Balakrishnan. Mitigating congestion in wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 134–147, New York, NY, USA, 2004. ACM Press.

[HM85] A. Haurie and P. Marcotte. On the relationship between nash-cournot and wardrop equilibria. *Networks*, 15:295–308, 1985.

[Jai90] Raj Jain. Congestion control in computer networks: issues and trends. *IEEE Network*, pages 24–30, 1990.

[JMB01] D. Johnson, D. Maltz, and J. Broch. *DSR The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks*, chapter 5, pages 139–172. Addison-Wesley, 2001.

[JT00] Ramesh Johari and David Tan. End-to-end congestion control for the internet: delays and stability. Technical Report 2000-2, Statistical Laboratory, University of Cambridge, 2000.

[Kel97] F. Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecommunications*, 8:33–37, 1997.

[Kel03] Frank Kelly. Fairness and stability of end-to-end congestion control. *European Journal of Control*, 9:159–176, 2003.

[Kes93] S. Keshav. A control-theoretic approach to flow control. *Proceedings of the conference on Communications architecture and protocols*, pages 3–15, 1993.

[KESI+01] I. Kassabalidis, M.A. El-Sharkawi, R.J. Marks II, P. Arabshahi, and A.A. Gray. Swarm intelligence for routing in communication networks, November 2001.

[KHR02] Dina Katabi, Mark Handley, , and Charles Rohrs. Internet congestion control for future high bandwidth-delay product environments. August 2002.

[Kle64] L. Kleinrock. *Communication nets: stochastic message flow and delay.* McGraw Hill, 1964.

[Kle76] L. Kleinrock. *Computer Applications*, volume 2 of *Queueing Systems.* Wiley, 1976.

[KLM96] Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[KM97] S. Kumar and R. Miikkulainen. Dual reinforcement q-routing: An on-line adaptive routing algorithm. In *Proceedings of the Artificial Neural Networks in Engineering Conference*, 1997.

[KMBL99] Peter Key, Derek McAuley, Paul Barham, and Koenraad Lavens. Congestion pricing for congestion avoidance. Technical Report MSR-TR-99-15, Microsoft Research, Feb 1999.

[KMT98] F. Kelly, A. Maulloo, and D. Tan. Rate control in communication networks: Shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49:237–252, 1998.

[KST01] K. Kar, S. Sarkar, and L. Tassiulas. A simple rate control algorithm for maximizing total user utility. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM'01)*, 2001.

[KV05] Frank Kelly and Thomas Voice. Stability of end-to-end algorithms for joint routing and rate control. *SIGCOMM Comput. Commun. Rev.*, 35(2):5–12, 2005.

[KVR95] Lampros Kalampoukas, Anujan Varma, and K. K. Ramakrishnan. An efficient rate allocation algorithm for ATM networks providing max-min fairness. In *HPN*, pages 143–154, 1995.

[LA02] Richard J. La and Venkat Anantharam. Utility-based rate control in the internet for elastic traffic. *IEEE/ACM Transactions on Networking*, 10(2):272–286, 2002.

[LBB04] S. Lee, S. Banerjee, and B. Bhattacharjee. The case for a multi-hop wireless local area network. In *Proceedings of INFOCOM*, 2004.

[LBS03] S. Liu, T. Basar, and R. Srikant. Controlling the internet: A survey and some new results. In *Proceedings of IEEE Conference on Decision and Control*, 2003.

[LOR+01] Dean H. Lorenz, Ariel Orda, Danny Raz, , and Yuval Shavitt. How good can IP routing be? Technical Report TR: 2001-17, DIMACS, 2001.

[Mal98] G. Malkin. RIP Version 2. RFC 2453 (Standard), November 1998.

[Mar03] Peter Markbach. Priority service and max-min fairness. *IEEE/ACM Transactions on Networking*, 11, October 2003.

[Mas85] L. G. Mason. Equilibrium flows, routing patterns and algorithms for store-and-forward networks. *Large Scale Systems*, 8:187–209, 1985.

[Moy89] J. Moy. OSPF specification. RFC 1131 (Proposed Standard), October 1989. Obsoleted by RFC 1247.

[MR99] Laurent Massoulie and James Roberts. Bandwidth sharing: Objectives and algorithms. In *Proceedings of IEEE INFOCOM (3)*, pages 1395–1403, March 1999.

[Mye91] Roger B. Myerson. *Game theory: analysis of conflict*. Harvard University Press, 1991.

[NS] The network simulator - ns-2. http://www.isi.edu/nsnam/ns.

[Pig20] A.C. Pigou. *The economics of welfare*. MacMillan, London, England, 1920.

[QYZS03] Lili Qiu, Yang Richard Yang, Yin Zhang, and Scott Shenker. On selfish routing in internet-like environments. In *Proceedings of SIGCOMM*, All ACM Conferences, pages 151–162, Karlsruhe, Germany, august 2003. ACM.

[RB02] Bozidar Radunovic and Jean-Yves Le Boudec. A unified framework for max-min and min-max fairness with applications. Technical report, I & C School of Computer and Communication Sciences, 2002.

[RB03] Bozidar Radunovic and Jean-Yves Le Boudec. Rate performance objectives of multihop wireless networks. Technical report, I & C School of Computer and Communication Sciences, June 2003.

[RK04] Vivek Raghunathan and P. R. Kumar. A wardrop routing protocol for wireless networks. In *Proceedings of IEEE CDC*, 2004.

[RM98] J.W. Roberts and L. Massoulié. Bandwidth sharing and admission control for elastic traffic. In *Proceedings of ITC Specialist Seminar*, Yokohama, October 1998.

[Rou02] T. Roughgarden. *Selfish Routing.* PhD thesis, Cornell University, May 2002.

[RT00] Tim Roughgarden and Eva Tardos. How bad is selfish routing? In *IEEE Symposium on Foundations of Computer Science*, pages 93–102, 2000.

[SB98] R.S. Sutton and A.G. Barto. *Reinforcement learning: an introduction.* MIT Press, March 1998.

[SDC97] Devika Subramanian, Peter Druschel, and Johnny Chen. Ants and reinforcement learning: A case study in routing in dynamic networks. In *IJCAI (2)*, pages 832–839, 1997.

[Seg77] A. Segall. The modelling of adaptive routing in data–communication networks. *IEEE Transactions on Communications*, 1:85–95, October 1977.

[SK91] T.J. Socolofsky and C.J. Kale. TCP/IP tutorial. RFC 1180 (Informational), January 1991.

[TGSE01] Hongsuda Tangmunarunkit, Ramesh Govindan, Scott Shenker, and Deborah Estrin. The impact of routing policy on internet paths. In *Proceedings of INFOCOM*, pages 736–742, 2001.

[TKL+04] T.Guven, C. Kommareddy, R. La, M.A. Shayman, and B. Bhattacharjee. Measurement based optimal multi-path routing. In *Proceedings of INFOCOM*, 2004.

[TS02] L. Tassiulas and S. Sarkar. Maxmin fair scheduling in wireless networks. In *Proceedings of INFOCOM*, volume 10, pages 320–328, June 2002.

[VGLA99] Srinivas Vutukury and J. J. Garcia-Luna-Aceves. A simple approximation to minimum-delay routing. In *Proceedings of SIGCOMM*, pages 227–238, 1999.

[VGLA01] Srinivas Vutukury and J. J. Garcia-Luna-Aceves. MDVA: A distance-vector multipath routing protocol. In *Proceedings of INFOCOM*, pages 557–564, 2001.

[Voi04] Thomas Voice. A global stability result for primal-dual congestion control algorithms with routing. *ACM SIGCOMM*, 34(3), July 2004.

[Wan03] Jun Wang. *Load Balancing In Hop-By-Hop Routing With And Without Traffic Splitting.* PhD thesis, University of Illinois at Urbana-Champaign, October 2003.

[War52] J.G. Wardrop. Some theoretical aspects of road traffic research. In *Proceedings of the Institution of Civil Engineers*, volume 1, pages 325–378, 1952.

[WC92] Z. Wang and J. Crowcroft. Analysis of shortest-path routing algorithms in a dynamic network environment. *ACM SIGCOMM CCR*, 22, 2:63–71, 1992.

[WEC03] Chieh-Yih Wan, Shane B. Eisenman, and Andrew T. Campbell. Coda: congestion detection and avoidance in sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 266–279, New York, NY, USA, 2003. ACM Press.

[XQYZ04] Haiyong Xie, Lili Qiu, Yang Richard Yang, and Yin Zhang. On self adaptive routing in dynamic environments — an evaluation and design using a simple, probabilistic scheme. Technical Report YALEU/DCS/TR1289, Computer Science Department,Yale University, May 2004.