Network utility maximization for delay-sensitive applications in unknown communication settings

THÈSE N^{O} 8732 (2018)

PRÉSENTÉE LE 2 OCTOBRE 2018

À LA FACULTÉ DES SCIENCES ET TECHNIQUES DE L'INGÉNIEUR

LABORATOIRE DE TRAITEMENT DES SIGNAUX 4

PROGRAMME DOCTORAL EN GÉNIE ÉLECTRIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Stefano D'ARONCO

acceptée sur proposition du jury:

Prof. A. P. Burg, président du jury
Prof. P. Frossard, directeur de thèse
Prof. G. Simon, rapporteur
Prof. E. Steinbach, rapporteur
Prof. D. Atienza Alonso, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2018

"*Your scientists were so preoccupied with whether or not they could that they didn't stop to think if they should*"

— "*Dr. Ian Malcolm*" in Jurassic Park, 1993

Abstract

In the last decades the Internet traffic has greatly evolved. The advent of new Internet services and applications has, in fact, led to a significant growth of the amount of data transmitted, as well as to a transformation of the data type. As a matter of fact, nowadays, the largest amount of traffic share consists of multimedia data, which do not represent classical Internet data. Due to the increasing amount of traffic, the network resources might be scarce, and in such cases it becomes extremely important to optimize network transmission in order to provide a satisfying service to the users. Although methods for maximizing the network utility in scenarios with limited resources have been studied extensively, the evolution of the Internet services poses continuously new challenges that require novel solution methods to meet the transmission requirements. In this thesis we propose novel solutions methods to network utility maximization problems that arise in the context of nowadays network communications. In particular we analyze problems related to delay-sensitive Internet applications and rate allocation in unknown network settings.

In the first problem we study how to effectively allocate the transmission rates in a multiparty videoconference system. The main contribution of this chapter is an approximate fast rate rate allocation method that is able to adapt quickly to changes in the videoconference conditions. This fast adaptation cannot be achieved with classical network utility maximization solving methods, as they are usually based on iterative approaches. In this case we leverage the particular structure of the problem to design a novel distributed solving method which proves to be very effective when compared to baseline solutions.

The next problem that we address is the design of a congestion control algorithm for delay-sensitive applications. One of the main problems of existing delay-based congestion control algorithms is that they tend to achieve an extremely low throughput when competing against loss-based algorithms. In order to overcome this difficulty we propose a novel adaptive controller based on a bandit problem approach. The adaptive controller tries to infer how the network responds, in terms of rate-delay pair at equilibrium, when changing the delay sensitivity of an underlying delay-based congestion control. Once the network response is inferred, the controller selects the sensitivity that leads to the best trade-off between the transmitting rate and the experienced delay.

In the final problem, we analyze the design of an overlay rate allocation systems to be used when: the amount of available network resources is not known, and the user congestion

feedback cannot be used as valid signal to reach the optimal rate allocation. Such a scenario appears when an Internet application wants to maximize a certain utility metric, but, at the same time, it must operate using a specific congestion control algorithm that is completely unaware of the application utility. To solve this problem we design a distributed system that coordinates the users in order to perform active learning on the amount of network resource. Adopting such a method reveals to be the key to an effective maximization of the long term application utility for the entire system.
To summarize, in this thesis we address specific problems that arise in nowadays Internet communication. In particular the considered problems deal with challenges related to delay sensitive communications, where it is important to adapt quickly and maintain a low end-to-end transmission latency, and communications in unknown environments, where it is essential to carefully take into account the existing uncertainties in order to achieve good system performance in the long term.

Keywords: Network Utility Maximization, Rate Adaptation, Network Congestion Control, Adaptive Systems, Bandit Problems, Optimal Learning, Bayesian Inference, Decision Making.

Sommario

Negli ultimi anni il traffico della rete Internet si é sviluppato in modo consistente. L'avvento di nuovi servizi e applicazioni della rete Internet ha infatti portato ad una significativa crescita della quantità di dati trasmessi, come anche ad una trasformazione della tipologia di dati stessi. La maggior parte dei dati trasmessi oggigiorno infatti, consiste di dati multimediali (audio e video), i quali non rappresentano una tipologia di dati solitamente trasmessa nella rete. A causa del aumento del traffico le risorse di rete potrebbero rivelarsi limitate. In questi casi diviene fondamentale ottimizzare le trasmissioni al fine di provvedere un servizio soddisfacente agli utenti. Nonostante metodi per la massimizzazione della utilità di rete in scenari con risorse limitate siano stati studiati estensivamente, l'evoluzione dei servizi Internet pone incessantemente nuove difficoltà che richiedono nuovi metodi risolutivi al fine di rispettare i requisiti delle trasmissioni. In questa tesi proponiamo nuove metodi di soluzione a problemi per la massimizzazione della utilià di rete che sorgono nel contesto delle moderne reti di comunicazioni. In particolare, i problemi analizzati sono relativi a applicazioni per la rete Internet sensibili al ritardo e problemi di allocazione della banda in reti affette da incertezza.

Nel primo problema analizziamo come allocare i tassi di trasmissione in un system per videoconferenze multiutente. Il principale contributo di questo capitolo é un metodo approssimato per la rapida allocazione dei tassi di trasmissione che sia in grado di potersi adattare velocemente alla variazione delle condizioni della videoconferenza. Questo rapido adattamento non può essere raggiunto con metodi classici per la massimizzazione della utilità di rete, in quanto questi ultimi sono solitamente basati su metodi iterativi. In questo caso, invece, usiamo a nostro vantaggio la particolare struttura del problema e ideiamo un nuovo metodo di risoluzione distribuito, il quale si dimostra molto efficace rispetto ad altre metodi di riferimento.

Il secondo problema che affrontiamo, é la progettazione di un algoritmo per il controllo della congestione per applicazioni sensibili al ritardo. Uno dei problemi principali degli algoritmi per il controllo della congestione basati sul ritardo di pacchetto é il fatto che tendano a raggiungere un tasso di trasmissione estremamente basso quando competono per le risorse di rete contro algoritmi basati sulle perdite di pacchetto. Al fine di superare questa problematica, proponiamo un nuovo controllore adattivo basato su un approccio *bandit problem*. Il controllore adattivo cerca di inferire come la coppia tasso di trasmissione - ritardo all'equilibrio cambi quando la sensibilità al ritardo di un algoritmo per il controllo della congestione basato sul ritardo viene modificata. Dopo che la reazione della rete é

stata stimata il controllore seleziona la sensibilità che permette di raggiungere il miglior compromesso fra il tasso di trasmissione raggiunto e il ritardo subito.
Nel problema finale analizziamo la progettazione di un sistema per la allocazione dei tassi di trasmissione sovrapposto, da essere utilizzato quando la quantità di risorse disponibili non é nota, e il segnale di retroazione della rete non può essere utilizzato dai singoli utenti al fine di ottimizzare la allocazione dei tassi di trasmissione. Questo scenario si presenta quando alcune applicazioni della rete Internet vogliono massimizzare una certa metrica di utilitá, ma sono costrette ad operare utilizzando un algoritmo per il controllo della congestione che risulta completamente agnostico riguardo a tale metrica. Per risolvere questo problema progettiamo un sistema distribuito che coordina gli utenti al fine di compiere un apprendimento attivo delle risorse di rete. Il fine ultimo del sistemo é quello di massimizzare nel lungo termine la utilitá a livello applicazione dell'intero sistema.
Per riassumere, in questa tesi affrontiamo specifici problemi che sorgono nelle moderne comunicazioni della rete Internet. In particolare, i problemi considerati, affrontano problematiche legate a comunicazioni sensibili al ritardo, dove é importante adattarsi velocemente e mantenere una bassa latenza fra i due nodi della rete, e comunicazioni in ambienti incogniti, dove é essenziale tenere conto delle incertezze esistenti al fine di offrire buone prestazioni del sistema nel lungo termine.

Parole chiave: Massimizzazione della Utilità di rete, Adattamento del Tasso di Trasmissione, Controllo della Congestione di Rete, Sistemi Adattativi, *Bandit Problems*, Apprendimento Ottimale, Inferenza Bayesiana, Processi Decisionali.

Contents

List of Figures

List of Tables

1 Introduction

In the last decades the amount of data transmitted in the Internet has increased remarkably, and, according to recent forecasts, it is expected to grow even more in the future [1]. Alongside with this growth, the type of network traffic has also changed: as novel Internet services appear, new kinds of transmission requirements arise. For instance, differently from classical Web traffic, interactive communication applications require to maintain a low communication latency in order to guarantee good experience to the users. Video streaming is another example of a relatively recent Internet application. As opposed to classical Web traffic, where the users mostly measure the communication quality in terms of connection speed, in video streaming, the quality perceived by the users relates to the image rendering quality. As the quality of a video depends on many factors (including connection speed), in order to maximize the overall user satisfaction, a network system should in general behave differently for video streaming and for simple Web traffic. The fact that many requirements of the new services offered in the Internet were not considered in its initial design, combined with the growth of these new Internet applications, calls for novel optimization frameworks for network communication.

When there is the need to optimize the data transmission in order to maximize the users' benefit while meeting the constraints imposed by the network infrastructure, the Network Utility Maximization (NUM) framework is extremely useful [2]. This framework allows to formulate a well defined optimization problem for resource allocation and to design a distributed solution method which can serve as a basis to build an effective network protocol. In this thesis we analyze several NUM problems related to current Internet communication challenges with applications that are both delay-sensitive and have specific user requirements. In the next section we briefly introduce a basic NUM problem that provides the common background for the specific problems tackled in this thesis, then we present the main contribution of the thesis.

1.1 Network Utility Maximization Background

Arguably the seminal paper in [2] is one of the works that has contributed the most to the popularity of the NUM framework. The NUM framework has then established as a powerful tool for the design and analysis of many communication protocols [3]. The general formulation of a NUM problem usually involves *i*) an objective function composed by a sum of utility functions, each of which is associated with a specific network user (or flow), and *ii*) a set of constraints which basically define the limitations of the communication system. The goal is then to design a network protocol that resembles a distributed solution method and achieves the optimal rate allocation.

The utility functions play a key role in NUM problems and ideally they should map all the aspects that characterize the communication in terms of benefit to a specific user. The mapping is obviously not unique and depends on the type of communication and on the model used for the user benefit. For instance, in the scenario of a simple data bulk transfer, one can model the user utility as a function of the transmission rate; whereas in video streaming, one should not only take into account the transmission rate but also the quality of the displayed video or the communication latency. The concept of utility is actually closely related to the one of Quality of Service (QoS) and Quality of Experience (QoE); the former represents a way to measure the quality of a service provided to the users, whereas the latter represents a measure of the users' satisfaction for a specific service [4]. Though not exactly, the three concepts are clearly strongly related. In the continuation of this work we will mainly refer to the concept of utility. Depending on the problem under study, we model the user utility differently. Our goal is to include the most significant factors that affect the users' satisfaction, while keeping the model as simple as possible. In fact, more complex models, though being able to describe the user's satisfaction more accurately, are also much more challenging to deal with from an optimization point of view.

In order to provide a general overview of the NUM framework, we now formulate a very simple problem that represents the common basis for the studies in the next chapters. One of the most basic NUM problems is the one depicted in Fig 1.1. In this case N users want to share a set of M network links with limited transmission capacities. The users' routes, i.e., the set of links employed by each user, are considered to be fixed. The variables over which the optimization is carried out, are the users' sending rates. The users want to adapt those in order to maximize the overall utility $\mathcal{U}$. Mathematically the problem can be formulated as follows:

$$\begin{aligned} \underset{\mathbf{x}}{\text{maximize}} \quad & \mathcal{U}(\mathbf{x}) = \sum_{n=1}^{N} u_n(x_n) \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{c} \\ & \mathbf{x} \in \mathcal{X}, \end{aligned} \tag{1.1}$$

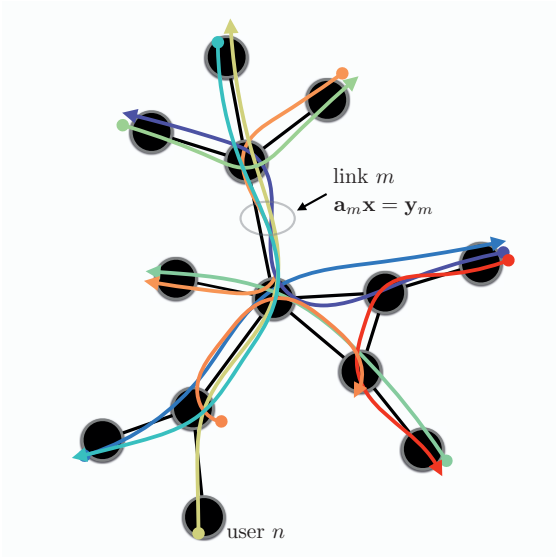


Figure 1.1 – Overview of the scenario considered in the basic NUM problem. Each user resides on a node of the network and transmits data to another network node. The colored arrows represent the communication routes of the different users.

where the vector $\mathbf{c}$ represents the available resources, i.e., the capacities of the network links. $\mathcal{X}$ represents the domain of the utility functions, which usually coincides with the positive orthant, as negative transmitting rates are meaningless. The routing information is represented by the matrix $\mathbf{A}$, which is a $M \times N$ binary matrix, where the element a_{nm} is equal to 1 if user n employs link m, and 0 otherwise. We denote the rate passing through link m by $y_m = \mathbf{a}_m\mathbf{x}$. Then $u_n(x_n)$ denotes the utility function of user n and represents the mapping between the transmitting rate x_n of user n and the resulting benefit. The utility functions are usually assumed to be concave functions [3], this in order to model elastic communication requirements and at the same time have a convex optimization problem.

Due to the fact that the number of users N and links M can grow very large, it is not possible to gather all the optimization variables in a single node and solve the optimization problem of Eq. (1.1) in a centralized way. As a result NUM problems must usually be solved using online distributed algorithms. One viable solution to solve the above problem in a distributed way is by dual-decomposition, or other decomposition methods [5]. In dual decomposition, we first need to write the Lagrange dual function of the original problem:

$$g(\boldsymbol{\lambda}) = \sup_{\mathbf{x}\in\mathcal{X}} \left\{ \mathcal{U}(\mathbf{x}) - \boldsymbol{\lambda}^{\mathsf{T}}(\mathbf{A}\mathbf{x} - \mathbf{c}) \right\}, \tag{1.2}$$

where $\boldsymbol{\lambda}$ is a vector of size M whose entries represent the dual variables associated to the

link capacity constraints. The dual problem corresponds then to solving:

$$\underset{\boldsymbol{\lambda}\geq 0}{\text{minimize}} \quad g(\boldsymbol{\lambda}). \tag{1.3}$$

Provided that the utility functions are concave, and that Slater's condition holds, then the primal problem in Eq. (1.1) and dual problem in Eq. (1.3) have a zero duality gap [6]. We can then solve the original NUM problem by solving the dual problem, which can be done by iterating over the following steps:

$$\begin{cases} x_n^t = \underset{x' \in \mathcal{X}_n}{\arg\max}\{u_n(x') - x'\mathbf{a}_n^\mathsf{T}\boldsymbol{\lambda}^t\} & n = 1...N \quad \text{(1.4a)} \\ \lambda_m^{t+1} = \left(\lambda_m^t + \epsilon\left(\mathbf{a}_m\mathbf{x}^t - b_m\right)\right)^+ & m = 1...M, \quad \text{(1.4b)} \end{cases}$$

where ϵ represents the step size and $()^+$ denotes the projection onto the positive orthant. Eq. (1.4a) represents the primal update and it basically corresponds to finding the $\mathbf{x}$ that verifies Eq. (1.2). Note that the Lagrangian of the original problem decouples with respect to the primal variables, i.e., it can be written as a sum of functions each depending on a single x_n. This is because each utility function depends only on a single user rate and the constraints are linear. Eq. (1.4b) corresponds to the dual variables update and it simply represents a step in the direction of the gradient of $g(\boldsymbol{\lambda})$[1]. The key feature of Eq. (1.4) is that the primal and dual updates can be computed independently for each user n and for each link process m. In fact, to update the rate x_n, all that is needed is the sum of the dual variables of the links employed by user n; similarly, to update the dual variable of link m, one needs only the transmitting rate of link m. By simply using this local information, we can run iteratively Eq. (1.4) and converge to the optimal rate allocation. The structure of the iterative method in Eq. (1.4) resembles the one of several NUM problems with different complexities.

Usually the set of operations described above is divided among two different sets of entities, namely users and links. The users collect the feedback from the network and adapt their transmission rates; at the same time, the network infrastructure, which represents the available resources, generates a feedback signal based on the users usage. One interesting aspect is that the network feedback is typically not something that is explicitly computed by a network node, but rather coincides with the evolution of a physical quantity that can then be measured by the users at the endpoints. For instance, in congestion control problems the network feedback to the user actions may coincide with the evolution of the queueing delay experienced by the packets in the inner network buffers. It is, however, important to point out that, even if generally exists, some sort of real implicit network feedback that the users can observe, as we will see in one of the analyzed problems, this implicit signal cannot always be used as a valid feedback signal to steer the rates to the

[1] The gradient can be replaced by a subgradient when the dual function is non-differentiable.

optimal allocation.

In this section we have introduced the NUM framework and described a possible solving method for a simple problem. Though the instances of distinct NUM problems might be different, the structure of the problems and the solution methods are usually rather similar to the one described above. The key point is usually to find an adequate decoupling of the system in order to obtain an effective distributed solving method. Alongside decoupling, other challenges may arise from particular constraints imposed by the environment or the application. For instance, delay-sensitive applications might require the adoption of a particular solving method in order to meet the time constraints. In other cases, potential uncertainties on the network environment might complicate the system design, which necessitates a more advanced and complex online solution method. These are exactly the challenges that are addressed in this thesis.

1.2 Thesis Outline

In this thesis we tackle some specific instances of NUM problems, which deal with two specific challenges that arise in today's Internet communication: *i*) guarantee low latency (fundamental for interactive communications) and *ii*) operate in unknown environments (which is often the case in a changeable and heterogenous network as the Internet). We now provide a more detailed description of the contributions that are eventually presented.

In Chapter 2, we analyze the problem of rate allocation in multiparty videoconferencing systems. Multiparty videoconferencing systems have been gaining a lot of popularity in the last years as they offer the possibility to have a virtual and rich communication experience among a large number of remote people. These applications have, however, large requirements in terms of both network and computational resources. In multiparty videoconferences, the design challenges stem from the fact that, ideally, all the users would like to have access at the same time to a high quality version of the audio and video representations of all the participants. Unfortunately, due to the potentially large number of participants, and the heterogeneity of users' transmitting capacity, this is usually not possible, so that the data transmission has to be carefully optimized.

The multiparty videoconferencing systems are usually either based on expensive central nodes, called Multipoint Control Units, with transcoding capabilities, or on a peer-to-peer architecture where users cooperate to distribute more efficiently the different video streams. Whereas the first class of systems requires an expensive central hardware, the second one completely depends on the redistribution capacity of the users, which sometimes might neither provide sufficient transmitting capacity nor be reliable enough.

In this work we propose an alternative solution where we use a central node to distribute the video streams, and we maintain the hardware complexity and the computational

requirements of this node as low as possible, e.g. it has no video decoding nor transcoding capabilities. We formulate the rate allocation problem as a NUM problem that aims at maximizing the participants' utility. The proposed algorithm is able to prioritize the different video streams according to the users' activity. As the latter can change unexpectedly, our solution is able to track the sudden variations and adapt the transmission rates as fast as possible. We propose two different distributed algorithms for solving the optimization problem. The first algorithm is able to find an approximate solution of the rate allocation problem in a one-shot execution. The second algorithm, based on Lagrangian relaxation, performs iterative updates of the optimization variables in order to gradually increase the value of the objective function. Both algorithms nicely complement each other. If executed in sequence, they allow to achieve a quick approximate rate reallocation in case of a sudden change of the system conditions, as well as a precise refinement of the variables for a better allocation. Experimental results show that our rate allocation algorithm is able to properly optimize the users' utility, providing the ability to adapt to sudden variations by quickly finding an approximate solution that is later refined. We also illustrate the benefits of our solution in terms of network usage and overall utility, when compared to a baseline heuristic method operating on the same system architecture.

While Chapter 2 studies a rate allocation problem under the assumption of using a congestion control algorithm suitable for real-time applications, we specifically address in Chapter 3 the problem of designing a novel congestion control algorithm for delay-sensitive communication. It is extremely important for this sort of congestion control algorithms to be resilient to different network conditions and to be able to effectively maximize the communication quality. In real-time applications, such as video telephony or videoconference applications, the congestion control algorithm, which is responsible for adjusting the transmission rate according to the network conditions, should typically be able to reach the largest possible rate (in oder to allow the application to achieve a high video quality) at the minimum possible delay (in order to guarantee a good interactivity). At the same time, it should also guarantee a fair share of the network resources when competing with other communication protocols, in particular loss-based congestion protocols. These two objectives actually conflict with each other: in order to achieve the largest rate with the minimum delay, the delay-based congestion control should be extremely sensitive to delay variations; at the same time, it should also be ideally immune to delay variations to achieve perfect coexistence with loss-based protocols. To solve this trade-off we propose a new learning-based adaptive controller that tunes the delay sensitivity of an underlying delay-based congestion control algorithm, according to the estimated network conditions. We first define a simple low-dimensional model for the network response. We then formulate a bayesian bandit problem for the selection of the delay sensitivity of the congestion control algorithm. By solving the bandit problem using an optimal learning method, we are able to maximize the long term utility provided to the users. Experimental results demonstrate the effective operation of the proposed

method and its ability to adapt to unknown network conditions in order to attain the best communication quality in all the possible scenarios.

Another related problem that arises in rate allocation, corresponds to the scenario where the congestion feedback measured by the users cannot directly be used as a valid constraints violation penalty to steer the transmitting rates to their optimal value. To overcome this limitation, in Chapter 4, we design an overlay rate allocation scheme that attempts to infer the actual amount of available network resources while coordinating the users rate allocation. In order to achieve this goal we first define a very general congestion signal that network users are able to detect. We then define a probabilistic model that relates this congestion signal to the actual available network resources in order to be able to infer their value. We then propose a new coordination scheme that performs active learning in order to quickly estimate the resources and improve the service performance. By adopting an optimal learning formulation we are able to balance the trade-off between an accurate estimation and an effective resources exploitation, in order to maximize the long term utility delivered to the users. Experimental results show the validity of the employed method. In particular, they show that a greedy algorithm, which leads to a passive resource estimation, would result in having extremely poor performance; whereas, thanks to the active learning method, we are able to effectively estimate the network resources and coordinate the users.

In summary, this thesis addresses several problems that arise in the context of communication networks, with particular emphasis towards delay-sensitive applications and communication in unknown environments. In Chapter 2 we consider the problem of how to achieve a fast overlay rate allocation in a multiparty videoconference system. In Chapter 3 we analyze the challenges that arise in the design of a congestion control algorithm for delay sensitive applications. Finally, in Chapter 4 we study the design of an overlay rate allocation system to optimize the transmission rates when the amount of available network resources is not known.

2 Rate Allocation in Multiparty Videoconferencing Systems

In this chapter[1] we tackle the problem of the rate allocation among a set of users participating in a videoconference. Videoconference applications allow several users to communicate together using audio/video streams and thus to provide a rich communication experience. Unfortunately, when the number of participants grows large and the resources are scarce the quality offered to the users typically degrades. In such scenarios it is fundamental to adapt properly and quickly the transmission rates in order to provide a good communication quality to the users.

More specifically, the video distribution problem in multiparty videoconferencing takes different forms, depending on the network architecture that is used for the data transmission. Since multicast technology is not widely deployed in the Internet, the most naïve implementation of a videoconferencing system is the one where each user sends his own video flow directly to all the other users. If the number of participants is equal to N, then $N-1$ video flows share the download link of each endpoint and $N-1$ replicas of the source stream share the upload link. This architecture is particularly problematic in the case of asymmetrical connections, such as asymmetrical digital subscriber lines (ADSL), where the upload capacity rapidly becomes the main bottleneck for large values of N. An alternative solution consists in using a Multipoint Control Unit (MCU) with transcoding capabilities. The MCU is an endpoint used as a communication bridge by the videoconference participants. In this case every sender sends its video only to one node (the MCU) mitigating the constraints on the upload link. The MCU, which is usually connected to the Internet with some high capacity links, transcodes the video of each sender and forwards it to all the other participants [7], possibly in different versions. This solution, however, exhibits important scalability problems due to the huge

[1] Part of this chapter has been published in:
- S. D'Aronco, S. Mena and, P. Frossard. Distributed rate allocation in switch-based multiparty videoconference. *Proceedings of the 7th International Conference on Multimedia Systems*. ACM, 2016.
- S. D'Aronco, S. Mena and, P. Frossard. Distributed rate allocation in switch-based multiparty videoconferencing system. *Transactions on Multimedia Computing, Communications, and Applications*. ACM, 2017

computational load required by the MCU for the transcoding operations. In order to alleviate the scalability problem derived from a single MCU some works proposed to adopt a peer-to-peer solution, [8, 9, 10]. Instead of having a unique central node, the users' endpoints compose a peer-to-peer network in order to improve the video delivery capacity. Although this solution is extremely scalable, it strongly relies on the upload capacities of the users' endpoints. In practice, these may not provide sufficient capacity and induce large communication delays.

In this chapter we aim at designing a multiparty videoconferencing system that offers a trade-off between a fully centralized solution with transcoding capabilities and a complete peer-to-peer solution that relies only on peers' resources. We focus on an architecture that keeps the computational requirements of the central node extremely low compared to a MCU with transcoding capabilities. Similarly to peer-to-peer systems the intelligence resides completely in the users' endpoints and the rate optimization process is fully distributed, which helps preserving system scalability.

In more details, we make the following key assumptions: *i*) the central node can enable application layer multicast communication, *ii*) the users' endpoints are capable of encoding their video streams at multiple rates (single rate encoding is included as trivial scenario), *iii*) a suitable transport protocol is available for real-time multimedia applications. The central node, also called the *switch node* in the remainder of this chapter, is used by the users as a communication hub. First, it offers a video packet forwarding service. In this way every sender can implement a 2-hop application layer multicast tree for the video delivery, alleviating the constraint on the upload link of the users. Second, it provides a coordination service among the senders and receivers in order to reach an effective utility aware rate allocation. The complexity of the central node is kept as low as possible: the computation of the layer rates, as well as the forwarding policies of the video packets, are computed by the videoconference participants in a distributed manner. The central node is thus simply a sort of "application layer" switch, hence the name *switch node*. Having low complexity at the central node is not only important for preserving the scalability of the system in case of multiple parallel videoconference sessions. From the perspective of videoconferencing providers, a low complexity central node requires less computation capabilities making the hardware cheaper and eventually decreasing the operative costs of the videoconferencing service.

More in detail, the operation of our system is the following. We first associate to every user a utility function, which depends on the activity (or importance) of the users and on the video characteristics. The utility functions and the upload/download capacity constraints are used to define an optimization problem that reflects the entire videoconference setup. We then propose two methods for solving this optimization problem. The first one provides a fast and efficient way to obtain an approximate solution. The second one is an iterative method that gradually improves an initial guess in order to achieve a higher objective value. Thanks to the structure of the problem, both methods can be

implemented in a distributed way preserving the overall scalability of the system. The two proposed algorithms are actually executed in sequence every time the parameters of the original optimization problem change, e.g., when the relative importance of the users varies. In this way we can exploit the features of both methods: the fast algorithm modifies the rate allocation in a single step trying to reach quickly a good approximate solution, the iterative algorithm then refines the approximate solution in order to further improve the objective function.

We carefully evaluate the performance of our system in the network simulator NS3 by replicating realistic network settings. We use the Network-Assisted Dynamic Adaptation (NADA) [11] as congestion control algorithm to send the media data streams and we perform the proposed rate allocation methods on top of it. The conducted experiments show that our system is able to properly allocate resources for optimizing the utility measure of the users. The fast algorithm provides a quick good approximate solution to the rate allocation problem, while the iterative method provides a better solution at the price of slower convergence.

The remainder of this chapter is composed as follows. The problem settings and formulation are described in Section 2.1. In Section 2.2 and Section 2.3 we introduce the fast and iterative solution methods. We discuss the algorithm implementation in Section 2.4. In Section 2.5 we provide some simulation results of the implemented system. In Section 2.6 we discuss the related work on multiparty videoconferencing systems. Finally, conclusions are given in Section 2.7

2.1 Mixed Integer Non-linear Programming Formulation

2.1.1 Problem Settings

We target a scenario where N users participate in a videoconference. All the participants are both senders and receivers, thus the video of every user should ideally be received by all the other users.

The proposed system architecture is a hub topology with the hub node corresponding to the switch node. Fig. 2.1 depicts a simple topology example. Every user establishes a bidirectional connection with the switch node using a general Congestion Control (CC) algorithm suitable for real-time communications. Multiple streams of video data are sent from a single user node to the switch node, and vice versa. In our solution, all streams from the switch to a user share a single download session. Another session is active on the reverse path for uploading the user's video streams. This way we can improve the responsiveness of the system when the rates of the video streams need to be modified. The idea of coupling several Real-time Transport Protocol (RTP) flows in order to gain more flexibility is similar to the one described in the internet draft [12] developed in the

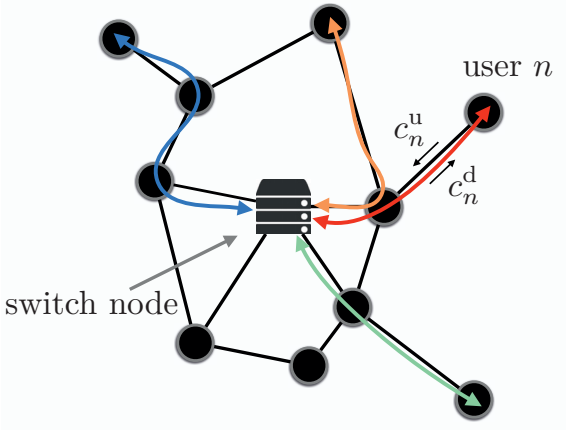


Figure 2.1 – System example.

context of the RTP Media Congestion Avoidance Technique (RMCAT) working group [13]. Real-time CC algorithms maximize the sending rate and at the same time try to limit the end-to-end delay experienced by the user. We identify with c_n^{d} the downloading rate achieved by the CC algorithm from the switch node to the endpoint of user n. Similarly we denote with c_n^{u} the uploading rate from the user endpoint to the switch node. The download and upload rates are actually time-varying, however, in our analysis we neglect the time dependency to make the notation lighter.

We assume that every user is able to encode its video into one or multiple streams at different bitrates. Encoding a larger amount of streams is obviously more hardware demanding. In this work we let the maximum number of encoded streams to be different among the videoconference participants, so that we can emulate the possible hardware heterogeneity of the endpoints. Although the proposed framework can easily be adapted to the case of Multiple Descriptor Coding (MDC) or multiple independent video streams, we consider the specific case of a Scalable Coding (SC), e.g., Scalable Video Coding (SVC) [14], which offers a compromise between adaptation to receivers' capacity heterogeneity and overall resource requirements. In SC, a video is encoded using different layers, namely a base layer, and one or more enhancement layers. Users can increase video quality by stacking several enhancement layers on top of each other. The advantages of SC with respect to independent coding is illustrated by a simple example. Consider the case where one user wants to serve a video available at two rates, e.g., 0.5 Mbps and 1 Mbps. With SC the sender can encode the video progressively in two layers: one base layer of 0.5 Mbps; and an enhancement layer of 0.5 Mbps, the user will then send a total of 1 Mbps. Both layers are necessary at the receiver to decode the full quality stream. In the case of independent coding, the user needs to send a total of 1.5 Mbps, since the two streams are independent. SC enables a more efficient communication as it avoids redundancy in the data transmission. The implementation of such scalable method however adds additional overhead to the transmitted data, therefore the effective rate, in terms of video quality,

reduces when more quality layers are used. Finally, we indicate with L_m the maximum number of different SC layers that the user m can encode, and we denote with r_{ml} the rate required to decode the l-th layer of user m. Thus, r_{ml} is not the rate of the single l-th layer, but the cumulative rate of all the layers that are required to decode the layer l. The coding rate of layer l can thus be written as $r_{ml} - r_{m(l-1)}$ for $1 < l \leq L_m$ (with $r_{m0} = 0$).

Similarly to other works [9, 10], we treat the video streams of the users differently based on their content. In a videoconference, not all the video streams are equivalent: *i*) the video content of some users might be more complex and require a higher encoding bitrate than the one of other senders for the same quality, *ii*) some users might be more active than others in the videoconference. When the rate allocation of the users is properly computed, a larger rate should be reserved to the most demanding and high priority users in order to maximize the overall utility of the videoconference. In order to design our utility aware rate allocation system, we need to define a mathematical model to measure the utility of a user. We define $u_m(r)$ as the utility of receiving the stream of user m at rate r. The utility function $u_m(r)$ is assumed to be a strictly concave increasing function of the rate, and embeds both the video complexity of the scene, and the importance that the user m plays in the videoconference. The total utility of the receiver n, modeled as a sum of the utilities of the different streams, can be written as:

$$\mathcal{U}_n = \sum_{m=1, n \neq m}^{N} u_m \left(r_m \right). \tag{2.1}$$

An intuitive choice for the utility function could be any sort of video quality metric multiplied by a scalar gain that reflects the user's importance in the videoconference. In this problem formulation we model the utility of receiving a video stream exclusively from its rate. In interactive communication however, the communication delay is also a critical quantity for the overall quality of the communication and, ideally, it should also be taken into account. We however neglect this quantity because our rate allocation is carried out on top of the real-time CC algorithm that is actually responsible for achieving a low communication delay. In our scenario we assume that each real-time congestion control session seeks for the best trade-off between the overall transmitting rate and the experienced delay independently of the layer rate allocation. Since the overlay rate allocation carried out by our algorithm has no effect on the experienced delay, we can safely ignore this quantity in our utility model.

As mentioned above, there is usually a distortion penalty with SC coding, and the coding efficiency of the SC decreases with the number of encoded layers for a given total encoding rate. In order to model this effect the utility function should also depend on the layer number l, and the utility should ideally decrease when l increases. For the sake of simplicity we neglect this dependency in our model as the distortion penalty is negligible as long as the layer rates are large enough. It is however possible to include this behavior

in the utility function for a more precise model if necessary.

2.1.2 Problem Formulation

Given the settings described in the previous section, the rate allocation problem for maximizing the utility of the videoconference can be stated as follows:

$$\underset{\{\mathbf{z}_n\},\{\mathbf{r}_m\}}{\text{maximize}} \quad \sum_{n=1}^{N}\sum_{m=1}^{N}\sum_{l=1}^{L_m} z_{nml}u_m(r_{ml}) \tag{2.2a}$$

$$\text{subject to} \quad \sum_{m=1}^{N}\sum_{l=1}^{L_m} z_{nml}r_{ml} \leq c_n^{\mathrm{d}}, \quad \forall n \tag{2.2b}$$

$$r_{ml} \leq c_m^{\mathrm{u}}, \quad \forall\, l,m \tag{2.2c}$$

$$\sum_{l=1}^{L_m} z_{nml} = 1 - \delta_{nm}, \quad \forall n,m \tag{2.2d}$$

$$r_{ml} \in [R_{\min}, R_{\mathrm{MAX}}], \quad \forall m,l \tag{2.2e}$$

$$z_{nml} \in \{0,1\}, \quad \forall n,m,l \tag{2.2f}$$

The above optimization problem aims at maximizing the overall utility of the users by selecting the video layers that every user has to receive and by allocating the rates of the different layers that each sender has to encode. The variables of the optimization problem are: *i)* z_{nml}, the decision variables for selecting which layer l of user m should be received by user n ($\mathbf{z}_n$ denotes a matrix of size $M \times \max\{L_m\}$ containing the layers selection of user n); *ii)* r_{ml}, the rates of the different video layers ($\mathbf{r}_m$ denotes a vector of size L_m containing the layer rates of user m). The objective function of the optimization problem corresponds to the sum of the receivers' utility functions defined in Eq. (2.1).

The first set of constraints (2.2b) represents the download capacity constraints, which restrict the sum of the rates of the received layers to be no larger than the download capacity c_n^{d}. The second set of constraints (2.2c) defines the limit on the upload capacity of the users; practically the largest cumulative layer rate of user m has to be smaller than or equal to the upload capacity c_m^{u}. In (2.2d) we impose that every user gets exactly one version of every other source stream and no version of his video stream (δ_{nm} denotes the Kronecker delta). The next constraints define the limits on the values that the variables can take: constraints (2.2e) define some possible maximum and minimum encoding rates for each sender, while constraints (2.2f) limit the value of the decision variables to belong to the set $\{0,1\}$.

Note firstly that in the case where independent video coding is used instead of SC, the constraints (2.2c) have to be changed. Since the encoding streams are independent in

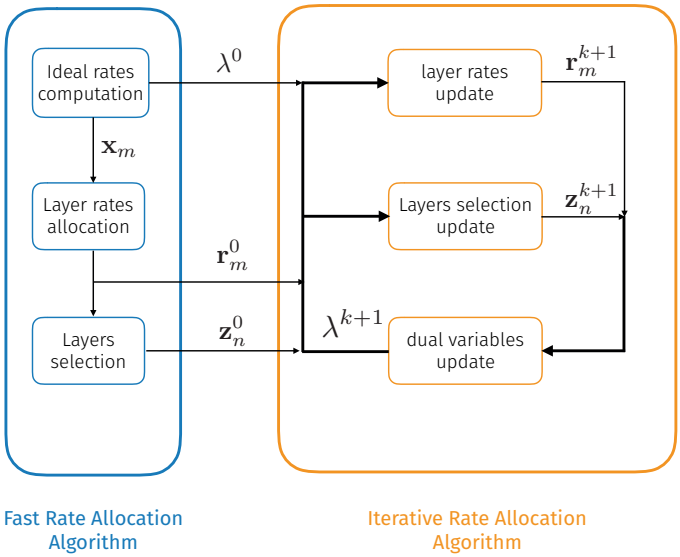


Figure 2.2 – Steps of the proposed rate allocation algorithm.

this case, the new constraints become $\sum_{l=1}^{L_m} r_{ml} \leq u_m$. Finally note that the parameters of the above optimization problem, e.g., download/upload capacities and users' weights, are time-varying. The problem in Eq. (2.2) needs to be reevaluated whenever parameters change. The quick computation of the new optimal allocation is therefore fundamental to guarantee a good utility value to the users in dynamic conditions.

The above problem represents a Non-convex Mixed-Integer Nonlinear Programming (MINLP) problem. Non-convex MINLPs are generally $\mathcal{NP}$-hard [15] and therefore it is extremely difficult to find the global optimal solution. We therefore focus in the next sections on the design of fully distributed algorithms in order to find good suboptimal solutions for Eq. (2.2) and at the same time preserve the scalability of the system.

In the next sections we describe two methods that we use for finding a suboptimal solution to the aforementioned problem. The first method provides a fast and approximate solution and is based on decomposing the original problem into three easier subproblems. The second method is an iterative method based on a Lagrangian relaxation.

2.2 Fast Rate Allocation Algorithm

The intuition behind the fast rate allocation method is the following. When the available layer rates of the senders are fixed, every receiver can easily and independently compute the best combination of the layers to maximize the utility function. This layer selection problem is actually the discrete version of the one where receivers are free to choose

the rate of every sender as a continuous variable. By asking the receiver to solve the continuous version we can collect a list of *ideal rates* that the receivers want to get; using these ideal rates, the senders can then set up a list of available rates in order to best satisfy the different receivers. Finally receivers can select from the available rates computed by the senders, the ones that maximize their utility. The original rate allocation problem is therefore divided into three subproblems as shown in Fig. 2.2. We now describe in details the individual steps that compose the fast rate allocation method.

2.2.1 Ideal Rate Computation

The ideal rates of user n, denoted by $\mathbf{x}_n$, can be computed using the download capacity and the importance of the different participants. In mathematical terms, the ideal rates of user n are the solution to the following optimization problem:

$$\mathbf{x}_n = \arg\max_{\mathbf{x}'_n} \quad \sum_{\substack{m=1 \\ n \neq m}}^{N} u_m(x'_{mn}) \tag{2.3a}$$

$$\text{subject to} \quad \sum_{\substack{m=1 \\ n \neq m}}^{N} x'_{nm} \leq d_n. \tag{2.3b}$$

Since we assume concave and non-decreasing utility functions the solution to this optimization problem is unique and can be easily computed using basic convex optimization algorithms [6]. The objective function of this problem corresponds to the receiver utility function as defined in Eq. (2.1), while the constraint simply imposes that the sum of the rates is smaller than the downloading rate set by the congestion control of user n. This optimization problem is solved independently by every receiver. By combining the results of problem in Eq. (2.3) for all the N receivers we obtain for each sender a set of $N-1$ ideal rates that can be used as guideline to decide the layer rates to make available.

2.2.2 Layer Rates Allocation Problem

This subproblem is the most complex to solve among the three steps composing the fast rate allocation method. It corresponds to the rate allocation of the different video layers at each sender based on the ideal rates collected from the receivers. The optimal layer allocation is carried out similarly to [16, 17]. The method is based on the following intuition: if a receiver n requests an ideal rate of x_{nm} for sender m, then the corresponding utility is certainly maximized if the sender m encodes a video layer with a cumulative rate exactly equal to x_{nm}. If a slightly lower rate is available for the receiver n the utility will be surely smaller than the ideal one. On the other hand, if a slightly higher rate is available, then the overall utility is also going to be smaller. In fact, in order to select a

higher rate, the receiver has to give up some transmission rate that is ideally planned to be used for other senders. This modifies the ideal (optimal) rate allocation and decreases the overall receiver utility. Using the above observations the sender can leverage the ideal rates of the receivers in order to find a suitable set of encoded layer rates to maximize performance of the receivers.

In order to proceed we first need to introduce a second concept of utility, namely the layer utility, written as:

$$u_{\text{layer}}(x_{nm}, r_{ml}) = \begin{cases} x_{nm}/r_{ml} & : x_{nm} < r_{ml} \\ r_{ml}/x_{nm} & : x_{nm} \geq r_{ml} \end{cases} \tag{2.4}$$

This function attempts to model the loss of utility when a receiver n with ideal rate x_{nm} receives instead a video layer with cumulative rate equal to r_{ml}. The choice of the utility function in Eq. (2.4) is not unique. It is however important that *i*) its maximum value is achieved when $x_{nm} = r_{ml}$, *ii*) it is non-decreasing for $0 \leq r_{ml} \leq x_{nm}$ and *iii*) non-increasing for $x_{nm} \leq r_{ml}$. Following the above intuition we can assume that, when multiple encoded rates are available, every receiver selects the ones that are closest to the ideal rates. We can therefore define the overall layer allocation utility as:

$$\mathcal{U}_{\text{layer}}(\mathbf{x}_m, \mathbf{r}_m) = \sum_{\substack{n=1, \\ n \neq m}}^{N} \max_{r_{ml}} u_{\text{layer}}(x_{nm}, r_{ml}) \tag{2.5}$$

where $\mathbf{r}_m$ denotes a vector containing the L_m layer rates of sender m. Note that the higher the value of $\mathcal{U}_{\text{layer}}$ the better the layer rates fit the ideal rates received.

The layer rates allocation problem can be solved efficiently with a dynamic programming approach by making the following observation. When a layer l is added on top of the other $l-1$ layers, the only receivers that might be interested in switching to this new layer are the ones that are using the second highest layer $l-1$. Obviously, a user that is expected to choose a layer lower than $l-1$ is not be interested in selecting a new layer higher than $l-1$. This observation can be stated more formally as follows:

$$\mathcal{V}^*_{\text{layer}}(\mathbf{x}_m, l, r_{ml}) = \underset{r_{m,l-1}}{\text{maximize}} \left\{ \mathcal{V}^*_{\text{layer}}(\mathbf{x}_m, l-1, r_{m,l-1}) + \delta_l(\mathbf{x}_m, r_{m,l-1}, r_{ml}) \right\}. \tag{2.6}$$

$\mathcal{V}^*_{\text{layer}}(\mathbf{x}_m, l, r_{ml})$ represents the optimal layer utility value when l encoding layers are used with the cumulative rate of the last layer equal to r_{ml}. Note that $\mathcal{V}^*_{\text{layer}}$ and $\mathcal{U}_{\text{layer}}$ both represent the total layer utility, but in terms of different input variables. The meaning of Eq. (2.6) is the following: the largest possible utility for layer l with cumulative rate r_{ml} is the sum of two terms: the first term is the best utility that we have using $l-1$ layers with the highest cumulative rate of $r_{m,l-1}$; and the second term is the utility gain

of adding a layer l at rate r_{ml}. This layer can only affect the users with a rate larger than $r_{m,l-1}$, thus the possible increase in the utility $\delta_l(\cdot)$ depends only on the layer rates $r_{m,l-1}$, r_{ml} and obviously the ideal rates $\mathbf{x}_m$.

We explain now how we solve the problem of Eq. (2.6) using a dynamic programming procedure. The first step is to discretize the possible values that the layer rates can take, we denote this quantity by Δ_r. The discretization step is a trade-off between the computational complexity and the accuracy of the solution. We fix $l = 1$ and we evaluate $\mathcal{V}^*_{\text{layer}}(\mathbf{x}_m, 1, r_1)$ for $r_{m,1}$ varying gradually from $R_{\min}, R_{\min} + \Delta_r, R_{\min} + 2\Delta_r, ..., R_{\text{MAX}}$. We then increase the value of $l = 2$, and we use the computed values of $\mathcal{V}^*_{\text{layer}}(\mathbf{x}_m, 1, r_{m,1})$ to calculate $\mathcal{V}^*_{\text{layer}}(\mathbf{x}_m, 2, r_{m,2})$ according to Eq. (2.6). We proceed in the described way until the maximum number of layers L_m for sender m is reached. We can then track back the optimal set of layer rates starting from the value of $\mathcal{V}^*_{\text{layer}}(\mathbf{x}_m, L_m, r_{m,L_m})$. Since the original problem requires that all the users receive the video of all the other participants, we force the first layer to be equal to the smallest ideal rate, in this way we can guarantee that every receiver is able to select at least the base layer from all the other participants.

We now briefly discuss the drawbacks of the above method to select the set of video layers. In order to measure the true overall layer utility the sender should know which layer is going to be chosen by each receiver, which is unfortunately an information that is not available. In fact, the receiver selection of the layers to receive from the sender m, depends on the layer rates of all the other senders as all the streams compete for the same download capacity at the receiver side. To replace this missing information, in the above procedure, we assume that the user n selects the layer that leads to the highest layer utility. Whereas this does not represent the finial choice for the videoconference receivers, the policy followed by the receivers is usually close to the approximated one. Moreover, it is thanks to this assumption that we are able to come up with a distributed fast allocation method.

2.2.3 Layer Selection Problem

After the layer rates allocation step, every sender m has fixed the rates of the L_m layers using the ideal rates. The final step consists in the selection, by the receivers, of the layers they want to receive based on the download capacity and the importance of the

users. The optimization problem can be stated as follows:

$$\mathbf{z}_n = \underset{\mathbf{z}'_n}{\operatorname{argmax}} \quad \sum_{m=1}^{N} \sum_{l}^{L_m} z'_{nml} u_m(r_{ml}) \tag{2.7a}$$

$$\text{subject to} \quad \sum_{m=1}^{N} \sum_{l=1}^{L} z'_{nml} r_{ml} \leq d_n \tag{2.7b}$$

$$\sum_{l=1}^{L_m} z'_{nml} = 1 - \delta_{nm} \quad \forall\, m \tag{2.7c}$$

$$z'_{nml} \in \{0, 1\} \quad \forall n, m, l. \tag{2.7d}$$

This problem represents the discretized version of the problem in Eq. (2.3). In fact, as for Eq. (2.3), the objective function of this problem represents the utility function of the receiver n, the download capacity constraint establishes the limit of the amount of data that can be downloaded, whereas the second set of constraints impose the limit of receiving at most one layer from each sender, as imposed in the original problem. This problem is a constrained Integer Linear Programming (ILP) problem [18] and can be solved exactly by using an ILP solver.

With respect to the original problem of Eq. (2.2), the rate allocation provided by the fast algorithm is not guaranteed to be optimal as the assumptions made in order to solve the second subproblem are not always valid. This solution method, however, makes it possible to solve the overall problem in a distributed manner and in a single round of operations which is a large advantage in practice.

2.3 Iterative Rate Allocation Algorithm

The method described in the previous section does not represent the only approach that can be used to find an approximate solution to Eq. (2.2). In this section we develop a second iterative algorithm, which is based on a Lagrangian relaxation of the original problem. Compared to the previous method, which is able to find the solution in a single round of operations, the new procedure iteratively modifies the optimization variables in the attempt to improve the value of the objective function while respecting the constraints.

As we want to have a distributed algorithm, we decompose the rate allocation problem among the different users following the decomposition methods that are usually used for NUM problems [5]. The fundamental feature that permits to decompose NUM problems is the fact that the objective function corresponds to a sum of utilities, each depending on a single sending rate. As a result, every sender is able to compute the derivative of the objective function with respect to the generated layer rates, hence to optimize the rates

locally and independently of the other variables. In our case, this is only partially true however: the gradient of the objective function computed with respect to the layer rate r_{ml}, does not depend on all the other layer rates but it depends on the layer selection variables $\{\mathbf{z}_n\}$ (similar reasoning is true for layer selection variables: optimizing over $\mathbf{z}_n$ can be done independently of the layer selections of the other users but not of the variables $\{\mathbf{r}_m\}$). In order to have a distributed iterative algorithm we need to update the layer rates and the layer selection variables alternatively, by fixing the other ones.

In a distributed algorithm it is not only necessary to decouple the objective function but also the constraints. We can recognize different types of constraints in Eq. (2.2): (2.2e)-(2.2f) are bounds on individual variables and can be handled locally. Constraints (2.2c) and (2.2d) impose conditions on the vectors $\mathbf{r}_m$ and $\mathbf{z}_n$ respectively, since $\mathbf{r}_m$ and $\mathbf{z}_n$ are computed by a single client node they can also be handled locally. The download capacity constraints of Eq. (2.2b) are the constraints that relate all the variables of the optimization problem and does not allow for a distributed solution. We can apply a dual decomposition method by plugging the download capacity constraints in the objective function and solve the problem in a distributed way. The Lagrangian relaxation of the original optimization problem of Eq. (2.2) can be written as follows:

$$
\begin{aligned}
g(\boldsymbol{\lambda}) = \underset{\{\mathbf{z}_n,\mathbf{r}_m\}}{\text{maximize}} \quad & \sum_{n=1}^{N}\sum_{m=1}^{N}\sum_{l=1}^{L_m} z_{nml}u_m(r_{ml}) - \lambda_n\left(z_{nml}r_{ml} - c_n^{\mathrm{d}}\right) && \text{(2.8a)}\\
\text{subject to} \quad & r_{ml} \le c_m^{\mathrm{u}} \;\; \forall\, l,m && \text{(2.8b)}\\
& \sum_{l=1}^{L_m} z_{nml} = 1 - \delta_{nm} \;\; \forall\, n,m && \text{(2.8c)}\\
& r_{ml} \in [R_{\min}, R_{\mathrm{MAX}}] \;\; \forall m,l && \text{(2.8d)}\\
& z_{nml} \in \{0,1\} \;\; \forall n,m,l. && \text{(2.8e)}
\end{aligned}
$$

where $\lambda_n \ge 0$ corresponds to the Lagrangian multiplier, or dual variable, associated to the download capacity constraints of user n ($\boldsymbol{\lambda}$ denotes the entire vector of dual variables). The problem in Eq. (2.8), however, still hides two further complications: *i*) as it is still non-convex recovering the global optimal would require too many computations, *ii*) the solution cannot be found in a distributed way with respect to both sets of variables $\{\mathbf{z}_n\}$ and $\{\mathbf{r}_m\}$ at the same time, as the two groups are still coupled. To find an approximate solution to the problem we then alternate between the two groups of variables, if we fix the layer rates we can optimize locally over the layers selection and vice versa.

For any fixed set of values $\boldsymbol{\lambda}$ the solution to (2.8) provides an upper bound to the original problem of Eq. (2.2) [6]. The dual problem corresponds to the minimization of this upper

bound:

$$\underset{\boldsymbol{\lambda}\geq 0}{\text{minimize}} \quad g(\boldsymbol{\lambda}) \tag{2.9}$$

Unfortunately in this case the original problem is not convex, thus the duality gap is not guaranteed to be zero and by solving the dual problem we might not find the optimal solution to the original one. The overall iterative method thus consists in optimizing alternatively over $\{\mathbf{z}_n\}$, $\{\mathbf{r}_m\}$ and, $\boldsymbol{\lambda}$. In the following we describe more in detail the individual updates, the separate steps are also depicted in Fig. 2.2.

2.3.1 Layer Rates Iterative Update

The update of the layer rates is simply done by taking a small step in the direction of the gradient of the objective function of Eq. (2.8a) computed with respect to $\mathbf{r}_m$:

$$\Delta_{ml} = \alpha \left(\sum_{n=1}^{N} z_{nml}^{k} u'_m(r_{ml}^{k}) - \sum_{n=1}^{N} z_{nml}^{k} \lambda_n^{k} \right) \tag{2.10a}$$

$$r_{ml}^{k+1} = \max\{R_{\min}, \min\{r_{ml}^{k} + \Delta_{ml}, R_{\text{MAX}}, u_m\}\}, \tag{2.10b}$$

where $u'_m(r)$ denotes the derivative of the utility function, Δ_{ml} represents the variation of the rate variable r_{ml} and $\alpha > 0$ is a simple parameter that controls the step length of the update equation. The second equation is needed in order to respect the constraints in Eq. (2.8b)-(2.8d). The primal variables $\mathbf{z}_n$ and the dual variables $\boldsymbol{\lambda}$ are fixed in this step.

Note that in some cases the derivative of the objective function of Eq. (2.8a), computed with respect to some layer rates, might vanish, which results in a null variation of the rate in Eq. (2.10). This happens when the layer is not selected by any receiver, i.e., $\sum_{n=1}^{N} z_{nml}^{k} = 0$. In this case, the layer does not contribute to the objective function value and the derivative is therefore null. If a layer is not selected in one iteration, it is very unlikely that it will be selected in the future ones if its rate is kept fixed. Therefore, to avoid this pitfall, we can randomly change the rate of the layers that are not selected in order to promote exploration of new solutions.

2.3.2 Layer Selection Iterative Update

The layer selection variables are discrete and it is not possible to apply a gradient ascent step as above. We rather solve an integer programming problem similar to Eq. (2.8) with respect to $\mathbf{z}_n$ with an additional constraint that limits the variation of $\mathbf{z}_n$ with respect to

its previous value. The problem is the following:

$$\mathbf{z}_n^{k+1} = \underset{\mathbf{z}'_n}{\text{argmax}} \quad \sum_{m=1}^{N} \sum_{l}^{L_m} z'_{nml} u_m(r_{ml}^k) - \lambda_n^k \left(\sum_{m=1}^{N} \sum_{l=1}^{L_m} z'_{nml} r_{ml}^k \right) \tag{2.11a}$$

$$\text{subject to} \quad \sum_{l=1}^{L_m} z'_{nml} = 1 - \delta_{nm} \quad \forall\ m, n \tag{2.11b}$$

$$\sum_{l=1}^{L_m} \sum_{m=1}^{N} (z'_{nml} - z_{nml}^k)^2 \leq 2 \tag{2.11c}$$

$$z'_{nml} \in \{0, 1\} \quad \forall n, m, l. \tag{2.11d}$$

This problem represents an integer programming problem with quadratic constraints and it is solved independently by every receiver. In order to limit the variable variations we introduce the quadratic constraint Eq. (2.11c). This constraint limits the difference between the old layer selection variable and the new one, by allowing the receiver to change the received layer for one sender at each iteration. This constraint is meant to avoid abrupt variations of the layer selection variables and to improve the stability of the algorithm.

2.3.3 Dual Variables Iterative Update

The last update concerns the dual variables. As mentioned previously the update of the dual variables coincides with a step in the direction of the negative gradient of Eq (2.8a) computed with respect to $\boldsymbol{\lambda}$ followed by a projection onto the positive orthant:

$$\lambda_n^{k+1} = \left(\lambda_n^k + \beta \left(\sum_{m=1}^{N} \sum_{l=1}^{L_m} z_{nml}^{k+1} r_{ml}^{k+1} - d_n \right) \right)^+, \tag{2.12}$$

where $\beta > 0$ is a simple parameter that limits the step length of the dual variable update, and $()^+$ denotes the projection onto the positive orthant. We would like to stress the fact that dual variables are updated according to a very intuitive rule: if a constraint is violated the associated lambda will grow; on the other hand, if a constraint is neither violated nor tight the dual variable will decrease. Finally note that the dynamic update in Eq. (2.12) does not admit equilibrium points that are infeasible. This can easily be understood by setting the dual variable variation to zero in Eq. (2.12).

2.4 Full Algorithm Implementation

In the previous sections we described two possible approaches to find an approximate solution to Eq. (2.2). Whereas the fast algorithm is based on the similarity between the problem in Eq. (2.3) and Eq. (2.7), the iterative algorithm is based on dual decomposition analogously to the many other NUM solution methods. Also, the features of the algorithm are radically different. The fast algorithm does not need an initial guess and provides a solution that is likely to be good, but it does not offer a way to further improve this solution. On the other hand, the iterative algorithm is able to gradually improve the solution but it requires an initial guess. Moreover, since the problem is not convex, local maxima are possible. Hence, a good initial guess is extremely important to achieve good performances. From this perspective, the two algorithms are not mutually exclusive but actually can be combined in order to achieve better solution. As depicted in Fig. 2.2, we can first run the fast algorithm to obtain a good initial guess of the solution, and then refine this guess using the iterative algorithm and improve the value of the objective function.

We explain now the full implementation of the videoconferencing system. The system is composed of two types of applications: the client application and the switch node application. In Fig. 2.3 a simplified block diagram with the client and the switch node applications is depicted. We briefly recall here which information can be measured locally by the different parts of the system. The client nodes have access to the value of their own upload and download capacity exclusively and to the utility functions of all the videoconference participants. The capacity values can be obtained from the CC whereas the utility can be extrapolated from the audio and video data received from the different users. For example, the video quality can be estimated by using some no-reference techniques and the relative importance of each stream can be inferred by detecting the current speaker (see for example speech activity detection methods, e.g., [19]). The switch node instead knows all the upload and download capacities of all the users from the CC algorithm, but it is not able to measure the utility function since it has no decoding capabilities.

2.4.1 Fast Rate Allocation Algorithm Implementation

We first describe the implementation of the fast rate allocation algorithm proposed in Section 2.2. The fast algorithm is triggered by the client nodes that independently solve the ideal rate computation problem described in Subsection 2.2.1. If the computed rates are different from the old ones, they are communicated to the switch node using a reliable protocol.

Upon reception of the new ideal rates from any of the receivers, the switch node quickly schedules the transmission of the updated ideal rates to all the senders. The small

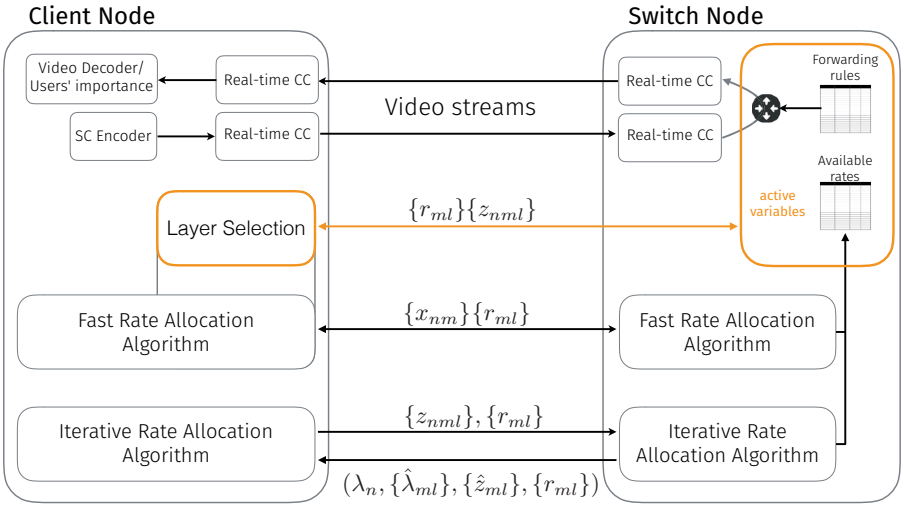


Figure 2.3 – System architecture.

waiting time before the transmission of the ideal rates permits to collect new ideal rates from other receivers before proceeding to the next rate allocation step. This ultimately reduces the communication overhead of the distributed algorithm. The users do not only communicate the new ideal rates to the switch node, but also the dual variable associated to the download capacity constraint of Problem (2.3). This information will be used later by the switch node in the execution of the iterative rate allocation algorithm. When the users receive the new ideal rates forwarded by the switch node they independently compute the new layer rates using the algorithm described in Subsection 2.2.2 and send the new rates to the switch node. When the switch node has collected the corresponding values from each client, it forwards the complete list of available layers to all receivers.

Upon the new values of the available rates are received, the users perform two operations: *i*) the sender subpart adapts the actual layer rates to the new values, *ii*) the receiver subpart solves the Layer selection problem described in Subsection 2.2.3 and sends the output of this optimization problem to the switch node. The switch node then updates the forwarding table according to the new rules communicated by the receivers. The forwarding table is a data structure that is maintained by the switch node. It contains a list of pair values (user identifier and layer identifier) for every user that indicates which layers should be forwarded to each endpoint. Every time a video packet is received by the switch node, the application checks which receivers have requested this pair of user-layer identifier and forwards the packet accordingly. Note that if the users have a different Round-Trip Time (RTT) the updates of the layer rates and the layers selections are slightly desynchronized. As a result, the capacity constraints might be violated for a short period of time causing an increase in the communication delay. However, as it is shown in Section 2.5, the desynchronization effect for typical RTT values causes a limited delay increase. In the case where the delay increase may lead to a severe quality

degradation of the interactive communication it is possible to control and limit the delay increase by properly dropping some packets or by improving the system implementation in order to take into account explicitly the desynchronization in the layer rates update.

At this point the fast rate allocation algorithm is completed and in order to improve the rate allocation solution we trigger the iterative algorithm.

2.4.2 Iterative Rate Allocation Algorithm Implementation

We now describe the implementation of the iterative rate allocation algorithm of Section 2.3, which uses the output of the fast rate allocation algorithm as initial guess. The switch node first executes a dual variable update according to Eq. (2.12)[2]. Using the new values of the dual variables and the current layer rate selection, the switch node computes the following quantities which are needed by the senders to compute the layer rates update of Eq. (2.10):

$$\hat{z}_{ml}^k = \sum_{n=1}^{N} z_{nml}^k \qquad \hat{\lambda}_{ml}^k = \sum_{n=1}^{N} z_{nml}^k \lambda_n^k. \tag{2.13}$$

The variable $\hat{z}_{ml}^k$ simply corresponds to the number of users receiving the layer l of sender m, whereas $\hat{\lambda}_{ml}^k$ is the sum of the dual variables associated to all the download links that are used by layer l of user m. The switch then communicates to each user the complete list of available rates ($\{\mathbf{r}_m^k\}$), the dual variable of the download capacity λ_m^k, the cumulative dual variable $\hat{\lambda}_{ml}^k$ and the total receivers $\hat{z}_{ml}^k$ for all the layers generated by the user m.

Each user, using the received information, updates the primal variables from Eq. (2.10) and the integer programming of Eq. (2.11). The users then communicate the updated variables to the switch node. When the switch node receives the new variables from all the users the dual variables are updated and the next iteration starts.

When we apply a primal-dual algorithm, we know that, if the equilibrium point exists, it has to be feasible. We might however achieve feasibility only asymptotically. In a realistic implementation we would like to obtain a solution that achieves feasibility in a finite amount of time. Secondly, it is also advisable, in case the download capacity is noisy, to have a small safety margin between the total downloading rate and the download capacity. In our implementation the switch node therefore uses a discounted value for the download capacities in the dual variables update, namely γc_n^{d}, with $\gamma = 0.98$.

The time taken to execute one step of the iterative algorithm depends on the RTTs

[2]The initial values for the dual variables are the ones collected from the ideal receiving rates computed in Eq. (2.3).

between the switch node and the users' endpoints. In fact, when new dual variables are computed the next iteration is executed when all the users communicate the updated primal variables. This time is roughly equal to the maximum RTT plus the time required to solve the integer programming problem of Eq. (2.11). Lower communication delays do not only improve videoconference QoE but also the convergence time of the iterative algorithm. The iterative algorithm can be executed continuously in order to track small variations of the utility functions or variations of the upload/download capacities. Note that the users' RTTs pose an upper bound on the exchange rate of information among the nodes, but we can also decrease the iteration speed in order to save network resources if the objective function has reached a sufficiently high value.

2.4.3 Further Implementation Details

We briefly list here some other implementation details in the design of our videoconferencing system.

1. Every stream from the switch node to one of the receivers goes through a sending buffer located at the switch node. These buffers are drained according to the sending rate of the congestion control algorithm. They help to handle situations where the download capacity changes suddenly, or where minor synchronization errors among endpoints may generate a sporadic surplus of packets. This occurs when the layer rates and the layers selections change, for example. In our particular implementation we simply use a droptail management policy for the output buffers [20]. More advanced scheduling techniques can also be used at this stage without affecting the design of the switch node. The maximum size of the buffers is set in such a way to limit the maximum queuing delay to 100 ms; when this limit is reached, other incoming packets are discarded. It is worth noting that packet losses at this stage are not taken into account by the congestion control, since the sessions from the sender to the switch node, and from the switch node to the receiver(s) are completely decoupled.

2. The forwarding rules that are actually used by the switch node can be updated at any time by the users. For example, if a sudden reduction of the download capacity occurs, then the users can recompute the optimal selection using the current layers and send the new rules to the switch node. The new rules will immediately become effective. This fast update is completely decoupled from the optimization algorithms, this is a sort of emergency action in case of sudden capacity (or utility) variations.

3. In Fig. 2.3 it can be seen that in the proposed implementation the variables of the iterative algorithm are not the same as the active variables used in the videoconference, the switch node is responsible for deciding when to update the active values. Obviously, there are different options here. One could decide that the primal variables computed by the algorithm at every iteration are always the

active ones. This method is dangerous because the variables might actually provide an infeasible solution of the original problem. Therefore we prefer to adopt a different solution. In our implementation the users solve the problem in Eq. (2.2.3) periodically, every second for example with the most updated layer rates of the iterative algorithm $\{\mathbf{r}_m^k\}$ and they communicate the optimal utility to the switch node. The switch node updates the active layer rates only if the expected total utility is larger than the current one, basically, we never perform a rate update that causes a drop of the overall utility function.

4. If system conditions change remarkably (e.g., huge capacity variations or new users joining the call) it is convenient to restart the allocation process from the fast rate allocation algorithm. The restart of the fast algorithm is simply triggered by sending new ideal rates to the switch node. The decision whether to restart the process or not has to be taken by looking at the variations of the utility functions and of the upload/download capacity. In our implementation we restart completely the process if *i*) a transmission capacity variation larger than 250 kbps occurs *ii*) the speaker of the videoconference changes.

5. As already mentioned both, the fast and the iterative, algorithms are fully distributed among the users and the switch node. In terms of computations the most expansive operation consists in computing the layer rates from the ideal rates. To carry out this operation we use a dynamic approach, which complexity depends on the number of layers and the discretization step that we use. Fortunately the iterative algorithm is able to refine the layer rates computed using the fast algorithm to provide a higher overall utility. As a result, it is not necessary to use excessively small discretization steps for the dynamic programming method, preserving then the complexity of the algorithm.

2.5 Experimental Results

We study now the performance of the proposed algorithm. The system has been implemented in the network simulator NS3 [21]. The simulation results illustrate the behavior of the system and show performance comparisons between the fast and iterative algorithms. We also show the benefits of the proposed solution over a heuristic and non-optimized rate allocation method that employs the same system architecture.

2.5.1 Experimental Setup

The system described in Section 2.4 has been fully implemented in the network simulator. We use the NADA congestion control algorithm to send the media packets between nodes, with the implementation described in [11]. It is worth noting, however, that the operation of the proposed rate allocation method is independent of the underlying CC algorithm

Table 2.1 – Settings for the 10 users scenario

Download capacity	[4 5 3.5 7 10.5 9 12.5 13 13.5 14] Mbps
Upload capacity	[0.7 0.7 0.7 1 1.4 1.5 2.1 1.8 2.0 1.8] Mbps

used by the system.

Similarly to many other works on NUM we use the logarithm of the rate in order to model the utility functions:

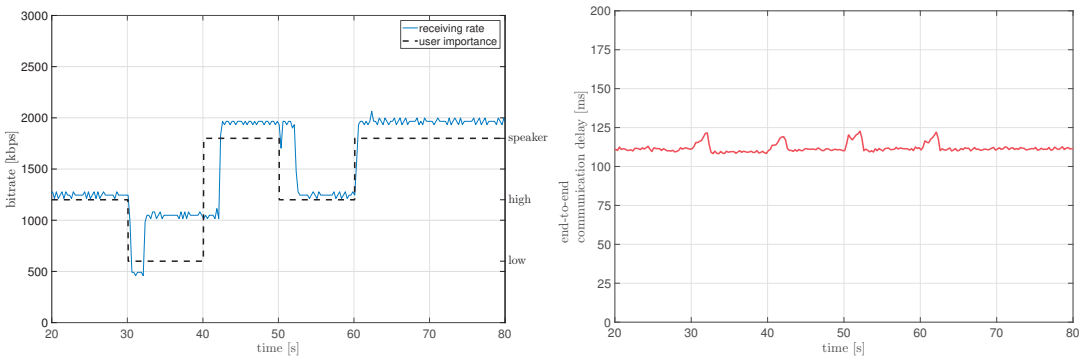
$$u_m(r_m) = w_m \log(r_m). \tag{2.14}$$

The coefficient w_m embeds the dependency of the utility function on the video content of user m. In the following simulations we assume that w_m represents only the users' activity and we assume to have the same video complexity for the different streams. The coefficient w_m can take the following values: 1 (low activity user), 2 (high activity user) and 3 (speaker). The weight of the user m can be obtained from the video-audio information coming from user m or can be signaled in the media packets. Receivers become aware of the importance of the other users simply by inspecting the incoming packets.

We consider a scenario with 10 users that participate in a videoconference. In order to have realistic transmission capacities settings we set the capacities of the links to different speed tiers that are represent realistic connection from common internet service provider. The values of the capacity for the upload/download links are listed in Table 2.1, we use heterogeneous capacity values in order to stress the performance of the proposed system. We finally assume that all the users are able to encode up to 3 video layers.

2.5.2 Fast Algorithm Evaluation

In the first test we evaluate how quickly the fast rate allocation algorithm can react to users' utility variations. We vary abruptly the users' importance during the videoconference and observe the reaction of the rate allocation algorithm. Due to the large number of participants, it is convenient to focus on a single user and analyze that particular download rate evolution. In Fig. 2.4 we show the evolution of the bitrate and the end-to-end delay of the video stream sent from user 9 to user 5. The algorithm is able to track the variation of the user importance by providing fast rate adaptation (see Fig. 2.4a). The time required to adapt the rate corresponds to 2s, which strongly depends on the RTTs between the switch node and the users. It is easy to observe that the time required by the fast algorithm for completing the rate adaptation is approximately equal, in the worst case scenario, to three times the largest RTT between the switch node and the users plus the waiting time adopted by the switch node before forwarding the ideal rates (see Subsection 2.2). Fig. 2.4b shows the evolution of the end-to-end communication delay



(a) Receiving rate (b) End-to-end delay

Figure 2.4 – Transmitted video stream from user 9 to user 5 (user 9 importance is shown)

between the two considered users. This metric mostly depends on the CC algorithm used rather than the proposed videoconferencing system. However, it is important to verify that the small desynchronization between the adaptation of the layer rates and the layer selection update does not cause large delay variations in the data transmission. When the layer rates change, the experienced delay slightly grows, but this increase is contained within 20 ms (similar results hold for the other users) which confirms that the effect of the desynchronization does not compromise the quality of the communication. Note that at time 30s and 60s the rate adaptations are almost instantaneous. In this case, it means that among the layers that are available at the time where the utilities change, there already exists a combination that can increase the utility function of user 5. Therefore the user selects this solution immediately, before the actual conclusion of the fast iterative algorithm, as discussed in the second paragraph of Subsection 2.4.3.

2.5.3 Iterative Algorithm Evaluation

In this subsection we evaluate how the layer rates evolve according to the iterative rate allocation algorithm. Since the purpose of the iterative algorithm is to progressively refine the rate allocation, and in order to evaluate the results more easily, we keep the activity of the users constant over time but we assign to each user a different importance level, namely the importance level assigned to each user are: $\mathbf{w} = [1\ 1\ 1\ 2\ 1\ 1\ 2\ 2\ 3\ 1]$. The algorithm works as described in Section 2.4: we first execute the fast algorithm described in Section 2.2 and then we use the approximate solution as initial guess for the iterative algorithm of Section 2.3.

Fig. 2.5 and Fig. 2.6 depict the evolution of the layer rates, the total number of receivers and the cumulative dual variables associated to user 6 during the execution of the iterative algorithm[3]. The upper plot in Fig. 2.5 shows the rate of the layers encoded by the user. In the initial phase we can notice the increase of the rate of layers 1 and 3. The increase

[3]In this case we consider user 6 in order to emphasize some specific features of the proposed method.

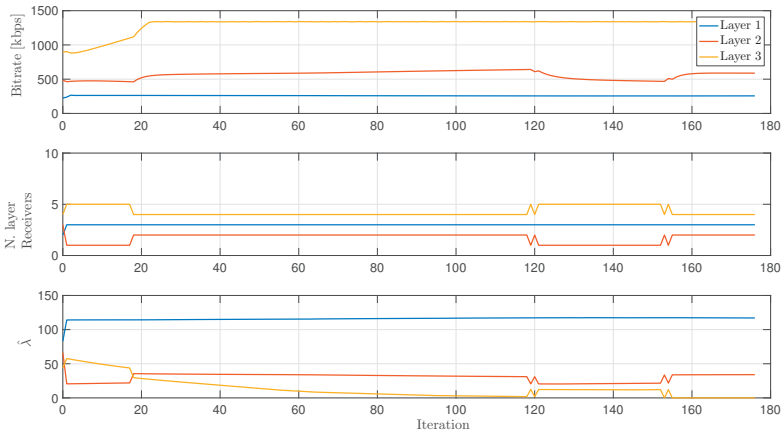


Figure 2.5 – Encoding bitrate, total receivers and cumulative dual prices for video layers generated by user 6.

of these two layers reveals one of the pitfalls that might affect the fast algorithm and that the iterative algorithm can correct. In this case other videoconference participants ideally want a lower rate from user 6 and some higher rate from other users. However, since other participants have a limited upload capacity (see Table 2.1) and are not able to provide the full requested rate, some free bandwidth becomes available this is ultimately allocated to users with a larger upload capacity, such as user 6 by the iterative algorithm, which results in a more efficient use of the upload link. The number of receivers of each layer and the cumulative dual prices are shown in the central and lower plot of Fig. 2.5 respectively. Note that the total number of receivers for all the layers is equal to 9, which means that all users in the videoconference receive the video stream of user 6, which is a constraint of the original Problem (2.2). In Fig. 2.6, some variables related to the receiving side of user 6 are depicted. In particular, the upper plot describes the total receiving rate at every iteration and the user download capacity, whereas the bottom plot illustrates the dual variable associated to the download constraint. As discussed in Section 2.3 the dual variable simply evolves according to the overuse or underuse of the capacity. When the rate exceeds the capacity the dual variable increases whereas when the constraint is not violated the dual variable decreases.

Using the same scenario, we show in Fig. 2.7 the evolution of the objective function of Eq. (2.8) as a function of the iteration. The objective function decreases, which is in agreement with the fact that Eq. (2.8) is an upper bound of the original problem of Eq. (2.2) that is minimized by the iterative algorithm. The red line is obtained by solving Eq. (2.2) with respect to the layers selection variables $\{\mathbf{z}_n\}$ with the layer rates fixed to the most recent iteration. Due to the non-convexity of the original problem we have that the upper bound is not tight to the global optimal value, thus the gap between the two lines is not guaranteed to become zero. At the beginning the objective value drops, because the iterative algorithm needs to violate the constraints in order to build up the dual variables, making all the layer rates too large to provide a good solution. When the

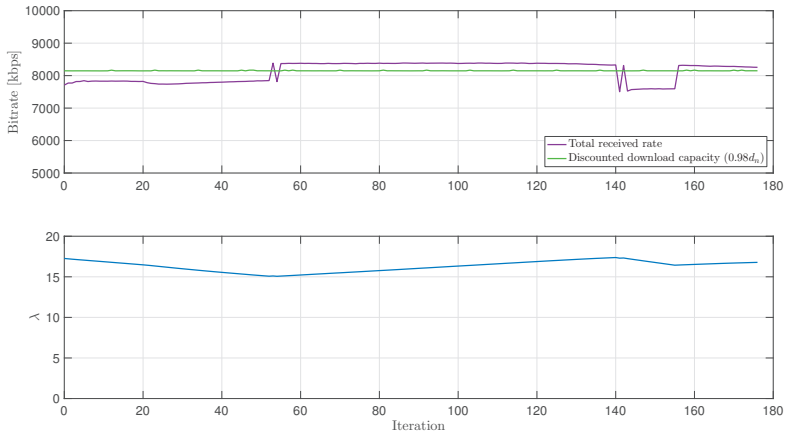


Figure 2.6 – Total received video bitrate, download capacity and dual variable associated to the download capacity of user 6.

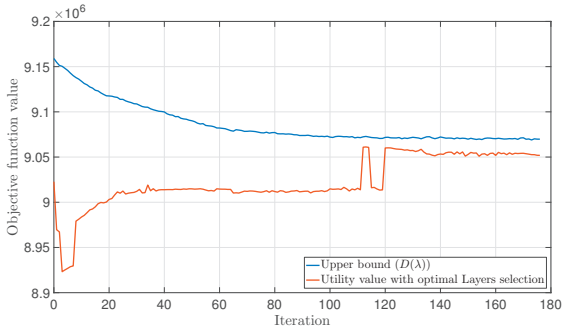


Figure 2.7 – Evolution of the Objective function of Problem (2.8).

dual variables are large enough they push back the solution of the primal-dual algorithm towards the feasible set of the original problem. At this point the layer rates are very close to the boundary of the feasible set, which translates in an efficient usage of the links and a higher total utility.

Finally a brief discussion on the number of iterations required by the iterative algorithm. In this simulation the iterative algorithm requires about 120 iterations in order to converge to the optimized allocation. If we consider that a time equal to the largest RTT between the switch node and the users' nodes is required in order to execute one iteration of the algorithm, and assuming realistic RTTs between $100 - 400$ms, then the time required to find the solution is about $12 - 50$s. This is too large to use the iterative algorithm for fast rate adaptation, and this is why we combine it with the fast rate allocation algorithm, which instead is able to quickly converge to a good solution. Nevertheless the iterative algorithm, as we will see in the next subsection, is very effective in terms of utility maximization and channel utilization.

Table 2.2 – Available layers used in the baseline solution

Number of available layers	Cumulative Layer rates
1	$1/4 \cdot c_n^{\mathrm{u}}$
2	$[1/4\ 1/2] \cdot c_n^{\mathrm{u}}$
3	$[1/4\ 1/2\ 3/4] \cdot c_n^{\mathrm{u}}$
4	$[1/8\ 1/4\ 1/2\ 3/4] \cdot c_n^{\mathrm{u}}$
5	$[1/8\ 1/4\ 3/8\ 5/8\ 7/8] \cdot c_n^{\mathrm{u}}$

2.5.4 Performance Comparisons

In order to further compare the quality of the solutions achieved by the fast and the iterative algorithms we test them for different numbers of encoded layers. In this case we use the same network and users' importance as in the previous simulation but we vary the number of encoded layers for the users. The maximum number of layers varies from 1 to 5, and for each scenario we evaluate the total utility, the average upload and download link utilization after convergence for the videoconference participants. We also compare the performance with a baseline solution where the rates of the encoded layers are fixed. The layer rates used for the baseline solution are listed in Table 2.2. The rates have been selected in order to offer a trade-off between an efficient use of the upload capacity and having a good probability of guaranteeing that all the users are able to receive all the streams. For this baseline method, the receiver algorithm is the same as in our proposed solutions: every user selects the combination of layers that maximizes his own utility function.

The results of these simulations are depicted in Fig. 2.8 where we see that the combination of the fast and iterative algorithms always outperforms the baseline method. The fast algorithm is also very efficient in terms of total utility, but it is not always able to use the transmission capacity in the most efficient way. The baseline solution obviously achieves lower values for all the metrics but the performance penalty decreases with the number of layers, since the optimization of the variables then becomes less important. When only few layers are available for a large user population, a proper optimization of the rates becomes important, and this is exactly the target scenario for our algorithm.

2.6 Related Work

We review here some of the most relevant works that address the problem studied in this chapter. Most of the existing works in the literature tackle the design of a multiparty videoconference problem based on peer-to-peer architectures. Compared to our system that relies on the existence of a central helper node, the peer-to-peer architectures have a higher number of decision variables in the rate allocation process. An example of such variables is the video stream route. Most of these works are based on the construction of a

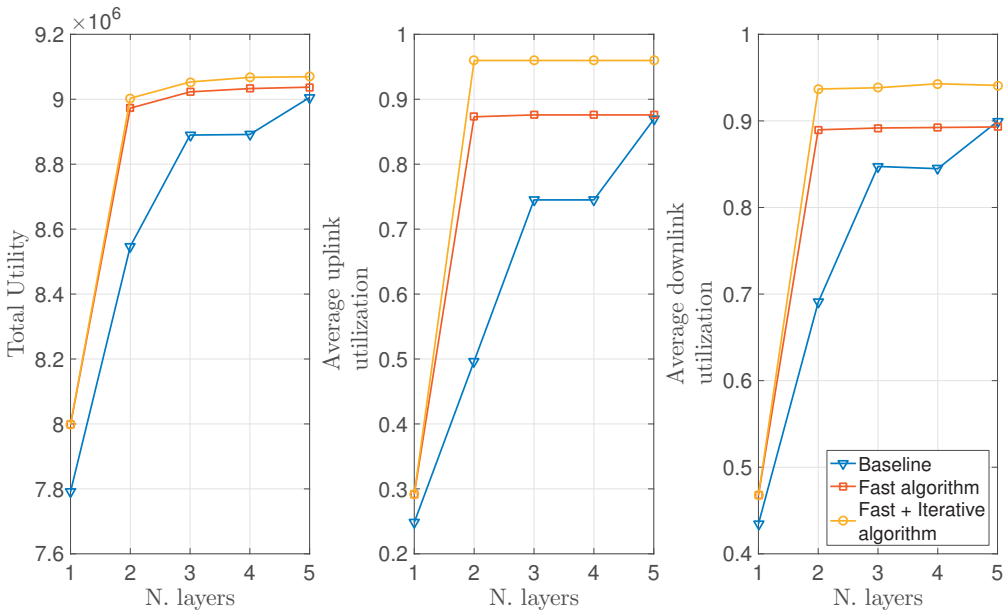


Figure 2.8 – Total utility, upload link utilization and download link utilization for the three methods under evaluation.

set of 2-hop multicast trees that employ user's nodes as central node to redistribute packets. In [8] the authors consider a peer-to-peer multiparty videoconferencing system and show that under specific assumptions, such as the peer uplinks being the only bottlenecks, the rate region achievable by using a limited number of mutualcast trees [22] is equal to the rate region achievable using inter-session network coding, making peer-to-peer solutions extremely attractive. In [10] the authors adopt a similar framework as the previous work but consider general topologies where the bottlenecks can be located anywhere in the network. In both studies, however, every user encodes a single video representation. This might not provide sufficient performance in the scenario where the users' download links have heterogenous values. The work in [9] focuses instead on a multi-rate scenario with upload link constraints only. In this work, every user is required to encode $N-1$ video layers in order to perfectly match the link constraints and use the network resources in an efficient way. Finally, a recent work [23] extends the previous framework to the case with both upload and download capacity constraints. The last two works however assume no constraint on the number of encoded video versions that a user can generate. In reality, the computational load required to every client node for the encoding process might become too heavy when the number of participants to the videoconference grows large.

In our case we have a fixed central node that is used to redistribute the packets among the different participants, thus our problem is simpler than the peer-to-peer ones from this perspective. In addition, we do not have limitations on the location of the bottleneck links. However, differently from all the previous multi-rate works, we can further impose

a specific constraint on the number of encoded streams that every user can generate, making our solution more applicable in scenarios with a large number of users.

The authors in [24] analyze how some of the commercial video conferencing solutions (specifically: Google+, Ichat and Skype) implement multiparty videoconferences. The analysis showed that both, the fully peer-to-peer, with a single encoded stream per sender, and the server based solution, with multiple encoded streams, are used by commercial solutions. Furthermore, another commercial solution [25] uses a simple central communication bridge in order to enable a 2-hop application layer multicast tree for the video delivery. The users encode the video with a finite set of rates and send them to a central node. The central node then decides which layers to forward to each receiver according to the download link capacity. Similarly to our system, this method offers a good compromise with a reliable central node to improve communication with no strong computational requirements for transcoding. The solution of the optimal rate allocation and the transmission policy are however not known nor available for all the analyzed commercial solutions. Finally, in [26] the authors develop a multiparty system with an architecture similar to the aforementioned solution and to our proposed system. Analogously to our work, this study is also motivated by the advantage and efficiency of having a simple central node with no transcoding capabilities that applies different forwarding policies to different flows. In order to limit the network usage, the central node forwards to the endpoints only a subset of the videoconference flows. The forwarding decision is made according to the users' importance level (based on the audio activity). However, as for the aforementioned commercial solutions, this study does not tackle the specific problem of the bitrate selection in the presence of heterogenous capacities among the videoconference participants.

2.7 Conclusions

In this chapter we propose a method for solving the rate allocation problem in a multiparty video conferencing system. The main contribution of the chapter is the design of a distributed method, composed of two subparts. The first subpart is able to guarantee a quick rate adaptation in the case where videoconference conditions change abruptly, and a second subpart provides precise refinement of the solution for an efficient network usage. The proposed videoconferencing system has been implemented in a network simulator and its performance have been compared with a baseline solution. The results show the benefits that can be achieved by the combined use of the two algorithms in terms of fast rate adaptation, resource utilization and utility provided to the users.

The designed system represents actually an overlay rate allocation method, as we have assumed to have access to a congestion control algorithm suited for real-time communication, which is then able to maximize the sending rate while minimizing the experienced delay. The reason why we split the congestion control algorithm design from the one of

the intra-rate allocation of the videoconference, is that the congestion control algorithm could never achieve an adaption speed as the one of the fast rate allocation method, which finds the solution in a one shot execution. Moreover, the design of a congestion control able to maximize the rate while minimizing the delay, is an extremely challenging task, as we will see in the next chapter. In terms of system architecture it is then much more convenient to consider the design of the two components separately. To further improve the proposed videoconference system we should consider the case where the switch node is not actually a single node but it is actually build in the cloud. The advantage in this case is that, when users are located far away in the network, the latency of the communication is further reduced, as flows can be merged locally for subgroups of users. On the other hand such extension also increases the degrees of freedom existing in the architecture design of overall system, raising new design challenges.

3 A Bayesian Bandit Approach to Adaptive Delay-based Congestion Control

In the previous chapter we have analyzed the problem of how to allocate transmission rates among the participants of a multiparty videoconference. In the system design we have assumed to have access to a congestion control that is able to maximize the transmitting rate of the users while limiting the experienced delay of the communication. Due to the heterogeneity of applications that have to coexist in the Internet, the design of such a congestion control algorithm is extremely challenging and it still represents an open problem in networking. In this chapter[1] we specifically analyze the design of a congestion control algorithm for delay-sensitive Internet applications that is able to operate effectively in a wide range of scenarios.

The congestion control algorithms form the subpart of the Internet applications that are responsible for adapting the sending rate according to the network condition. Congestion control algorithms can be seen as distributed methods to solve optimal network resource allocation problems. From practical point of view, the CC algorithms can be divided in two categories, namely: loss-based and delay-based algorithms. Loss-based controllers are widely deployed over the internet (e.g., TCP) and use congestion events triggered by packet losses to perform rate adaptation. This class of controllers does not take into account any type of delay measurement, such as the One-Way Delay (OWD) or the RTT. Since there is no control on the latency that the packets might experience on their route, large delays can be experienced in the case of large buffers in the inner network nodes. On the other hand, delay-based congestion control algorithms can overcome the increasing delay issue by detecting congestion events from OWD measurements. Delay-based congestion control algorithms seem therefore suitable for low delay applications since they are able to adapt the sending rate to the evolution of the delay. Unfortunately, most of the delay-based CC algorithms are not truly effective when sharing the network resources with loss-based

[1] Part of this chapter has been published in:
- S. D'Aronco, L. Toni, S. Mena, X. Zhu and, P. Frossard. Improved Utility-Based Congestion Control for Delay-Constrained Communication. *IEEE/ACM Transactions on Networking*. IEEE/ACM 2017.
- S. D'Aronco and P. Frossard. A Bayesian Bandit Approach to Adaptive Delay-based Congestion Control. *Packet Video Workshop*. ACM, 2018

algorithms. In fact, loss-based CC algorithms always need to fill the buffers of the inner network nodes before reducing their sending rate. As a result, a delay-based algorithm sharing a bottleneck link with a loss-based algorithm might experience a very large queueing delay that reduces considerably its sending rate. Therefore, there is a real need for a congestion control that could enable low delay communication whenever possible and that is robust against the presence of loss-based algorithms.

The key point is that a delay-based CC for a real-time application should behave differently depending on the different network scenarios in order to guarantee the best communication quality to the user. Let us consider for instance the scenario depicted in Fig. 3.1. When the network is shared with a loss-based algorithm (user A in the figure), the delay-based congestion control should ideally ignore the delay measurements; instead when no loss-based algorithm is active (user B in the figure), the loss-based algorithm should adapt the rate using the delay measurements. Distinguishing between the two cases is not trivial for a single CC instance operating on an individual network endpoint. The difficulties stem from the fact that the network status, e.g. presence of a loss-based algorithm or not, cannot be directly observed but rather has to be inferred by looking at the rate-delay evolution of the delay-based algorithm.

In this chapter we propose a new learning-based adaptive scheme for a low-delay CC algorithm that is able to infer the network conditions efficiently and adapt the behavior of the congestion control accordingly. More in detail, we adopt an adaptive control scheme that tunes a specific parameter, in this case the delay sensitivity, of an underlying delay-based CC algorithm. By observing how this parameter changes the equilibrium point of the delay-based CC, defined as the rate-delay equilibrium, the adaptive controller infers the hidden network conditions and selects the parameter that offers the best rate-delay trade-off. To speed up the inference of the network conditions, the adaptive controller performs active learning by taking into account the relationship between the uncertainty of the network response and parameter tuning. We define an approximate low-dimensional model of the network response, which is able to map the delay sensitivity to the achieved rate and delay at equilibrium. We then introduce a utility measure, specifically defined to model real-time communication, that maps the equilibrium rate-delay pair of the communication to the user satisfaction. Finally, we combine the probabilistic network model and the utility measure to define a Bayesian bandit problem, which we solve using an optimal learning formulation [27]. The advantage of our approach is that, by combining the probabilistic network model with a utility measure we are able to specifically predict the future expected utility as a function of the sensitivity parameter. By doing this we can then take decisions in a foresighted way and maximize the long term communication utility.

In order to evaluate the proposed method we implement the adaptive controller in the NS3 network simulator [21] and assess its performance under different network conditions. Our results demonstrate the validity of the proposed method, as it effectively adapts

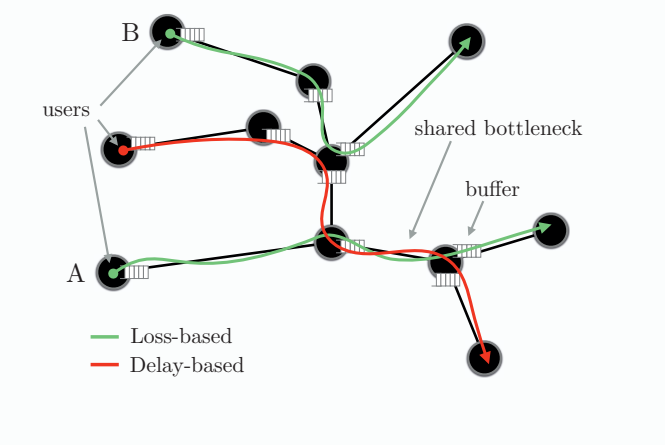


Figure 3.1 – Network system model.

to a large variety of scenarios. We also compare the proposed controller to other CC algorithms specifically designed for real-time applications. The results show that the adaptivity of the proposed scheme is able to outperform existing methods and achieve better overall performance in the considered scenarios.

The remainder of the chapter is structured as follows. In Section 3.1 we provide some network congestion control preliminaries and we formalize the problem. In Section 3.2 we describe the design of the proposed learning-based adaptive controller. In Section 3.3 we describe the controller implementation. In Section 3.4 we present the experimental results. We provide a review of the related works in Section 3.5. Finally, conclusions are presented in Section 3.6.

3.1 Congestion Control Preliminaries

3.1.1 Congestion Control Settings

The congestion control problem basically relates to the problem of Eq. (1.1), where a set of users wants to share some network resources while maximizing a certain utility metric. More specifically, we consider N users sharing a set of M network links. The optimization problem then reads:

$$\begin{aligned} \underset{\mathbf{x}}{\text{maximize}} \quad & \mathcal{U}(\mathbf{x}) = \sum_{n=1}^{N} u_n(x_n) \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{c} \\ & \mathbf{x} > 0. \end{aligned} \tag{3.1}$$

The notation corresponds with the one introduced earlier: the sending rate of the user n is denoted by x_n, and $\mathbf{x}$ denotes the vector containing all the users sending rates; u_n is an increasing and concave utility function associated to user n (in this case, for simplicity, we assume that the domain of the utility functions is the positive orthant); the routing information is embedded in the $M \times N$ binary matrix $\mathbf{A}$; and c_m denotes the transmitting capacity of link m. The rate passing through link m can be written as $\mathbf{a}_m\mathbf{x} = y_m$. As shown in Fig. 3.1, each link is preceded by a buffer that accumulates the data in surplus when y_m exceeds the link capacity c_m. The queueing delay at the buffer of link m, $q_m(t)$, evolves over time as follows:

$$\dot{q}_m = \frac{1}{c_m} \left(y_m - c_m\right)^+_{q_m} . \tag{3.2}$$

Here we denote by $\dot{z}$ the derivative of the variable z with respect to time, and $(z)^+_x$ means that the z is projected onto the positive orthant if $x \leq 0$, and it basically guarantees to have a positive queuing delay. A data unit, or packet, that arrives at link m at time t takes a time equal to $q_m(t)$ before being actually sent over the link m. The total queuing delay time experienced by user n is equal to the sum of all queueing delays of the route: $\mathbf{a}_n^\mathsf{T}\mathbf{q}$. The total time required for a packet to be transmitted is then composed by the sum of the queueing delay plus the propagation delay of the route:

$$d_n = \mathbf{a}_n\mathbf{q} + d_{0n}, \tag{3.3}$$

where d_{0n} represents the propagation delay of the route of user n and it is assumed to be constant over time. Note that the propagation delay actually depends on the size of the data packet transmitted, in this work however we consider to operate with fixed size packets. Each network buffer has a limited storage capacity. In the event that this amount is reached, further incoming packets are discarded. The amount of packets that are lost obviously depends on the incoming rate. More specifically, the loss ratio for link m when the buffer is full corresponds to:

$$l_m(y_m) = \left(\frac{y_m - c_m}{y_m}\right)^+_{y_m - c_m} . \tag{3.4}$$

We can then write the loss ratio of user n by taking into account the loss probabilities of the links composing the user's route:

$$p_n = 1 - \prod_{m=1}^{M} a_{mn} \left(1 - l_m\right), \tag{3.5}$$

which, if the loss ratios of the links are small, can be approximated by

$$p_n \simeq \mathbf{a}_n^\mathsf{T}\mathbf{l}. \tag{3.6}$$

In the next subsection, we describe how it is possible to solve the classical NUM problem of Eq. (3.1) in a distributed way using the user experienced delay or the user loss ratio. Note that both quantities are physical quantities that can be measured by the users at the endpoints, so that no explicit communication is required.

3.1.2 Congestion Control Algorithms

We discuss now the most common approaches to solve the NUM problems in the congestion control framework. The first method is an approximate penalty method, which resembles loss-based congestion control algorithms, while the second method is based on a dual decomposition, and resembles delay-based congestion control algorithms.

One way to simplify Eq. (3.1) consists in having a penalty function that maps the violation level of the capacity constraints into a negative utility:

$$\underset{\mathbf{x}>0}{\text{maximize}}\ \mathcal{U}(\mathbf{x}) - \sum_{m=1}^{M} \int_0^{y_m} g_m(z)dz \tag{3.7}$$

where $g_m(z)$ represents the penalty function for link m, when the link rate is equal to z. The function g_m must be in this case a positive and increasing function of the link rate y_m. We can find the optimal value of the problem in Eq. (3.7) using a gradient-based method; we then obtain the following rate update equation:

$$x_n^{t+1} = \left(x_n^t + \alpha_n \left(u_n'(x_n^t) - \sum_{m=1}^{M} a_{mn} g_m(y_m^t) \right) \right)^+, \tag{3.8}$$

where $u_n'(x_n)$ is the derivative of the utility function with respect to x_n, and α_n is a positive parameter that controls the update rate. If we use the loss ratio of link m as penalty, i.e., $g_m(y_m) = l_m(y_m)$, and assume small loss ratios, i.e., $\mathbf{a}_n^\mathsf{T}\mathbf{l} \simeq p_n$, then we observe that users need to know only their own utility function and their loss ratio p_n in order to compute the rate update. The users can easily estimate their loss ratios by observing the number of dropped packets at the receiving node. The main drawback of this solution is that, at the equilibrium, losses are always experienced, which means that bottleneck buffers are always full and large queuing delays might be experienced.

Another possible way to solve the NUM problem is by dual decomposition [5]. The optimization problem can then be solved in a distributed way by using a primal-dual algorithm similar to the one of Eq. (1.4) based on the following update equations:

$$\begin{cases} x_n^{t+1} = \left(x_n^t + \alpha_n \left(u_n'(x_n^t) - \sum_{m=1}^{M} a_{mn} \lambda_m^t \right) \right)^+ & n = 1...N \quad (3.9a) \\ \lambda_m^{t+1} = \left(\lambda_m^{t+1} + \epsilon_m \left(y_m^t - c_m \right) \right)^+ & m = 1...M, \quad (3.9b) \end{cases}$$

where λ_m is the dual variable associated to the capacity constraint of link m, and ϵ_m is a positive parameter that controls the update rate of the dual variable λ_m. Note that compared to Eq. (1.4), we take here only a single step in the direction of the gradient for the primal variables. This is common for congestion control problems as it resembles a sort of low-pass filtering to counteract potential signal noise, and helps to have more stable sending rates. We can clearly see that, if we set $\nu_m = 1/c_m$, then Eq. (3.9b), represents the discrete version of the queueing delay dynamics defined in Eq. (3.2). As a result, the users can compute Eq. (3.9a) by monitoring the delay experienced by the transmitted packets. In addition, from Eq. (3.9), we notice that the queuing delay at equilibrium cannot be null in general, as otherwise the rate would grow indefinitely. For a given utility function, the delay experienced by a user at the equilibrium depends on the bottleneck capacity: generally, the larger the bottleneck capacities c_m, the larger the user rates x_n, and the lower the equilibrium queueing delay. We refer to the amount of queueing delay that a delay-based CC needs to experience (create) to maintain a stable sending rate as the self-inflicted queuing delay.

Existing CC algorithms are strongly based on the distributed schemes we presented in this section. From Eq. (3.9) we can easily understand why delay-based CC algorithms tend to starve when competing against loss-based algorithms. Loss-based algorithms make the queuing delays, which represent the dual variables $\boldsymbol{\lambda}$ in Eq. (3.9), grow large, leading to a small equilibrium rate of Eq. (3.9a) in the presence of large network buffers. The challenge in the design of delay-based CC algorithms consists in modifying the rate update equation at the users' endpoints in order to have a limited experienced delay when possible and, at the same time, to not starve when competing with loss-based algorithms. In the next subsection, we provide more insights and details about why it is essential for a delay-based CC algorithms to adapt to different network conditions in order to guarantee good performance.

3.1.3 Network Response and Equilibrium points

As discussed in the previous subsection delay-based CC algorithms essentially adjust the sending rate according to the delay measurements: higher delays translate to higher congestion penalties and thus to a lower sending rate. A delay-based CC is at equilibrium when the penalty resulting from the experienced delay has the necessary strength to keep the sending rate constant. The curve passing by the equilibrium points on a rate-delay plane, corresponds to the self-inflicted delay curve of the CC and it basically defines the sensitivity of the CC to the experienced delay.

Interestingly, given a network in some fixed conditions and a flow transmitting data between two nodes, we can think about characterizing the flow route with the rate-delay pairs that represent equilibrium points for the network system. In order to better understand this idea let us consider a data flow over a single network link of capacity

c and maximum buffer size of q_{MAX} and a null propagation delay. We analyze three different network scenarios, namely: *i*) the link is not shared with any other flow, *ii*) the link is shared with a loss-based algorithm, and *iii*) the link is shared with a delay-based algorithm. The three cases are also depicted in Fig. 3.2.

1. **Single user** In this case the network link is not shared with any other users. The rate-delay equilibrium points that can be achieved can be clustered in three regions: *i*) the sending rate is smaller than the capacity c and the equilibrium queuing delay is zero, *ii*) the sending rate is equal to c and the queuing delay can take any value between 0 and the maximum buffer size q_{MAX}, *iii*) the sending rate is larger than c and the only valid equilibrium delay is equal to q_{MAX}. We can thus see that the equilibrium point is not unique and the value at which the entire system converges depends on the design of the congestion control.

2. **Link shared with a loss-based algorithm** As loss-based algorithms do not adapt to delay variations and always fill the network buffers, the only equilibrium value for the delay, in this case, is the one that corresponds to the maximum buffer size. Regarding the sending rate, ideally any value represents a valid equilibrium point, as the competing flow will always take the amount of free transmission capacity in order to maximize its own utility and fill the buffer.

3. **Link shared with a delay-based algorithm** In this final case, we consider that the network link is shared with a delay-based CC algorithm that implements an update rate equation similar to Eq. (3.9a). In this case, the value of the delay at equilibrium that we observe, by varying the sending rate x, resembles the self-inflected delay curve of the competing flow. Larger sending rates reduce the available capacity for the competing flow making the experienced delay increase. Obviously the equilibrium delay cannot exceed the size of the buffer.

From the perspective of a network endpoint, there is no unique rate-delay equilibrium point, but rather a curve defined on the rate-delay plane. Moreover as discussed above this curve can be significantly different depending on the network condition. We can thus think to represent the network, as perceived by the flow endpoint, by a curve representing valid rate-delay equilibrium pairs. Hereinafter we define this curve as the *network response*. The equilibrium point, in terms of user's rate and delay, at which the network converges depends then on both: the CC algorithm and the network response.

We might then ask if all the network equilibrium points offer the same communication quality or if some of them are better than others. To answer this question, we define first a utility function that actually models the requirements of delay-sensitive communication. It is rather intuitive that for a delay-sensitive communication, the two quantities that affect the most its quality are the transmitting rate and the communication latency. We thus introduce an application utility function $v(x, d)$ that maps the equilibrium sending

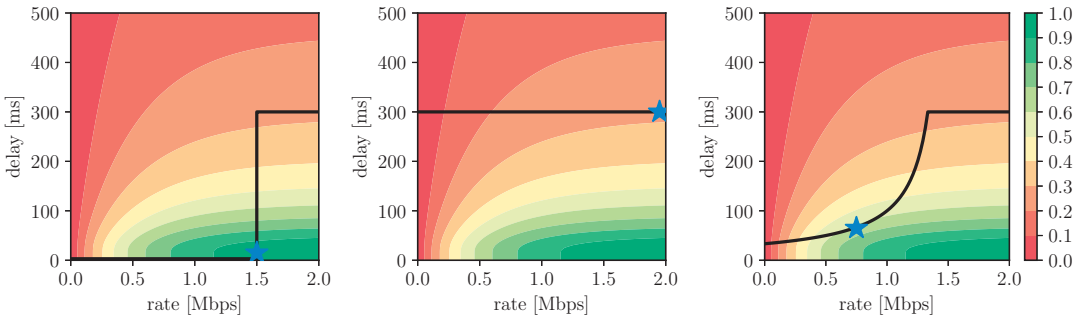


Figure 3.2 – Representation of possible network responses for a single link with capacity $c = 1.5$ Mbps and maximum buffer size $q_{\text{MAX}} = 300$ ms. The different scenarios are (from left to right): *i*) single user case, *ii*) shared link with a loss-based algorithm, and *iii*) shared link with a delay-based algorithm. The black line represents the rate-delay equilibrium points of the network for the different scenarios. The colors represent the value of the utility function v as a function of rate and delay of the flow under consideration. Finally, the star denotes the network equilibrium point that has the largest utility value.

rate x and the equilibrium experienced delay d to a utility value. In this case we do not make any specific assumption on the shape of the function v. In Fig. 3.2 we consider a utility function v which favors high rates and low communication delays, as it is usually the case. It can be seen that, in the three cases, all the points on the network response curve do not lead in general to the same utility: equilibrium points in the bottom-right part of the positive orthant offer a much better communication quality.

The obvious consideration that arises is the following: as the network in general offers a set of different rate-delay equilibrium points, we ideally want a CC algorithm that adapts its behavior in order to change the equilibrium reached by the overall system, and maximizes the delay-sensitive application utility. The main challenge in this problem stems from the fact that the network response is not known a priori and can only be estimated by sensing properly the network. This means that the CC should, at the same time, vary its behavior to sense the network, and leverage the collected information to reach the best equilibrium point. In the next section we specifically investigate this problem and propose an effective learning-based solution.

3.2 Learning-based Adaptive Controller

In this section we describe the proposed solution to the problem introduced in Subsection 3.1.3. We start by discussing how the CC should sense the network in order to estimate the network response. We then build a low-dimensional probabilistic model for the network response, which allows us to perform inference efficiently. Finally, we use this probabilistic network model to define a bandit problem for the maximization of the

communication utility v introduced previously.

3.2.1 Network Sensing Method

As mentioned in the previous section the network response is not known by the user endpoint, and it has to be estimated by sensing properly the network. A naïve method to perform such a task is simply to set a constant sending rate, then wait for the network to adapt and finally observe the experienced delay. While this method looks straightforward, it also has some serious drawbacks. In fact, the controller completely ignores the received congestion signals, as the rate is kept firmly constant till the network system adapts. From the point of view of the network system, this naïve approach is highly deprecated as it does not lead to an elastic behavior and might destabilize the network and severely degrade the quality of service provided to all the network users. An alternative way is to set a target delay and then have a control law that adapts the rate in order to reach the target value. The drawback with this approach is that it may lead to episodes of user throughput starvation as some delay values might not represent equilibrium points for any rate. For instance, let us consider the case where the target delay is set to a value smaller than the network buffer size, and the flow in question shares the network link with a loss-based algorithm, which constantly fills in the network buffers. In this case, the controller strives to reduce the experienced delay vainly, eventually reaching a zero throughput. Another possible solution to this problem consists in using an underlying delay-based CC whose delay sensitivity can be adjusted by tuning a certain parameter. A delay-based CC algorithm is characterized by the self-inflicted delay curve; this curve, together with the network response, defines the rate and delay at equilibrium. By varying the delay sensitivity, we basically move the self-inflicted delay curve of the CC, and consequently also the equilibrium point; this ultimately allows us to sense the network. The advantage of this method, compared to the previous ones, is that we have an elastic controller that adapts to network congestion events avoiding extreme behaviors that could severely harm the communication quality of the network users. This is the method we decide to pursue for our adaptive controller design. In order to proceed with the design we then need to choose a specific underlying delay-based CC.

Although the proposed method can be easily modified to several delay-based CC algorithms, we consider here a CC derived from the one proposed in [28]. The algorithm is characterized by the following rate update equation:

$$x^{t+1} = \left(x^t + \alpha x^t \left(\frac{w}{x^t} - \beta \frac{(d^t + \delta \dot{d}^t - d_{\mathrm{bl}})^+}{\mathrm{RTT}} \right) \right)^+ , \tag{3.10}$$

where x^t denotes the sending rate at time t, d^t is the experienced OWD, $\dot{d}^t$ is the time derivative of the OWD, RTT represents the round-trip time of the communication, w resembles the uplifting force of the rate update (in this case, the algorithm is designed

to use a logarithmic utility function $u(x) = w\log(x)$, thus $u'(x) = w/x$) and, d_{bl} is the baseline delay which discounts part of the experienced delay. The additional terms in Eq. (3.10) compared to Eq. (3.9a), such as the RTT scaling and the OWD derivative term, are required in order to achieve stability. A thorough discussion about the stability analysis of CC algorithms goes beyond the scope of this thesis, further information can be found in [28]. However it is important to know that, due to the delays that exist in the network, the rate-update equation has to meet specific criteria. This means that not all the parameters can be changed freely and independently. In particular, α, β, δ and w should be set in a way that guarantees the stability of the controller [28]. In order to tune the controller sensitivity we need to identify one parameter that affects the rate-delay equilibrium pair but does not compromise the stability of the controller. According to the stability analysis conducted in [28], the parameter d_{bl} represents a valid option to this end. Moreover, additively discounting the experienced delay is an operation that can ideally be applied to any delay-based CC algorithm. As a result, the adaptive learning-based controller developed in this chapter can be easily generalized to work with other underlying delay-based CC algorithms. Finally, note that the parameter d_{bl} represents a sort of negative delay sensitivity as the larger its value, the smaller the delay penalty. In the remainder of the section we focus on finding a rule to adjust the value of this parameter to the different network conditions in order to maximize the real-time communication utility v.

3.2.2 Low-dimensional Network Response Model

In the previous subsection, we have identified the parameter d_{bl} of the underlying delay-based CC, as the free variable that we can adjust in order to modify the rate-delay equilibrium point reached by the network. According to this choice, it is natural to model the network response as two functions, $x_{\text{eq}}(d_{\text{bl}})$ and $d_{\text{eq}}(d_{\text{bl}})$, which map the value of d_{bl} to the rate and delay at equilibrium, respectively. The goal is to infer these mappings from pointwise observations to eventually optimize the application utility v. In order to proceed, we shall define a parametric model for $x_{\text{eq}}(d_{\text{bl}})$ and $d_{\text{eq}}(d_{\text{bl}})$, so that we can leverage the collected observations to infer the model parameters and be able to predict the network response for any value of d_{bl}. Although there exist many probabilistic models that can be used to represent these functions [29], most of them make use of a large number of parameters to fit the observed data, and require a large amount of computations to perform inference and prediction. In order to have a realistic adaptive controller design we need to keep the amount of required computations as low as possible. Thus, we consider an approximate simplified model for the network response, which employs piecewise linear functions. This simplified model aims at describing the most significant cases that we can encounter in real environments; it also leverages the specific meaning of d_{bl} in the rate update rule of Eq. (3.10), and its impact on the equilibrium point.

In particular we consider to approximate the equilibrium rate network response with the following parameterized model:

$$x_{\text{eq}}(d_{\text{bl}}; \boldsymbol{\theta}_x) = \min(\theta_x^m d_{\text{bl}} + \theta_x^q, \theta_x^U), \tag{3.11}$$

where $\theta_x^m > 0$, θ_x^q and $\theta_x^U > 0$ are the model parameters ($\boldsymbol{\theta}_x$ denotes the vector made of the single parameters). This simple model is crafted to represent effectively the two regimes that we often encounter in congestion control. In the first region (small d_{bl}) the rate at equilibrium increases when increasing the delay discount d_{bl}. This part aims at modeling scenarios where the CC competes with other data flows for the same network link; or cases where the value of d_{bl} is not large enough to guarantee the full utilization of the available capacity. In the second region, we face a plateau phase where increasing the delay discount does not offer any advantage in terms of transmission rate. This part models the situation where the maximum transmitting capacity of the link is reached: even by further discounting the experienced delay, the sending rate cannot increase any more.

Similarly, we model the equilibrium delay network response with the following function:

$$d_{\text{eq}}(d_{\text{bl}}; \boldsymbol{\theta}_d) = \max(\theta_d^L, \min(\theta_d^m d_{\text{bl}} + \theta_d^q, \theta_d^U)), \tag{3.12}$$

where $\theta_d^m \geq 0$, θ_d^q, 0 and $\theta_d^U \geq \theta_d^L \geq 0$ are the model parameters ($\boldsymbol{\theta}_d$ denotes the vector made of the single parameters). Again, the shape is crafted to approximate the main types of network responses that we can have. In the first region, the delay at equilibrium is insensitive to d_{bl} and is equal to θ_d^L. This first region captures the fact that each communication route exhibits a propagation delay, which corresponds to the lowest possible experienced delay. The second part of the network response is characterized by an increasing equilibrium delay as a function of the delay discount. Finally, in the last region the experienced delay reaches a plateau phase. This part is needed to model the cases where the experienced delay matches the size of the network packet buffers. In this case, the network nodes discard other incoming packets and the delay cannot grow any larger.

The model developed in this section allows us to perform inference on the network response, which is represented by the parameters $\boldsymbol{\theta}_x$ and $\boldsymbol{\theta}_d$, from observations of the rate and delay at equilibrium for a fixed baseline delay d_{bl}. We denote by x_{obs}, and d_{obs} two random variables representing the observed data, and by $p(x_{\text{obs}}|\boldsymbol{\theta}_x, d_{\text{bl}})$ and $p(d_{\text{obs}}|\boldsymbol{\theta}_d, d_{\text{bl}})$ the likelihood probabilities for observing such values given the network parameters and the baseline delay. In order to perform Bayesian inference we simply need to combine these likelihood probabilities with a prior belief using the Bayes rule:

$$p(\boldsymbol{\theta}_x|x_{\text{obs}}, d_{\text{bl}}) = \frac{p(x_{\text{obs}}|\boldsymbol{\theta}_x, d_{\text{bl}})p(\boldsymbol{\theta}_x)}{p(x_{\text{obs}}|d_{\text{bl}})}, \tag{3.13}$$

where $p(\boldsymbol{\theta}_x|x_{\text{obs}}, d_{\text{bl}})$ is the posterior probability, $p(\boldsymbol{\theta}_x)$ is the parameters' prior, and $p(x_{\text{obs}}, d_{\text{bl}})$ is the marginal likelihood (note that the marginal likelihood does not depend on $\boldsymbol{\theta}_x$ and this term is only needed to have a valid posterior distribution that integrates to one). By using Eq. (3.13) we can process equilibrium points observations to modify, and improve, the knowledge on the parameters that define our network response model. Note that the same reasoning is valid for the delay response parameters $\boldsymbol{\theta}_d$, which have a posterior distribution defined as:

$$p(\boldsymbol{\theta}_d|d_{\text{obs}}, d_{\text{bl}}) = \frac{p(d_{\text{obs}}|\boldsymbol{\theta}_d, d_{\text{bl}})p(\boldsymbol{\theta}_d)}{p(d_{\text{obs}}|d_{\text{bl}})}, \tag{3.14}$$

In the next subsection we show how we can leverage the network model and the probabilistic framework developed in this subsection to formulate a bandit problem.

3.2.3 Bayesian Bandit and Optimal Learning

In Subsection 3.1.3 we have introduced the application utility $v(x, d)$, which depends on the sending rate and experienced delay. The purpose of this function is to map the experienced rate and delay at the equilibrium to the associated communication quality. The ultimate goal of the adaptive controller is obviously to maximize this metric. In our case the variable that we are free to adjust in order to maximize the communication quality is the delay baseline d_{bl}. Ideally we want to select the value of d_{bl} that maximizes the following problem:

$$\underset{d_{\text{bl}}}{\text{maximize}} \quad v(x_{\text{eq}}(d_{\text{bl}}; \boldsymbol{\theta}_x), d_{\text{eq}}(d_{\text{bl}}; \boldsymbol{\theta}_d)). \tag{3.15}$$

Unfortunately, in order to solve this problem we need to know the parameters of the network response $\boldsymbol{\theta}_x$ and $\boldsymbol{\theta}_d$. The value of these parameters is not known actually, but we may have access to a belief on these parameters. We can then think to maximize the expected utility, which we define as:

$$\mathcal{V}(d_{\text{bl}}, p(\boldsymbol{\theta})) = \mathbb{E}_{p(\boldsymbol{\theta})}\left[v(x_{\text{eq}}(d_{\text{bl}}; \boldsymbol{\theta}_x), d_{\text{eq}}(d_{\text{bl}}; \boldsymbol{\theta}_d))\right], \tag{3.16}$$

where $\boldsymbol{\theta} = [\boldsymbol{\theta}_x \boldsymbol{\theta}_d]$, $p(\boldsymbol{\theta})$ represents the belief on the network response parameters, and $\mathbb{E}$ is the expectation operation. $\mathcal{V}(d_{\text{bl}}, p(\boldsymbol{\theta}))$ represents the expected utility when using a delay baseline equal to d_{bl} with a belief on the network parameters equal to $p(\boldsymbol{\theta})$.

Eq. (3.16) defines what is the expected value of the utility for a single baseline delay selection. However, in our problem, the adaptive controller selects the baseline delay iteratively, therefore we should extend the performance measure to a sequence of actions.

If we consider an infinite series of decisions we can define the long term utility as:

$$\mathcal{V}^{\infty}(\{(d_{\text{bl}}^t, p^t(\boldsymbol{\theta}))\}) = \sum_{t=0}^{\infty} \gamma^t \mathcal{V}(d_{\text{bl}}^t, p^t(\boldsymbol{\theta})), \tag{3.17}$$

where $\gamma \in [0, 1)$ is the discount parameter for the future rewards. It basically describes how much we care about the future utility compared to the immediate ones. It is important to note that the beliefs $p^t(\boldsymbol{\theta})$) in Eq. (3.17) are related, as they they are consecutive posterior updates on the parameters $\boldsymbol{\theta}$ from the sequence of collected observations. The evolution of the belief is imposed by the Bayes rule of Eq. (3.13) (in this case the prior belief corresponds to $p^t(\boldsymbol{\theta})$ and the resulting posterior represents the future belief $p^{t+1}(\boldsymbol{\theta})$)). From this perspective, the selection of the delay sensitivity d_{bl} can be seen as a bandit problem, where the system has to iteratively select a value for d_{bl}, and at each iteration it gathers information on the system and a reward. As better knowledge about the network translates into a larger expected reward, the system should try to balance the exploitation of the current knowledge and the refinement of the network response belief to improve future performance. Ideally, we would like to find the value for d_{bl} associated to each possible belief $p(\boldsymbol{\theta})$), which maximizes the long term expected utility. This corresponds to solving the following problem:

$$\underset{\pi}{\text{maximize}} \;\; \mathbb{E}_{p\left(\{x_{\text{obs}}^t, d_{\text{obs}}^t\}|\{\pi(p^t(\boldsymbol{\theta}))\}\right)} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{V}\left(\pi(p^t(\boldsymbol{\theta})), p^t(\boldsymbol{\theta})\right) \right], \tag{3.18}$$

where π represents a policy that maps a belief on $\boldsymbol{\theta}$ to a delay baseline value, i.e., $\pi : p(\boldsymbol{\theta}) \rightarrow d_{\text{eq}}$; $p\left(\{x_{\text{obs}}^t, d_{\text{obs}}^t\}|\{\pi(p^t(\boldsymbol{\theta}))\}\right)$ corresponds to the probability of observing a sequence of equilibrium points $\{x_{\text{obs}}^t, d_{\text{obs}}^t\}$ given that a delay baseline equal to $\{\pi(p^t(\boldsymbol{\theta}))\}$ is applied to the underlying delay-based CC, marginalized over the prior belief on $\boldsymbol{\theta}_x$,

$$p\left(\{x_{\text{obs}}^t, d_{\text{obs}}^t\}|\{\pi(p^t(\boldsymbol{\theta}))\}\right) = \prod_t \int p((x_{\text{obs}}^t, d_{\text{obs}}^t)|\pi(p^t(\boldsymbol{\theta})), \boldsymbol{\theta}) p^t(\boldsymbol{\theta}) d\boldsymbol{\theta}. \tag{3.19}$$

Finally, note that the beliefs on $\boldsymbol{\theta}_x$ and on $\boldsymbol{\theta}_d$ evolve according to

$$p^{t+1}(\boldsymbol{\theta}_x) = p^t(\boldsymbol{\theta}_x|x_{\text{obs}}^t, \pi(p^t(\boldsymbol{\theta}))) \tag{3.20}$$

and

$$p^{t+1}(\boldsymbol{\theta}_d) = p^t(\boldsymbol{\theta}_d|d_{\text{obs}}^t, \pi(p^t(\boldsymbol{\theta}))), \tag{3.21}$$

respectively. The distributions $p^t(\boldsymbol{\theta}_x|x_{\text{obs}}^t, \pi(p^t(\boldsymbol{\theta})))$ and $p^t(\boldsymbol{\theta}_d|d_{\text{obs}}^t, \pi(p^t(\boldsymbol{\theta})))$ can be computed from Eq. (3.13) and Eq. (3.14).

In order to find the optimal policy for the closed loop problem of Eq. (3.18), we typically need to compute the state value function of our system using a dynamic programming

approach. However, due to the infinite dimension of the state space and the required expectation operations, this method quickly runs into computational problems. One way to work around the complexity problem consists in approximating the closed loop problem of Eq. (3.18) with a sequence of receding horizon open loop problems. As also suggested in [27], a rough but effective approximation corresponds to computing, at each time step t, the delay sensitivity d_{bl}^t that maximizes the long term expected utility, considering to update the belief only for a single future observation. Then, after taking a decision at t and observing the equilibrium $(x_{\text{obs}}^t, d_{\text{obs}}^t)$, the belief is updated and the same problem is solved again with the new belief. In this case the problem can be rewritten as:

$$\underset{d_{\text{bl}}^t}{\text{maximize}} \quad \mathcal{V}(d_{\text{bl}}^t, p^t(\boldsymbol{\theta})) + \frac{\gamma}{1-\gamma}\mathbb{E}_{p^t\left(x_{\text{obs}}, d_{\text{obs}} | d_{\text{bl}}^t\right)}\left[\mathcal{V}^\star\left(p^t(\boldsymbol{\theta}|x_{\text{obs}}^t, d_{\text{obs}}^t, d_{\text{bl}}^t)\right)\right], \quad (3.22)$$

where $\mathcal{V}^\star\left(p(\boldsymbol{\theta})\right)$ represents the maximum expected utility with a parameters' belief equal to $p(\boldsymbol{\theta})$. The receding horizon approach of Eq. (3.22) has largely simplified the problem, compared to Eq. (3.18), mainly for two reasons: *i*) being open loop we optimize over a scalar variable, d_{bl}, rather than the infinite dimension policy π, and *ii*) by limiting the learning horizon to one step we only need to compute the expectation over a single observation $(x_{\text{obs}}^t, d_{\text{obs}}^t)$.

This concludes our analysis on the operating principle of the proposed learning-based adaptive controller for delay-sensitive applications. In the next section, we describe more in detail the controller implementation, in particular the update of the belief on the parameters $\boldsymbol{\theta}_x$ and $\boldsymbol{\theta}_d$, and the solution of the optimization problem in Eq. (3.22).

3.3 Controller Implementation

The internal architecture of the adaptive controller is shown in Fig. 3.3. The operations of the adaptive controller can be summarized as follows: the adaptive controller sets a certain value for d_{bl} and monitors the evolution of the sending rate and experienced delay. When an equilibrium point is reached, the equilibrium detector forwards the information about the new observation to the network response inference block. At this point using Bayesian inference methods, the belief on the network response parameters is updated. Finally, the controller uses the new belief to solve the one-step receding horizon problem of Eq. (3.22) and to compute a new value for d_{bl} to be used in the next iteration. We now describe in more detail the implementation of the different subparts of the controller.

3.3.1 Equilibrium Detector

This subpart is responsible for monitoring the evolution of the sending rate and the experienced delay, and detecting when the underlying CC algorithm reaches an equilibrium point. In our implementation we seek for two types of equilibrium: *i*) a short term

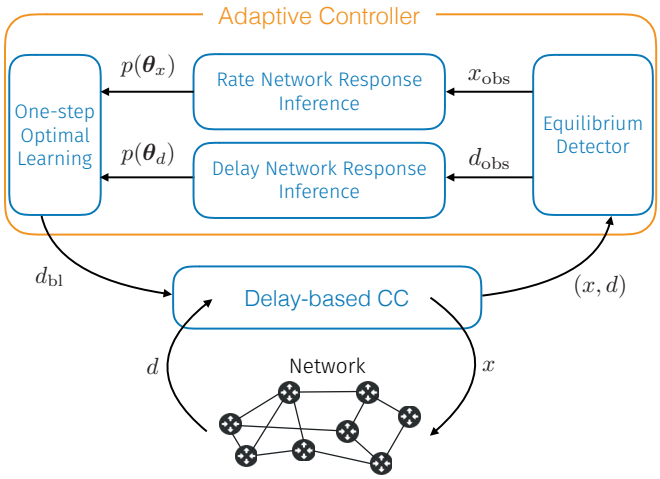


Figure 3.3 – Architecture of the proposed delay-based adaptive controller. The adaptive controller runs on the top of a delay-based CC. It infers the network response and adapts the delay sensitivity of the CC in order to achieve the best rate-delay trade-off.

equilibrium, where the variations of the rate and delay signals are below a certain threshold; and *ii*) a long term equilibrium, where the rate and delay might be varying in the short term but their average value over a time window of dozens of seconds is constant. The long term equilibrium is necessary in the scenario where the network link is shared with a TCP flow. In this case, due to the typical TCP sawtooth behavior, a local equilibrium is never achieved. By averaging the delay and the rate over large time windows, we can however observe that the system has reached a sort of limit cycle. In order to make use of our probabilistic model for the network response inference, we need to associate to different observations a likelihood probability $p(x_{\text{obs}}|\boldsymbol{\theta_x}, d_{\text{bl}})$. In both cases, namely short term and long term equilibrium, we assume that x_{obs} is distributed according to a normal distribution centered around the value dictated by Eq. (3.11):

$$x_{\text{obs}} \sim \mathcal{N}(\mu = x_{\text{eq}}(d_{\text{bl}}; \boldsymbol{\theta_x}), \sigma = \sigma_x). \tag{3.23}$$

In the case of the short term equilibrium we consider x_{obs} equal to the current sending rate, whereas in the case of the long term equilibrium we set x_{obs} equal to the average of the rate observed since the last change in d_{bl}. The standard deviation σ_x represents the standard deviation of the noise affecting the measurement. Note that the same likelihood model applies for the delay observations.

In order to speed up the network response inference process, we add the possibility to forward non-equilibrium observations to the inference method. For instance, if we detect that for a fixed value of d_{bl} the delay is increasing excessively, we could add an observation that imposes to have an equilibrium delay higher than the current one, for the used delay

discount d_{bl}. In order to model this type of observation we can use the probit model as the likelihood probability (other model such as the logistic function are also viable choices). In the case of the delay we have:

$$p(\text{higher delay is true}|d_{\text{obs}}, d_{\text{bl}}, \boldsymbol{\theta_d}) = \Phi\left(\frac{d_{\text{eq}}(d_{\text{bl}}; \boldsymbol{\theta_d}) - d_{\text{obs}}}{\sigma_d}\right), \tag{3.24}$$

where Φ denotes the cumulative density function of a standard normal random variable. The above equation returns the probability for the event of having an equilibrium delay larger than d_{obs} given d_{bl} and $\boldsymbol{\theta_d}$. What we observe in this case is the increasing delay, and the current delay value d_{obs}, see Fig. 3.4. The inference method can use the probit function to perform inference on the network parameters. What happens intuitively is that the inference method basically rules out all the values of $\boldsymbol{\theta_d}$ that lead to a delay at equilibrium $d_{\text{eq}}(d_{\text{bl}}; \boldsymbol{\theta_d})$ lower than d_{obs}. Note that also in this case, σ_d represents the noise affecting the measurement of d_{obs}. In our implementation we forward to the network response inference block non-equilibrium observations only when the delay grows excessively with respect to the value predicted by the current network response belief.

For the rate we never forward to the inference method information about non-equilibrium points. We avoid this operation as we have seen from simulation results that, when competing against TCP flows, the TCP sawtooth behavior causes misleading and faulty non-equilibrium observations that worsen the performance of the overall system.

3.3.2 Rate and Delay Network Response Inference

The rate and delay network response inference is carried out by two separate blocks that are responsible for processing all the observations gathered by the equilibrium detector. They derive an updated posterior belief for the parameters $\boldsymbol{\theta}_x$ and $\boldsymbol{\theta}_d$. One aspect that we have not included in the model of Subsection 3.2.2, which is however part of the real implementation, is the fact that the network parameters $\boldsymbol{\theta}_x$ and $\boldsymbol{\theta}_d$ might be changing over time. In fact as users might join or leave the network, the network response might change as well. Hence, we allow the parameters $\boldsymbol{\theta}_x$ and $\boldsymbol{\theta}_d$ to evolve according to a Markov model. The evolution of the parameters is thus defined by a transition probability $p(\boldsymbol{\theta}^{t+1}|\boldsymbol{\theta}^t)$. In our model we simply consider as transition probability a set of univariate normal distributions, one for each single parameter θ of the network model:

$$\boldsymbol{\theta}^{t+1}|\boldsymbol{\theta}^t \sim \mathcal{N}\left(\boldsymbol{\mu} = \boldsymbol{\theta}^t; \boldsymbol{\Sigma} = \text{diag}(\boldsymbol{\nu})\right), \tag{3.25}$$

where $\mathcal{N}$ denotes a multivariate normal distribution. The mean of the distribution in this case is equal to the old parameters values and the covariance is a diagonal matrix with entries on the diagonal equal to the vector $\boldsymbol{\nu}$. The vector $\boldsymbol{\nu}$ simply represents how much we allow each parameter to change from one step t to the next.

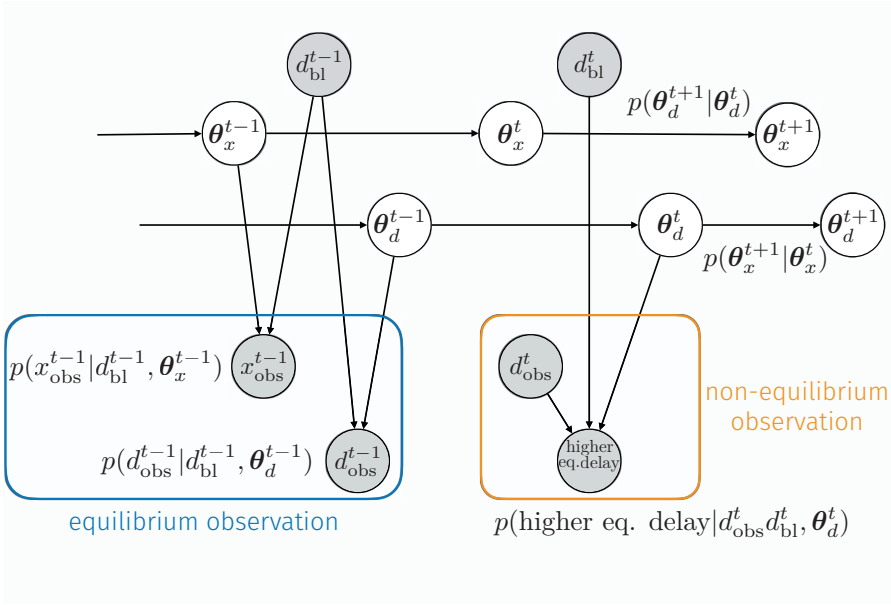


Figure 3.4 – Graphical model of the network response inference.

The complete graphical model for the network inference is depicted in Fig. 3.4. We are interested in finding the probability distributions $p^t(\boldsymbol{\theta}_x^t)$ and $p^t(\boldsymbol{\theta}_d^t)$ representing the belief on the parameter $\boldsymbol{\theta}_x^t$ and $\boldsymbol{\theta}_d^t$ at time t given the prior belief and all the observations collected up to time t. The posterior distributions are mainly needed in order to predict the expected utility $\mathcal{V}(d_{\text{bl}}, p^t(\boldsymbol{\theta}^t)$. There are mainly two possible ways to compute expectations under a distribution: deterministic approximate inference methods and sampling methods. In this case we use the deterministic approximate methods as they are considered faster and require less computations to obtain an approximate result [30]. These methods attempt to minimize a divergence measure between the true posterior, $p^t(\boldsymbol{\theta}_x^t)$ ($p^t(\boldsymbol{\theta}_d^t)$), and a second distribution, $q(\boldsymbol{\theta}_x^t)$ ($q(\boldsymbol{\theta}_d^t)$), which belongs to a fixed and predefined distribution family. The key point is that computing expectation is much simpler over the target distribution than over the original one. Several methods exist in order to achieve this goal, and a thorough discussion of these methods goes beyond the scope of this work; we refer the reader to [29] for an overview of such algorithms.

In our case we implement the Expectation Propagation (EP) algorithm on the graphical model of Fig. 3.4 [30, 31]. This method basically tries to fit the distribution of the unknown parameters with a distribution that belongs to the exponential family. We fit the distributions $p^t(\boldsymbol{\theta}_x^t)$ and $p^t(\boldsymbol{\theta}_d^t)$ with a product of univariate normal distributions, each related to a single parameter of the network response model. Approximating the two beliefs with a multivariate distribution with correlated components would certainly lead to a better approximation; however it would also require more complex operations, such as the computation of several multidimensional integrals. In order to keep the computational cost of the proposed controller as low as possible, we favor to have a

less precise but computationally lighter approximation. The EP algorithm with a fully factorized distribution corresponds to an iterative message passing algorithm among the nodes of the graphical model of Fig. 3.4, we provide in Appendix A a detailed description of the EP implementation. The output of the two network response inference blocks is a set of univariate normal distributions, one for each parameter of the approximate network response model. These distributions can then be used to predict the expected value of the application utility v, and the value of the equilibrium rate and delay, for any value of d_{bl}.

3.3.3 One-step Optimal Learning

The final subpart of the adaptive controller is the one responsible for solving the one-step optimal learning problem of Eq. (3.22). The inputs to this block are the current beliefs obtained from the network response inference method $q(\boldsymbol{\theta}_x^t)$ and $q(\boldsymbol{\theta}_d^t)$. To lighten the notation, we drop the t index of the parameters and we consider $\boldsymbol{\theta}_x$ and $\boldsymbol{\theta}_d$ to represent the current network response. Furthermore, note that the bandit problem formulation of Subsection 3.2.3 can be straightforwardly modified to take into account the parameters evolution at each step t. The only difference for the two case is that in the dynamic case we should consider the fact that the belief over the parameters is modified between two consecutive steps as effect of the Markovian dynamics of the parameters.

As the objective function of Eq. (3.22) is not convex and the optimization variable is a simple scalar value, we find the optimal value by doing a grid search over d_{bl}. For each value of d_{bl} we must compute the value of the objective function, which is composed of two terms: the immediate expected utility $\mathcal{V}(d_{\text{bl}}, q(\boldsymbol{\theta}))$, where $q(\boldsymbol{\theta}) = q(\boldsymbol{\theta}_x)q(\boldsymbol{\theta}_d)$, and the expected best future value of the same quantity. The key point in order to compute these quantities efficiently is to have access to the distributions $p(x_{\text{eq}}|d_{\text{bl}})$ and $p(d_{\text{eq}}|d_{\text{bl}})$. The two distributions basically represent the prediction distribution for the rate and delay after marginalizing out the parameters of the network response model. Fortunately, as the approximate belief $q(\boldsymbol{\theta})$ is composed of univariate normal distributions, and the network response of Eq. (3.11) and Eq. (3.12) basically corresponds to max (and min) operators over pairs of normal random variables, we can easily calculate the mean and the variance of x_{eq} and d_{eq} [32]. We can then approximate $p(x_{\text{eq}}|d_{\text{bl}})$ and $p(d_{\text{eq}}|d_{\text{bl}})$ using normal variables with the mean and variance computed before. In this case, the value of $\mathcal{V}(d_{\text{bl}}, q(\boldsymbol{\theta}))$ can simply be computed as:

$$\mathcal{V}(d_{\text{bl}}, q(\boldsymbol{\theta})) = \int v(x_{\text{eq}}, d_{\text{eq}}) p(x_{\text{eq}}|d_{\text{bl}}) p(d_{\text{eq}}|d_{\text{bl}}) dd_{\text{eq}} dx_{\text{eq}}. \tag{3.26}$$

Since $p(x_{\text{eq}}|d_{\text{bl}})$ and $p(d_{\text{eq}}|d_{\text{bl}})$ are two univariate normal distributions, the above integral can easily be computed using basic numerical integration methods. Note that the distribution $q(\boldsymbol{\theta})$ is hidden in the right-hand side of the equation inside the prediction probabilities $p(x_{\text{eq}}|d_{\text{bl}})$ and $p(d_{\text{eq}}|d_{\text{bl}})$.

In order to compute the second term, which represents the average of best future value of expected utility, we need to consider all the possible values for the rate-delay equilibrium observations, infer all the different future beliefs, maximize the future expected utility, and finally compute the mean for the different possible observations. We approximate this operation in the following way: *i*) we approximate the normal distributions $p(x_{\text{eq}}|d_{\text{bl}})$ and $p(d_{\text{eq}}|d_{\text{bl}})$ with discrete distributions using a limited number of bins (usually between 10 and 20); *ii*) for each bin we consider the value of the bin centroid as the future observation and we run the inference method for the network response assuming a prior belief equal to $q(\boldsymbol{\theta}_x)$ and $q(\boldsymbol{\theta}_d)$; *ii*) we then use the computed future belief to find the maximum expected utility using Eq. (3.26) ; *iv*) we finally average the best expected future utilities by using the bin weights of the discretized $p(x_{\text{eq}}|d_{\text{bl}})$ and $p(d_{\text{eq}}|d_{\text{bl}})$.

After carrying out these operations for all the values of d_{bl} on the grid we can select the value that maximizes the objective function and apply it as working parameter to the underlying delay-based CC. The system then waits for a new delay sensitivity update, which is triggered by the equilibrium detector block in case of new observations are collected.

3.4 Experimental Results

We implement the proposed controller in the Network Simulator NS3 [21] in order to evaluate its performance. We compare the proposed method with two other CC algorithms suited for real-time communication, namely the NADA algorithm [11] and the Self-Clocked Rate Adaptation for Multimedia (SCReAM) algorithm [33][2]. Both algorithms have been recently developed in the context of the RMCAT working group [13] and represent state-of-the-art designs. The mode of operation of the two algorithms is rather different. In order to guarantee satisfying performance in all the possible scenarios, the SCReAM algorithm, similarly to our method, employs an adaptive scheme, whereas the NADA algorithm employs a non-linear mapping of the delay to improve coexistence with loss-based algorithms.

3.4.1 Experimental settings

The parameters of Eq. (3.10) are set to the following values: $\alpha = 1/1.25$, $\delta = 8\text{RTT}$, $\beta = 0.1$ and $w = 20$ kbps. Alongside the delay-based update rule, we also monitor the packet losses to adapt the sending rate in case of operating in a loss-based environment. In order to guarantee the TCP friendliness of our controller, we enforce the sending rate to be smaller than the rate computed according to the TCP Friendly Rate Control (TFRC) [34] algorithm.

[2]For the algorithms implementation we used the source code available online at https://github.com/cisco/ns3-rmcat and https://github.com/EricssonResearch/scream, for NADA and SCReAM respectively.

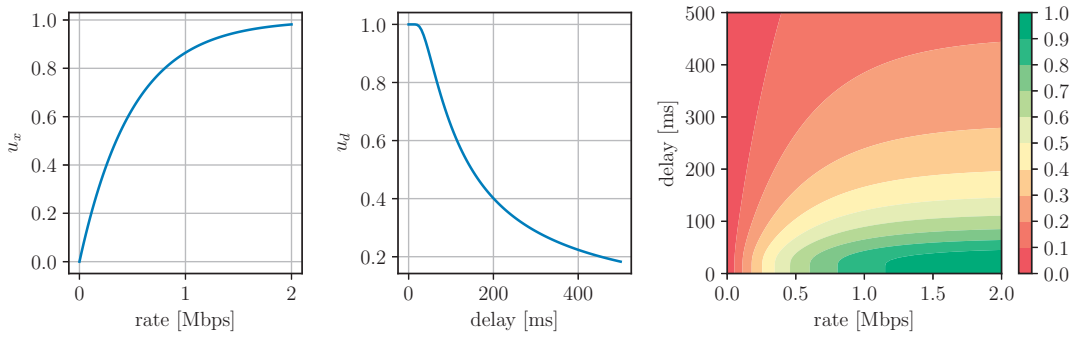


Figure 3.5 – Utility used to model delay-sensitive communication. The first and second plot show the rate utility v_x and delay utility v_d, respectively. The third plot shows the full utility value v on the rate-delay plane.

We consider an application utility $v(x, d)$, which factorizes in two functions, $v(x, d) = v_x(x)v_d(d)$, which are defined as follows:

$$v_x(x) = 1 - \exp(-\max(0, 4x)) \qquad v_d(d) = 1 - \exp(-\frac{100}{\max(d - 5, 1e - 9)}), \quad (3.27)$$

where the rate x is measured in Mbps, and the delay d in ms. The two functions have been chosen empirically, and simply aim at representing some reasonable utility model for a delay-sensitive application in order to evaluate the proposed algorithm. The functions are depicted in Fig. 3.5. The range of possible values for the delay discount is set to $d_{\text{bl}} \in [-25, 500]$ ms.

We test the proposed controller on a single link network. To simulate different network conditions, we vary the capacity of the network and the size of the network buffer. Moreover, to test the TCP coexistence, we simply initiate a TCP connection that goes through the same link.

3.4.2 Operation Analysis

In the first set of simulations we assess the mode of operation of the proposed algorithm. In a first scenario, we consider a single user transmitting data through a network link with a capacity of 0.75 Mbps, a buffer size of 500 ms, and a minimum propagation delay of 27 ms. The simulation results are depicted in Fig. 3.6. The different plots show the temporal evolution of the transmission rate, the experienced delay, and the value of the delay discount d_{bl}. Moreover the two bottom plots show the final belief on the network response. We can observe that, in the initial stage, the adaptive controller tries to decrease the delay sensitivity in order to achieve a larger rate. However, as the delay is growing significantly while the sending rate remains constant, the controller infers that the maximum sending rate is already achieved. The controller then tries to increase the

utility by reducing gradually the experienced delay. After a certain point, the value of delay discount becomes too low and does not lead to a full utilization of the capacity, (between 60 and 80 s). The controller then sets the parameter d_{bl} to a value that allows to achieve the maximum sending rate with a delay close to the minimum one.

An interesting aspect that we can notice is that the value of d_{bl} that guarantees the best utility is not equal to the propagation delay, but is actually slightly smaller. This is because d_{bl} represents the threshold after which the underlying CC considers the delay as a penalty. Since, in order to achieve a finite sending rate, the penalty has to be larger than zero, the experienced delay must also be larger than d_{bl}. The optimal value for d_{bl} is the one that, when subtracted from the minimum propagation delay, creates a penalty that has the right strength to produce a sending rate equal to the bottleneck capacity. That is why we allow d_{bl} to take negative values. In fact, when the bottleneck capacity is low, the penalty must reach rather high values to counteract the uplifting force in the rate update equation in Eq. (3.10). By using negative values for d_{bl} the adaptive controller is able to achieve low queuing delays even in these scenarios. Finally, we can notice that the network response for large values of d_{bl} is not estimated correctly. However, the uncertainty is still rather high, therefore the controller is not confident about the true network response in this region. Nevertheless, the controller correctly believes that the sending rate cannot get any higher even for large values of d_{bl}, and thus it does not seek for improving the network response belief in that specific region. This erroneous belief is due to the fact that we are actually using a piecewise linear function to fit the network response, which in reality has a more complex form. As a result, the inference might sometimes be erroneous in regions where not enough recent samples are collected.

In the second simulation, we verify the mode of operation of the algorithm when competing against a TCP flow. We consider a user sharing a network link with a TCP flow, we set the link capacity to 1.5 Mbps, the buffer size to 200 ms and the minimum propagation delay of the link to 25 ms. The simulation results are depicted in Fig. 3.7. Initially, the adaptive controller tries to vary a little bit the delay without actually affecting neither the rate nor the experienced delay. The controller then tries to increase the baseline delay by some consistent amount. It observes that, while the delay is not changing, the rate is actually increasing. After collecting other samples, the controller infers correctly what is the maximum delay of the route and that the experienced delay is actually insensitive to the parameter d_{bl}. The delay sensitivity is then reduced to the minimum making the delay penalty null. At this point the rate dictated by the delay-based update rule grows indefinitely, however, as in this case losses are experienced, the TFRC congestion algorithm limits the sending rate, guaranteeing the TCP friendliness, as shown in Fig. 3.7.

With this experiment we conclude the analysis on the mode of operation of the algorithm. In the next section we focus on the performance achieved by the algorithm in different network scenarios.

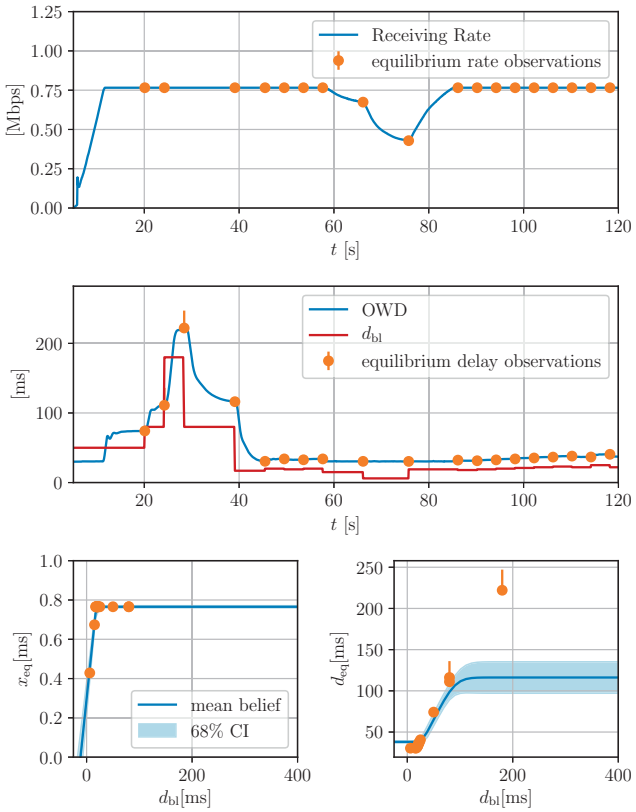


Figure 3.6 – Single user employing the proposed controller over a single bottleneck link. Evolution of the experienced rate and delay (top). Final belief (mean and 68% Confidence Interval) on the network response for equilibrium rate and delay (bottom).

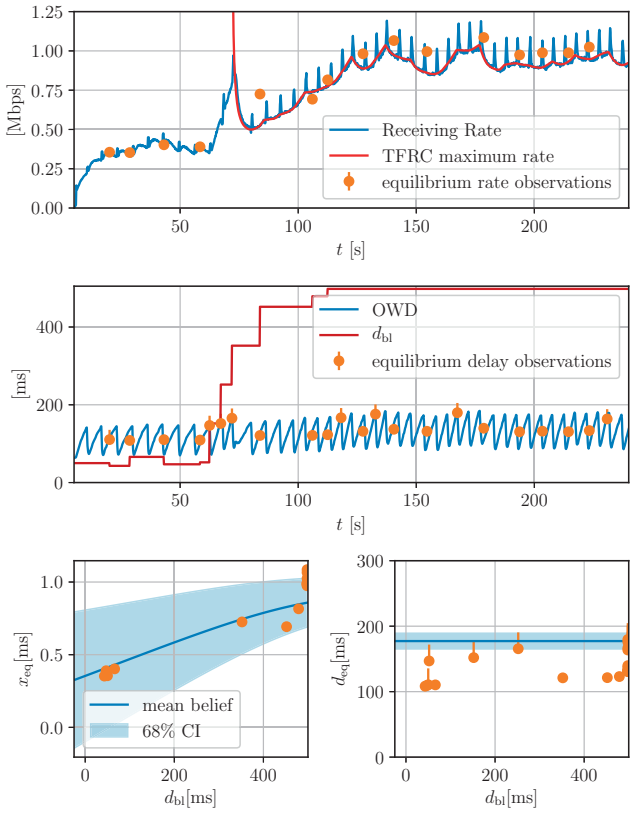


Figure 3.7 – Single user employing the proposed controller competing with TCP flow. Evolution of the experienced rate and delay (top). Final belief (mean and 68% Confidence Interval) on the network response for equilibrium rate and delay (bottom).

3.4.3 Comparison with Other Congestion Control Algorithms

We now evaluate two of the most important performance metrics of a delay-based CC algorithms: *i*) the ability to fully utilize the available capacity, and *ii*) the ability to limit the amount of queueing delay that is created in the network buffers (self-inflicted delay). We run the proposed controller on a single network link for different values of the bottleneck capacity. We then collect periodically the sending rate and the experienced delay. The box-plots shown in Fig. 3.8 depict the statistics of the collected results over a communication of 300s. The controller is able to achieve, in all the different scenarios, a channel utilization close to the maximum one. At the same time, it attains relatively low queueing delays. This means that the adaptive controller is able to effectively adapt to the different network conditions and to provide a good utility to the user.

In order to have some comparisons with other existing methods, we conduct the same tests using NADA and SCReAM. The simulation results are depicted in Fig. 3.9 and Fig. 3.10 for NADA and SCReAM, respectively. The NADA algorithm is the method that is able to achieve the most stable transmission rate. This is due to the fact that it does not employ an adaptive scheme; thus, the equilibrium point never changes. Another notable difference is that, for the NADA algorithm we can perfectly recognize the fact that smaller bottleneck capacities lead to larger delays (i.e., penalties) in order to reach the equilibrium rate. Such a clear pattern cannot actually be seen for SCReAM and for our algorithm. This is due to the adaptive schemes adopted: the controllers adapt to the different network conditions in order to reduce the experienced delay regardless of the capacity value, therefore the self-inflected delay curves do not possess any specific shape. As final remark, we can observe that, considering all the different scenarios, the proposed method can guarantee a more effective behavior, it is in fact able to achieve a small queuing delay for all the different capacity values.

In the last set of simulations, we aim at analyzing the ability of the algorithm to avoid starvation when competing with TCP flows. To this end, we initiate a TCP flow and a flow using the proposed controller on the same bottleneck link. We conduct the same experiment for different values of the bottleneck capacity and for different values of packet buffer size. We then measure the average receiving rate for the flow using the proposed controller, over a window of 500 s. In Fig. 3.11 we show the outcome of these simulations. More specifically we show the average sharing ratio, which is the ratio between the average transmission rate of the proposed algorithm and the bottleneck capacity, as well as the average delay discount d_{bl} over the entire simulation. The adaptive controller correctly estimates, for most of the simulated scenarios, that the best strategy consists in increasing d_{bl} to make the controller insensitive to the experienced delay. In fact, due to the TCP flow, it is not possible to reduce the experienced delay by any means. Among all the simulated scenarios, the worst sharing ratio, achieved for a buffer size of 500 ms and a capacity of 1.5 Mbps, is equal to 0.27, which can still be considered a satisfying performance. We conduct the same test using the NADA and SCReAM algorithms, results

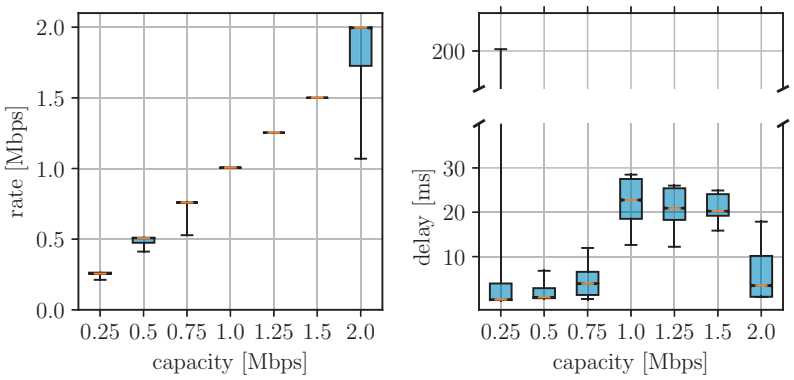


Figure 3.8 – Statistics of the transmission rate and experienced queuing delay for a single user employing the proposed controller for different values of bottleneck capacity. The box-plots show the following percentiles $[10, 25, 50, 75, 90]$.

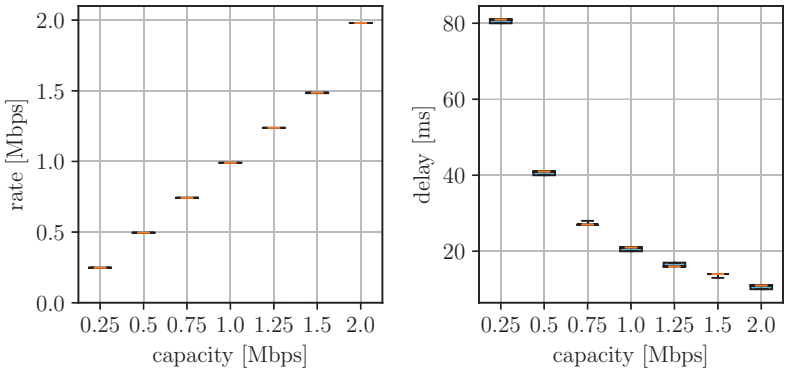


Figure 3.9 – Statistics of the transmission rate and experienced queuing delay for a single user employing the NADA algorithm for different values of bottleneck capacity. The box-plots show the following percentiles $[10, 25, 50, 75, 90]$.

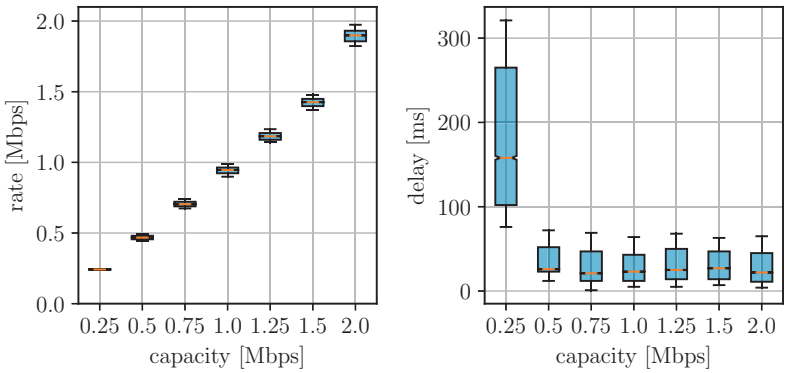


Figure 3.10 – Statistics of the transmission rate and experienced queuing delay for a single user employing the SCReAM algorithm for different values of bottleneck capacity. The box-plots show the following percentiles $[10, 25, 50, 75, 90]$.

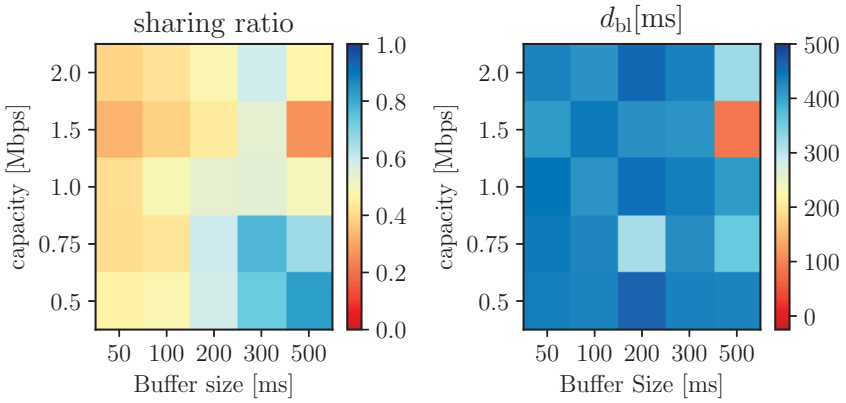


Figure 3.11 – Average sharing ratio and delay discount d_{bl} when competing against a single TCP flow, for different values of bottleneck capacity and network buffer size.

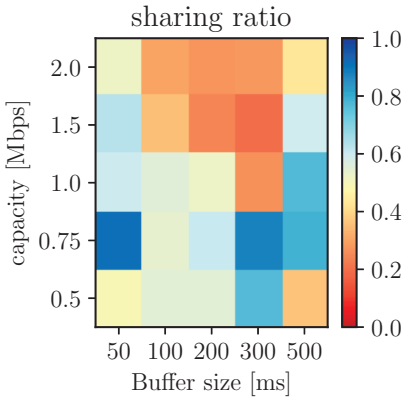


Figure 3.12 – Average sharing ratio for the NADA algortihm when competing against a single TCP flow, for different values of bottleneck capacity and network buffer size.

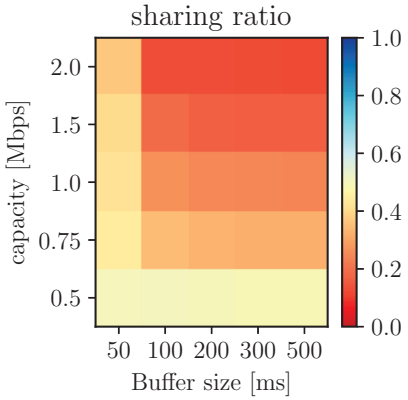


Figure 3.13 – Average sharing ratio for the SCReAM algorithm when competing against a single TCP flow, for different values of bottleneck capacity and network buffer size.

are shown in Fig. 3.12 and Fig. 3.13, respectively. The adaptive solution of the SCReAM algorithm appears to be rather ineffective. The controller achieves low sharing ratios for large buffers sizes and large link capacities. The NADA algorithm instead, though not being an adaptive scheme, can effectively avoid starvation when competing against a TCP flow. However, in this case, the sharing ratio has a quite large variability (its ideal value is 0.5). Depending on the network conditions, bottleneck capacity and buffer size, the value of the sharing ratio changes remarkably. Our proposed method instead, is able to achieve a more homogenous sharing ratio among all the different network conditions.

3.5 Related Work

In the last decades several delay-based CC algorithms have been proposed. The target applications of these algorithms are however different. For example novel versions of delay-based TCP, e.g., FAST TCP [35], have been proposed with the aim of improving the TCP performance in some specific network scenarios, as for instance in networks with a large bandwidth-delay product. Another largely used delay-based CC is the Low Extra Delay Background Transport (LEDBAT) [36]. The goal of LEDBAT is to have a congestion control algorithm that is less aggressive than TCP and that can be used for low-priority background traffic. This algorithm is in fact supposed to be outclassed by other TCP flows when sharing a common bottleneck link, as the TCP flows should represent high-priority transmission. These algorithms overall have different design objectives than the method proposed in this chapter.

The CC algorithm proposed in this chapter is meant to be used for high-priority real-time communication. Among others, most of the recent delay-based CC algorithms, that share the same objective as ours, have been proposed in the context of RMCAT, a working group of the IETF [13]. This working group aims at developing a standard for congestion control algorithms for real-time media applications. Existing advanced solutions for real-time Internet communication can be divided into two categories: namely robust and adaptive algorithms. Robust solutions are the ones that design a delay-based CC with constant parameters with the goal of providing a satisfying performance in all the possible scenarios. In this category we can cite the NADA algorithm [11]. This algorithm adopts a non linear mapping of the experienced delay in order to be robust against loss-based algorithms, similarly to [37]. The delay penalty basically increases linearly with respect to the experienced delay, up to a certain delay value; it then decreases to zero for large delays. The motivation behind this method is that, if an extremely large queueing delay is achieved, it means that the channel is shared with a loss-based algorithm, and because of this the delay measurements should be ignored. The main advantages of the robust approach are that it is simple, and, as it does not require online parameter tuning, it can converge quickly. On the other hand, it requires an accurate offline tuning of the parameters in order to reach satisfying performance in all the possible scenarios. Adaptive schemes instead, sense the network conditions and adjust some internal parameters online

in order to maximize the CC performance for each specific scenario. In this category we can cite the SCReAM algorithm [33], and the Google congestion control algorithm [38]. Both methods are based on an online adaptation of a parameters that control the delay-sensitivity of the algorithms. The goal is to reduce the sensitivity in the cases where the experienced delay cannot be properly controlled. Both methods, however, do not try to build a belief on the global network response, and are rather based on small local variations of the online tunable parameters. Moreover they also do not make use of an application utility function that describes the quality of the communication; therefore they are not able to tune the parameters in order to be optimal under a specific metric. In this chapter, we overcome these limitations by having an adaptive scheme that tries to build an exhaustive global belief of the network response, and, at the same time, tunes the parameters in order to maximize a specific communication metric. By doing this, the controller can weight the benefit and the risk of any action taken when adapting the delay sensitivity. Obviously such approach increases the computational cost of the congestion control method, however, with a proper design this additional cost can be kept limited, in order to not prevent the operation of overall method

3.6 Conclusions

In this chapter we propose a method for tuning the rate-delay sensitivity of a delay-based CC algorithm in order to maximize the communication utility of a delay-sensitive Internet application. The proposed solution is based on modeling the problem as a bandit problem and using an optimal learning approach to guarantee an efficient inference of the optimal sensitivity. More specifically, we first abstract the network as two functions that map the sensitivity of the controller to the rate-delay equilibrium pairs. We then create an approximate low-dimensional model for the network response and define a probabilistic framework for the inference of such a response. We then leverage the resulting Bayesian framework to solve a one-step optimal learning formulation and select the controller sensitivity that optimizes the long term application utility. Experimental results show that the proposed control is able to effectively adapt to different network conditions as it outperforms state-of-the-art algorithms that share the same design objective.

The main contribution of this chapter is actually to show the validity of adaptive delay-based solutions for Internet communication. It is however important to notice that adaptive solutions reveal a potential hazardous drawback. In fact, when multiple users employing adaptive controller share some common resources, they do not seek for maximization of the overall network utility, but to maximize their own utility exclusively. This might lead to cases where the equilibrium point is unlikely to represent the best rate-delay pair. Nevertheless, we believe that, even considering this potential drawback, due to the large heterogeneity of CC algorithms coexisting in the internet and the large amount of loss-based traffic, adaptive solutions still represent the most promising methods for delay-sensitive Internet communication.

4 Online Resource Inference in Network Utility Maximization Problems

In Chapter 3, we have analyzed the challenges that arise when Internet applications have different utilities and congestion measure. We have seen that, in this case, it is beneficial for some congestion control algorithms to adapt their behavior depending on the current network conditions in order to avoid an unfair and ineffective sharing of the network resources. In some cases we might face the opposite problem: a group of applications with the same utility measure might want to achieve an overall utility maximization, but the feedback observed from the network infrastructure does not represent a valid signal to steer the users to the optimal rate allocation. In this chapter[1] we investigate whether it is possible for these applications to reach the optimal allocation by exchanging information collected from the different endpoints.

At first sight, the scenario considered in this chapter might seem uncommon. However, interestingly enough, the rate allocation problem of one of the most common Internet applications, HTTP Adaptive Streaming (HAS) [39, 40], can be considered as belonging to the class of problems considered here. HAS represents nowadays the standard technology for video streaming over the internet and it employs as communication protocols HTTP over TCP. The key point is that HAS users sharing a common link tend to converge to a rate-fair allocation if they tune the rate according to their individual congestion measurements. As it is shown in [41, 42], such a rate-fair allocation is not efficient for video streaming and users should rather be coordinated at the application level in order to reach a higher quality of service. The same problem may arise whenever a network application used by different users is forced to employ a specific congestion control algorithm (e.g., TCP), while it would be better to coordinate the transmitting rates to match a specific rate allocation. If these network applications have no knowledge about the amount of available resources, the optimal rate allocation is hard to find, and classical NUM algorithms cannot help in this scenario. In this chapter, we aim at filling this gap

[1] Part of this chapter has been published in:
- S. D'Aronco and P. Frossard. Online Resource Inference in Network Utility Maximization Problems. Accepted *Transactions on Network Science and Engineering*. IEEE, 2018.

and to propose a way for extending the NUM framework to more general scenarios where resource constraints are not known a priori and the private congestion level perceived by the users cannot be used straightaway.

More specifically, in this chapter, we consider a set of users sharing different network links. The users want to transmit data across the network while optimizing the system efficiency. We assume that the coordination system knows how the network nodes are connected but does not know the transmitting capacity of the links. We further assume that, when the overall transmitting rate exceeds the network capacity, only some of the network users may detect the congestion event. The goal is then to design a distributed overlay allocation method that tunes the sending rate of all the users, so that even the users that do not detect any congestion event can adjust their sending rate to their optimal value.

In order to achieve this goal, we first separate the problem in two subproblems, one corresponding to a classical NUM problem and one corresponding to a resource inference problem. Due to the general and complex model assumed for the congestion signals the resource inference cannot be done efficiently using a passive method and the adoption of an active learning method is crucial in order to speed up the resources estimation. In the analyzed framework, performing active learning comes at the expenses of reducing the service provided to the users. In order to guarantee good system performance in both short and long term, we formulate the problem as an optimal learning problem [27], similar to the scheme adopted in Chapter 3. The original optimal learning problem is however computationally intractable, hence we carefully approximate the different mathematical steps and develop a decentralized algorithm to solve a simplified version of the problem. The algorithm distributes the operations among different processes, each associated to a single specific network link. These processes are not bounded to be executed on any specific network node, and only require communication among each other and with the different users, enabling a flexible deployment. The experimental results show the effectiveness of the proposed method and the advantage that an optimal learning formulation offers with respect to a naïve strategy that greedily optimizes the immediate performance.

This chapter is organized as follows. In Section 4.1 we introduce the problem settings and state the problem formulation. In Section 4.2 we describe how we can find an approximate solution to the problem. In Section 4.3 we provide an overview of the overall system implementation. Results from computer simulations are provided in Section 4.4. In Section 4.5 we provide a discussion on the related work. Finally conclusions are provided in Section 4.6.

4.1 Problem Settings and Formulation

4.1.1 Problem Settings

In this chapter we aim at solving the classical NUM problem introduced in Chapter 1:

$$\begin{aligned} \underset{\mathbf{x}}{\text{maximize}} \quad & \mathcal{U}(\mathbf{x}) = \sum_{n=1}^{N} u_n(x_n) \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{c} \\ & \mathbf{x} \in \mathcal{X}, \end{aligned} \tag{4.1}$$

where all the quantities coincide with the ones defined in Eq. (1.1). As described in Chapter 1 this problem can easily be solved in a distributed manner by dual decomposition. In this chapter however, we analyze the scenario where the capacity vector $\mathbf{c}$ is unknown, and thus the dual variable update, in Eq. (1.4b), cannot be explicitly computed.

In order to have some hope to find a solution to this problem we need a sort of network feedback signal that we can measure when applying an unfeasible rate allocation. In this problem we assume to have access to a per user feedback signal which however might be biased and penalize some users more then others, as a result it cannot be used straightforwardly by the users to update the sending rate. In order to be as general as possible we define a binary user congestion signal that can be seen as a generalization of many other congestion signal definitions. More in detail the considered feedback signal model is the following. We denote by the random binary variable z_m^t the occurrence of a congestion event on link m. Its probability is given by:

$$p(z_m^t | c_m, y_m^t) = \left[\sigma(\kappa(y_m^t - \rho c_m))\right]^{z_m^t} \left[1 - \sigma(\kappa(y_m^t - \rho c_m))\right]^{1 - z_m^t}, \tag{4.2}$$

where $\sigma()$ denotes the sigmoid function, y_m^t corresponds to the sum of the users sending rates passing through link m at time t, $y_m^t = \mathbf{a}_m \mathbf{x}^t$. κ and ρ are positive scalar parameters that can be used to tune the steepness and the location of the sigmoid function. Eq. (4.2) tells us that the larger the requested transmission rate for link m is, the larger the probability to face a congestion event. By using the sigmoid function instead of a step function in Eq. (4.2) we can account for some possible noise in the network, e.g., transmission bursts and noisy estimates of y_m. As the binary variable z_m^t represents the occurrence of a congestion event on link m at time t, the binary variable v_n^t represents the detection of a congestion event on the route of user n at time t ($v_n^t = 1$ when a congestion is detected by user n).

In this problem we assume that the user congestion variables $\mathbf{v}^t$ and link congestion variables $\mathbf{z}^t$ are related by the following conditions:

a) if $v_n^t = 1$ then at least one variable z_m^t with $a_{mn} = 1$, has to be equal to one. This

condition tells us that if all the links used by the user do not trigger any congestion, then the user cannot observe a congestion event.

b) If for all the users with $a_{mn} = 1$, the variables v_n^t are zero, then $z_m^t = 0$. This condition tells us that if link m triggers a congestion then one of the users employing link m has to observe a congestion event.

If two vectors $\mathbf{z}^t$ and $\mathbf{v}^t$ do not verify the above conditions, then, they are not consistent and the probability to observe a pair of inconsistent vectors is zero. In order to model mathematically the above conditions we introduce the function $\Psi_{\mathbf{v}^t}(\mathbf{z}^t)$. This function is parametrized by a vector $\mathbf{v}^t$, and maps a vector $\mathbf{z}^t$ to the set $\{0, 1\}$. Given a pair of vectors $\mathbf{v}^t$ and $\mathbf{z}^t$, $\Psi_{\mathbf{v}^t}(\mathbf{z}^t)$ is equal to one if the two vectors verify the above conditions and equal to zero otherwise. More specifically the function is defined as follows:

$$\Psi_{\mathbf{v}^t}(\mathbf{z}^t) = \prod_{n:\ v_n^t=1} \psi_n^1(\mathbf{z}^t) \prod_{m:\ \mathbf{a}_m^{\mathrm{T}}\mathbf{v}^t=0} \psi_m^0(z_m^t), \tag{4.3}$$

with:

$$\psi_n^1(\mathbf{z}^t) = 1 - \prod_m (1 - a_{nm} z_m^t), \quad \psi_m^0(z_m^t) = 1 - z_m^t. \tag{4.4}$$

The factors ψ_m^0 correspond to the above condition *b*) whereas the factors ψ_n^1 are associated to condition *a*). Note that the vector $\mathbf{v}^t$ selects the factors ψ_n^1 and ψ_m^0 that are active in full term $\Psi_{\mathbf{v}^t}$.

What we actually observe at each iteration t is not the link congestion vector $\mathbf{z}^t$, which represents a latent variable in our model, but the user congestion signals $\mathbf{v}^t$ and the users' rates $\mathbf{x}^t$ (since we assume to know the routing matrix the vector $\mathbf{y}^t$ is also known). As shorthand we denoted by $\mathcal{D}^t$ the observed data at time t: $\mathcal{D}^t = (\mathbf{y}^t, \mathbf{v}^t)$. What we aim to do is to use the observed variables in order to infer the value of the constraint vector $\mathbf{c}$, which is the quantity we are interested in, in order to find the optimal rate allocation. We can represent the above probabilistic model using the graphical model of Fig. 4.1. As can be seen the $\mathbf{z}^t$ variables are generated from the vectors $\mathbf{c}$ and $\mathbf{y}^t$ but they are related to each other depending on the value of the observed vector $\mathbf{v}^t$. Combining Eq. (4.2)-(4.4) for all the observation t we obtain:

$$p(\{\mathbf{z}^t\}|\mathbf{c}, \{\mathcal{D}^t\}) = \frac{1}{Z} \prod_t \Psi_{\mathbf{v}^t}(\mathbf{z}^t) p(\mathbf{z}^t|\mathbf{c}, \mathbf{y}^t), \tag{4.5}$$

where $p(\mathbf{z}^t|\mathbf{c}, \mathbf{y}^t) = \prod_m p(z_m^t|c_m, y_m^t)$ and Z corresponds to a normalization constant required to obtain a valid posterior distribution. Denoting with $p^0(\mathbf{c})$ the prior knowledge on the parameters $\mathbf{c}$ we can write down the joint distribution between the latent variables

$\{\mathbf{z}^t\}$ and the parameters $\mathbf{c}$:

$$p(\{\mathbf{z}^t\}, \mathbf{c}|\{\mathcal{D}^t\}) = p(\{\mathbf{z}^t\}|\mathbf{c}, \{\mathcal{D}^t\})p^0(\mathbf{c}). \tag{4.6}$$

Finally the posterior on $\mathbf{c}$ can be obtained by marginalizing out all the latent variables $\{\mathbf{z}^t\}$, leading to:

$$p(\mathbf{c}|\{\mathcal{D}^t\}) = \sum_{\{\mathbf{z}^t\}} p(\{\mathbf{z}^t\}, \mathbf{c}|\{\mathcal{D}^t\}). \tag{4.7}$$

The factor graph associated to Eq. (4.5)-(4.6) is depicted in Fig. 4.2.

In our scenario we sequentially collect the observable data $\mathcal{D}^t$, therefore we can see the belief on $p(\mathbf{c})$ as something evolving with t:

$$p^{t-1}(\mathbf{c}, \mathbf{z}^t|\mathcal{D}^t) = \frac{1}{Z^t}\Psi_{\mathbf{v}^t}(\mathbf{z}^t)p(\mathbf{z}^t|\mathbf{c}, \mathbf{y}^t)p^{t-1}(\mathbf{c}), \tag{4.8}$$

$$p^t(\mathbf{c}) = p^{t-1}(\mathbf{c}|\mathcal{D}^t) = \sum_{\mathbf{z}^t} p^{t-1}(\mathbf{c}, \mathbf{z}^t|\mathcal{D}^t). \tag{4.9}$$

We can think of $p^t(\mathbf{c})$ as the prior at time t, which is equal to the posterior at time $t-1$ ($p^{t-1}(\mathbf{c}|\mathcal{D}^t)$).

In this subsection we have formalized the user congestion signal $\mathbf{v}$ along with a probabilistic model able to generate a posterior distribution on the constraint vector $\mathbf{c}$ using the observed quantities $\mathbf{v}$ and $\mathbf{y}$.

4.1.2 Problem Formulation

Analogously to the classical NUM solving method of Eq. (1.4), we need to design a closed loop algorithm that selects at each step t a certain rate vector $\mathbf{x}^t$. It then observes the congestion feedback signal $\mathbf{v}^t$, and uses it to compute a new value of the sending rate $\mathbf{x}^{t+1}$. This process should continue till ideally the optimal allocation $\mathbf{x}^\star$ of the problem in Eq. (4.1) is reached.

There is no unique solution to the design of such feedback control system. One possible option consists in having a single loop algorithm where the allocation method at each iteration t computes under a defined policy a new allocation vector $\mathbf{x}^{t+1}$. While this design choice ideally allows to achieve better performance, as it does not impose any particular structure on the control system, it also poses several design challenges. The controller should, in fact, find a map between the prior belief $p(\mathbf{c})$, the users' requests

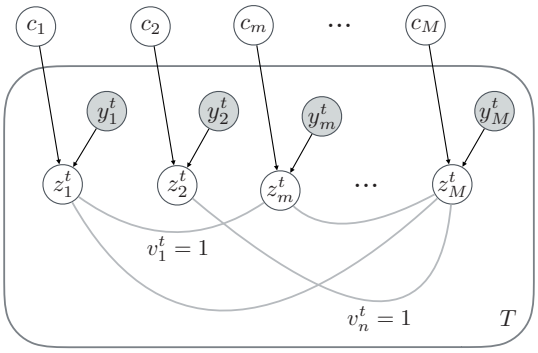


Figure 4.1 – Graphical model of the considered scenario. Grey variable nodes represent quantities that are observed at each step t.

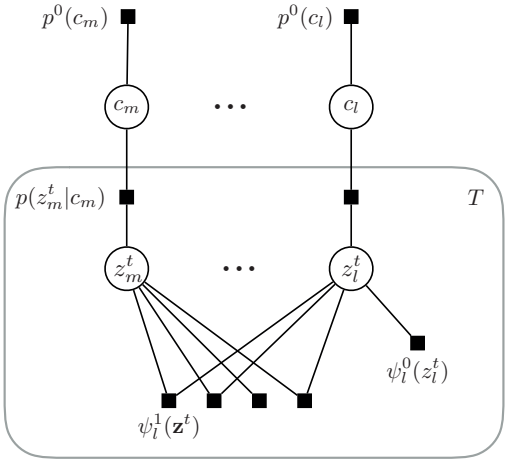


Figure 4.2 – Factor graph of one single observation t of the graphical model depicted in Fig. 4.1. Each factor $\psi_n^1(\mathbf{z}^t)$ forces at least one of the connected variables $\mathbf{z}^t$ to be one. The factor $\psi_n^0(z_l^t)$ force z_l^t to be zero.

$\{\mathbf{x}^t\}$ and the feedbacks $\{\mathbf{v}^t\}$ (along with the probabilistic model that relates $\mathcal{D}^t$ to $\mathbf{c}$), to find the rate allocation $\mathbf{x}^{t+1}$. Due to the large dimension of the input and output space this design choice is unpractical

An alternative design choice consists in an adaptive control architecture [43], where we separate the resource allocation method in two separate loops. An inner loop responsible for the users coordination, where $\mathbf{c}$ is seen as a controller parameter, and an outer loop responsible for the inference of the $\mathbf{c}$ parameter. A high-level block diagram of this system is depicted in Fig. 4.3. In this case the design of the controller is much simpler as the user rate allocation process is separated from the resource inference. The inner loop simply has to solve at step t an instance of the NUM problem with the available resources set by the vector $\hat{\mathbf{c}}^t$. In this case any method capable to solve the NUM problem, as for instance the one in Eq. (1.4), can be used. The outer loop is responsible for the inference of the true $\mathbf{c}$ vector. It receives as input the samples $\mathcal{D}^t$, and, based on the current belief $p^t(\mathbf{c})$, it then selects the best value of $\hat{\mathbf{c}}^t$ under some defined policy. Note that the belief $p^t(\mathbf{c})$ represents a hyperstate of the controller that evolves according to Eq. (4.8)-(4.9). Differently from the inner loop there is no off-the-shelf solution for the outer loop controller, the design of such subsystem is the main focus of this work. As final remark, in our design we consider to update the outer loop, and compute a new parameter $\hat{\mathbf{c}}$, after the inner loop converges to equilibrium. In this case, the observed data $\mathcal{D}^{t+1}$ corresponds to the equilibrium point of the inner loop at iteration t, which is the optimal point of the NUM problem of Eq. (4.1) when the $\mathbf{c} = \hat{\mathbf{c}}^t$, i.e., $\mathbf{y}^t = \mathbf{y}^\star(\hat{\mathbf{c}}^{t-1})$.

The design of the outer loop controller is analogous to the problem solved in the previous chapter, and it corresponds to finding a policy $\pi(\cdot)$ that maps the current belief on the constraint vector $\mathbf{c}$ into a value of the constraint vector to be used as parameter for the inner loop ($\pi : p(\mathbf{c}) \rightarrow \hat{\mathbf{c}}$) while maximizing the performance of the system. We can measure the performance of the system at each step t as the expected optimality gap when using as constraint vector for the classical NUM algorithm of the inner loop $\hat{\mathbf{c}}^t$ instead of the true unknown value $\mathbf{c}$. In order to do this, we introduce a loss function $L(\mathbf{c}, \hat{\mathbf{c}})$. Ideally the loss function models the difference in overall users' utility attained by using the estimate $\hat{\mathbf{c}}$ instead of the true value $\mathbf{c}$. Unfortunately, an exact evaluation of the optimality gap is highly expensive in terms of computations since it would require to solve the NUM problem of Eq. (4.1) twice. Instead we can consider a simpler and rather classic loss function such as the squared l^2 distance:

$$L(\mathbf{c}, \hat{\mathbf{c}}) = ||\mathbf{c} - \hat{\mathbf{c}}||_2^2. \tag{4.10}$$

The idea is that the closer $\hat{\mathbf{c}}$ is to the true vector $\mathbf{c}$ the smaller the optimality gap. Using the belief $p(\mathbf{c})$ and the loss function we can then define the risk as the expected value of the loss function:

$$\mathcal{R}(\hat{\mathbf{c}}, p(\mathbf{c})) = \mathbb{E}_{p(\mathbf{c})}\left[L(\mathbf{c}, \hat{\mathbf{c}})\right]. \tag{4.11}$$

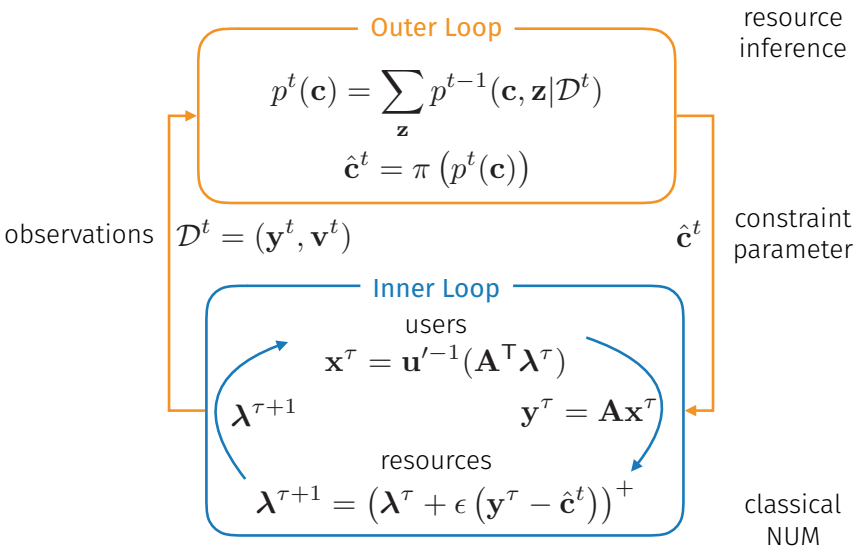


Figure 4.3 – High-level block diagram of our system. The method is composed by two separate loops. The inner loop represents a simple primal-dual distributed algorithm for solving the NUM problem (e.g., Eq. (1.4)). The outer loop is the method proposed in this work, see Section 4.2, for the selection of the sequence of constraints $\{\hat{\mathbf{c}}^t\}$ to be used as input parameter for the inner loop.

The risk corresponds to our performance metric at each step t and it represents the expected suboptimality of the rate allocation algorithm when using a resource estimate equal to $\hat{\mathbf{c}}$. Having defined the risk for a single step, we can easily extend the same metric to an entire sequence of belief-estimates pairs $\{(p^t(\mathbf{c}), \hat{\mathbf{c}}^t)\}$:

$$\mathcal{R}^\infty(\{(\hat{\mathbf{c}}^t, p^t(\mathbf{c}))\}) = \sum_{t=0}^{\infty} \gamma^t \mathcal{R}(\hat{\mathbf{c}}^t, p^t(\mathbf{c})), \tag{4.12}$$

where $\gamma \in [0, 1)$ represents a discount factor that permits to compute the cumulative risk over an infinite time horizon.

Maximizing the expected long term performance of our system is equivalent to minimizing the long term risk of being suboptimal, corresponding to Eq. (4.12). Note that consecutive beliefs $p^t(\mathbf{c}))$ are dependent; more precisely, they evolve according to Eq. (4.8)-(4.9), and the evolution depends on the value of the observed data $\mathcal{D}^t$. By combining the different elements we can formulate an optimal learning problem for the long term risk minimization:

$$\underset{\pi(\cdot)}{\text{minimize}}\ \mathbb{E}_{p(\{\mathcal{D}^t\}|\{\pi(p^{t-1}(\mathbf{c}))\})}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(\pi(p^t(\mathbf{c})), p^t(\mathbf{c}))\right] \tag{4.13}$$

where the beliefs evolves according to

$$p^t(\mathbf{c}) = \sum_{\mathbf{z}^t} p^{t-1}(\mathbf{c}, \mathbf{z}^t | \mathcal{D}^t), \tag{4.14}$$

$p\left(\{\mathcal{D}^t\}|\{\pi(p^{t-1}(\mathbf{c}))\}\right)$ corresponds to the probability of observing a sequence $\{\mathcal{D}^t\}$ for $t = 1...\infty$ given that a constraint vector $\hat{\mathbf{c}}^t = \pi(p^{t-1}(\mathbf{c}))$ is used at step t for the inner control loop, marginalized over the initial belief on $\mathbf{c}$, i.e.,

$$p(\{\mathcal{D}^t\}|\{\pi(p^{t-1}(\mathbf{c}))\}) = \prod_t \int p(\mathcal{D}^t|\pi(p^{t-1}(\mathbf{c})), \mathbf{c}) p^{t-1}(\mathbf{c}) d\mathbf{c}. \tag{4.15}$$

Since the observed data $\mathcal{D}^t$ is a random variable, the transition from $p^{t-1}(\mathbf{c})$ to $p^t(\mathbf{c})$ is also stochastic. As a result we need to average the objective function of Eq. (4.13) over the possible transitions.

The discount parameter γ, has the same meaning of Eq. (3.18) and describes how much we care about the future performance. Having $\gamma > 0$ allows to find a policy that tries to reduce not only the immediate risk but also the future ones by reducing the variance of the future beliefs on $\mathbf{c}$. As we will see, in our problem values of $\hat{\mathbf{c}}^t$ that reduce the current risk, do not coincide with those that reduce the expected value of the future risk. Using an optimal learning formulation we are able to explicitly take into account this trade-off and perform a smart choice for $\hat{\mathbf{c}}^t$ at each step t.

4.2 Approximate Solution

In this section we describe how it is possible to approximate the problem in Eq. (4.13) in order to be able to find an effective solution. We first limit the time horizon over which we optimize similarly to what is done in Chapter 3, we then describe how it is possible to approximate the true posterior on $\mathbf{c}$ using deterministic approximate inference, and finally we introduce a mean field approximation in order to deal with large network systems.

4.2.1 Receding Horizon Adaptation

Similarly to Chapter 3, to find the optimal policy for the closed loop problem of Eq. (4.13) is too demanding in terms of computations. Thus we approximate the closed loop problem with a series of receding horizon open loop problems.

Using this approach at each step t we aim at finding the estimate $\hat{\mathbf{c}}^t$ given the current belief $p^t(\mathbf{c})$ that minimizes the risk of the current step t plus an additional term corresponding to the discounted infinite sum of the expected minimum immediate risk at step $t + 1$.

Hence, we aim at solving the following optimization problem:

$$\underset{\hat{\mathbf{c}}'}{\text{minimize}} \quad \mathcal{R}(\hat{\mathbf{c}}', p^t(\mathbf{c})) + \frac{\gamma}{1-\gamma}\mathbb{E}_{p^t(\mathcal{D}|\hat{\mathbf{c}}')}\left[\mathcal{R}^\star(p^t(\mathbf{c}|\mathcal{D}))\right] \tag{4.16}$$

where $\mathcal{R}^\star(p^t(\mathbf{c}|\mathcal{D}))$ corresponds to the minimum risk for a belief equal to the posterior distribution.

In the next subsection we describe how to use approximate inference methods in order to find a convenient expression of the belief $p^t(\mathbf{c})$, which is necessary for the computation of the risk. We then deepen into the details of an approximate solution of the problem in Eq. (4.16) for large network systems.

4.2.2 Approximate Posterior Inference

The probability distribution $p^t(\mathbf{c})$ represents the belief on the parameter $\mathbf{c}$ given our original prior $p^0(\mathbf{c})$ and all the observations collected so far $\{\mathcal{D}^t\}$. The distribution $p^t(\mathbf{c})$ is used to evaluate the risk of different estimates $\mathcal{R}(\hat{\mathbf{c}}, p^t(\mathbf{c}))$. This operation involves averaging over all the possible values of $\mathbf{c}$ and it is in general computationally expensive. As in the previous chapter we decide to use a deterministic approximate inference methods in order to find a simple distribution $q(\mathbf{c})$ which resembles the true belief $p^t(\mathbf{c})$. More specifically, we implement the EP algorithm [30] on the graphical model of Fig. 4.1, using as target distribution for the parameter vector $\mathbf{c}$ a fully factorized distribution $q(\mathbf{c})$ composed of M univariate lognormal distributions. The reasons for using this specific distribution is twofold: *i*) the lognormal distribution has positive support, like the possible values of the link capacities, *ii*) as required by the EP algorithm it belongs to the exponential family. Note that considering a multivariate lognormal distribution with correlated components would certainly lead to a more accurate approximation of the true distribution, but it also has a higher storage and computational cost (the storage cost of the second order moments of the M univariate distributions grows linearly with M, whereas, for the multivariate distribution it grows quadratically with M). As we consider a fully factorized target distribution, we basically associate to each variable node of the factor graph, depicted in Fig. 4.2, a univariate distribution with tunable parameters. For the z nodes, this distribution is simply a Bernoulli distribution; whereas for the c nodes, as mentioned earlier, is the lognormal distribution. As the EP algorithm with a fully factorized distribution requires only to exchange messages locally among the different nodes of the factor graph, we can leverage the structure of the problem in order to distribute the operations and build a full distributed inference system. The iterative exchange of information among the nodes changes the parameters of these distributions till they converge to the value that minimizes a divergence measure with respect to the true distribution $p^t(\mathbf{c})$. After convergence, the outcome for the $\mathbf{c}$ parameter is a set of M univariate lognormal distributions that resemble the true belief $p^t(\mathbf{c})$. At this point instead of computing the risk using the highly complex distribution $p^t(\mathbf{c})$ we can use the

approximate distribution $q(\mathbf{c})$, which, being of a simple form, allows for a closed form expression of the risk $\mathcal{R}(\hat{\mathbf{c}}, q(\mathbf{c}))$. for a specific description of the implementation on the factor graph of Fig. 4.2, we refer the reader to the Appendix B.

4.2.3 Mean Field Approximate Solution

We now focus on the solution of problem of Eq. (4.16). Considering the quadratic loss introduced in Subsection 4.1.2 and the factorized approximate belief $q(\mathbf{c})$, we can express the risk $\mathcal{R}(\hat{\mathbf{c}}, q(\mathbf{c}))$ in a simple form:

$$\begin{aligned}\mathcal{R}(\hat{\mathbf{c}}, q(\mathbf{c})) &= \int ||\hat{\mathbf{c}} - \mathbf{c}||_2^2 q(\mathbf{c}) d\mathbf{c} \\ &= \sum_{m=1}^{M} \int (\hat{c}_m - c_m)^2 q(c_m) dc_m \\ &= \sum_{m=1}^{M} \left((\hat{c}_m - \mu_m)^2 + \sigma_m^2 \right),\end{aligned} \tag{4.17}$$

where (μ_m, σ_m^2) represents the mean and the variance of the distribution $q(c_m)$. The last expression of Eq. (4.17) consists of a sum of M parts, each corresponding to a single network link. Each part is composed by a sum of two terms: the square of the deviation from the mean of $q(c_m)$ plus the variance of $q(c_m)$. If we substitute Eq. (4.17) in Eq. (4.16) we obtain an objective function composed by two parts, the squared l^2 distance of $\hat{\mathbf{c}}$ from the mean of the current belief (immediate risk) plus a term that depends on the future belief (future risk). Considering a posterior distribution computed by applying the EP algorithm and using the current $q(\mathbf{c})$ as the true prior distribution, the second part of the objective function in Eq. (4.16) corresponds the sum of the variances of the future belief:

$$\mathbb{E}_{p(\mathcal{D}|\hat{\mathbf{c}})} \left[\mathcal{R}^\star(p(\mathbf{c}|\mathcal{D})) \right] \simeq \mathbb{E}_{p(\mathcal{D}|\hat{\mathbf{c}})} \left[\mathcal{R}^\star(q(\mathbf{c})) \right] = \mathbb{E}_{p(\mathcal{D}|\hat{\mathbf{c}})} \left[\sum_m \sigma_m^{2\prime} \right], \tag{4.18}$$

where $\sigma_m^{2\prime}$ denotes the posterior variance for parameter c_m, and we leveraged the fact that the risk is minimized when $\hat{c}_m = \mu_m$. The computation of the future posterior variance after observing data $\mathcal{D}$ can be done by simply running loopy belief propagation, similarly to Subsection 4.2.2. However, the difficulties reside in taking the expectation over all the possible observations $\mathcal{D}$. This operation is complicated for two reasons: *i*) the number of possible combinations of the vector $\mathbf{v}$ grows exponentially with the number of users N, *ii*) we do not actually assume to have a generative model for the observations $\mathbf{v}$. As a workaround for these two impediments, we propose to adopt a mean field approximation of the network, and to consider a worst case scenario for the observations $\mathbf{v}$. The main motivation behind this approach is the following. Finding the true optimal constraint vector $\hat{\mathbf{c}}$ is extremely complicated. Even for the approximate receding horizon problem

of Eq. (4.16), it would involve a large amount of computations, which are difficult to handle for large M and N. Therefore we ask whether we can optimize each entry of $\hat{\mathbf{c}}^t$ independently by considering a mean interaction of all the network links.

As mean field approximation we consider a single network link with a lognormal distribution $q(\tilde{c})$ with mean and variance equal to $(\tilde{\mu}, \tilde{\sigma}^2)$. We can write down the mean field version of Eq. (4.16) as:

$$\underset{\hat{c}}{\text{minimize}} \;\; (\hat{c} - \tilde{\mu})^2 + \frac{\gamma}{1-\gamma} \mathbb{E}_{p(\tilde{\mathcal{D}}|\hat{c})} \left[\tilde{\sigma}^{2\prime} \right]. \tag{4.19}$$

The above equation simply represents the optimal learning formulation for a single mean field link. The quantity $\tilde{\mathcal{D}} = (\tilde{y}, \tilde{\mathbf{v}})$ represents the observed data related to the mean field link. We now need to define an approximate relation between the parameter $\hat{c}$ and the future link rate $\tilde{y}$. The constraint vector $\hat{c}$ is used by the inner loop algorithm responsible for solving the classical NUM problem. Because of the shape of the utility functions, the users strive to utilize the resources as much as possible tending to make the constraints tight. As a result, we can consider, as a first approximation, $\tilde{y} = \hat{c}$.

We now need to find an approximate expression for how the posterior variance of the mean field link $\tilde{\sigma}^{2\prime}$ varies with respect to $\mathcal{D}$. In order to proceed we need to consider separately the cases where the mean field link triggers a congestion, $\tilde{z} = 1$, and when it does not, $\tilde{z} = 0$.

case $\tilde{z} = 1$

When the mean field link triggers a congestion event we know for sure that there must be at least one observation $\tilde{v} = 1$ (see Subsection 4.1.1). This observation corresponds to a factor node in the factor graph that connects the mean field link to other network links, see Fig. 4.4. If the average route length of the network is L_R, then the factor node is connected on average to the mean field link plus other $L_R - 1$ links. In this case the posterior variance of $\tilde{c}$, denoted by $\tilde{\sigma}^{2\prime}$, corresponds to the variance of the following distribution:

$$p'(\tilde{c}) = \left(\Big(1 - \prod_{l < L_R - 1} p(z_l = 0 | y_l) \Big) p(\tilde{z} = 0 | \tilde{y}, \tilde{c}) + p(\tilde{z} = 1 | \tilde{y}, \tilde{c}) \right) q(\tilde{c}) / Z_{\text{norm}}, \tag{4.20}$$

where Z_{norm} is a normalizing constant, and

$$p(z_l = 0 | y_l) = \int dc_l p(z_l = 0 | c_l, y_l) p(c_l). \tag{4.21}$$

The factor

$$1 - \prod_{l<L_R-1} p(z_l = 0|y_l), \tag{4.22}$$

of Eq. (4.20) is equal to the probability to trigger a congestion event by the other $L_R - 1$ links composing the route. If the probabilities $p(z_l = 0|y_l)$ are small $\forall l$ then it is easy to see that the variance of $p'(\tilde{c})$ is basically equal to the current one $\tilde{\sigma}^{2\prime} = \tilde{\sigma}^2$. The only way to achieve a gain in the posterior variance is to have a high probability $p(z_l = 0|y_l)$ for all the other links composing the route. However, note that in the mean field model all the links are supposed to adopt the same strategy, as a result if $p(z_l = 1|y_l) \simeq 0$ for all the links, it is also true that $p(\tilde{z} = 1|\tilde{y}) \simeq 0$ for the mean field link. In this case the considered event of a congestion by the mean field link would be extremely unlikely to happen, and by extension also the variance reduction. One way to bypass the problem is to consider that the network links can belong to two different classes: class A and B. Links of class A have a probability of triggering a congestion event equal to p_{A1}, whereas links of class B have probability p_{B1}; more specifically p_{A1} (similarly for p_{B1}) is defined as

$$p_{\mathrm{A1}} = p(z = 1|y = \hat{c}_{\mathrm{A}}) = \int dc p(z|c, y)p(c) = 1 - p_{\mathrm{A0}}. \tag{4.23}$$

If we consider $p_{\mathrm{A1}} > p_{\mathrm{B1}} \simeq 0$, with the mean field link belonging to class A, and the other links composing the route belonging to class B, then Eq. (4.20) can actually lead to drastically change the mean field posterior belief. By using the class notation Eq. (4.20) becomes

$$p'(\tilde{c}) = \Big(\Big(1 - p_{\mathrm{B0}}^{L_R-1}\Big)\, p(\tilde{z} = 0|\tilde{y}_A, \tilde{c}) + p(\tilde{z} = 1|\tilde{y}_A, \tilde{c})\Big) q(\tilde{c})/Z_{\mathrm{norm}}, \tag{4.24}$$

In the worst case scenario of a single user feedback v for the mean field link, the probability of having such route is equal to $(1 - \alpha)^{L_R-1}$. In all the other cases, i.e., when there is another link of the route that belongs to class A, we consider to have a posterior distribution with the same variance of the current belief.

case $\tilde{z} = 0$

The second case corresponds to the event where the mean field link does not trigger a congestion event. In this scenario the belief variance can be reduced only if none of the users employing the link observes a congestion event, see Subsection 4.1.1. In this case we can consider as worst case scenario the condition where a user observes no congestion only when none of the employed links is congested. The probability for an uncongested mean field link to have no users with $\tilde{v} = 1$ is then equal to:

$$p_{\mathrm{A0}}^{\alpha(L_C-1)} p_{\mathrm{B0}}^{(1-\alpha)(L_C-1)}, \tag{4.25}$$

where L_C is the average number of links that share at least one user with the mean field link, see Fig. 4.4. In mathematical terms L_C is the average number of non-zero entries for the rows of the matrix $\mathbf{A}\mathbf{A}^\mathsf{T}$. In this case the new variance is equal to the variance of the following distribution:

$$p'(\tilde{c}) = p(\tilde{z} = 0|\tilde{y}_A, \tilde{c})q(\tilde{c})/Z_{\text{norm}}. \tag{4.26}$$

Finally, we need to consider the same analysis when the mean field link belongs to class B. It is easy to see that in this case due to the shape of the likelihood function and the assumption $p_{\text{B1}} \simeq 0$, the belief variance would basically remain constant even if we could observe the hidden variable $\tilde{z}$. As a result we assume a worst case scenario where the expected posterior variance for the links of class B remains constant.

Combining the two scenarios we can write down an approximate expression for the expected future variance of the mean field model:

$$\begin{aligned}\mathbb{E}[\tilde{\sigma}^{2\prime}] =&(1-\alpha)\tilde{\sigma}^2+\\ &\alpha\Big(p_{\text{A1}}\Big((1-\alpha)^{L_R-1}(\tilde{\sigma}^{2\prime}_{\text{A1}}-\tilde{\sigma}^2)+\tilde{\sigma}^2\Big)+\\ &p_{\text{A0}}\Big(p_{A0}^{\alpha(L_C-1)}p_{B0}^{(1-\alpha)(L_C-1)}(\tilde{\sigma}^{2\prime}_{\text{A0}}-\tilde{\sigma}^2)+\tilde{\sigma}^2\Big)\Big),\end{aligned} \tag{4.27}$$

where $\tilde{\sigma}^{2\prime}_{\text{A1}}$ and $\tilde{\sigma}^{2\prime}_{\text{A0}}$ correspond to the variance of the distribution in Eq. (4.24) and Eq. (4.26) respectively. In Eq. (4.27) we consider that the only scenarios where the belief variance changes is when a link belongs to class A and the aforementioned worst case condition are observed. In all the other scenarios the belief variance does not change. Note that, since we lack of a generative model for $\mathbf{v}$, the conducted analysis does not aim at producing an accurate model for the expected posterior variance, but rather at modeling some situations where we expect to have a considerable variance reduction of the belief on $\tilde{c}$.

The developed model can now be used in order to minimize the mean field objective function of Eq. (4.19). As independent variables of our model we consider α, p_{A1} and p_{B1}, which are all defined over the interval $[0, 1]$. α here represents the ratio of links that belong to class A. Note that since $p(z = 1|y = \hat{c})$ is monotonically increasing, it can be inverted, and we can find a map from p_{A1} to $\hat{c}_{\text{A}}$, which is denoted by $\hat{c}_{\text{A}}(p_{\text{A1}})$. The final problem then becomes:

$$\underset{p_{\text{A1}},p_{\text{B1}},\alpha}{\text{minimize}}\ \alpha(\hat{c}_{\text{A}}(p_{\text{A1}})-\tilde{\mu})^2+(1-\alpha)(\hat{c}_{\text{B}}(p_{\text{B1}})-\tilde{\mu})^2+\tilde{\sigma}^2+\frac{\gamma}{1-\gamma}\mathbb{E}[\tilde{\sigma}^{2\prime}](p_{\text{A1}},p_{\text{B1}},\alpha). \tag{4.28}$$

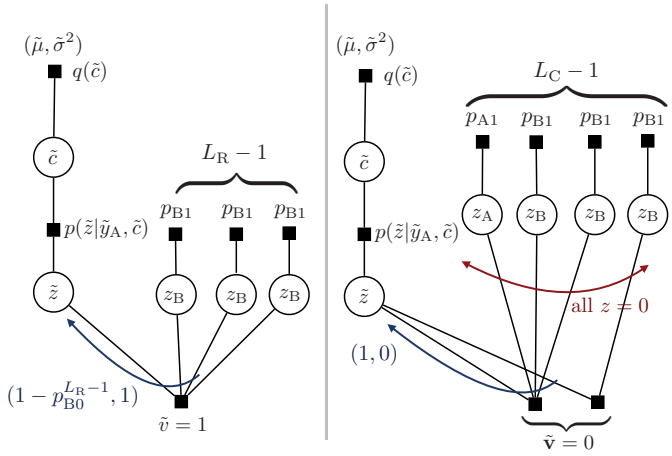


Figure 4.4 – On the left, factor graph for the case of congestion event signal observed from the users. The information flowing upwards to the variable node c of the mean field link affects the variance of the posterior belief. On the right, worst case scenario for observing no congestion signals from the users of one network link. We request in this case that all the links connected to the mean field through all of its users to be feasible.

All the terms in Eq. (4.28) are well defined and since the optimization problem is non-convex, but it involves only three variables, we can simply solve it by performing a grid search.

In Fig. 4.5 we show how the expected posterior variance and the value of the objective function vary as a function of the optimization variables. More specifically we show the approximate expected posterior variance as a function of α and p_{A1}. Empirically we observed that the minimizer is usually located at $\alpha \simeq 1/L_R$. This is intuitively reasonable, as, in order to reduce the posterior variance in the case of link congestion, we need to have routes that are composed by one link of class A and all the other links of class B. Having one link every L_R that belongs to class A increases the chances of observing such routes. The minimizer with respect to p_{A1} has instead a more complex dependency with respect to the other variables and parameters of the model, however, we observed empirically that it is usually located between 0.5 and 0.8 making rather uncertain the possibility of observing a congestion. The p_{B1} parameter instead minimizes the future variance when is set to 0. In fact, in order to increase the chances to reduce the variance of the links of class A p_{B1} should be as low as possible. However, a low p_{B1} increases the deviation from the mean belief and consequently the immediate risk for the links of class B. The optimal learning formulation allows to easily take care of this trade-off. To this extent we can observe in Fig. 4.5 that the minimizer of the objective function with respect to p_{B1} is usually located around $0.01 - 0.05$. Note that the optimal value of p_{B1} is consistent with the assumption $p_{\mathrm{B1}} \simeq 0$ made in the analysis conducted above. If this condition was not true at the optimal point, then we could not have used the approximate mean

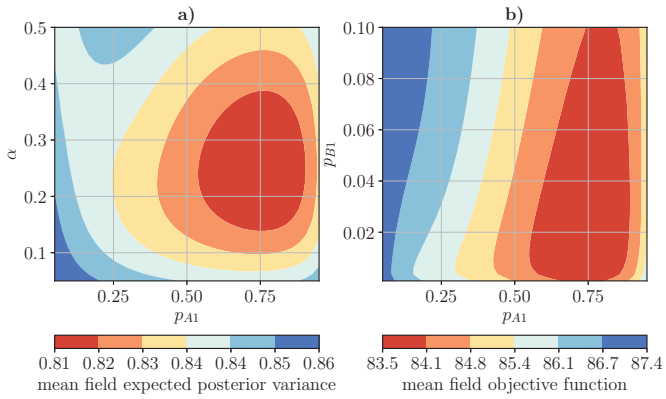


Figure 4.5 – **a**) Numerical evaluation of the posterior expected variance of the mean field model defined in Eq. (4.27). The figure shows how this quantity changes as a function of p_{A1} and α with a fixed $p_{\mathrm{B1}} = 0.01$. **b**) Numerical evaluation of the objective function of the mean field model defined in Eq. (4.28). The figure shows how this quantity varies as a function of p_{A1} and p_{B1} ($\gamma = 0.99$). For both figures, the other involved parameters have been set to $L_R = 4$, $L_C = 20$, and $\tilde{\sigma}^2 = 0.85$, respectively.

field variance prediction developed in this section. Finally, we noticed that as long as the sigmoid function of Eq. (4.2) is sharp with respect to the mean field belief $q(\tilde{c})$, i.e., large variance of $q(\tilde{c})$, then the optimal parameters $(p_{\mathrm{A1}}, p_{\mathrm{B1}}, \alpha)$ are not sensitive to the value of $\tilde{\sigma}$. As a result, as long as we are uncertain about the capacity values, we are not forced to re-optimize the mean field parameters every time the belief changes.

In the next section we summarize the complete algorithm proposed in this chapter and we discuss the key points of its implementation.

4.3 Overview of the Complete Algorithm

The complete set of operations required to run our algorithm is reported in Alg. 1. The overall system is composed by two groups of processes, namely the N user and the M link processes (each responsible for an individual network link). First, each user n requests the prices λ_m of the employed links to the different resource processes and computes the sending rate according to Eq. (1.4a) (lines 3-4). The users then forward the rate x_n^t and the feasibility signal v_n^t to all the process of the employed links (line 5). At this point the resource processes compute independently the link rate y_m^t simply by adding up all the users rates and update the price using Eq. (1.4b) (lines 7-8). The next operation consists in updating the dataset $\{\mathcal{D}^t\}$ with the new observed datapoint (line 9). In order to limit the size of the dataset we subsample the points that are included, and we set a maximum number of stored points (we discard the older points when the maximum size is reached). In this way we can reduce the number of operation required by the EP algorithm. A more detailed description of the updating operation can be found in the Appendix C.

When the inner loop has converged, we recompute the value of the constraint vector $\hat{\mathbf{c}}$ (line 10). Note that we can establish the convergence of the inner loop algorithm by monitoring the variation of the vectors $\boldsymbol{\lambda}^t$ and $\mathbf{y}^t$. Alternatively we can simply update the vector $\hat{\mathbf{c}}$ on a fixed time basis. The next operation corresponds to running the EP algorithm in a distributed way among the M link processes using the data available in the dataset (line 11). Once the belief is updated we use the results from the mean field analysis to set $\hat{\mathbf{c}}$. In order to execute this step we need to know the optimal values for $(\alpha, p_{\mathrm{A1}}, p_{\mathrm{B1}})$. This operation can either be computed by each link process independently, or it can be computed by one process and forwarded to the all the link processes. Since we consider the network topology and the sigmoid function characterizing the probability of a link congestion to not change in time, we can compute the optimal values in advance using the information on the network topology and the initial prior on the network links, then hardcode the optimal values of $(\alpha, p_{\mathrm{A1}}, p_{\mathrm{B1}})$ in the link processes. As shown in the results section this simple strategy is sufficient to provide satisfying performance of the proposed algorithm. Since we know that links belonging to class A are the ones that are likely to reduce their variance, instead to assign randomly at each step t αM links to class A and $(1-\alpha)M$ to class B, we assign to class A the αM links with the largest variance (line 15-16). Before doing this operation we set the variance of the links that were underutilized in the previous time step to zero (lines 12-14). The intuition is that if links are underutilized, then it is useless to reduce the uncertainty on their capacity value as they are associated to loose constraints, and the optimal rate allocation does not depend on them. Finally, after the class assignment each link process, given its current belief computes the value of $\hat{c}_m$ that matches the probability to trigger a congestion equal to the value of its class. Note that the class assignment operation can be done in a fully distributed way by using a consensus algorithm. More details on its implementation can be found in the Appendix C.

The operation, listed in Alg. 1 are executed continuously. The belief on $\mathbf{c}$ is expected after some steps to converge, shrinking the probability density around some value of $\mathbf{c}$, that should match the true value of the link capacities.

We now briefly discuss at a high-level the complexity of the algorithm in terms of communication and storage cost. The inner loop of the proposed system corresponds to a classical NUM algorithm: at each step t each user communicates with the resource processes of the links composing the route to collect the link prices and forward the sending rate and congestion signal. Considering an average route made of L_R links, this operation involves a transmission of $\mathcal{O}(NL_R)$ messages. The execution of the EP algorithm is the most expensive part in terms of communication and storage requirements of the entire algorithm. Each link process m has to store, for each episode t in the dataset, the route of the users who employed link m and detected a congestion event $v_n = 1$ (plus the link rate y_m). Therefore, if we denote by N_{link} the average number of users per link, the storage cost is $\mathcal{O}(N_{\mathrm{link}}L_R)$ for each element in the dataset and for each link process. $N_{\mathrm{link}}L_R$ basically corresponds to the average number of factor

Algorithm 1 Complete algorithm

loop
  for each user n **do**
    collect $\lambda_m^t \forall m \in n$
    compute and apply rate $x^t(\mathbf{a}_n^\mathsf{T}\boldsymbol{\lambda}^t)$
    Forward v_n^t and $x_n^t \forall m \in n$
  for each link m **do**
    ${\lambda_m}^{t+1} = \max(0, {\lambda_m}^t + \epsilon(y_m^t - \hat{c}_m^t))$
    $\hat{c}_m^{t+1} = \hat{c}_m^t$
    update dataset $\{\mathcal{D}^t\}$
  if inner loop converged **then**
    compute $q^t(\mathbf{c})$ using EP
    for each link m **do**
      if $\lambda_m^t = 0$ **then**
        $\sigma_m^2 = 0$
    assign to class A $M\alpha$ links with largest σ_m^2
    assign to class B other links
    for each link m **do**
      find $\hat{c}_m \ : \ p(z' = 1|\hat{c}_m) = p_m^{\text{class}}$
    $\hat{c}_m^{t+1} = \hat{c}_m$

nodes ψ_n^1 connected to each variable node z_m^t in the factor graph of Fig. 4.2. This means that, in terms of communication requirements, one complete update of the incoming messages for all the variable nodes in the factor graph requires an exchange of $\mathcal{O}(N_{\text{link}}L_R)$ messages for each link process and for each episode in the dataset. EP on a factor graph with loops has to be executed multiple times on the entire dataset before convergence. Unfortunately, it is not easy to quantify the number of iterations required by the EP algorithm to converge. In our implementation we stop the EP update after ten iterations, since empirically we have observed that they are usually sufficient to provide a good belief estimate. Considering fixed the amount of refinements required by the EP algorithm, the overall cost, in terms of communication and storage, at time t is $\mathcal{O}(N_{\text{link}}L_R t)$ for each process M. The complexity grows linearly with time because the dataset is obviously growing. In order to avoid this we limit the size of the dataset to the last $S_\mathcal{D}$ observations. The key point in the complexity analysis is the dependency of L_R with respect to N_{link}. The answer to this question however depends on the network topology. For instance, if the network belongs to the class of small-world networks, and user routes correspond to the shortest routes between the source and destination nodes, then L_R grows logarithmically with respect to the number of total network nodes, making the overall complexity of the algorithm more tractable. Finally, the mean field optimization is basically independent of the numbers of links and users. The class assignment task uses a simple method derived from classical consensus average algorithms [44], and simply involves exchange of local messages among the M link processes. This concludes the discussion on the

implementation of the proposed algorithm, in the next section we show and discuss the performance of the proposed method.

4.4 Experimental Results

In order to test the proposed method we generate some random networks with different numbers of links and users, more specifically we generate scale-free networks according to Price's model. We assign to each user a log-shaped utility function $u_n(x) = w_n \log(x)$, where w_n is a random positive parameter, and a random route connecting two (non adjacent) nodes of the network. After the routing matrix is set, we sample the prior distribution of the link capacities in order to set their true value. We consider that each link has a lognormal distribution with mean and standard deviation equal to (μ_m^0, σ_m^0). We set $\mu_m^0 = 2.7N_m$ where N_m is equal to the number of users using the link and $\sigma_m^0 = 0.27N_m$. Regarding the parameters of the sigmoid function of Eq. (4.2) we fix $\rho = 0.95$ and $\kappa = 5((1-\rho)c_m)^{-1}$. In this case we have that when $y_m = 0.95c_m$ there is a 0.5 probability for link m to trigger a congestion event, whereas when $y_m = c_m$ the probability goes up to about 0.99. We optimize the values of $(p_{\mathrm{A}1}, p_{\mathrm{B}1}, \alpha)$ at the beginning of each simulation using the L_{R} and L_{C} of the network in use, the average mean and variance of the prior distribution of the different links, and the parameters κ and ρ defined above. In order to generate the observation vector $\mathbf{v}$ at each step t, we first generate the link congestion vector $\mathbf{z}$ using Eq. (4.2) and then we randomly pick a value of $\mathbf{v}$ among the ones that are consistent with the model specified in Subsection 4.1.1.

In the first test we run the algorithm on a network with 12 links and 200 users and a discount factor of $\gamma = 0.99$, Fig. 4.6 shows the evolution of some quantities involved in the algorithm operation. The plots show the evolution of the mean value of the lognormal belief of $q^t(c_m)$, the 95% Confidence Interval (CI) of the belief, the value of $\hat{\mathbf{c}}^t$, and finally, the sum of the users sending rates for link m, y_m. The selection of $\hat{\mathbf{c}}^t$ is consistent with the results of the mean field approximation. For example, between $t = 0$ and $t = 200$, the value of $\hat{b}_1$ (which has a large variance) is set to a value that is slightly larger than the mean value, whereas for link 5 and 6 $\hat{c}$ is set to a value below the mean of their belief. We can verify our assumption about the approximation $\hat{\mathbf{c}} = \mathbf{y}$. As it can be seen, the requested link rate y_m tends to converge to the value of the parameter $\hat{c}_m$. When it does not reach the value $\hat{c}_m$ it is because the link capacity is too large and the link is underutilized at equilibrium. The assumption $\hat{\mathbf{c}} = \mathbf{y}$ holds for the remaining active links. Concerning the underutilized links, we can observe that, since the rate y_m reached by the inner loop is much lower than the mean value of the capacity belief, the uncertainty of the constraint cannot be reduced and it remains rather high. However, in this case, there is no need to reduce the variance of these capacities as they do not affect the optimal rate allocation. For the active links, the algorithm continues to sample at values of y_m close to the mean value of the belief. As a result, the variance sequentially shrinks and the mean value approaches the true value of c_m.

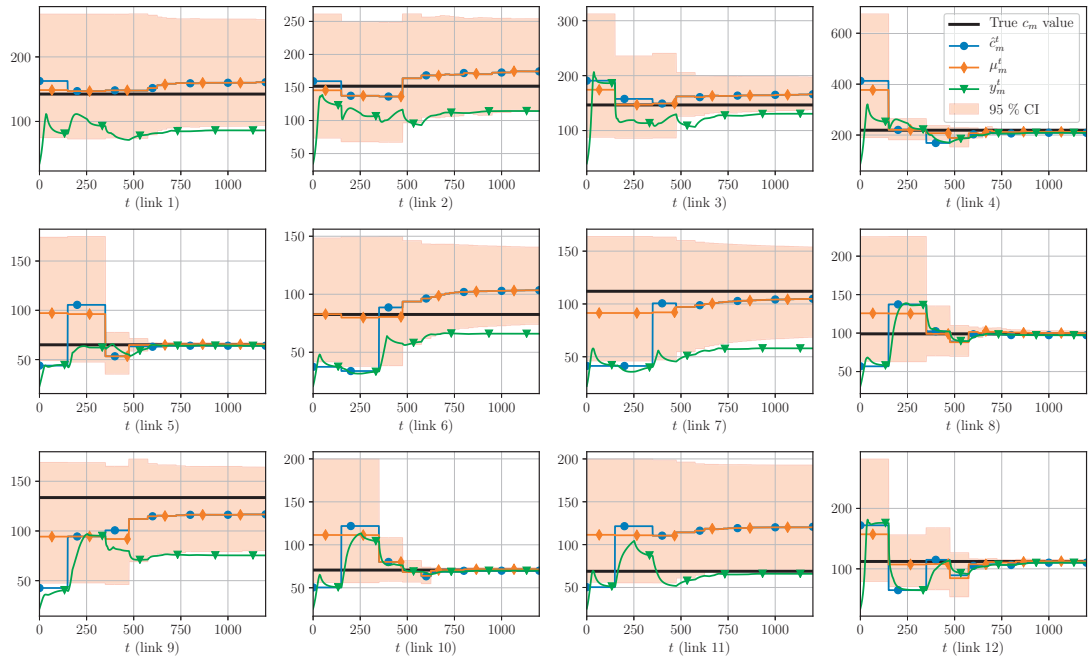


Figure 4.6 – Evolution of the most important quantities employed by the proposed algorithm for all the M links. Each subplot shows: the true c_m value, the evolution of the mean value of the lognormal belief of c_m, the 95% Confidence Interval (CI) of the belief, the value of the decisions $\hat{\mathbf{c}}^t$, and finally, the users sending rate through link m y_m.

In order to evaluate the performance of the rate allocation algorithm we compare it to a greedy algorithm that simply minimizes the immediate risk (i.e., $\gamma = 0$) and matches the value of $\hat{\mathbf{c}}$ with the mean value of the current available belief. The greedy algorithm does not seek for values of $\hat{\mathbf{c}}$ that are expected to reduce the future belief uncertainty, and resembles a passive method that does not perform any active learning. For our algorithm we set the values of the future risk discount γ to 0.98, 0.99 and 0.995. We use a topology with 48 links and 800 users. We draw ten different sets of samples of $\mathbf{c}$ from the prior distribution and ten different sets of values for the parameters w of the utility functions. We then run the proposed algorithm with different γ values and the greedy method on the same ten different random settings. In Fig. 4.7 we plot the evolution of the average value over the different runs, and the standard deviation, of the mean absolute percentage error for the capacity values. More specifically the metric used corresponds to:

$$e_{\text{links}} = 100\% \sum_m \frac{|\mu_m - c_m^{\text{true}}|}{M c_m^{\text{true}}} \tag{4.29}$$

where μ_m represent the mean of the current belief. The greedy algorithm is actually able to reduce the parameters uncertainty by a larger value in the very first stages but then it struggles to further reduce the uncertainty which remains constant around 22% throughout the entire simulation. Running the system with $\gamma \simeq 1$ instead leads to lower

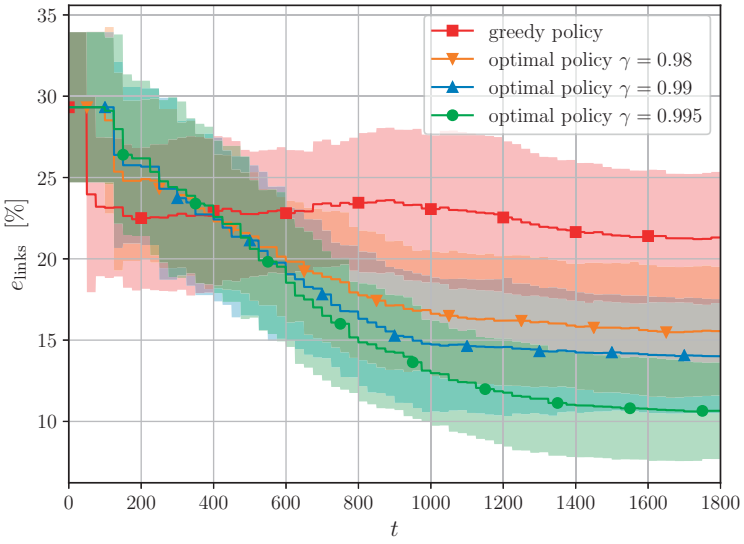


Figure 4.7 – Evolution of the mean absolute percentage error between μ_m^t and the true vector c_m^{true}. The plot shows the average and standard deviation of the metric among ten different runs.

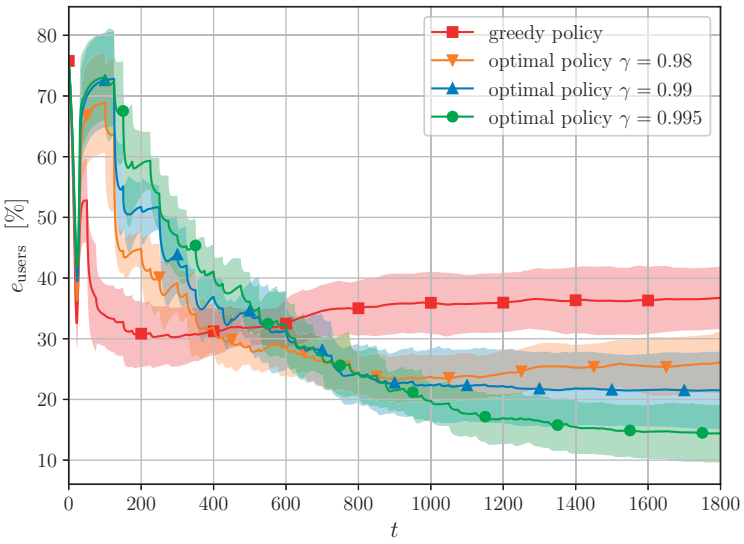


Figure 4.8 – Evolution of the mean absolute percentage error between the rates x_m^t used and the optimal rates $x_n^\star$. The plot shows the average and standard deviation of the metric among ten different runs.

values for the mean absolute percentage error, between 10% and 17% for the different values of γ. The error decreases slowly at the early stages, however the reduction is more persistent and eventually achieves much lower mean error. Moreover we can see that larger values of γ lead to lower long term error. Due to the existence of underutilized links we cannot expect the error to decrease to zero. In fact for these links the average error is expected to remain large since the belief variance does not decrease.

Another metric that can be used to evaluate the algorithm is the evolution of the mean absolute percentage error of the users' sending rate with respect to the optimal ones:

$$e_{\text{users}} = 100\% \sum_n \frac{|x_n - x_n^\star|}{N x_n^\star} \tag{4.30}$$

This quantity is not sensitive to potential underutilized links and directly measures how the users' rates are close to the optimal ones. The evolution of this metric is shown in Fig. 4.8. As for the previous figure, we show the mean and standard deviation among ten different runs. In order to reduce the future risk our algorithm sacrifices the immediate one, achieving a larger error in the first steps with respect to the greedy strategy. However, in the final steps, when the capacity beliefs are more accurate, the users' rates tend to be closer to the optimal ones and our method, for $\gamma = 0.995$, settles around $e_{\text{users}} \simeq 15\%$. The greedy algorithm instead is not able to significantly reduce the error for the future steps, and settles to an average percentage error of 37%.

We further evaluate the performance of the algorithm for different network sizes. We generate five different networks, with $M = [24, 36, 48, 60, 72]$ links and $N = [400, 600, 800, 1000, 1200]$ users. For each network we draw ten different sets of values for $\mathbf{c}$ and w. Though the network size differs for the different simulations we generate the users routes in order to have an average route length of 4 links for each topology. We run the simulation for the greedy policy and for γ equal to 0.98, 0.99 and 0.995. As for the previous tests we compute the mean absolute percentage error of the capacity estimates and the users' rates. We then plot the mean and standard deviation for the ten different runs after 1800 iterations. The results are depicted in Fig. 4.9 and Fig. 4.10. The best performance is achieved with $\gamma = 0.995$, with e_{users} between $15 - 30\%$. the greedy strategy achieves the worst performance for all the different network sizes, with about $38 - 45\%$ of error with respect to the optimal user rate. The results show that the performance of the algorithm is not strongly correlated with the network size. The performance discrepancies among the different sizes are likely due to the different random realizations of the network topology.

In the final simualtion we investigate how the average length of the user routes affects the performance of the proposed method. We use a network topology with $M = 48$ and we generate ten different realizations of the user populations for different values of the average route length, with $L_R = [3, 4, 5]$. We then compute the mean and standard

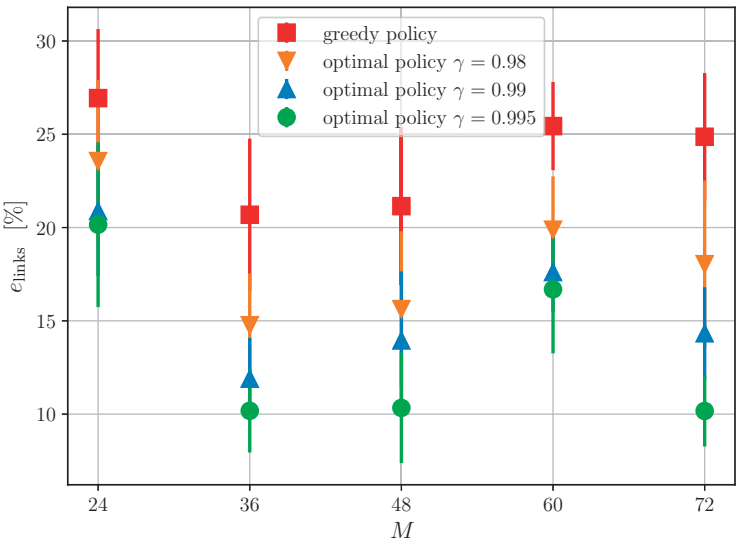


Figure 4.9 – Mean absolute percentage error between μ_m^t and the true vector c_m^{true} for networks with a different number of links M. The plot shows the average and standard deviation of the metric among ten different runs.

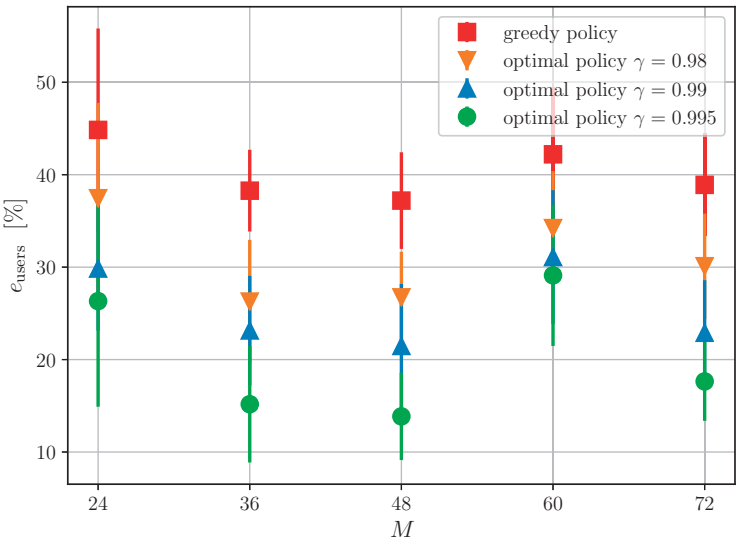


Figure 4.10 – Mean absolute percentage error between the rates x_m^t used and the optimal rates $x_n^\star$ for networks with a different number of links M. The plot shows the average and standard deviation of the metric among ten different runs.

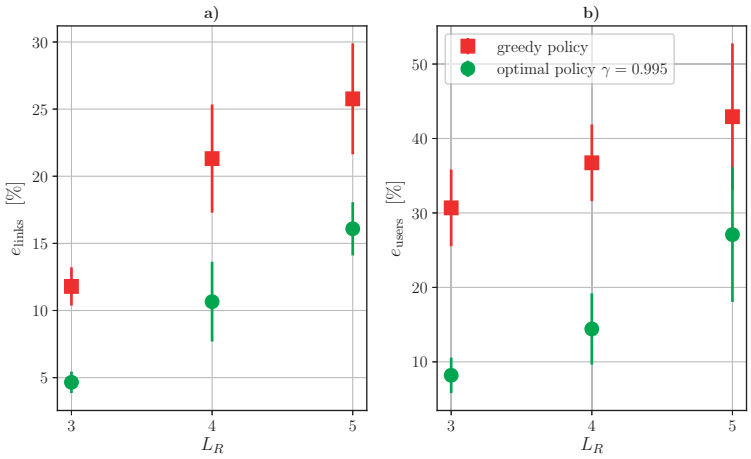


Figure 4.11 – **a**) Mean absolute percentage error between μ_m^t and the true vector c_m^{true} for different average route length L_R. **b**) Mean absolute percentage error between the rates x_m^t used and the optimal rates $x_n^\star$ for different average route length L_R The plot shows the average and standard deviation of the metric among ten different runs..

deviation of the same metrics used in the previous experiments for the different scenarios. The results are shown in Fig. 4.11. The results show that, when the average route length increases the system performance decrease for both the greedy policy and the proposed foresighted method. However the proposed method always outperforms the greedy policy. This is somewhat expected since longer routes make it more challenging for the inference method to correctly estimate the value of the latent variables $\mathbf{z}$. As a consequence, the value of the link capacities is also harder to infer.

4.5 Related Work

Some works, motivated by similar arguments to ours, analyze how classical NUM algorithms are affected by potential noisy measurements of the feedback signals [45, 46]. In particular these studies consider the case where the congestion measurements performed by the users are corrupted by a biased noise which prevents the network from achieving the optimal rate allocation. The aim of this study however, compared to the latter ones is different: whereas these works try to assess the optimality gap due to biased feedbacks, in this chapter we try to infer the available resources in order to implement an overlay NUM system that is then robust to biased feedback. We are not aware of any work on NUM that tries to build a user coordination system without having access to direct measurements of the available resources. The only known work that attempts to solve a NUM problem with unknown constraints is [47]. The authors however assume to have access to measurements of the constraint vector perturbed by a zero mean noise. In our case we rather assume to never have direct access to the link capacities but only to

indirect users congestion signal measurements. A somewhat similar problem is analyzed in [48]. In this NUM related work, the private congestion signals measured by the different users are heterogenous, leading to an inefficient rate allocation. The authors use a game theoretic framework in order to design a coordination system based on local users' beliefs without any explicit user communication. However in this study it is assumed that the congestion signals that the user can measure are linearly coupled with the actions, i.e., the sending rates, of all the users. In our case instead, we do not make such assumption, and the congestion signal observed by some users could be completely insensitive to the actions of other users that employ the same network resources. Therefore we are able to cover a larger class of NUM problems.

4.6 Conclusions

In this chapter we consider a specific instance of the NUM problem where the amount of the network resources is unknown and the private congestion signals of the users cannot be used to directly achieve the optimal rate allocation. The congestion signals, however, can be combined to infer the amount of available resources. The main contribution of the chapter is a distributed method that infers the available network resource and maximizes the overall utility using an optimal learning formulation. As shown by the conducted experiments, the optimal learning approach is fundamental to achieve satisfying performance, which cannot be attained using a naïve and greedy strategy.

We believe that the analyzed problem and proposed framework, though being in its early stages, could be of great interest for the future computer networks. For instance, in the Internet the challenges and requirements of the platforms using the network evolve faster than the underlying infrastructure. In this scenario if the new applications want to optimize the data transmission they might have to coordinate the users using only information from the endpoints, since these are the network parts that are accessible and can be updated more easily. The new applications might then need to infer what are the actual global resources available. Though being extremely challenging and complex, building an overlay allocation method that infers the available resources and adapts to different network conditions, might be the only viable solution if the lower layers, and the infrastructure of the communication network, cannot be changed.

As final remark, we would like to point out that the framework developed in this chapter can go beyond communication networks. Potentially, every network systems which involves the inference of specific link parameters from end-to-end route measurements rather then per link measurements, could ideally leverage the basic framework developed in this chapter.

5 Conclusions

In this thesis we have studied several problems that arise in the context of Internet communication. In particular we focused on two specific challenges: the first challenge is related to the existence of strict timing requirements for data transmission in real-time Internet applications, whereas the second one concerns the ability to maximize the communication utility in unknown network scenarios.

The first problem that we considered corresponds to designing a rate allocation method for a multiparty videoconferencing system. The main challenge stems from the fact that, as the videoconference conditions might change abruptly, it is important for the users' sending rates to adapt as quickly as possible to the new conditions. Most of the rate allocation systems adopt iterative distributed solving methods that do not lead to adaptation methods that are sufficiently fast for the considered problem. We thus leverage the structure of the optimization problem and design a fast non-iterative solving method that is able to find an approximate solution in a distributed way. The proposed algorithm is then completed by a second iterative method that refines the rate allocation in order to further improve the videoconference utility. One key aspect of the considered approach is that the rate allocation is done on top of the congestion control algorithm used by the users, which leads to an overlay rate allocation method. Such a design approach allows us to neglect many problems that arise in the context of congestion control for real-time communication.

The design of a congestion control algorithm for real-time Internet communication specifically corresponds to the second problem considered in this thesis. The main challenge is to design congestion control methods that achieve a large sending rate while limiting the experienced transmission delay. This goal is extremely hard to reach as the Internet is shared with other applications that might not have any timing constraints, e.g., TCP traffic. TCP traffic usually tends to build up large queues in the inner network nodes making the delay uncontrollable. In such conditions, the ideal real-time congestion control algorithm should understand that the delay cannot be reduced and to not react to delay

variations in order to compete fairly with the TCP traffic. To address this problem we develop an adaptive controller that iteratively selects the delay sensitivity of an underlying delay-based congestion control algorithm in order to achieve the best trade-off between the sending rate and the experienced delay. As, at each step, the adaptive controller collects a reward (i.e., the communication utility) and observes new information about the network, the problem represents a particular bandit problem instance. We then adopt an optimal learning method to solve the bandit problem and optimize the communication utility in the long term. The experiments prove the validity of the proposed approach compared to other methods representing state-of-the-art designs.

Finally, we analyze the problem of the rate allocation in unknown network settings. The ultimate goal here is to design a distributed system that coordinates a set of users in order to achieve the optimal rate allocation when the amount of available resources is not known. We assume that the private congestion signals that are observed by the individual users cannot be used to steer the transmission rates to the optimal allocation, as opposed to what happens usually in congestion control problems. Such scenario occurs for instance when we want to allocate the rates at the application level without having access to the used congestion control. Though the individual feedbacks cannot be used locally, we assume that, by combining them, we can infer the overall network congestion. We define then a probabilistic model that allows us to perform inference on the network resources by merging the individual congestion signals observed by the users. We then use the belief on the network resources in order to run an overlay rate allocation system using classical NUM solving methods. We formulate the problem as an optimal learning problem in order to speed up the inference procedure and quickly maximize the utility provided to the users. From the experimental results, we observe that, adopting an optimal learning approach is essential to infer correctly the amount of available resources. In fact, a baseline solution employing a greedy policy and a passive learning method achieves extremely poor system performance, in most of the tested cases.

Overall this thesis proposes different solutions to a collection of Network Utility Maximization problems. The main outcome of this thesis is twofold: *i*) overlay methods that manage the rate allocation at the application layer offer suitable solutions in complex scenarios. In fact, as more complex operations can be implemented at the application layer, there is the possibility to develop more advanced and smart decision making algorithms. *ii*) When designing an iterative decision making process in environments affected by uncertainty, to model properly, and be able to leverage, the relation between the actions taken and the uncertainty improvement of the unknown parameters, is fundamental to boost the system performance.

A Expectation Propagation Implementation for Network Response Inference

We discuss here in more detail the method used to perform inference on the parameters of the network response. We first extend the graphical model of Fig. 3.4 by including some latent variables to the probabilistic model. These latent variables allow for simpler computations when running the inference method and thus reduce the overall algorithm complexity. We then consider the specific computations carried out in the execution of the EP algorithm. We focus exclusively on the rate network response inference, as the two probabilistic models, network rate response and network delay response, are basically the same. Moreover, as the EP algorithm operates by exchanging local messages, we can analyze separately the variable nodes of the model in Fig. 3.4 related to the different time steps t. As we focus exclusively on the network rate response for a single time step t, we can lighten the notation by dropping the x and t subscript from the different symbols involved in the computations, e.g., the parameter $\boldsymbol{\theta}_x^t$ is simply denoted here as $\boldsymbol{\theta}$.

A.1 Latent Variable Model

The posterior belief on the parameters $\boldsymbol{\theta}$ can be generally written as:

$$p(\boldsymbol{\theta}|x_{\text{obs}}, d_{\text{bl}}) = \frac{p(x_{\text{obs}}|\boldsymbol{\theta}, d_{\text{bl}})p(\boldsymbol{\theta})}{p(x_{\text{obs}}, d_{\text{bl}})}. \tag{A.1}$$

The EP algorithm is based on the minimization of the KL divergence between the true posterior distribution and a second distribution belonging to a fixed family. In order to perform this minimization it is necessary to compute the moments of the true posterior distribution. When the probability distribution has no simple expression the only viable solutions to achieve this task are numerical integration methods or sampling methods. These methods, however, are rather expensive in terms of computations and might require a long time before convergence. In our case the likelihood function $p(x_{\text{obs}}|\boldsymbol{\theta}, d_{\text{bl}})$ is not easy to deal with, this is due to the non-linear interaction between the baseline delay d_{bl} and the parameters $\boldsymbol{\theta}$. One possible way to reduce the required computations consists

Appendix A. Expectation Propagation Implementation for Network Response Inference

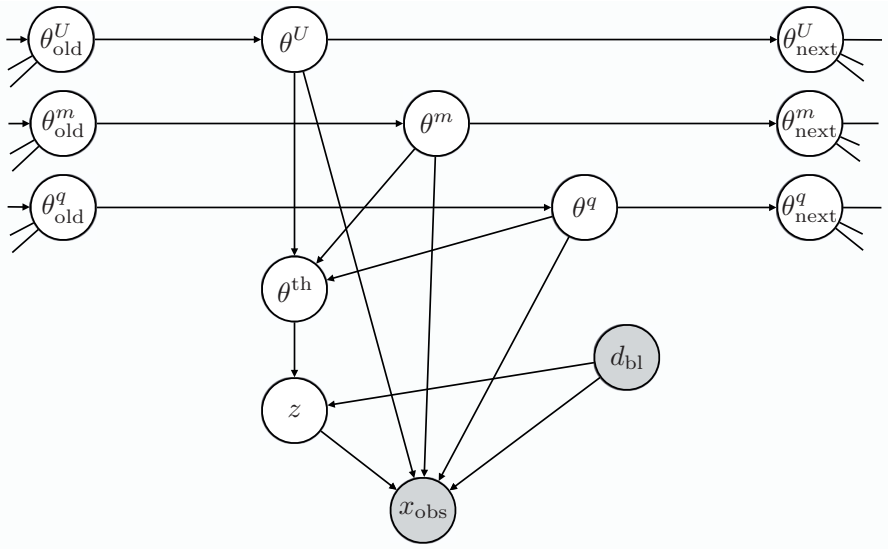


Figure A.1 – Graphical model for the rate network response with latent variables. The parameters $\boldsymbol{\theta}_{\text{old}}$ and $\boldsymbol{\theta}_{\text{next}}$ represent the previous and future network response, respectively.

in introducing latent variables to the model. Such an approach increases the number of variable nodes to the graphical model, but it also simplifies the relation among them, leading to a significant reduction of the overall algorithm complexity.

Considering our model, we notice that if we knew, whether the value of d_{bl} falls in the in the linear increase region or in the plateau region, then the inference would become much simpler, as we could completely decouple the different regions of the network response. To achieve this decoupling goal we need to identify the baseline delay that delimitates the transition between the increasing and the plateau region. We denote this point as θ^{th}, its value can easily be found using Eq. (3.11):

$$\theta^U = \theta^m \theta^{\text{th}} + \theta^q \qquad \rightarrow \qquad \theta^{\text{th}} = \frac{\theta^U - \theta^q}{\theta^m}. \tag{A.2}$$

Using the new parameter θ^{th} we define a latent binary variable z that is equal to one when d_{bl} is larger than θ^{th} and zero otherwise. We then use this binary variable to distinguish between the two regions simplifying the computations of the EP algorithm. The new probabilistic model including the latent variables is depicted in Fig. A.1. Although it might seem that the model has become more complicated, as the number of variable nodes has grown, the probability distribution relating the different variables are now much simpler. We list now the different relations.

- $\theta^{\text{th}} \mid \theta^U, \theta^q, \theta^m$ - The parameter θ^{th} basically represents redundant information as, according to the rate network response model of Eq. (A.2), when θ^U, θ^q, and θ^m are fixed the value of θ^{th} is deterministic. We can model this probability using a

Dirac delta function:

$$p(\theta^{\text{th}}|\theta^U, \theta^q, \theta^m) = \delta\left(\theta^U - (\theta^m\theta^{\text{th}} + \theta^q)\right). \tag{A.3}$$

Basically such distribution tells us that given θ^U, θ^q, and θ^m, there is only one possible value for θ^{th}.

- $z \mid \theta^{\text{th}}, d_{\text{bl}}$ - The latent binary variable $z \in \{0, 1\}$ simply tells us if the current baseline delay falls in the plateau region or not; we associate $z = 1$ with the event $d_{\text{bl}} > \theta^{\text{th}}$ and $z = 0$ with the complementary event. The probability is simply defined as:

$$p(z|\theta^{\text{th}}, d_{\text{bl}}) = \begin{cases} z & \text{if} \quad d_{\text{bl}} > \theta^{\text{th}} \\ 1 - z & \text{otherwise} \end{cases} \tag{A.4}$$

- $x_{\text{obs}} \mid z, \theta^U, \theta^q, \theta^m, d_{\text{bl}}$ - The likelihood probability for the variable x_{obs} apparently depends still on all the parameters of the model but in this case thanks to the variable z, it has a much easier form. We can write the probability as:

$$p(x_{\text{obs}} \mid z, \theta^U, \theta^q, \theta^m, d_{\text{bl}}) = \left(p(x_{\text{obs}}|\theta^U)\right)^z \left(p(x_{\text{obs}}|\theta^q, \theta^m, d_{\text{bl}})\right)^{1-z}. \tag{A.5}$$

The variable z here acts as a gate variable. It selects the increasing linear model if $z = 0$ and the plateau region if $z = 1$. If we consider as described in Section 3.2 to have that the probability of observing x_{obs} can be described as a normal distribution centered on the piecewise linear network model, we have:

$$p(x_{\text{obs}}|\theta^U) = \varphi(\frac{x_{\text{obs}} - \theta^U}{\sigma}), \tag{A.6}$$

and

$$p(x_{\text{obs}}|\theta^q, \theta^m, d_{\text{bl}}) = \varphi(\frac{x_{\text{obs}} - (\theta^m d_{\text{bl}} + \theta^q)}{\sigma}), \tag{A.7}$$

where σ denotes the standard deviation of noise measurement, and φ corresponds to the probability density function of a standard normal random variable. Note that the two probabilities defined in Eq. (A.6) and Eq. (A.7) are linear with respect to the parameters, thus simplifying the posterior computation.

In this section we have defined an equivalent probabilistic model for the rate network response that uses latent variables. The advantage is that the operations required by the EP algorithm are much simpler for this new model compared to the original one. Finally, note that for the delay network response, the z variable is not a binary variable, as in that case it must be able to classify three regions: the initial plateau, the linear phase, and again the final plateau. The structure of the model is, however, analogous to the one

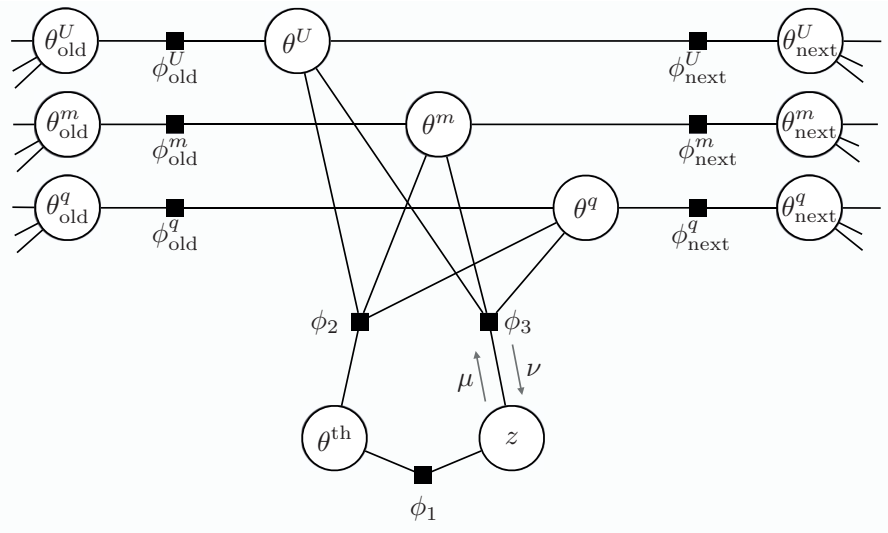


Figure A.2 – Factor graph associated with the Graphical of Fig. A.1.

presented in this section.

A.2 EP Algorithm and Factor Graph

The basic operations of the EP on a graphical model and a fully factorize target distribution consist in exchanging local messages among the different nodes. In order to proceed we first introduce the factor graph of the graphical model of Fig. A.1. The associated factor graph is shown in Fig. A.2, where we have removed the variable nodes associate to observed data, as they can be thought as being included in the connected factors.

We give now a brief overview of the EP algorithm on factor graphs [30, 31]. We denote by μ and ν the incoming and outgoing messages of the variable nodes, respectively. These messages are possibly unnormalized probability density (mass) functions defined over the domain of the variable node. The incoming messages can be thought as "opinions" associated to the factor node generating the message about the value of the variable node. The final belief on a variable node, given all the available information, can be obtained by combining (multiplying) all the different incoming messages. In the EP algorithm in order to have an approximate posterior distribution that belongs to a defined family, the incoming messages are projected onto the space of the considered target distribution of the variable. The projected incoming messages, denoted by $\tilde{\mu}$, can be computed in the following way. We consider here a generic variable node connected to S factor nodes, the goal is to compute the project incoming message for the factor node r. To do this we need to minimize the following KL divergence measure:

$$q = \arg\min_{q'} \ \text{KL}(\mu_r \prod_{s<S, s\neq r} \tilde{\mu}_s || q'), \tag{A.8}$$

where q represents the approximate posterior distribution which belongs to the considered target family. In order to solve Eq. (A.8) we simply need to match the sufficient statistics between the two distributions of the KL divergence. The projected message can then be computed according to:

$$\tilde{\mu}_r = \frac{q}{\prod_{s<S, s\neq r} \tilde{\mu}_s}. \tag{A.9}$$

The outgoing message from a variable node to a factor node, is formed by multiplying all the projected incoming messages of all the other factor nodes connected to the variable node. The EP algorithm is an iterative algorithm which must be executed multiple times in order for the incoming messages of the different nodes to converge. In order to have a very efficient implementation of the EP algorithm it is important to be able to compute efficiently Eq. (A.8). Ideally we would like to solve the equation in a closed form in order to avoid numerical integration.

We now describe the required computations associated to each factor node of our model. Recall that the target distribution for the parameters θ^U, θ^m, θ^q, and θ^{th} is a univariate normal distribution, whereas for the latent binary variable z, we use the Bernoulli distribution. We now consider, one by one, the factor nodes composing the probabilistic model.

- $\phi_{\text{old}}^{\cdot}$ and $\phi_{\text{next}}^{\cdot}$ - These factors represent the propagation of the parameters beliefs from the past and future observations. As the outgoing messages from the parameters are normal distributions, and the transition probability (Eq. (3.25)) is also a normal distribution, then, the incoming messages from these factors are all already valid distributions. As a result, in this case, we do not need to solve Eq. (A.8) as $\mu = \tilde{\mu}$, and we simply include the incoming message in the approximate posterior q.

ϕ_1 - This factor is connected to two variables z and θ^{th}, and it resembles the probability that the baseline delay d_{bl} falls or not in the plateau region. In order to compute the incoming message for the z node we should compute the probability that, according to the outgoing message from θ^{th}, the current d_{bl} belongs to the plateau region. As the outgoing message from the θ^{th} node is a normal distribution, we can simply compute the incoming z message using the cumulative density function of the normal distribution.The incoming message for the variable θ^{th} instead, corresponds to a step function plus a constant. Also in this case, by using the properties of the truncated normal distributions, we are able to recover the sufficient statistics and solve Eq. (A.8) without computing any numerical integration.

- ϕ_2 - This factor represents the relation that exists between the parameters of the network response. As using four parameters represents an overdefined model, we used a Dirac delta function to model the relation among the parameters values, see Eq (A.3). Solving Eq. (A.8) for any of the variable nodes, results into computing a

tridimensional integral. In order to avoid expensive numerical integration procedure we use a rough but effective approximation. The key intuition is that when the mass of the parameters beliefs is located away from zero, the lognormal distribution approximates accurately enough a normal one. Moreover, the lognormal distribution is stable under multiplication and division. In order to obtain an approximate expression for the incoming messages, we basically transform normal distributions to lognormal ones, when two variables need to be multiplied/divided in Eq. (A.2), mapping then the resulting distribution back to a normal one. By doing this we are able to obtain approximate incoming messages from the ϕ_2 factor to the different parameters avoiding numerical integration.

- ϕ_3 This last factor is the one that is associated to the observation x_{obs}. In this case the z variable is fundamental to simplify the computations. To understand this let us consider the incoming message for the parameter θ^U. We can think about this incoming message as the superposition of the beliefs: one corresponding to $z = 0$ and the other to $z = 1$. If $z = 1$ it means that the active region is the plateau one, and thus the likelihood corresponds to the normal distribution of Eq. (A.7). In the case $z = 0$ the parameter θ^U does not play any role in the likelihood, thus the observation is completely uninformative and the message corresponds to a constant value. As a result the incoming message is a normal distribution plus a constant term. Having this simple form, we are then able to solve analytically Eq. (A.8). The same considerations apply for the parameters θ^m and θ^q.

As already said the EP algorithm is an iterative method. Thus we need to iterate over the factor nodes of the factor graph in order to progressively refine the incoming messages and compute the approximate posterior. We recompute the incoming messages of the different nodes for $I^{\text{EP}}_{\text{MAX}} = 25T$ iterations, where T denotes the number of observation in the dataset. In our case we limit the dataset to have at most 10 observations. If new observations are collected, the older ones are then excluded from the dataset. The output of the EP algorithm is a set of univariate normal distributions q, one for each variable node, which resembles the posterior distribution. Note that the approximate distribution q of each variable node is given by the product of the projected incoming messages $\tilde{\mu}$.

B Expectation Propagation Implementation for Network Resource Inference

The implemented EP algorithm on the considered factor graph of Fig. B.1 is reported in Alg. 2. We denote by μ and ν the incoming and outgoing messages respectively, of the different variable nodes. These messages are possibly unnormalized probability density (mass) functions defined over the domain of the variable node. Their subscripts denote the variable node and factor node of the message[1]. Since the factor graph, depicted in Fig. B.1, has loops, the incoming messages of the variable nodes, z_m^t and c_m, need to be iteratively refined (line 1) till they converge to their final value. All the messages are initialized using non-informative distribution (lognormal distribution with infinite variance for the c_m variables and uniform distribution for the binary variables z_m^t). At each iteration the algorithm updates the incoming messages $\mu_{c_m z_m^t}$ parallelizing the operations over M different processes. Each process m computes the incoming message to the z_m^t from the c_m using the messages from the $t' \neq t$ observations and the prior information (line 4). Then we iterate over the individual factors of $\Psi_{\mathbf{v}^t}()$ to update the incoming messages of the latent variables. Each process m computes the outgoing message from the z_m^t node to the factor node $\psi_n^1()$ associated to any observation n with $v_n^t = 1$ (lines 5-7). As next step, each process collects all the outgoing messages $\nu_{z_l^t \psi_n^1}$ for $l \neq m$ and $a_{ln} = 1$ and computes the incoming message $\mu_{z_m^t \psi_n^1}$ (lines 8-9):

$$\mu_{z_m^t \psi_n^1}(r_m) = \sum_{\substack{l \neq m \\ r_l \in \{0,1\}}} \psi_n^1(\mathbf{r}) \prod_{\substack{l \neq m \\ a_{ln}=1}} \nu_{z_l^t \psi_n^1}(r_l), \tag{B.1}$$

where we have emphasized the fact that the messages μ and ν are actually functions. A naïve computation of this quantity is rather expensive in terms of required operations. However, since $\psi_n^1(\mathbf{r})$ is equal to one when at least one r_m variable employed by user n is equal to one and zero otherwise, we can compute $\psi_n^1(\mathbf{r})$ as the complementary event of

[1] To make the notation more clear the messages to and from the factor node $p(z_m^t | c_m, y_m)$ are indexed by the variable nodes c_m and z_m^t.

Appendix B. Expectation Propagation Implementation for Network Resource Inference

Algorithm 2 Implementation of the EP algorithm.

for $i < I_{\text{MAX}}^{\text{EP}}$ **do**
  for each observation to process t **do**
    for each link m **do**
      update $\mu_{z_m^t c_m}$
    for each n with $v_n^t = 1$ **do**
      for each link m with $a_{mn} = 1$ **do**
        compute $\nu_{z_m^t v_n^t}$
        collect $\nu_{z_l^t v_n^t}\ \forall l \neq m,\ a_{ln} = 1$
        update $\mu_{z_m^t v_n^t}$
      compute $\mu_{c_m z_m^t}$
      $q = \arg\min_{q'} \ \text{KL}(\mu_{c_m z_m^t} \prod_{t' \neq t} \tilde{\mu}_{c_m z_m^{t'}} || q')$
      $\tilde{\mu}_{c_m z_m^t} = \frac{q}{\prod_{t' \neq t} \tilde{\mu}_{c_m z_m^{t'}}}$

having all the r_m variables equal to zero. We therefore obtain:

$$\begin{aligned} \mu_{z_m^t \psi_n^1}(0) &= \prod_{\substack{l \neq m \\ a_{ln}=1}} (\nu_{z_l^t \psi_n^1}(0) + \nu_{z_l^t \psi_n^1}(1)) - \prod_{\substack{l \neq m \\ a_{ln}=1}} \nu_{z_l^t \psi_n^1}(0) \\ \mu_{z_m^t \psi_n^1}(1) &= \prod_{\substack{l \neq m \\ a_{ln}=1}} (\nu_{z_l^t \psi_n^1}(0) + \nu_{z_l^t \psi_n^1}(1)) \end{aligned} \tag{B.2}$$

if we consider to normalize the outgoing messages $\nu_{v_l^t z_n^t}$ we obtain the simpler form:

$$\begin{aligned} \mu_{z_m^t \psi_n^1}(0) &= 1 - \prod_{\substack{l \neq m \\ a_{ln}=1}} \nu_{z_l^t \psi_n^1}(0) \\ \mu_{z_m^t \psi_n^1}(1) &= 1. \end{aligned} \tag{B.3}$$

This is a simple operation that consists in the product of the outgoing messages associated with the event $z_m^t = 0$ for the observation $v_n^t = 1$. The factors $\psi_m^0()$ can easily be handled locally by each link process. Once all the incoming messages from all $\mathbf{v}^t$ factors are updated, the incoming message $\mu_{c_m z_m}^t$ can be computed. The EP algorithm then aims at minimizing the following divergence in order to obtain the belief for c_m:

$$q = \arg\min_{q'} \ \text{KL}(\mu_{c_m z_m^t} \prod_{t' \neq t} \tilde{\mu}_{c_m z_m^{t'}} || q'), \tag{B.4}$$

where $q(c_m)$ represents the lognormal distribution characterizing the belief on c_m. In order to minimize Eq. (B.4) we simply need to match the sufficient statistics between the two distributions of the KL divergence. In order to match the sufficient statistics we need to perform integration over the first argument of the KL divergence of Eq. (B.4). As

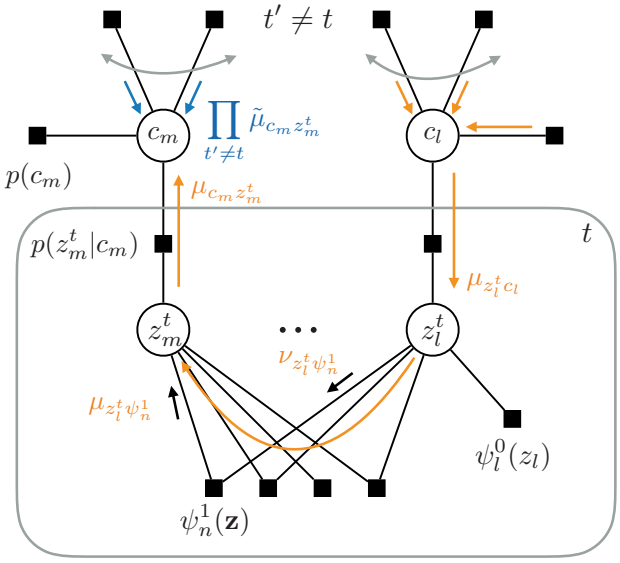


Figure B.1 – Factor graph of one single observation t showing the messages between the variables nodes exchanges during the EP algorithm execution.

the integral is a simple one dimensional integral of a well behaved function any common numerical integration method can be used to achieve this task. The message $\tilde{\mu}_{c_m z_m^t}$ which corresponds to the lognormal approximation of $\mu_{c_m z_m^t}$ is then set to:

$$\tilde{\mu}_{c_m z_m^t} = \frac{q}{\prod_{t' \neq t} \tilde{\mu}_{c_m z_m^{t'}}}. \tag{B.5}$$

Note that all the terms in Eq. (B.5) are lognormal distributions, which belong to the exponential family, therefore the above operation simply consists in algebraical manipulation of the distribution parameters. The EP algorithm allows also to add a damping factor (ϵ_{EP}) in the update of the incoming messages μ (and $\tilde{\mu}$) see [31]. This modification prevents the messages from changing drastically with respect to their previous value; the equilibrium points do not change but it can improve the convergence of the algorithm, in our implementation we used $\epsilon_{\text{EP}} = 0.5$ The above steps are executed for each datapoint in the dataset and for $I_{\text{MAX}}^{\text{EP}} = 5$ iterations. At this point, the execution stops and the lognormal distribution $q(\mathbf{c})$ defined by the most updated values of the messages $\tilde{\mu}_{c_m z_m^t}$ approximates the true posterior $p(\mathbf{c}|\{\mathcal{D}^t\})$.

C Detailed Algorithm Implementation for the Distributed Network Resource Inference

The detailed complete set of operations required to run our algorithm is reported in Alg. 3. The overall system is composed by two groups of processes, namely the N users and the M link processes (each responsible for an individual network link). As described in the main Chapter 4 lines (3-8) correspond to the execution of the classical NUM algorithm. The next operations are instead specific to our system.

The next step corresponds to updating the dataset $\{\mathcal{D}^t\}$ with observations $\{\mathbf{y}^t, \mathbf{v}^t\}$. Though our design optimizes the choice of $\hat{\mathbf{c}}^t$ by taking into account the expected posterior variance when observing exclusively the point $\{\mathbf{y}^t, \mathbf{v}^t\}$ after the inner loop has converged, we could as well use the information collected during the inner loop dynamics. However, points that share similar link rates $\mathbf{y}^t$ provide redundant information to the inference method, moreover a higher number of points increases also the computation cost of the running the EP algorithm. In order to have a trade-off between the two cases we adopt a simple heuristic that consists in adding one observations to the dataset every T_S time steps (lines 10-12). Moreover in order to limit the amount of computations of the EP algorithm we limit the maximum size of the dataset to $S_{\mathcal{D}}$. When the maximum size is reached we discard the older points to make space for the new ones. The most recent values of the approximated incoming messages $\tilde{\mu}_{c_m z_m^t}$ for the discarded points are however embedded in the prior belief. When the inner loop has converged (line 13) we execute the distributed EP algorithm, see Appendix B, on our graphical model using the current dataset and obtain the new fitted distribution $q^t(\mathbf{c})$ (line 14).

At this point we need to compute the new value of $\hat{\mathbf{c}}$ to use in the next iterations. In order to set the value of $\hat{\mathbf{c}}$ according to the mean field approximation, we need a rule to assign each link either to class A or to class B. Our solution consists in assigning to class A the links with the larger variance. According to our model, the links of class A are actually the ones expected to reduce more significantly their belief variance. Before assigning the links to the different classes it is however beneficial to modify the current variances σ_m^2 for the links that are actually underutilized as their values do not affect

Appendix C. Detailed Algorithm Implementation for the Distributed Network Resource Inference

Algorithm 3 Complete algorithm

1: **loop**
2: **for** each user n **do**
3: collect $\lambda_m^t \forall m \in n$
4: compute and apply rate $x^t(\mathbf{a}_n^\mathsf{T}\boldsymbol{\lambda}^t)$
5: Forward v_n^t and $x_n^t \forall m \in n$
6: **for** each link m **do**
7: $y_m^t = \mathbf{a}_m^\mathsf{T}\mathbf{x}^t$
8: ${\lambda_m}^{t+1} = \max(0, {\lambda_m}^t + \epsilon(y_m^t - \hat{c}_m^t))$
9: $\hat{c}_m^{t+1} = \hat{c}_m^t$
10: **if** modulo$(t, T_S) = 0$ **then**
11: **for** each link m **do**
12: update dataset $\{\mathcal{D}^t\}$ with point (y_m^t, v_n^t)
13: **if** $|\lambda_m^{t+1} - \lambda_m^t| \leq V_\lambda \ \forall m$ **then**
14: compute $q^t(\mathbf{c})$ using EP
15: **for** each link m **do**
16: **if** $\boldsymbol{\lambda}^t = 0$ **then**
17: $\boldsymbol{\sigma^2} = 0$
18: Initialize $\hat{\mathbf{r}}^i = 0$, $\mathbf{r}^i = 0$, $i = 0$
19: **while** $|h_m^i - h_m^{i-1}| \leq V_h \ \forall m$ **do**
20: **for** each link m **do**
21: $r_m^i = \underset{r' \in \{0,1\}}{\arg\max}\ r'(\sigma_m^2 - h_m^i)$
22: $\hat{r}_m^{i+1} = 1 - \mathbf{l}_m^\mathsf{T}\hat{\mathbf{r}}^i + \Delta r_m^i$
23: $h_m^{i+1} = \left(1 - \mathbf{l}_m^\mathsf{T}\mathbf{h}^i + \delta_h(\hat{r}_m^i - \alpha)\right)^+$
24: $p_m^{\text{class}} = r_m^i p_{\text{A1}} + (1 - r_m^i)p_{\text{B1}}$
25: **for** each link m **do**
26: find $\hat{c}_m \ : \ p(z' = 1|\hat{c}_m) = p_m^{\text{class}}$
27: $\hat{c}_m^{t+1} = \hat{c}_m$

the rate allocation. We set the variance σ_m^2 of these links to zero, basically forcing them to belong to class B (line 15-17). We design a distributed algorithm for the selection of the αM largest links with the largest variance σ_m^2 among the M link processes. We can formulate the problem as an optimization problem:

$$\begin{aligned} \underset{\mathbf{r}}{\text{maximize}} \quad & \mathbf{r}^\mathsf{T}\boldsymbol{\sigma}^2 \\ \text{subject to} \quad & \mathbf{1}^\mathsf{T}\mathbf{r} \leq \alpha M \\ & \mathbf{r} \in \{0,1\}^M, \end{aligned} \tag{C.1}$$

where $\mathbf{r}$ is a binary vector representing the class membership of link m: if $r_m = 0$ then the link m belongs to class B and to class A otherwise. For simplicity we assume here that $\alpha M \in \mathbb{N}$ and that the solution to problem in Eq. (C.1) is unique. If these conditions hold we can solve exactly the above problem using a dual method, which allows to decompose

the solution method among the M processes. We consider the Lagrange relaxation of Eq. (C.1):

$$\underset{\mathbf{r}\in\{0,1\}^M}{\text{maximize}}\ \mathbf{r}^\mathsf{T}\boldsymbol{\sigma}^2 - h\left(\mathbf{1}^\mathsf{T}\mathbf{r} - \alpha M\right), \tag{C.2}$$

where h denotes the Lagrange multiplier of the inequality constraint. Given the value of the dual variable h each process can compute independently the value of r_m. This operation is extremely simple, basically $r_m = 1$ if $\sigma_m^2 > h$ and zero otherwise. In order to solve the dual problem the variable h can be iteratively updated using a gradient descent method:

$$h' = \left(h + \delta_h\left(\hat{r} - \alpha\right)\right)^+, \tag{C.3}$$

where $\hat{r}$ represents the mean value of the elements of $\mathbf{r}$ and δ_h is a positive parameter that controls the step length. Unfortunately this update operation requires a central entity which knows the value of $\hat{r}$ and forwards the dual variable h to all the M link processes. Ideally we prefer to have an algorithm that is completely distributed, and does not require a central entity, therefore we modify the iteration in the following way. We first create M copies of the dual variable h and M estimates of the mean value $\hat{r}$, one for each link process. Each process then, iteratively executes Eq. (C.3) using its local copies and runs in parallel a consensus algorithm on both variables [44]. In our case we need to solve a dynamic average consensus problem, which means that the nodes have to agree on the average value of a M dimensional input signal that is varying over time (in our case the M-dimensional signals are the dual variables $\mathbf{h}$ and the estimates $\hat{\mathbf{r}}$). In order to run a consensus algorithm among the M link processes we define a connected undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the nodes $\mathcal{V}$ correspond to the M processes and the edges $\mathcal{E}$ denote the pairs of processes that exchange messages in the class assignment method. We define by $\mathbf{L}$ the normalized Laplacian matrix of the graph $\mathcal{G}$. Many works focused on how to optimize communication among the agents in order to speed up the convergence of average consensus algorithms (see [49]) in this work, however, we simply assume that the M links processes define a random connected communication network.

The iterative steps of the class assignment algorithm are the following:

$$\begin{cases} \mathbf{r}^i = \underset{\mathbf{r}'\in\{0,1\}^M}{\arg\max}\ \mathbf{r}'^\mathsf{T}(\boldsymbol{\sigma}^2 - \mathbf{h}^i) & \text{(C.4a)} \\ \begin{bmatrix} \hat{\mathbf{r}}^{i+1} \\ \mathbf{h}^{i+1} \end{bmatrix} = \left[\begin{array}{c|c} \mathbf{I}-\mathbf{L} & \mathbf{0} \\ \hline \delta_h^i\mathbf{I} & \mathbf{I}-\mathbf{L} \end{array}\right] \begin{bmatrix} \hat{\mathbf{r}}^i \\ \mathbf{h}^i \end{bmatrix} - \left[\begin{array}{c|c} \mathbf{I} & \mathbf{0} \\ \hline \mathbf{0} & -\delta_h^i\mathbf{1} \end{array}\right] \left[\Delta\mathbf{r}^i | \alpha\right] & \text{(C.4b)} \\ \mathbf{h}^{i+1} = (\mathbf{h}^{i+1})^+ & \text{(C.4c)} \end{cases}$$

where $\Delta\mathbf{r} = \mathbf{r}^i - \mathbf{r}^{i-1}$ denotes the variation of vector $\mathbf{r}$ with respect to previous iteration, and $\mathbf{I}$ denotes the identity matrix. Each iteration is composed by three steps. First

each process selects the optimal value of r_m according to the local copy of the dual variable h_m^i. The variables $\hat{\mathbf{r}}$ are then updated according to the following dynamics $\hat{\mathbf{r}}^{i+1} = (\mathbf{I} - \mathbf{L})\hat{\mathbf{r}}^i + \Delta\mathbf{r}^i$. Considering null initial conditions, we have that, under a steady input $\mathbf{r}$ the vector $\hat{\mathbf{r}}$ converges to $\mathbf{1}\sum_m r_m/M$ (similar dynamic consensus methods have been proposed in [50, 51]). Note that, at each step ,each link process has to communicate with the other processes that correspond to its neighbors on the graph $\mathcal{G}$ in order to collect the values of $\hat{r}_l^i$. At the same time, the dual variables $\mathbf{h}$ evolve according to $\mathbf{h}^{i+1} = (\mathbf{I} - \mathbf{L})\mathbf{h}^i + \delta(\hat{\mathbf{r}}^i - \mathbf{I}\alpha)$. The operations are basically the same as before, except that each link process integrates the difference between the target ratio α and the local estimate of the mean value of $\mathbf{r}$. The use of the Laplacian matrix $\mathbf{L}$ in this case assures that possible discrepancies among the entries of the vector $\mathbf{h}$ are damped, making all the link processes to agree on the same value of the dual variable. Finally, the variables $\mathbf{h}$ are constrained to be non-negative to be consistent with to Eq. (C.3). Executing iteratively Eq. (C.4) the link processes agree on the class assignment, and the links that have a largest uncertainty are assigned to class A (lines 18-24). At this point each link process computes the value of $\hat{c}_m$ that makes $p(z_m = 1|\hat{c}_m)$ equal to the probability of the link class (lines 25-26).

Finally, all the parameters used in our simulations for Alg. 3 are given in Table C.1.

Table C.1 – Parameters settings

Parameter	Value	Parameter	Value
ϵ_m	$0.5/\hat{c}_m$	T_S	25
V_λ	10^{-3}	V_h	10^{-3}
δ_h^i	$0.1/\sqrt{i}$	γ	$0.98 - 0.995$
$S_\mathcal{D}$	25		

Bibliography

[1] C. V. N. Index, "The zettabyte era–trends and analysis," *Cisco white paper*, 2017.

[2] F. P. Kelly, A. K. Maulloo, and D. K. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research society*, vol. 49, no. 3, 1998.

[3] M. Chiang, S. H. Low, A. R. Calderbank, and J. C. Doyle, "Layering as optimization decomposition: A mathematical theory of network architectures," *Proceedings of the IEEE*, vol. 95, no. 1, 2007.

[4] K. Brunnström, S. A. Beker, K. De Moor, A. Dooms, S. Egger, M.-N. Garcia, T. Hossfeld, S. Jumisko-Pyykkö, C. Keimel, M.-C. Larabi, *et al.*, "Qualinet white paper on definitions of quality of experience," 2013.

[5] D. P. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 8, 2006.

[6] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[7] "Cisco video and telepresence architecture design guide," *Cisco*, 2012.

[8] M. Chen, M. Ponec, S. Sengupta, J. Li, and P. A. Chou, "Utility maximization in peer-to-peer systems," in *SIGMETRICS Performance Evaluation Review*, ACM, 2008.

[9] M. Ponec, S. Sengupta, M. Chen, J. Li, P. Chou, *et al.*, "Multi-rate peer-to-peer video conferencing: A distributed approach using scalable coding," in *International Conference on Multimedia and Expo*, IEEE, 2009.

[10] X. Chen, M. Chen, B. Li, Y. Zhao, Y. Wu, and J. Li, "Celerity: a low-delay multi-party conferencing solution," in *International Conference on Multimedia*, ACM, 2011.

[11] X. Zhu, R. Pan, M. Ramalho, S. de la Cruz, C. Ganzhorn, P. Jones, and S. D'Aronco, "NADA: A unified congestion control scheme for real-time media." IETF, 2015.

[12] M. Welzl, S. Islam, and S. Gjessing, "Coupled congestion control for RTP media." Internet-Draft, 2015.

[13] "RTP Media Congestion Avoidance Techniques (RMCAT)." IETF Working Group, 2012.

[14] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the h. 264/avc standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 9, 2007.

[15] S. Burer and A. N. Letchford, "Non-convex mixed-integer nonlinear programming: a survey," *Surveys in Operations Research and Management Science*, vol. 17, no. 2, 2012.

[16] Y. R. Yang, M. S. Kim, and S. S. Lam, "Optimal partitioning of multicast receivers," in *International Conference on Network Protocols*, IEEE, 2000.

[17] J. Liu, B. Li, Y. T. Hou, and I. Chlamtac, "On optimal layering and bandwidth allocation for multisession video broadcasting," *IEEE Transactions on Wireless Communications*, vol. 3, no. 2, 2004.

[18] A. Schrijver, *Theory of Linear and Integer Programming*. John Wiley & Sons, Inc., 1986.

[19] K. Sakhnov, E. Verteletskaya, and B. Simak, "Dynamical energy-based speech/silence detector for speech enhancement applications," in *World Congress on Engineering*, 2009.

[20] D. E. Comer, *Internetworking with TCP/IP*, vol. 1. Pearson, 2006.

[21] G. F. Riley and T. R. Henderson, "The ns-3 Network Simulator Modeling and Tools for Network Simulation," in *Modeling and Tools for Network Simulation*, Springer Berlin Heidelberg, 2010.

[22] J. Li, P. A. Chou, and C. Zhang, "Mutualcast: An efficient mechanism for one-to-many content distribution," in *SIGCOMM Asia Workshop*, ACM, 2005.

[23] E. Kurdoglu, Y. Liu, and Y. Wang, "Dealing with user heterogeneity in p2p multi-party video conferencing: Layered distribution versus partitioned simulcast," *IEEE Transactions on Multimedia*, vol. 18, no. 1, 2016.

[24] Y. Xu, C. Yu, J. Li, and Y. Liu, "Video telephony for end-consumers: Measurement study of google+, ichat, and skype," in *Conference on Internet Measurement Conference*, ACM, 2012.

[25] A. Eleftheriadis, "SVC and video communications," *White Paper, Vydio*, 2011.

[26] B. Grozev, L. Marinov, V. Singh, and E. Ivov, "Last n: relevance-based selectivity for forwarding video in multimedia conferences," in *25th Workshop on Network and Operating Systems Support for Digital Audio and Video*, ACM, 2015.

[27] W. B. Powell and I. O. Ryzhov, "Optimal learning," 2012.

[28] S. D'Aronco, L. Toni, S. Mena, X. Zhu, and P. Frossard, "Improved utility-based congestion control for delay-constrained communication," *IEEE/ACM Transactions on Networking*, vol. 25, no. 1, 2017.

[29] C. M. Bishop, *Pattern recognition and machine learning*, ch. 8. springer, 2006.

[30] T. Minka, "Expectation propagation for approximate bayesian inference," in *Conference in Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers Inc., 2001.

[31] T. Minka, "Divergence measures and message passing," tech. rep., Microsoft Research, 2005.

[32] S. Nadarajah and S. Kotz, "Exact distribution of the max/min of two gaussian random variables," *IEEE Transactions on very large scale integration (VLSI) systems*, vol. 16, no. 2, 2008.

[33] I. Johansson and Z. Sarker, "Self-clocked rate adaptation for multimedia." IETF, 2015.

[34] M. Handley, S. Floyd, J. Padhyea, and J. Widmer, "TCP friendly rate control (TFRC): Protocol specification." IETF, 2003.

[35] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: motivation, architecture, algorithms, performance," *IEEE/ACM transactions on Networking*, vol. 14, no. 6, 2006.

[36] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, "Low extra delay background transport (LEDBAT)." IETF, 2012.

[37] Ł. Budzisz, R. Stanojević, A. Schlote, F. Baker, and R. Shorten, "On the fair coexistence of loss-and delay-based TCP," *IEEE/ACM Transactions on Networking*, vol. 19, no. 6, 2011.

[38] S. Holmer, H. Lundin, G. Carlucci, L. D. Cicco, and S. Mascolo, "A google congestion control algorithm for real-time communication." IETF, 2013.

[39] T. Stockhammer, "Dynamic adaptive streaming over http –: Standards and design principles," in *Conference on Multimedia Systems*, ACM, 2011.

[40] L. De Cicco and S. Mascolo, "An adaptive video streaming control system: Modeling, validation, and performance evaluation," *IEEE/ACM Transactions on Networking*, vol. 22, no. 2, 2014.

[41] S. Petrangeli, J. Famaey, M. Claeys, S. Latré, and F. De Turck, "Qoe-driven rate adaptation heuristic for fair adaptive video streaming," *ACM Transactions Multimedia Computing, Communications and Applications*, vol. 12, no. 2, 2015.

[42] S. D'Aronco, L. Toni, and P. Frossard, "Price-based controller for utility-aware HTTP adaptive streaming," *IEEE MultiMedia*, no. 99, 2017.

[43] K. J. Åström and B. Wittenmark, *Adaptive control*. Courier Corporation, 2013.

[44] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, 2007.

[45] M. Mehyar, D. Spanos, and S. H. Low, "Optimization flow control with estimation error," in *INFOCOM*, IEEE, 2004.

[46] J. Zhang, D. Zheng, and M. Chiang, "The impact of stochastic noisy feedback on distributed network utility maximization," *IEEE Transactions on Information Theory*, vol. 54, no. 2, 2008.

[47] M. Zargham, A. Ribeiro, and A. Jadbabaie, "Network optimization under uncertainty," in *Annual Conference on Decision and Control*, IEEE, 2012.

[48] Y. Su and M. Van Der Schaar, "Linearly coupled communication games," *IEEE Transactions on Communications*, vol. 59, no. 9, 2011.

[49] D. Jakovetic, J. Xavier, and J. M. Moura, "Weight optimization for consensus algorithms with correlated switching topology," *IEEE Transactions on Signal Processing*, vol. 58, no. 7, 2010.

[50] D. P. Spanos, R. Olfati-Saber, and R. M. Murray, "Dynamic consensus on mobile networks," in *IFAC world congress*, Prague Czech Republic, 2005.

[51] M. Zhu and S. Martínez, "Discrete-time dynamic average consensus," *Automatica*, vol. 46, no. 2, 2010.

Stefano D'Aronco

+41 78 633 57 49
email: stefano.daronco@epfl.ch / stefano.dar@outlook.com

RESEARCH INTERESTS

Adaptive Systems, Reinforcement Learning, Bayesian methods, Decision-making under uncertainty, Multi-agent Systems, Network Systems

EDUCATION

École Polytechnique Fédérale de Lausanne (EPFL) - Lausanne, Switzerland 2014-2018
PhD in Electrical Engineering.

- Dissertation Title: *Network Utility Maximization for Delay-Sensitive Applications in Unknown Communication Settings*
- Supervisor: Prof. Pascal Frossard

Università degli Studi di Udine - Udine, Italy 2010-2013
M.Sc. in Electronic Engineering - 110/110 cum laude

- Visiting Student **Technische Universiteit Delft** February 2012 - July 2012
- Dissertation Title: *Random Number Generation by Sampling a Radioactive Source*
- Supervisor: Prof. Riccardo Bernardini

Università degli Studi di Udine - Udine, Italy 2007-2010
B.Sc. in Electronic Engineering - 110/110 cum laude

- Dissertation Title: *Design and implementation of a LCR meter for impedance measurements on biological samples*
- Supervisor: Prof. Antonio Affanni

WORK EXPERIENCE

EPFL, Lausanne, Switzerland September 2013 - February 2014
Signal Processing Laboratory (LTS4)
Internship - Development of streaming algorithms for delay-sensitive Internet communication (in collaboration with Cisco Systems)

SCIENTIFIC SKILLS

Expertise: Networking, Convex Optimization, Bayesian inference, Probabilistic Graphical Models, Machine Learning, Multi-armed Bandit Problems, Control Theory, Internet Communication

Languages & Software: C/C++, Python, Matlab, Network Simulator (NS3)

ACADEMIC SERVICES

Reviewer: IEEE Transactions on Control of Network Systems, IEEE Multimedia, ELSEVIER Signal Processing: Image Communication

Teaching: Digital Signal Processing (EPFL), Supervised several student projects

SELECTED UNIVERSITY PROJECTS AND COURSES

Courses (EPFL): Unsupervised and Reinforcement Learning in Neural Networks, a Network Tour of Data Science, Optimal Control, Bayesian Computation

Project (Optimal Control): *Foresighted HTTP Adaptive Streaming for Video on Demand Service*
Design of a double horizon Model Predictive Controller to optimize the long-term video quality for HTTP adaptive streaming systems.

Project (Bayesian Computation): *Polytope Estimation Using Bayesian Inference*
Development of a method for computing an approximate posterior distribution of a polytope from labeled data.

PUBLICATIONS

Conference and Workshop Papers

- **S. D'Aronco**, and P. Frossard. *A Bayesian Bandit Approach to Adaptive Delay-based Congestion Control*. ACM Packet Video Workshop. 2018.
- **S. D'Aronco**, L. Toni, and P. Frossard. *Price-Based Controller for Quality-Fair HTTP Adaptive Streaming*. IEEE International Symposium Multimedia. 2016.
- F. Chiariotti, **S. D'Aronco**, L. Toni, and P. Frossard. *Online learning adaptation strategy for DASH clients*. ACM Multimedia Systems Conference. 2016.
- **S. D'Aronco**, S. Mena, and P. Frossard. *Distributed rate allocation in switch-based multiparty videoconference*. ACM Multimedia Systems Conference. 2016.

Journal and Magazine Papers

- **S. D'Aronco**, and P. Frossard. *Online Resource Inference in Network Utility Maximization Problems*. IEEE Transactions on Network Science and Engineering. 2018. (Accepted).
- **S. D'Aronco**, S. Mena, and P. Frossard. *Distributed Rate Allocation in Switch-Based Multiparty Videoconferencing System*. ACM Transactions on Multimedia Computing, Communications and Applications. 2017.
- **S. D'Aronco**, L. Toni, and P. Frossard. *Price-Based Controller for Utility-Aware HTTP Adaptive Streaming*. IEEE MultiMedia. 2017.
- **S. D'Aronco**, L. Toni, S. Mena, X. Zhu, and P. Frossard. *Improved Utility-Based Congestion Control for Delay-Constrained Communication*. IEEE/ACM Transactions on Networking. 2016.

Patents

- S. Mena, **S. D'Aronco**, and X. Zhu. *Switching Between Loss-Based and Delay-Based Mode for Real-Time Media Congestion Controllers*. US9584420. 2017.
- S. Mena, L. Toni, and **S. D'Aronco**. *Congestion control for media flows*. US9549016. 2017.

Other

- X. Zhu, R. Pan, M. Ramalho, S. Mena, P. Jones, J. Fu, and **S. D'Aronco**. *NADA: A Unified Congestion Control Scheme for Real-Time Media*. IETF Internet-Draft. 2015.

HONORS AND AWARDS

- Best paper award at IEEE International Symposium Multimedia 2016.
- Best paper award at ACM Multimedia Systems Conference 2016.

LANGUAGES

Italian: mother tongue
English: fluent
French: beginner

OTHER

Nationality: Italian