University of Padova

Academic year 2018-2019 (796th)

Department of Information Engineering (DEI)

Doctoral thesis in Information Engineering

Intelligence in 5G networks

Author: Federico Chiariotti

Supervisor: Andrea Zanella

To the people who were there when I needed them most: Giulia, Edoardo, Martina, Flaminia, Francesco

Abstract

Over the past decade, Artificial Intelligence (AI) has become an important part of our daily lives; however, its application to communication networks has been partial and unsystematic, with uncoordinated efforts that often conflict with each other. Providing a framework to integrate the existing studies and to actually build an intelligent network is a top research priority. In fact, one of the objectives of 5G is to manage all communications under a single overarching paradigm, and the staggering complexity of this task is beyond the scope of human-designed algorithms and control systems.

This thesis presents an overview of all the necessary components to integrate intelligence in this complex environment, with a user-centric perspective: network optimization should always have the end goal of improving the experience of the user. Each step is described with the aid of one or more case studies, involving various network functions and elements.

Starting from perception and prediction of the surrounding environment, the first core requirements of an intelligent system, this work gradually builds its way up to showing examples of fully autonomous network agents which learn from experience without any human intervention or pre-defined behavior, discussing the possible application of each aspect of intelligence in future networks.

Contents

Chapter 1

Introduction

Although Artificial Intelligence (AI) is already an important part of our daily lives, seamlessly performing tasks such as speech processing and image recognition without us even noticing, defining exactly what intelligence means is still a controversial question, particularly so when computers are involved: machines have already overcome humans' performance in well-defined tasks with limited and consistent environments, but replacing humans when creativity and adaptability are required is a more ambitious proposition. In order to solve this problem, we need a coherent definition of intelligence which can be applied to both biological and artificial agents, providing a framework for research on the theory and applications of AI.

In a 2007 review [1] of the use of the term in psychology, philosophy and AI research, Legg and Hutter came up with this definition:

> *Intelligence measures an agent's ability to achieve goals in a wide range of environments.*

Although the definition is still very broad, it can be formulated mathematically [2] and it is extremely useful in understanding the trends in the application of AI in communication networks and protocol design [3].

An obvious example is Self-Organized Networking (SON) [4], a set of statistical and learning techniques that is already applied in 4G cellular networks. SON's use cases are self-organization, self-optimization and self-healing: the network needs to be able to provide the best possible service under changing propagation and traffic demand conditions, while saving power and optimizing several of its parameters without human intervention. The similarity between SON's goals and the definition of intelligence above should be clear to the reader: a complex modern cellular network can forgo human intervention and optimize itself only by being intelligent.

While SON represents a first step towards intelligence in networks, the various techniques that fall under the definition are not integrated and are often blind to one another, causing conflicts and suboptimal network settings [5]; while several *ad hoc* algorithms have been proposed to solve this issue [6, 7], these efforts are still partial and unsystematic. However, moving towards a more integrated and functional network intelligence is one of the major goals of 5G, the new generation of cellular networks [8], which puts a greater emphasis on AI techniques and network awareness [9] to deal with the increased network complexity.

Several factors contribute to the complication of communication networks. Firstly, the ever-growing demand for mobile data traffic, which has already increased 4,000 fold in the past 10 years [10] and is expected to continue doing so at a steady rate, strains cellular networks' capabilities and requires ever more optimized strategies. The wide variety of applications and services that 5G will need to support is another major complicating factor: applications such as live video conferencing, Augmented Reality (AR), or vehicular networking impose strict latency and throughput requirements that strain the network's capabilities. The integration of the Internet of Things (IoT) and Smart City paradigms into 5G also contributes to make the network harder to manage, since supporting both human-type and Machine to Machine (M2M) communications [11] is a complex problem. The densification of cellular networks by deploying small cells [12] also opens new possibilities to optimize the access network, but it makes the burden of complexity even heavier, as does the integration of new radio technologies such as mmWave [13].

The combination of these factors makes it very likely that intelligence will not only be an improvement for the 5G network, but a vital necessity, and the vision of a cognition-based network is becoming closer to reality [14].

In this kind of network, tools such as Reinforcement Learning (RL) [15] and unsupervised representation learning [16] will make intelligent, context-aware decisions with high-level objectives, integrating different technologies and supporting the demands of very different applications organically. In other words, network elements will be able to see the big picture and optimize users' Quality of Experience (QoE) instead of maximizing a simpler Quality of Service (QoS) metric [17], thanks to their awareness of the environment [18].

In this work, we present our first steps towards the definition of such a network, providing case studies showing how intelligence can benefit different aspects of communication networks. Chapter 2 presents the theoretical background of this thesis, describing several machine learning algorithms in the fields of prediction, regression and RL, while the following chapters describe our work in applying these techniques to future networks and applications. First, Chapter 3 shows how networks can benefit from prediction and data-driven approaches by presenting three case studies of prediction of network variables, which may then be exploited by the network to provide user-centric optimized services. In Chapter 4, we combine that prediction with traditional optimization algorithms , showing how this anticipatory approach can lead to more intelligent decisions on the Transport layer. Going back to the definition of intelligence, we can state that such a system satisfies the criteria, and is therefore intelligent, but something is still missing: the intelligent behavior is designed and pre-programmed into the system, and the design effort is considerable and needs to be repeated for every problem, considering all possible aspects and performing extensive testing and theoretical work. In Chapter 5, we present a use case of RL as a possible way to avoid this work: RL agents learn optimal behavior by trial and error without any predefined model, and although they are still not a one-size-fits-all solution, their performance when dealing with uncertainty in networking scenarios is impressive. In Chapter 6, we present Smart Cities as one of the main components of the 5G system. Bike sharing, one of the most important Smart City services, is used as an example to show how the introduction of data analytics and intelligence can improve the performance of this kind of systems. Chapter 7 takes the concept a step further, and presents the idea for a symbiotic relationship between the Smart City and the underlying communication network; more specifically, the sensors in a Smart City can provide data to optimize the network in an intelligent way. As in all the previous chapters, we provide case studies and examples showing the benefits of intelligence in realistic settings. Finally, we draw our conclusions in Chapter 8.

Chapter 2

A review of machine learning techniques

In this chapter, we present a review of the machine learning techniques we use in later chapters; several prediction techniques are described, along with a review of the theory of RL and the recent developments in the field. The algorithms we present are commonly used in the literature [3, 19], but this is by no means an exhaustive survey of the prediction techniques in the literature [20, 21].

Perception is a critical component of intelligence [22]: an agent cannot develop intelligent strategies to act and accomplish goals in an environment if it is not aware of the environment's nature and the effects an action might have on it. A natural extension of the concept of perception is prediction [23]: an intelligent agent does not just passively register the environment through its sense organs or artificial sensors, but it builds a model of reality and expects certain things to happen in the immediate future.

In a networking context, the possible benefits of predicting factors such as the future traffic demand or the location of users are clear: considering a stochastic knowledge of the future can improve the performance of network optimization algorithms, as the additional information enables more intelligent choices. The concept of *anticipatory networking* [24] is gaining traction in the research community, as the highly volatile nature of wireless networks requires foresightedness. Since a prediction might be required at different layers of the networking stack and for processes with different statistics, it might have different timescales and dimensions, involving vastly different amounts of data.

RL represents a further step in the evolution of intelligence in communication networks: while prediction techniques can be useful in the design of classical optimization algorithms, RL agents can autonomously learn how to interact with the environment without the need for an explicitly pre-programmed behavior. This approach reduces the necessary design effort, enabling agents to act without a human designing its responses [25], learning how to make its own decisions like a biological brain [26].

2.1 Prediction techniques

In this section, we describe a variety of prediction tools from the literature; in Chapter 3 we will use them in a series of case studies of prediction of network parameters, using different datasets and timescales. Our objective is to showcase the techniques and how they can be applied in a more systematic fashion in future networks, automatically generating predictions of all the relevant network variables and providing a full awareness of the present and future environment so that each network element can act intelligently in it.

2.1.1 Graphical Bayesian models

The Graphical Bayesian (GB) model can be represented by the graph shown in Figure 2.1. As the Bayesian model only works for discrete attributes, the dynamic interval of any continuous process needs to be discretized into M classes. The memory-n Bayesian model uses the past n samples as features, resulting in M^n possible combinations of the input vector. The predictor is essentially a classifier, in which the future sample of the random process is the correct class.

The multimodal classifier is implemented by a Dirichlet distribution [27] over the M-dimensional simplex, which is parameterized by a real non-

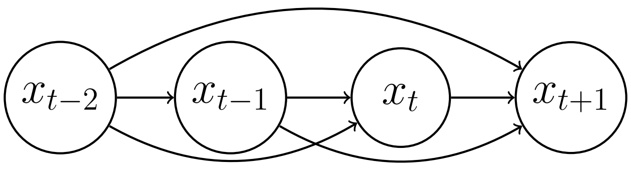


Fig. 2.1: Representation of the graphical Bayesian model with 3-state memory.

negative vector $\boldsymbol{\alpha}$:

$$P(\mathbf{x} = (x_1, \ldots, x_M)|\boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^{M} x_i^{\alpha_i - 1}, \tag{2.1}$$

where the normalizing constant $B(\boldsymbol{\alpha})$ is the multivariate Beta function [28]. The random vector $\mathbf{x} = (x_1, \ldots, x_M)$ is a probability distribution over the M classes, such that X_i represents the probability that the given sample is in the i-th class, with $i \in \{1, \ldots, M\}$ (i.e., the probability distribution of the next sample of the studied process). The first and second moments of X are given by:

$$E[x_i] = \frac{\alpha_i}{\sum_{j=1}^{M} \alpha_j} \tag{2.2}$$

$$\text{Var}[x_i] = \frac{\alpha_i \sum_{j \neq i} \alpha_j}{(\sum_{j=1}^{M} \alpha_j)^2 (\sum_{j=1}^{M} \alpha_j + 1)} \tag{2.3}$$

Intuitively, $E[x_i]$ is a measure of how probable the class i is with respect to the totality of the classes, and $\text{Var}[x_i]$ measures the uncertainty on that probability. The conjugate distribution of the Dirichlet distribution is the Dirichlet-multinomial distribution; Bayesian inference can be performed by generating a new parameter vector $\boldsymbol{\alpha}'$, defined as

$$\alpha_i' = \alpha_i + n_i, \tag{2.4}$$

where n_i is the number of observed transitions to class i. The prediction can be performed by taking the expected probability distribution of the next sample, given by

$$P(i) = \frac{\alpha'_i}{\sum_{j=1}^{M} \alpha'_j}. \tag{2.5}$$

The predicted class then corresponds to the maximum probability value.

2.1.2 Support Vector Machines

Support Vector Machines (SVMs), also called Support Vector Regressions (SVRs) when used for regression, [29,30], are learning machines that minimize the following cost function:

$$C = \sum_i E_\varepsilon(f_{\mathbf{w}}(\mathbf{x}^{(i)}) - x_{t+1}^{(i)}) + \lambda ||\mathbf{w}||^2, \tag{2.6}$$

where $f_{\mathbf{w}}$ is a function taking as input a memory-n feature vector $\mathbf{x}^{(i)} = (x_{t-n+1}, \ldots, x_t)$ and predicting a future sample $\hat{x}_{t+1}$, for a given training example i. The error z between this predicted sample and the actual sample at time $t+1$, x_{t+1}, is then fed to an ε-insensitive error function

$$e_\varepsilon(z) = \begin{cases} |z| - \varepsilon & \text{if} \quad |z| > \varepsilon \\ 0 & \text{otherwise} \end{cases}, \tag{2.7}$$

so that $f_{\mathbf{w}}$ is constrained to have a maximum absolute prediction error lower than a given constant ε for all the training data. The second term in (2.6) accounts for regularization: the trade-off between the minimization of the two terms is governed by the constant λ (the reader can refer to [31,32] for more details). In (2.6), all the training examples are assumed to lie in an "ε-tube" (see Figure 2.2). However, this is not verified in general, and (2.6) can be modified so as to allow for some tolerance in the prediction errors. Therefore, for each training example $\mathbf{x}^{(i)}$, it is possible to introduce *slack variables* ξ_i and ξ_i^*, where $\xi_i > 0$ is related to a point for which $(x_{t+1}^{(i)} - f_{\mathbf{w}}(\mathbf{x}^{(i)})) > \varepsilon$, and $\xi_i^* > 0$ is related to a point for which $(f_{\mathbf{w}}(\mathbf{x}^{(i)}) - x_{t+1}^{(i)}) < -\varepsilon$. Training examples are thus allowed to lie outside the ε-tube, as in Figure 2.2, provided

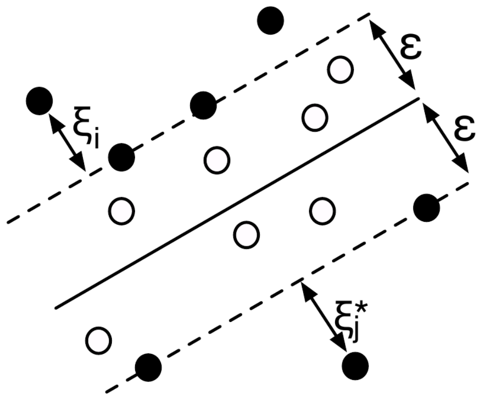


Fig. 2.2: Graphical representation of an ε-tube with slack variables.

that the corresponding slack variables are positive: this condition can be formulated as

$$-\varepsilon - \xi_i^* \leq x_{t+1}^{(i)} - f_{\mathbf{w}}(\mathbf{x}^{(i)}) \leq +\varepsilon + \xi_i. \tag{2.8}$$

The SVR problem then becomes

$$\min_{\mathbf{w}} \sum_i (\xi_i + \xi_i^*) + \lambda ||\mathbf{w}||^2, \tag{2.9}$$

subject to the constraints $\xi_i, \xi_i^* \geq 0$, and (2.8). It can be seen that only the examples outside the ε-tube contribute to the cost, with deviations being linearly penalized. Computing the dual formulation of (2.9), exploiting the Karush-Kuhn-Tucker conditions [33, 34], and assuming that $f_{\mathbf{w}}$ is simply a linear function of the inputs, i.e., $f_{\mathbf{w}}(\mathbf{x}^{(i)}) = \mathbf{w}\mathbf{x}^{(i)} + b$, it can be found that

$$\mathbf{w} = \sum_i (\mu_i - \mu_i^*)\mathbf{x}^{(i)}, \tag{2.10}$$

were μ_i and μ_i^* are the Lagrange multipliers. The prediction function then becomes

$$f_{\mathbf{w}}(\mathbf{x}) = \sum_i (\mu_i - \mu_i^*)(\mathbf{x}^{(i)}\mathbf{x}) + b. \tag{2.11}$$

In (2.10), the weight vector $\mathbf{w}$ is a function of the training examples $\mathbf{x}^{(i)}$; however, only those examples such that $\mu_i - \mu_i^* \neq 0$, called Support Vectors (SVs), have to be evaluated in (2.10) and (2.11). Finally, it is possible to allow the prediction function $f_{\mathbf{w}}$ to be non-linear in each training example $\mathbf{x}^{(i)}$, so as to allow better generalization over non-linear target functions. In fact, in (2.11), the SVs only appear inside scalar products, and (2.10) does not need to be calculated explicitly. Therefore, it can be proved that $< \mathbf{x}^{(i)}, \mathbf{x} >$ in (2.11) can be replaced by particular non linear functions $k(\mathbf{x}^{(i)}, \mathbf{x})$, known as *kernels*, which correspond to scalar products between non linear transformations of $\mathbf{x}^{(i)}$ and $\mathbf{x}$. Substituting $k(\mathbf{x}^{(i)}, \mathbf{x})$ in (2.11), we thus obtain the optimal prediction function in a non-linear *feature space*, rather than in input space:

$$f_{\mathbf{w}}(\mathbf{x}) = \sum_{i=1}^{m} (\mu_i - \mu_i^*) k(\mathbf{x}^{(i)}, \mathbf{x}) + b. \tag{2.12}$$

2.1.3 Linear regression techniques

Regression is a statistical method to fit data to a model. The simplest form of regression is *multiple linear regression* [35], which finds the linear model that minimizes a loss function of the error between the model outputs and the actual data; the least squares function is usually used as the loss function.

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}, \tag{2.13}$$

where $\mathbf{X}$ is the matrix of independent variables and $\mathbf{y}$ is the dependent variable. The parameter vector $\hat{\boldsymbol{\beta}}$ represents the fitted model [36].

We do not consider nonlinear regression in this work, although it is often used for prediction; in our view, the complexity of nonlinear regression makes more advanced learning techniques such as Neural Networks (NNs) or SVMs more useful tools. However, given the highly variable nature of networking data, we consider some regularization techniques that weight the loss function to constrain the resulting model and avoid overfitting the available data.

- *Ridge regression* [37] is a shrinkage method that adds a square penalty to the least squares loss, weighted by a regularization matrix λ_R.

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X} + \boldsymbol{\Gamma}^T\boldsymbol{\Gamma})^{-1}\mathbf{X}^T\mathbf{y}, \tag{2.14}$$

where $\boldsymbol{\Gamma} = \lambda_R \boldsymbol{I}$. Adding the regularization matrix helps reduce overfitting by penalizing models with very large parameters and mitigating the noise introduced by solving the inverse problem.

- *Lasso regression* [38] is a shrinkage method very similar to ridge regression that uses a linear penalty instead of a square penalty. Lasso regression does not have a closed-form solution, but it requires the use of convex optimization techniques.
- *Elastic net regression* [39] is a linear combination of the lasso and ridge regularization techniques, and is particularly useful when the number of predictors is larger than the number of observations and in the presence of highly correlated predictors.

2.1.4 Random Forest and k-Nearest Neighbors

The Random Forest (RF) method [40] operates by creating multiple decision trees [41] and giving the mean of the trees' prediction as an output. The forest only consider some of the features of each tree, avoiding the risk of overfitting.

The k-Nearest Neighbors (k-NN) method [42] is a simple learning algorithm, in which the output is the average between the k closest values in the feature space. There are significant theoretical similarities between RF and k-NN [43]; both can be viewed as weighted neighborhood schemes, but in RF the neighborhood of a point depends on the structure of the trees, and consequently also on the training set.

2.1.5 Neural Networks

The perceptron [44] is a simple method to learn a linear classifier, invented in 1957. The weighted sum of the inputs is passed through a non-linear activation function, then the output is quantized so that the classifier gives a binary response. The limited utility of the perceptron was due to its inability to recognize nonlinear patterns, but the study of NN gained new life when Multilayer Perceptrons (MLPs) were invented [45], stacking multiple layers to extend the algorithm's capabilty to recognize different patterns.

A MLP is a fully connected *feed-forward* network with one or more hidden layers. The neurons can be connected in several ways, which depend on the implemented architecture. Input neurons get activated directly by the environment state variables, while other neurons are activated through weighted connections from neurons residing in previous layers. Given the architecture, i.e., the way neurons are connected, and the activation functions, i.e., how the weighted input is reshaped by each neuron (and subsequently sent forward to the next layer), the whole system is completely determined by the connection weights and biases, which are both included in a single vector $\mathbf{w}$. Neural networks are trained by backpropagation [46], a simple gradient-based algorithm that updates each weight based on the error of the predicted output.

The output of any neuron in any layer $\ell \geq 1$ is obtained by first computing a *weighted* sum of all the outputs from the previous layer, and then evaluating it through a nonlinear activation function (the hyperbolic tangent, in our case). The final output vector, from the neurons in the last layer, is obtained through a cascade of operations of this type, by passing the output vector from any layer $\ell \geq 1$ to all neurons in the next layer $\ell+1$. A great deal of work has been carried out on deep learning architectures in the last decade [47,48]. For further details, see, e.g., [49].

This network has no memory, and this means that given an input vector, the final output vector only depends on the network's weights. In this case, if we desire to keep track of a process over time, the input vector has to be extended to contain this information. This entails a redefinition of the system state (to account for *current* and *past* samples), which corresponds to a larger number of neurons and to a higher complexity (in terms of training time and memory space).

On the other hand, Recurrent Neural Networks (RNNs) implement some

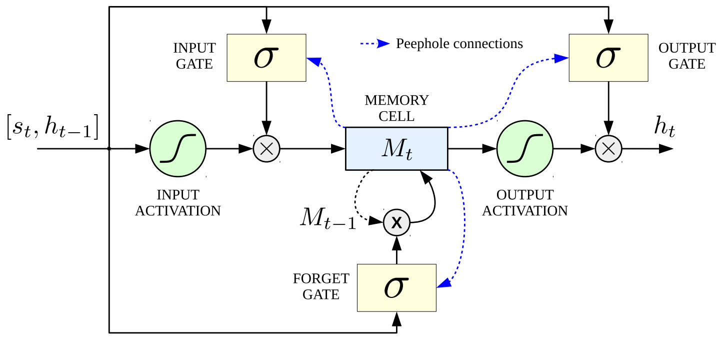


Fig. 2.3: Schematic diagram of an LSTM cell.

internal feedback mechanics that introduce memory, i.e., given an input vector, the network output depends on the network's weights *and* on the previous inputs. A schematic diagram of the LSTM internal structure is shown in Figure 2.3. The memory is implemented through a Memory Cell that allows storing or forgetting information about past network states. This is made possible by structures called gates that handle access to the Memory Cell. Gates are composed of a cascade of a network with sigmoidal activation function (σ) and a point-wise multiplication block. There are three gates in an LSTM cell: **1.** the *input gate*, that controls the new information that need to be stored in the Memory Cell, **2.** the *forget gate*, that manages the information to keep in the memory and what to forget, and **3.** the *output gate*, associated with the output of the cell (h_t). In addition, all the data that pass through a gate is reshaped by an activation function (usually an hyperbolic tangent). Optionally, peephole connections can be added to allow all gates inspect the current cell state, even when the output gate is closed [50]. Backpropagation Through Time (BTT) is usually used in conjunction with optimization methods to train RNNs [51].

2.1.6 Kalman filters

The Kalman filter [52] is the continuous extension of a Hidden Markov Model (HMM) [53]; it models a known discrete-time linear dynamic system with a hidden state $\mathbf{x}_k$, which it tries to estimate from the history of a correlated observation vector $\mathbf{y}_k$. The state vector is never observed directly, but the function mapping states to observations at time k is known:

$$\mathbf{y}_k = \mathbf{A}\mathbf{x}_k + \mathbf{w}_k, \tag{2.15}$$

where $\mathbf{A}$ is the matrix defining the linear system mapping states to observations and $\mathbf{w}_k \sim \mathcal{N}(0, \sigma_w^2)$ is a white Gaussian noise with a known correlation matrix $\mathbf{Q}$. The system state change equation is given by:

$$\mathbf{x}_k = \mathbf{B}\mathbf{x}_{k-1} + \mathbf{v}_k, \tag{2.16}$$

where $\mathbf{A}$ is the matrix defining the linear state change system and $\mathbf{v}_k \sim \mathcal{N}(0, \sigma_v^2)$ is a white Gaussian noise with a known correlation matrix $\mathbf{R}$. The system is completely defined by the tuple $(\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R})$, and the equations work even when it is time-dependent, as long as the tuple is known at all times.

The Kalman filter is the optimal estimator, and it works in two phases: a prediction (*a priori*) phase during which the filter estimates the future behavior of the system, and an update (*a posteriori*) phase during which the filter incorporates new observations and refines its estimate of the state.

The prediction phase extrapolates the currently known information to future states by iterating the system equations:

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{B}\hat{\mathbf{x}}_{k-1}. \tag{2.17}$$

It is also possible to estimate the error covariance matrix $\mathbf{P}$:

$$\mathbf{P}_{k|k-1} = \mathbf{B}\mathbf{P}_{k-1}\mathbf{B}^T + \mathbf{R}. \tag{2.18}$$

It is possible to predict future observations by applying the same method, and with a known error.

The update phase is based on the concept of innovation, which we denote by $\mathbf{z}_k$ and is defined as:

$$\mathbf{z}_k = \mathbf{y}_k - \mathbf{A}\hat{\mathbf{x}}_{k|k-1}. \tag{2.19}$$

The innovation is equivalent to the difference between the actual observation and the predicted one; its covariance $\mathbf{S}_k$ is calculated as

$$\mathbf{S}_k = \mathbf{Q} + \mathbf{A}\mathbf{P}_{k|k-1}\mathbf{A}^T. \tag{2.20}$$

The optimal Kalman gain $\mathbf{K}_k$ is then given by

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{A}^T\mathbf{S}_k^{-1}. \tag{2.21}$$

The update equation for the filter is

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\mathbf{z}_k, \tag{2.22}$$

and the *a posteriori* covariance of the state is

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{A})\mathbf{P}_{k|k-1}(\mathbf{I} - \mathbf{K}_k\mathbf{A})^T + \mathbf{K}_k\mathbf{Q}\mathbf{K}_k^T. \tag{2.23}$$

The main limitations of the Kalman filter are its linearity and the requirement to know the noise covariance matrix perfectly; a Kalman system designed for the wrong system will often perform poorly. However, it is possible to use the autocorrelation of the innovation to dynamically estimate the system noise and correct the filter. The adaptive Kalman filter [54] converges to the optimal Kalman filter starting with no knowledge of $\mathbf{Q}$ and $\mathbf{R}$ for any linear time-invariant system. The Autocovariance Least Squares (ALS) method [55] exploits the same principle, but it is now widely used because of its faster convergence. It is also possible to extend the Kalman filter to nonlinear systems using the unscented Kalman filter [56].

2.2 Reinforcement Learning

It is possible to combine one of the previously described prediction techniques with a classical optimization algorithm to design a foresighted system, and the possibility to intelligently exploit predictions to optimize a network is a powerful tool [24]. However, this combination of intelligence and traditional optimization techniques still requires a significant design effort; in order to call a network element fully intelligent, it should be able to act without a human designing its responses [25], learning how to make its own decisions like a biological brain [26].

This is possible through RL, a technique pioneered by Chris Watkins in the late '80s [57]. RL is inspired by behavioral psychology and experiments in animal learning, as agents start with almost no knowledge of the world and gradually learn the optimal behavior by trial and error. RL is fundamentally different from the techniques we described in Section 2.1, as the agent takes an active role in the learning instead of passively estimating a variable, and its first application was to board games. Since board games are inherently adversarial and have a rigid structure that can be easily modeled, it is easy to transpose them as RL problems and to evaluate agents' performance. In 1995, the RL algorithm TD-Gammon [58] was the first AI to play backgammon at the same level as human champions, and it led to several innovations in high-level strategy. Although the first computer to beat the human chess world champion, Deep Blue, did not use reinforcement but just a brute-force look-ahead strategy [59], RL sparked a revolution in the AI field after Mnih *et al.* combined it with deep learning, achieving human-level play or better on several classic arcade games using only visual input [60]. The same techniques were later used to beat the human world champion of Go, an extremely complex Chinese board game that was thought to be too complicated for machines [61]. A later version of the algorithm achieved the same results without any external training [62], mastering chess beyond human levels as well. Generalizing experience in one game to others is another step towards true intelligence [63], since the extensive training that these algorithms need does not always make them practical.

Although the field is very active and RL tools and algorithms are still being developed and improved, its applications to networks has already begun [64]: having a general learning agent limits the design effort and improves performance in several contexts, and we predict that RL is going to be widely used in 5G.

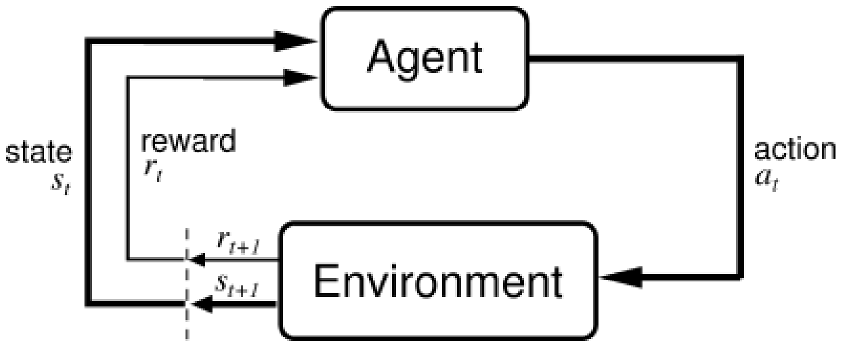


Fig. 2.4: Schematic diagram of an MDP.

2.2.1 Markov Decision Processes

The model that sparked the RL revolution is the Markov Decision Process (MDP) [65]: all the successful algorithms we cited above are based on this underlying model. An MDP, whose basic logic is shown in Figure 2.4, is defined by an action space A, a state space S, and a reward function $\rho : S \times S \times A \to \mathbb{R}$. The action $a_t \in A$, taken when the system is in a given state $s_t \in S$, determines the statistical distribution of the next state s_{t+1} and the reward $\rho(s_t, s_{t+1}, a_t)$ attained in step t. A policy is a function $\Pi : S \to A$ that maps states into actions.

The *expected long-term utility* achieved by an admissible policy Π when starting from state s_0 is defined as

$$R(s_0; \Pi) = \lim_{h \to +\infty} \mathrm{E}\left[\sum_{t=0}^{h} \lambda^t \rho(s_t, s_{t+1}, \Pi(s_t)) \middle| s_0, \Pi \right] \tag{2.24}$$

where $\lambda \in [0, 1)$ is a discount factor that ensures convergence, and $P(s_{t+1}|s_t)$ is the one-step conditional transition probability of the state process $\{s_t\}$. The equivalent recursive formulation, first derived by Bellman in [65], is given by:

$$R(s_0; \Pi) = \sum_{s_1 \in S} P(s_1|s_0)\rho(s_0, s_1, \Pi(s_0)) + \lambda R(s_1; \Pi). \tag{2.25}$$

The optimal policy $\Pi^*(\cdot)$ is finally found as:

$$\Pi^*(s) = \arg\max_{\Pi} R(s;\Pi)\,, \quad \forall s \in S\,. \tag{2.26}$$

The problem (2.26) can then be solved through dynamic programming algorithms such as Value Iteration (VI) [65], but the computational complexity becomes rapidly unmanageable as the size of the problem grows. A possible approach to deal with the curse of dimensionality is to adopt RL tools, such as Q-learning [66], which gradually converges to the optimal solution through trial-and-error, as explained next.

2.2.2 Q-learning

Q-learning is a RL algorithm introduced by Watkins in 1992 [66]. It works by maintaining a table of estimates $Q(s, a)$ of the expected long-term reward (given by (2.25) in our problem formulation) for each state-action pair (s, a).

The Q-learning algorithm can use various exploration policies to decide the next action based on the *Q-values*: the most common are the ε-greedy policy and Softmax. Both strategies are non-deterministic; the ε-greedy policy chooses the action with the highest Q-value with probability $1 - \varepsilon$, and a suboptimal action at random with probability ε. Softmax chooses an action according to the softmax distribution of Q-values:

$$P(a_t|s_t) = \frac{\exp\left(-\frac{Q(s_t,a_t)}{\xi}\right)}{\sum_{a\in A}\exp\left(-\frac{Q(s_t,a)}{\xi}\right)}\,, \tag{2.27}$$

where the parameter ξ sets the greediness of the algorithm. Greedier algorithms make suboptimal choices less often, but run a higher risk of getting stuck in local minima since they explore the state space less frequently.

In standard Q-learning, the Q-value $Q(s_t, a_t)$ is updated if the learning agent takes action a_t in state s_t. The future reward over the infinite time horizon is approximated as $\max_{a\in A} Q(s_{t+1}, a)$, i.e., the best Q-value of the future state s_{t+1}. If the Q-value matches the real expected long-term reward, Q-learning coincides with the Bellman formulation in (2.25), which exactly solves the problem. In practice, since the Q-values are to be learned at runtime, they only provide an approximation of the real long-term rewards, but it has been proved that they converge to the optimal rewards for sensible exploration policies. The Q-learning update function is given by:

$$\hat{R}(s_t, a_t) = \rho(s_t, s_{t+1}, a_t) + \max_a \lambda Q(s_{t+1}, a) \quad (2.28)$$

$$Q(s_t, a_t) \leftarrow Q(a_t, q_t) + \alpha(\hat{R}(s_t, a_t) - Q(s_t, a_t)) \ , \quad (2.29)$$

where the learning rate α sets the aggressiveness of the update and is usually decremented over time as the learning agent gets closer to convergence. The choice of the maximum Q-value in the bootstrap approximation (2.28) makes Q-learning an *off-policy* learning algorithm, since the greedy policy used in the long-term reward estimation (update policy) usually differs from the actual policy the learner uses (exploration policy). As the Q-values converge, the exploration policy should become greedier until it reaches convergence.

Limits of the Q-learning approach

The Q-learning approach is powerful, but it has some limitations: the algorithm provably converges to the optimal policy if its parameters are chosen correctly [66], but the convergence speed is an issue in complex problems. In [67], Kearns and Singh proved that, for an MDP with N states and A actions, the number of samples necessary for the expected reward to converge within ε of the optimal policy with probability $1-p$ is bounded by:

$$O\left(N\,A\,\varepsilon^{-2}\left(\log\left(\frac{N}{p}\right)+\log\left(\log\left(\varepsilon^{-1}\right)\right)\right)\right). \quad (2.30)$$

For fixed values of ε and p, the number of training steps is $O(NA\log(N))$, but the constant factor can be significant. Due to the curse of dimensionality, the number of states of the MDP tends to be very large for all but the most trivial problems, making standard Q-learning need a huge amount of training samples to reach convergence and obtaining a good trade-off between precision and adaptability.

When tackling networking problems, Q-learning has two main drawbacks:

- *Continuous state space.* The quantities defining the state space may often have values in some real interval. The definition of an MDP of manageable size involves a quantization of all the continuous variables in the state according to a predefined number of levels (dictated by the quantization step). The smaller the quantization step, the better the approximation of the actual (continuous) variable and the more accurate the fit between the MDP and the process it represents gets. However, the number of states grows very quickly as the quantization step gets smaller, and the best compromise between representation accuracy and number of states is often difficult to find;

- *Curse of dimensionality.* This is a direct consequence of the previous point, since the number of samples required for Q-learning to converge grows very quickly with the number of states, according to (2.30). Here, the problem is not only related to the convergence time, but also to the data availability: optimal policies may be hard to attain due to the need for too many data samples, which may not be available in practical settings.

With *deep Q-learning* we can avoid these issues by approximating the action-value function $Q(s, a)$ through a NN that returns the (estimated) Q-value $Q(s, a)$ for any given state and action pair (s, a). The network weights, once trained, will encode the mapping and replace the tables used by Q-learning. This allows the model to be fed with continuous variables, avoiding the quantization problem, and has the further desirable property that NNs, if properly trained, are able to *generalize*, providing correct answers (i.e., excellent Q-value approximations) even for points (s, a) that were never processed in the training phase. In other words, NNs act as universal approximators. This amounts to a reduction of the number of training samples that are required to reach a certain performance level; however, with this approach, the RL logic remains unchanged, and there is no restriction on the NN architecture to use.

2.2.3 Deep Q-learning

Conventional machine learning techniques are often limited in their ability to analyze data in their natural form. Usually, a good representation of the environment requires complex analysis and considerable expertise. This phase is commonly referred to as *feature engineering* and aims at finding a suitable representation of the raw data through a reduced set of features (*feature vector*), from which the learning system can extract useful environment information.

Representation learning consists of a set of mechanisms to automate this process: the learning machine is fed with raw data and discovers the best representation for detection or classification on its own. The deep learning methods we presented in Section 2.1.5 are representation learning techniques

Deep Q-learning combines a Q-learning approach with a deep learning framework to obtain optimal policies for any MDP. Learning systems of this kind, referred to as *Deep Q-networks (DQNs)*, have been used in many complex systems in different research fields with state of the art performance, although their development is quite recent.

The main difference between the standard Q-learning algorithm, as described in Section 2.2.2, and DQNs, is in the way of estimating the Q-value of each state-action pair, generalized by the function $Q(s, a)$, i.e., an approximation of the optimal action-value function $Q^*(s, a)$. While standard Q-learning keeps a table of values and updates each state-value pair separately, DQN uses a deep learning approach to approximate the Q-function. This can be achieved in two different ways:

1. a single deep network, fed with the current system state, is used to simultaneously estimate the Q-values for all possible actions;

2. one separate deep network is trained for each possible action, approximating a sub-space of the whole action-state set.

The first approach has the advantage of providing the entire set of Q-values (always needed to make the final decision) with a single computation.

Considering the MDP defined in Section 2.2.1, a loss function $\tilde{L}$ at iteration t is evaluated using the 4-tuple $e_t = (s_t, q_t, r_t, s_{t+1})$, which here is referred to as the *agent's experience* at time t. The loss function can be derived from the Bellman equation in (2.29) [60]:

$$\tilde{L}_t(s_t, a_t, r_t, s_{t+1}|\mathbf{w}_t) = \Big(r_t + \lambda \max_a \hat{Q}(s_{t+1}, a|\bar{\mathbf{w}}_t) - Q(s_t, a_t|\mathbf{w}_t)\Big)^2 \ , \quad (2.31)$$

where r_t is the reward accrued for segment t. Two deep NNs, with the same architecture, are considered. A first network, with weight vector $\mathbf{w}_t$, is updated for each new segment (at every time step t), and is used to build the Q-value map $Q(s_t, q_t|\mathbf{w}_t)$. A second NN, referred to as the *target network*, is accounted to increase the stability of the learning system [60], and its weight vector $\bar{\mathbf{w}}_t$ is updated every K steps (segments), by setting it equal to that of the first network and keeping it fixed for the next $K-1$ steps, i.e., $\bar{\mathbf{w}}_t = \mathbf{w}_t$ every K steps. The target network is used to retrieve the mapping $\hat{Q}(s_{t+1}, a|\bar{\mathbf{w}}_t)$ in (2.31). Another improvement is given by the implementation of a technique called *experience replay* [68]. The agent's experience $e_t = (s_t, a_t, r_t, s_{t+1})$ is stored into a *replay memory* $R = \{e_1, \dots, e_t\}$ after each iteration. In this way, a new loss function L_t that also accounts for the past experience can be considered. Specifically, a subset $R_M = \{e_1, \dots, e_M\}$ of M samples, with $e_j \in R$, $j = 1, 2, \dots, M$, is extracted uniformly at random from the replay memory R, and L_t is finally evaluated as an empirical mean over the samples in set R_M:

$$L_t(\mathbf{w}_t) = \frac{1}{M} \sum_{e_j \in R_M} \tilde{L}_t(e_j|\mathbf{w}_t) \ . \quad (2.32)$$

This leads to three main advantages: greater data efficiency, uncorrelated subsequent training samples and independence between current policy and samples [60].

The whole process can be divided into two consecutive phases, which take a different but fixed number of iterations: namely, the *update phase*, and the *test phase*.

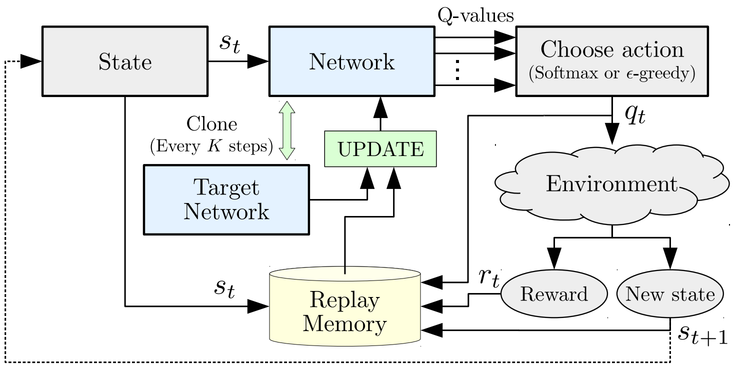


Fig. 2.5: Schematic diagram of an update iteration.

Update phase The exploration parameter, namely ε in the case of an ε-greedy policy or ξ for softmax (see Section 2.2.2), is gradually reduced. We recall that a smaller value for this parameter means that the policy tends to prefer the action that is considered to be optimal at the current training stage. Furthermore, at each iteration the network's weights are updated to minimize the loss function in (2.32). The Adam method is used as the gradient descent optimization algorithm: it implements an adaptive learning rate to provide a faster and more robust convergence [69].

Test phase The exploration parameter is set to zero, thus obtaining a greedy policy implementing the actions that are deemed optimal given the current system state and the mapping $Q(s_t, q_t|\mathbf{w}_t)$ from the first NN. For this phase, the weights $\mathbf{w}_t$ are frozen and are no longer updated for the whole duration of the test. The target network is not used in the test phase and all the performance evaluations are based on the results obtained during this second phase.

A schematic diagram of an update iteration is shown in Figure 2.5. First, the current environment state s_t is fed to the deep NN, which outputs an estimate of the Q-value for each possible action $q \in A$, i.e., the various representations in the adaptation set A. Then, an action a_t is chosen according to either an ε-greedy or softmax policy. Upon taking action q_t, the system moves to the new state s_{t+1} and a new reward r_t is evaluated according to (5.4). The newly acquired experience $e_t = (s_t, a_t, r_t, s_{t+1})$ is stored into the replay memory R. Hence, a batch of M samples is extracted, uniformly at random, from the replay memory and is used to update the network's weights. The loss function in (2.32) is minimized, using the mapping from the target network, i.e., $\hat{Q}(s_{t+1}, a|\bar{\mathbf{w}}_t)$, whose weights $\bar{\mathbf{w}}_t$ are updated every K steps.

Chapter 3

Prediction and anticipatory networking

As we explained in Chapter 2, the possibility to predict the future dynamics of several network parameters can significantly improve the performance of network optimization efforts [24]. In the context of 5G, the increasing complexity of cellular networks [70] and the strict QoS requirements of media applications will make intelligent management unavoidable; predictive approaches represent a first step towards such a model.

In this chapter, we focus on three case studies, predicting network variables and discussing their possible usefulness in optimization schemes. First, we try to estimate the long-term evolution of a wireless channel; this can make resource allocation easier for applications that work on timescales of hundreds of milliseconds or seconds, such as video streaming; prediction-based adaptive streaming systems have already been proposed [71]. Another aspect of supporting user activity is energy efficiency: nowadays, mobile devices often have to be recharged during the day to provide the required dependability [72]. While there is a huge research effort to increase the battery capacity and recharging speed, another approach looks at techniques to improve the energy efficiency of the devices, e.g., implementing smart energy-saving policies [73]. In this respect, an accurate estimate of the residual charge duration can be useful both to drive the energy-saving policies implemented by the operating system of the device and to let the user adopt energy-preserving strategies when using their mobile device.

Finally, the third case study looks at predictive approaches from a network perspective: using a publicly available dataset, it tries to predict the future traffic load for a given cellular base station in an urban scenario, exploiting spatio-temporal correlations. This could also be used in 5G network optimization in order to allocate resources foresightedly.

3.1 Predicting the wireless channel

In this section, we present two learning methods to predict the wireless channel gain on a long-term scale, without any inputs other than the time-averaged Received Signal Strength Indicator (RSSI). This prediction is performed by the Base Station (BS), which is the only fixed element in the network; the BS can indirectly learn the mobility patterns of the mobile users and the fading characteristics of the channel by observing patterns in their RSSI. Most of the efforts in the literature have focused on Multi-Input Multi-Output (MIMO) techniques on short time horizons, but several works [74] [75] have expressed the need for long-term accurate channel gain predictions. The two machine learning techniques we use in this context are GB models and SVR machines . The GB model can be used as a baseline, as it does not try to generalize its experience, but simply considers each class as a separate classification problem. SVRs are able to find and generalize patterns in the data, making better predictions with fewer data.[1]

3.1.1 State of the art

Wireless links are often modeled as Rayleigh fading channels. Most model-based prediction systems concentrate on short-term predictions of the fading envelope for wideband channels [77], and cannot be directly used for optimization at the higher layers.

[1]The work presented in this section was presented at IEEE ICNC 2017 and published in the conference proceedings [76].

Shen *et al.* predict future channel quality from receiver-side Channel State Information (CSI) [78], but the auto-regressive filters they use are only accurate on a timescale of a few milliseconds. The work in [79] proposes a prediction method tailored to Orthogonal Frequency Division Modulation (OFDM) and based on time-domain statistics with a slightly longer range, but the timescales for accurate predictions are still far below 100 ms.

Another auto-regressive model is proposed by Jarinova [80], but its predictions are extremely short-range and the order of the filter is a parameter that needs to be optimized carefully.

Several studies in the literature have attempted to solve the long-term channel prediction problem with machine learning techniques. A typical example is Ramanan and Walsh's channel prediction algorithm for sensor networks [81]; it is a distributed algorithm that employs message passing techniques to minimize the Kullback-Leibler divergence between the expected prior distribution and the actual posterior.

The problem of channel prediction is central in Cognitive Radio (CR) systems, and Demestichas *et al.* propose a GB model [82] to solve this issue. Their GB network predicts the future capacity for each possible CR configuration, and adapts to channel conditions online, but it is meant for Modulation and Coding System (MCS) selection rather than for optimization at higher layers.

Flushing *et al.* take an empirical approach [83], combining a probing mechanism with SVR to predict link quality in dense wireless networks. Thanks to mobility, the probing system can learn about several different topologies and extend this knowledge to larger, denser networks.

Finally, Liao *et al.* perform long-term channel prediction [75] using both spatial and temporal information. The authors propose a Gaussian Process (GP) regression, training the system through a series of routes on a city map. Their prediction method is robust against spatial errors, with better performance than both SVR and auto-regressive filters. Although their method is sound for large-scale scenarios, it does not deal with smaller cells and needs Global Positioning System (GPS) information from clients, which might not be available.

3.1.2 Studied scenario

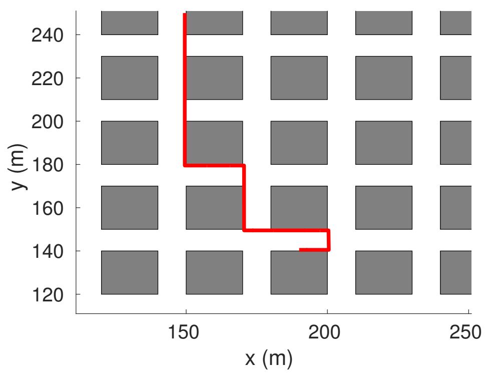
As we mentioned in Section 3.1, we used a GB model and an SVR to predict the future RSSI of a wireless channel, without external inputs such as GPS data. The two learning methods were trained on the same RSSI data, generated by simulating a realistic urban scenario. The wireless channel we considered used a 945 MHz downlink carrier frequency (one of the commercial bands used in Long Term Evolution (LTE)), and the users moved in a Manhattan grid of 100 buildings.

The grid we used is composed of 20 m wide square buildings, with 10 m wide one-way streets at each corner. The evolved Node Base (eNB) is placed at coordinates $(140, 140)$, on top of a building close to the center of the simulation area.

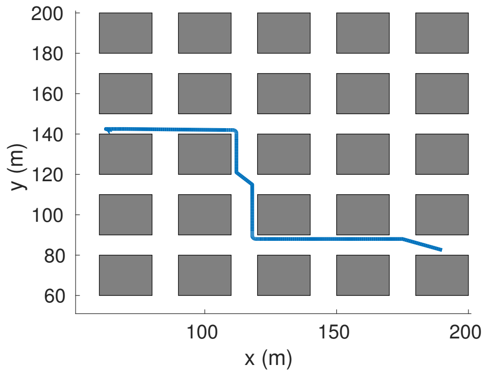
Propagation loss and fading The propagation loss was computed with the open-source system-level network simulator ns–3 [84]. In particular, we used the LTE module [85] and a radio propagation model called Hybrid Buildings Propagation Loss Model, which chooses the correct propagation model based on the reciprocal position of transmitter and receiver (both outdoors, both indoors, only one indoors). This model also takes into account the external wall penetration loss (for different types of buildings, i.e., concrete with windows, concrete without windows, stone blocks, wood), and the internal wall penetration loss.

We used the ns–3 simulation to create a square grid of path loss measures in our urban scenario, with a sampling distance of 33 cm. The path loss was then approximated as a linear combination of the 4 closest points in the grid, weighted by the relative distance. The main parameters of the ns–3 simulation are listed in Table 3.1.

The fading and shadowing processes were both simulated from well-known models. We used the log-normal model for shadowing, with a standard deviation of 4 dB and a correlation distance of 8 m. Doppler fading was modeled with a Rayleigh distribution, using the parameters listed in Annex B.2 of [86] and the Welch periodogram method. In the fading calculation, the node



(a) Pedestrian



(b) Vehicular

Fig. 3.1: Examples of trajectories for the two mobility models.

speed was assumed to be constant, simplifying the computation significantly with negligible error.

Mobility model We used two mobility models: *pedestrian* and *vehicular*. In both models, the user goes from point A to point B by choosing the direction that takes them closer to point B at each intersection.

In the *pedestrian* model, a person walks at a constant speed of 1.5 m/s along the side of the nearest building at a distance of 0.5 m. Road crossings are placed at each intersection, and the pedestrian waits for a random time

Parameter	Value
Downlink carrier frequency	945 MHz
Uplink carrier frequency	900 MHz
Resource block bandwidth	180 kHz
Available bandwidth	25 RB
eNB beamwidth	360° (isotropic)
TX power used by the eNB	43 dBm
eNB noise figure	3 dB
Number of buildings	100
Floors for each building	5
Radio Environment Map resolution	9 samples/m^2

Table 3.1: Path loss computation parameters

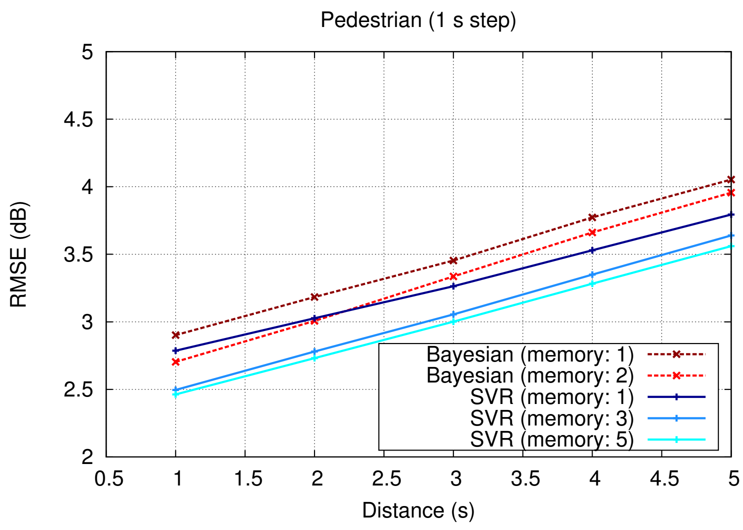


Fig. 3.2: Prediction error for the pedestrian scenario (1 s window).

between 0 and 5 seconds before crossing to wait for cars.

In the *vehicular* model, the driver keeps a constant speed of 15 m/s while driving straight, switching between the 3 lanes by moving at a 45 degree angle. Before a turn, the driver switches to the correct lane (e.g., they switch to the right lane before turning right), then slows down to 5 m/s with a constant deceleration in the 5 meters before the curve and makes a circular turn. After reaching the destination, the driver stops and reverses to slowly park on the curb, with a semi-circular trajectory.

The channel data were generated by running the urban scenario 5000 times for the pedestrian model and 10000 for the vehicular model, obtaining 3-4 days of data for the vehicular model and 20 hours for the pedestrian model (the car reaches its destination faster, so the traces are shorter). Two example trajectories for both models are shown in Figure 3.1.

3.1.3 Learning parameters and results

Both prediction methods were trained on the full available dataset, with two different sampling rates: the channel was averaged over a window of 1 s and 0.5 s.

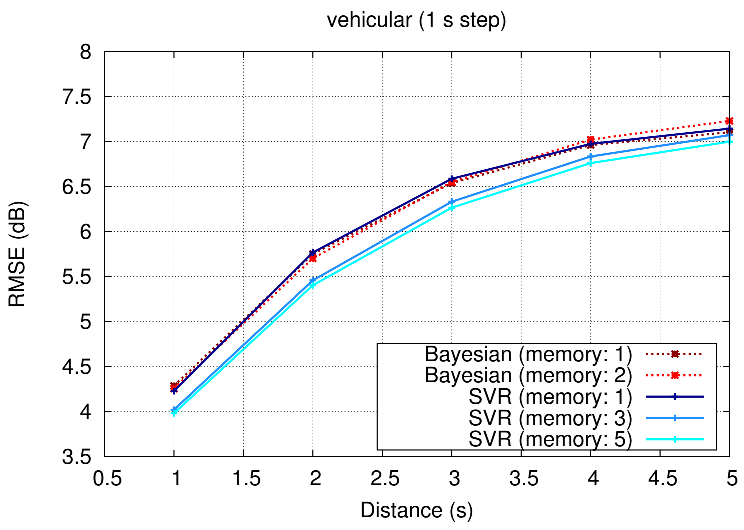


Fig. 3.3: Prediction error for the vehicular scenario (1 s window).

The Bayesian model used a Gaussian prior, centered on the last known channel sample; the probability vector for all classes was multiplied by a factor k to obtain the Dirichlet parameter vector $\boldsymbol{\alpha}$. Both the prior weight factor k and the variance σ of the Gaussian distribution were optimized as hyperparameters by cross-validation. The channel quantization step used to divide the data into classes was 2 dB.

As regards the SVR learning algorithm, we found the Radial Basis Function (RBF) kernel: $k(\mathbf{z}_i, \mathbf{z}_j) = e^{-\gamma||\mathbf{z}_i - \mathbf{z}_j||^2}$ to perform best with respect to other possible kernel choices. In this case, the hyperparameters of the model are γ and C in (2.9): a grid search on (γ, C) pairs was thus performed, and the one with the best cross-validation Root Mean Square Error (RMSE) was selected.

After cross-validation, the performance of both prediction methods was measured on a previously unknown test set.

Figure 3.2 shows the prediction RMSE for the pedestrian mobility model; the quality of the prediction is very good even with the simpler model, as pedestrians are slow and generally highly predictable. As expected, SVR clearly outperforms the naive Bayesian model, as it is able to generalize its experience and to better capture the features of the model. The gain

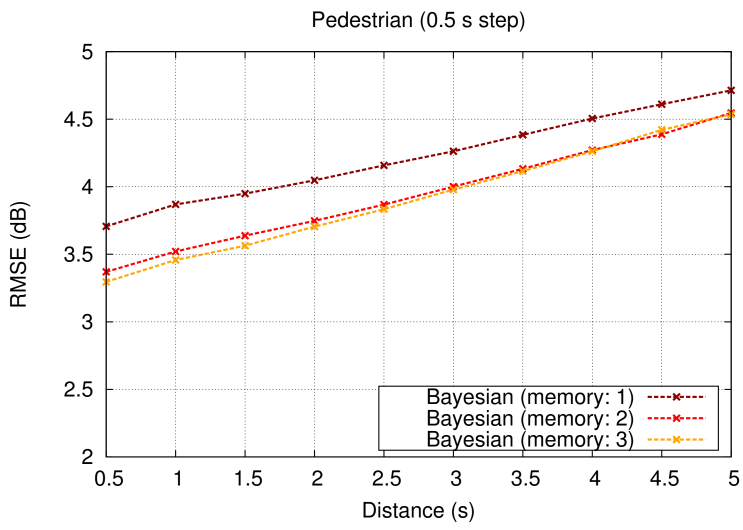


Fig. 3.4: Prediction error for the pedestrian scenario (0.5 s window).

of the longer memory is less pronounced for the Bayesian model, as it is overshadowed by the small size of the dataset (a longer memory means that a bigger dataset is necessary, and the memory-3 Bayesian model is not plotted, as its performance is not better than that with memory 2).

In the vehicular scenario, the RMSE is higher and the performance gap between the two methods is smaller (see Figure 3.3); the Bayesian model even outperforms the SVR if the prediction is more than 3 seconds ahead, but a prediction error of more than 7 dB is only slightly better than no prediction at all (the prediction RMSE when using a memoryless channel model is about 8 dB). This may be due to the high speed of the vehicles ($\sim$ 10 times the speed of the pedestrians), which makes accurate generalizations about the evolution of the channel hard.

Figures 3.4 and 3.5 show the performance of the Bayesian method with a channel sampling window of 0.5 s; due to the computational cost of the SVR training, its performance in this case has not been evaluated. The figures show that the trend in the performance of the Bayesian method is essentially the same, although the error is higher; the performance of the memory-3 Bayesian model shows that a longer memory is beneficial for the pedestrian model, but loses most of its benefits in the more chaotic vehicular scenario

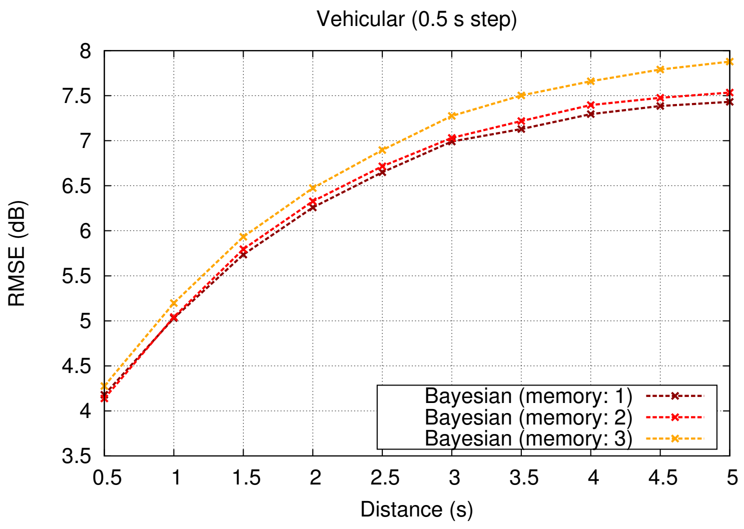


Fig. 3.5: Prediction error for the vehicular scenario (0.5 s window).

unless a bigger dataset is used.

Finally, Figure 3.6 shows the performance of the two predictors when they are trained with a reduced dataset: the two predictors were trained on 20% of the available data in the vehicular scenario with a 1 second step. The plot shows how the performance of the SVR degrades far less than that with the Bayesian model, thanks to the former's ability to generalize experience. In fact, the reduced-dataset SVR performs better than the full-dataset Bayesian model when the prediction distance is less than 5 seconds.

The training was performed with just a few hours of RSSI data, so a BS with multiple connected users might be able to quickly gather the necessary training data and achieve a high-quality prediction in a very short time. However, the computational cost of the training itself is not negligible; while SVRs show a clear performance gain in both scenarios, the Bayesian model might be enough for applications that need a lower precision. It is worth noting that the SVR can have a satisfactory performance even when trained using a reduced dataset, as shown in Figure 3.6; this makes it ideal if the limiting factor is not computational capability, but the size of the available dataset (e.g., in adaptive systems that are trained online to follow a time-varying scenario). The quality of the predictions is generally high, and the

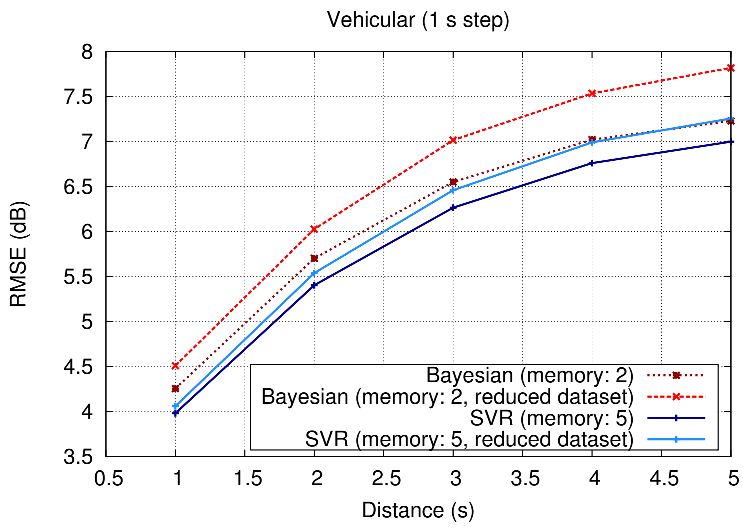


Fig. 3.6: Prediction error for the vehicular scenario (1 s window) including results using a reduced dataset.

RMSE is almost as low as the results shown in [75], but without the use of GPS data, thereby reducing the energy consumption.

3.2 Predicting battery usage in smartphones

As of today, most devices predict the battery depletion time by taking into account the mean duration of a full battery charge and the most recent battery-draining trend, disregarding time and location information [73]. In this section, we propose a novel method for the prediction of the battery depletion time that makes use of a NN to draw a personalized battery-usage pattern from time and location information provided by the mobile device. We apply our method to a dataset that includes mobility data collected by the LifeMap monitoring system at Yonsei University in Seoul [87]. Results show that our prediction method is by far more accurate than a popular method, currently employed in commercial mobile devices to predict the residual duration of the battery charge, and outperforms other machine-learning methods. We also provide insights on the importance of location information to the accuracy of such a prediction, and on the computational complexity of the proposed approach.[2]

3.2.1 State of the art

Several machine learning techniques have been applied to predict the battery charge duration of mobile devices. Most of the proposed solutions based the forecast on the current state of the device together with the battery-discharge history. For example, Zhao *et al.* [73] use the multiple linear regression prediction method: the discharge rate is calculated by comparing the current device status (e.g., CPU load, LCD brightness and I/O device usage) with the discharge rates that were observed in the past.

In [89], two NNs are trained in order to predict the discharging curve of batteries. Starting from some considerations on the electro-chemical nature of rechargeable batteries, the authors derive a non-linear discharging pattern. In order to model this behavior, they feed the NNs with some key parameters of the battery and device state, such as time, battery level, battery temperature, and resource usage in the past. The obtained predictor can estimate the remaining working time with an accuracy of 3%. However, their model was implemented and evaluated on a digital multimeter under controlled research conditions and no testing was performed on real world data. Wen *et al.* [90] propose to predict battery lifetime by using an online and offline calculation. First, a reference discharge curve is derived from a one-time, full-cycle, voltage measurement of a constant load, and is suitably transformed to reduce prediction complexity. Then, the forecasting is performed by mapping the current discharging data on the reference curve, using linear fitting or Least Squares methods.

An empirical approach to battery lifetime prediction is proposed in [91]. After analyzing a large dataset containing the discharging patterns of several users and training a general model, they built a tool to predict smartphone charge levels by classifying smartphone users based on the comparison of their discharging patterns with those in the dataset.

[2]The work presented in this section was presented at IEEE GLOBECOM 2017 and published in the conference proceedings [88].

In [92], Rakhmatov *et al.* study the problem of battery discharge time prediction from a theoretical point of view. Starting from the physical working principles of a lithium-ion electrochemical cell, a high-level model of the battery is built. The discharging coefficients of the model are estimated by simulation and statistical fitting of empirical data, and variable and constant loads are taken into account to improve the estimate of the residual operating lifetime of the device.

Most of the techniques mentioned above do not take individual users' discharging patterns and personal behavior information into account, but instead exploit the depletion discharging curve of a generic smartphone usage or focus on a short-term prediction mainly employing low-level information. Although current battery level and resource usage are fundamental to predict the battery discharge time, we advocate that a much more reliable and long-term estimation of the battery charge can be obtained by considering some space-time features of the device usage patterns. Our model will be able to capture time- and location-dependent user habits which can have a heavy impact on the battery lifetime [73]. Moreover, it is highly scalable to different types of devices and batteries and does not require manual tuning or reference parameters, unlike [90] or [73].

3.2.2 Data analysis

In this section, we discuss the proposed estimation method, the dataset we used and the selection of input and output parameters. Then, we explain the preprocessing operations performed on the data to ease the learning task of the NN.

Dataset and features The dataset from the LifeMap project [87] includes a wide collection of mobility data that was gathered from the smartphones of 6 graduate students from Yonsei University over 6 months. More specifically, data about the battery level, position and connectivity level of each smartphone were retrieved from the operating system and stored every 10 minutes. Data clearly span many charge and discharge cycles, but the battery charge reached a critically low level only on a few such cycles, since in many occasions the smartphones were recharged during the day, before the battery charge dropped below critical levels. Therefore, given this limitation of the dataset, we rely on the expected lifetime of the battery, as we will explain in the following paragraphs.

To design our predictor, we first need to identify the *features*, or *input* parameters, that have to be extracted from the dataset, and the *targets*, or *output* values, of the machine learning model. The choice of these parameters is discussed in the following.

Target parameter Since our purpose is to predict the battery discharge process over time, the output of the estimator should be a projection of the residual lifetime of the device in the future. To build the training set for our machine learning algorithm, hence, we need to find the (expected) residual lifetime of the battery charge in each of the time instants of the discharge phases where data were collected. For the discharge phases that end with a non-negligible residual battery level, the expected depletion time is estimated by linear extrapolation of the charge levels during that cycle. More specifically, let $(t_0, t_1, \dots t_n)$ be the set of sample times in a given discharge period, and $b(t_i)$ be the value of the battery level (on a 0-100 scale) measured at time t_i, $i = 0, 1, \dots, n$. If we take the whole discharge cycle into consideration, the expected depletion time is obtained as:

$$t_{\text{dep}} = t_0 - b(t_0)\frac{t_n - t_0}{b(t_n) - b(t_0)} \,. \tag{3.1}$$

Finally, the outputs y_i used to train our prediction model are obtained for each discharge cycle as

$$y_i = t_{\text{dep}} - t_i, \quad 0 \le i \le n \,. \tag{3.2}$$

An example of this analysis is shown in Figure 3.7. We observe that this estimate is more accurate when $b(t_n)$ is small, i.e., when the discharge cycle ends with an almost complete discharge of the battery, which is quite

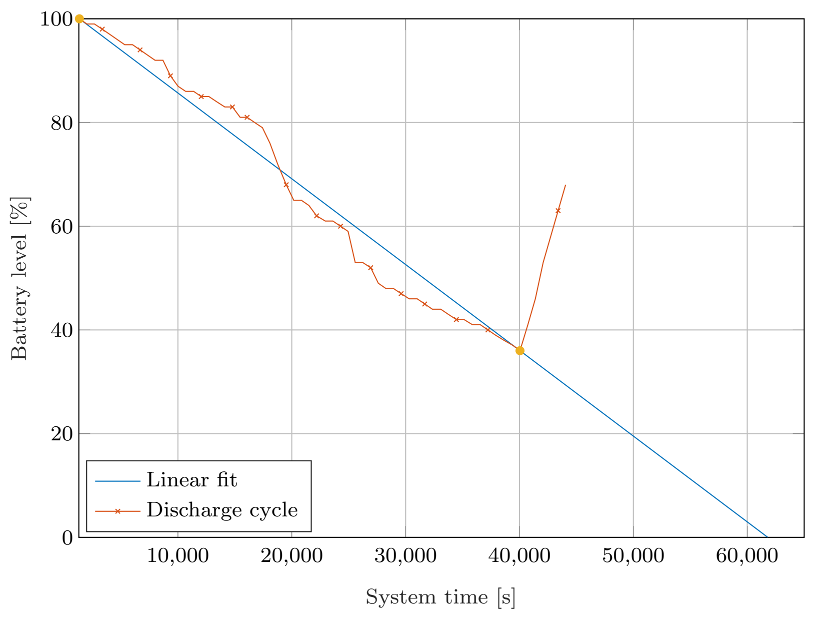


Fig. 3.7: Example of battery discharge rate estimation.

common for modern devices [72]. The linear estimation of the depletion time is a limit to the prediction accuracy, but the limit is inherent in the dataset.

Moreover, we observe that the estimate of the residual battery charge duration is more accurate when it becomes more useful, i.e., when the battery level is low.

Input features Our model considers multiple inputs in order to estimate the remaining battery lifetime, namely:

- Battery level: the current status of the battery is clearly essential for a meaningful prediction.

- Time of the day: this parameter is essential to allow the model to learn and recognize specific battery usage patterns during the day. For instance, a user may make a more intensive use of the smartphone while resting in the evening, rather than during the day, or *vice versa*. To capture these aspects, battery usage data need to be coupled with time information.

- Day of the week: along the same rationale, this feature can make the NN able to recognize patterns of battery usage that depend on the day of the week. For example, sports activities regularly practiced during the week may be associated to a lower usage of the smartphone (e.g., when left in the locker) or to a higher usage (e.g., due to activity-tracker applications), depending on the user's habits.

- Location: this feature can also discriminate different battery consumption patterns, which are often correlated with the user's location. The dataset includes data regarding the user movements and labels for each place that was visited. The locations are classified in 12 categories, such as home, workplace, leisure place, and so on.

- Movement: finally, we consider a boolean feature indicating whether the user is moving from one place to another or is static. This allows us to identify activities typically performed by users while moving on foot or with public transportation, e.g., phone calls or web browsing.

We advocate that such features should be sufficient to discriminate among the specific usage patterns of the different users, thus making it possible to customize the prediction engine.

Preprocessing To efficiently train the machine learning models, we first need to preprocess the input data [93]. In particular, we standardize features and outputs by removing the mean and scaling the amplitudes to get unit variance. By doing so, the distribution of individual features is close to the normal distribution with zero mean and unit variance.

The performance of these methods has been evaluated in terms of the coefficient of determination R^2, which is a very popular criterion to measure the performance of statistical models [94]. Denoting by y_i and $\hat{y}_i$ the actual and predicted values of the ith output, respectively, and by $\bar{y}$ the mean of the actual values, then the coefficient of determination is defined as

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2} \,. \tag{3.3}$$

Note that $R^2 \leq 1$ and the fraction gives the mean squared estimation error over the variance of the outputs. A perfect predictor will yield $R^2 = 1$, while a "dummy" prediction that always returns the expected value of the output, disregarding all inputs, would yield $R^2 = 0$.

In our tests, a NN with at least one month's worth of training data always outperformed all other methods and thus turned out to be the most suitable tool for battery discharge prediction. The parameters of the NN were chosen by performing an exhaustive search in the parameter space; each setting was evaluated by cross validation, and successive rounds refined the search space until convergence. The explored parameters were the size and shape of the hidden layers of the network, the activation function of hidden neurons and the regularization parameter α. We always used the "L-BFGS" [95] solver, an optimizer in the family of quasi-Newton methods, with 200 training iterations. Table 3.2 shows some of the selected configurations; the identity activation function is the simple $f(x) = x$, the logistic function is given by $f(x) = 1/(1 + e^{-x})$, and the Rectified Linear Unit (ReLU) is $f(x) = x$ if $x > 0$, $f(x) = 0$ otherwise.

We also report the R^2 score on the training and validation sets. The activation function and network structure strongly impact both the network prediction performance and the time required to train the network. The time

Architecture	α	Time [s]	R^2 (val.)	R^2 (train)
(100, 100, 100, 100 - identity)	0.01	2.479	0.036	0.040
(300 - logistic)	1e-05	2.212	0.036	0.040
(100, 100, 100, 100 - logistic)	0.01	1.064	0.000	0.000
(300 - ReLU)	0.001	40.875	0.580	0.738
(100, 100, 100 - ReLU)	0.01	54.795	0.656	0.842
(100 - tanh)	0.01	17.947	0.568	0.699
(300 - tanh)	0.01	57.476	0.529	0.642
(100, 100, 100 - tanh)	0.01	76.726	0.754	0.853
(200, 150, 100, 50 - tanh)	0.12	111.439	0.806	0.904

Table 3.2: Performance of the NN for different parameter settings. The architecture column contains the number of nodes in the hidden layers of the NN and the activation function.

required to perform the prediction is not listed in the table, since it is always lower than 100 μs.

We found that a distribution of the neurons across several layers allowed the model to detect more complex battery usage patterns, yielding in general better performance, as reported in Table 3.2. On the other hand, increasing the number of neurons does not always have a positive impact on the test score, and usually requires longer training phases (see rows 6 and 7 in Table 3.2).

The model chosen after cross validation consists of a NN with 4 hidden layers of 200, 150, 100 and 50 neurons each, respectively, shown in the last row of Table 3.2. The best performing activation function was the hyperbolic tangent function, i.e., $f(x) = \tanh(x)$. In order to avoid overfitting, the regularization parameter α has been set to the quite high value of 0.12.

Data processing, model training and performance evaluation have been carried out by using the Scikit-learn Python framework [96]. The test set corresponds to 10% of the available data for each user.

3.2.3 Results

We compare our prediction model with a baseline method that is very similar to the estimation that is adopted in today's smartphones. This deterministic method consists of taking the last 5 samples of the battery level history and obtain a linear approximation of the discharge pattern via Least Squares linear fit [90]. The linear approximation is then used to estimate the remaining charge duration (basically, the same method we used to extrapolate the charge lifetime in our training dataset). We also compare the performance of our NN with SVMs, k-NN regression and Linear Regression using the same input features and training/test sets. The plots shown in Figure 3.8 confirm that, for all the 6 users of the considered dataset, the machine learning models in general have better performance than the deterministic baseline method and, among the considered machine learning techniques, the NN achieves the best performance.

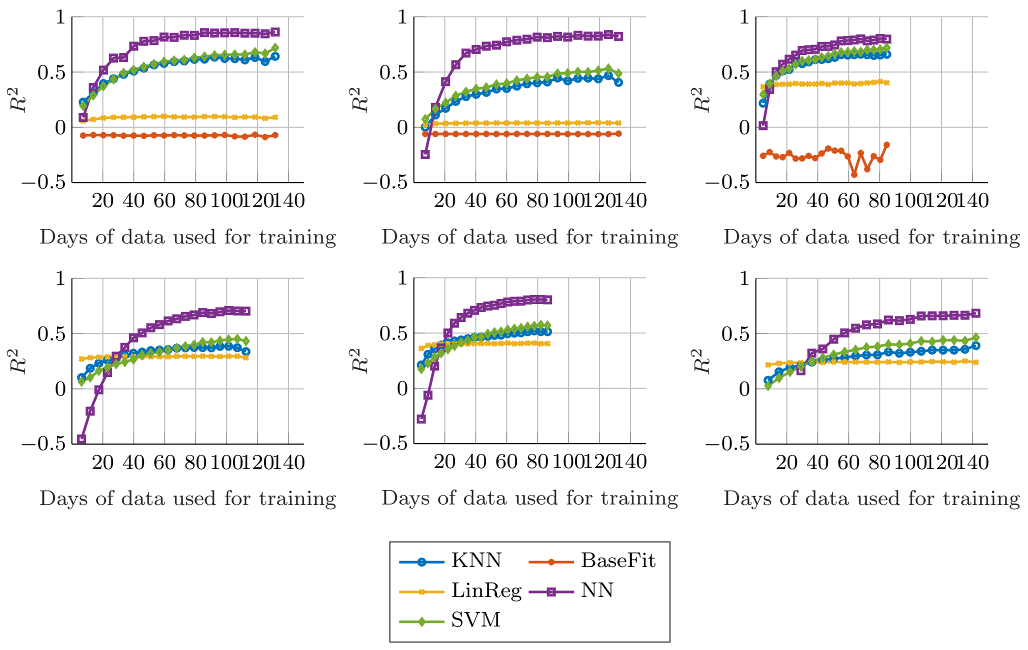


Fig. 3.8: Comparison of the performance of several models by averaging results from 20 random training/testing subsets.

Prediction performance and generalization The fully trained NNs reached a mean coefficient of determination of 0.7835, with a peak of 0.8620 for a user whose dataset was particularly clean, with fairly regular battery usage patterns.

As Figure 3.8 shows, the deterministic battery estimation paradigm is inefficient, as it bases its prediction only on the recent battery history, without considering more general patterns which can instead be accounted for by using location and time-based information that is available at almost no cost in every modern mobile device [97]. The clear advantage of such a model, however, is that it does not require any training.

In order to test the generalization and discrimination capabilities of our model, we trained the NN using the data of a single user and tested its prediction performance on the data of any other user. The results of this test are shown in Table 3.3, where each row reports the R^2 obtained for the different users when the network was trained with the data of the user in the corresponding row. The last row reports the results obtained when training the network with a mix of the data of all the users. It is clear that the NN generally performs poorly if trained on a user and tested on a different one, while the performance is good when training and test data are from the same user[3]. Moreover, the low performance values reported in the last row of the table confirm that our model must be trained only on personal data, otherwise it cannot learn accurately a user's personal patterns nor provide reliable predictions. This proves that our model can learn a user's behavior properly but can hardly generalize to other users. In other words, the learnt model is strictly personal and dependent on the user's specific habits.

As a consequence, the model needs to be trained with local information that can be collected only by the smartphone of each user, so that it is not possible to pretrain the network and deploy it on a general device; the NN must be trained while the device is being used by its owner and this process requires some time, as discussed later. On the other hand, the NN does not need to share information with other users or with service providers in order to be trained or provide the prediction. Considering the growing concerns for the user's privacy in location-based services [98] and the decreasing willingness of the users to share location information [99], these features of the proposed approach are desirable.

		Test user					
	ID	1	2	3	4	5	6
Train user	1	0.643	-0.0733	-0.293	-2.130	-3.508	0.642
	2	-0.268	0.645	-0.581	-4.083	-6.479	-0.048
	3	-0.195	-0.074	0.687	-2.490	-3.865	0.514
	4	-0.121	-0.068	-0.220	0.664	-3.173	0.714
	5	-0.110	-0.067	-0.223	-1.715	0.703	0.729
	6	-0.070	-0.101	-0.423	-0.335	0.670	0.761
	All	0.308	0.371	-0.167	-4.693	-3.725	-0.711

Table 3.3: R^2 values of NNs with about 3 months worth of training data. Each column reports the results of the testing on a different user, while the rows specify the user on which the training is performed. For the last row, the training is carried out by mixing data from all six students.

[3]Disjoint portions of each dataset are used for training and testing.

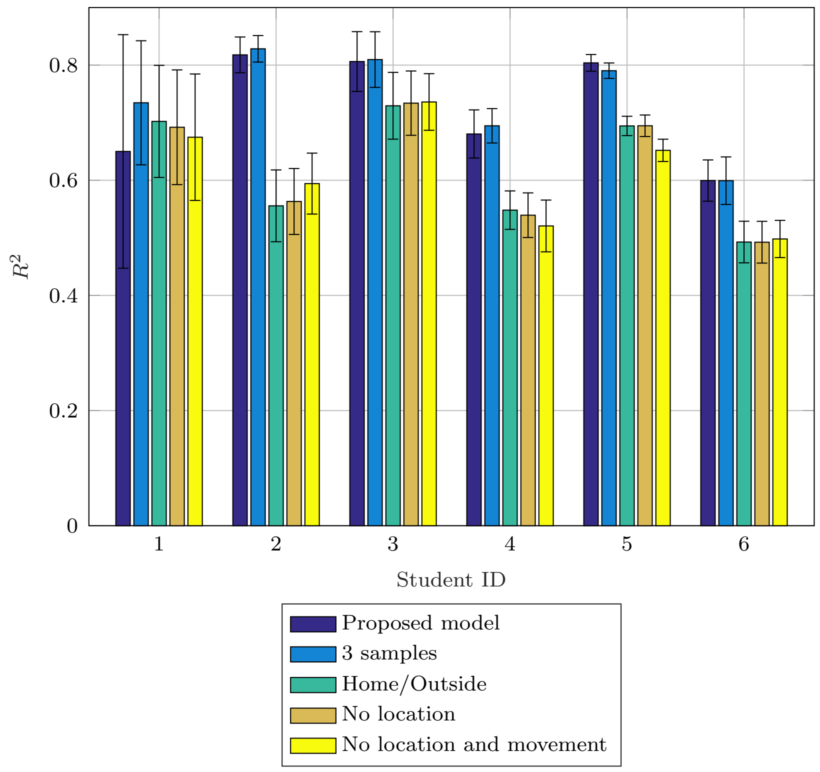


Fig. 3.9: R^2 score and 95% confidence intervals on test sets of NNs with different input vectors for every user (three months of training data).

Of course, during the training phase of the NN, the prediction of the battery charge lifetime can be obtained with other approaches, in order to guarantee the availability of the service to the user: when the training is complete, the prediction will become more precise, being tailored to the user's habits.

Effect of incomplete input on the prediction Our input vector consists of the current battery level, the time of the day, the day of the week, the location of the user, and the motion information. In order to gain insights on the relevance of the different input features on the prediction accuracy of the NN, we considered other input vectors, namely a larger input vector obtained by enriching the Proposed one with the last three samples of the battery level; then a smaller input vector with a boolean value "home/outside" in place of the full 12 location categories; then an input vector, named "No location",

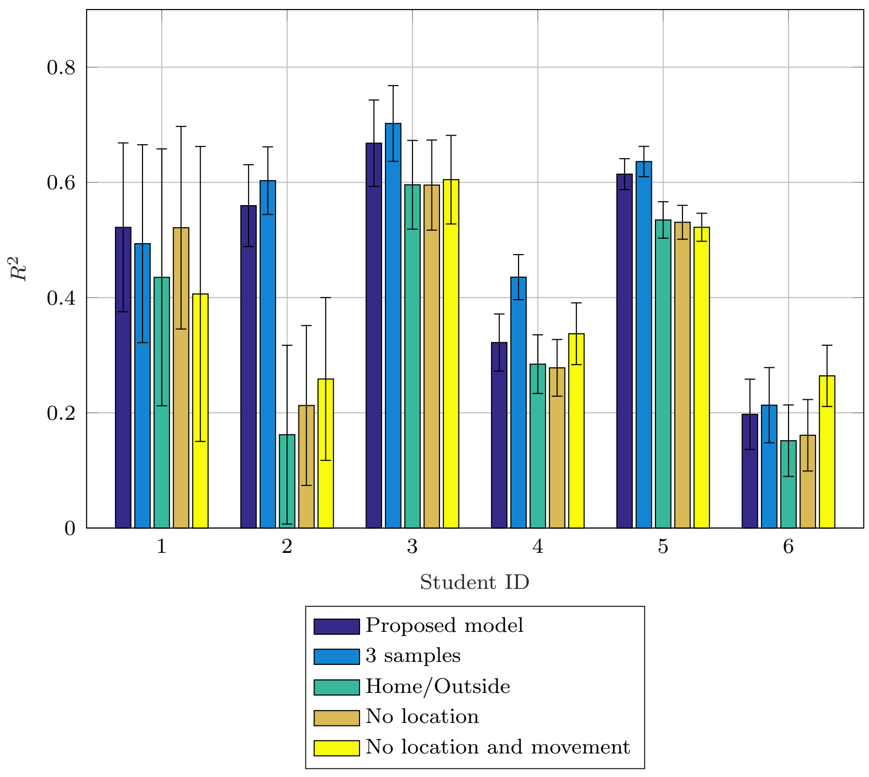


Fig. 3.10: R^2 score and 95% confidence intervals on test sets of NNs with different input vectors for every user (one month of training data).

without any location-based data; and finally an input vector without location and movement data, called "No Location and movement." Figures 3.9 and 3.10 show the R^2 score obtained for the different test users and for each of these input vectors, when considering one month (left-hand side plot) or three months (right-hand side plot) of training data.

We observe that, with one month of training data, the location-based NNs outperform the models with incomplete or no location data available for students 2, 3 and 5, while for the other subjects the different input vectors yield comparable results. However, with three months of training data, the advantage of a location-based input vector becomes more evident for all the subjects. Without location information, the "movement" feature becomes useless or detrimental: if the NN has no information on users' location, it should just disregard movement information. We can also notice that, in general, considering more than the last sample of battery level only brings

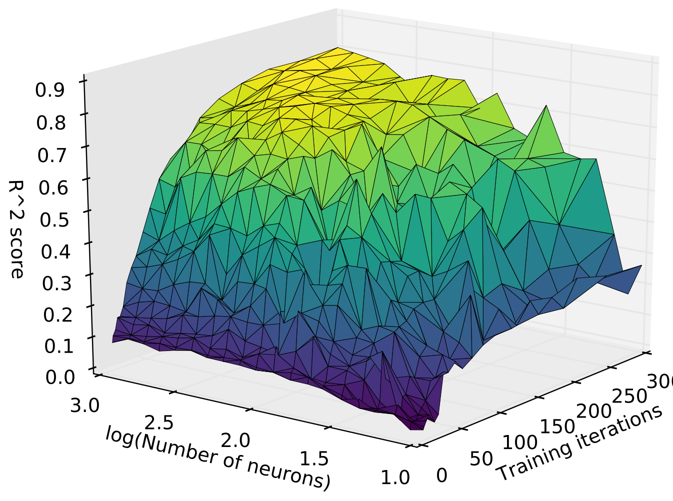


Fig. 3.11: R^2 metric for the NN as a function of model complexity, expressed as the logarithm of the total number of neurons and the training time in epochs.

small improvements in terms of R^2. Furthermore, from the confidence intervals in Figures 3.9 and 3.10 we can see that a larger amount of training data improves the stability of the prediction and reduces its sensitivity to noisy input data, as expected.

Computational complexity of the prediction Finally, we investigate the impact on the R^2 metric of the number of neurons and training iterations; a network with 500 neurons is a good trade-off between computational complexity and reliability of the model. Figure 3.11 shows the results of one of our experiments. Note that with as few as 10 neurons and 100 training iterations, the NN already outperforms the common predictors of mobile devices that always yield negative R^2 score during our experiments (see Figure 3.8). However, increasing the size of the model makes it more robust to variations and noise in the training step and improves its performance with respect to the other ML algorithms we tested. On the other hand, the increased computational costs of very large NNs have diminishing returns, as Figure 3.11

clearly shows: the NN we used offers a good trade-off between prediction accuracy and computational complexity.

The computational complexity of the model is reasonable. A training step of 200 epochs on 144 points, that represent a day's worth of collected data, requires 2 seconds of CPU time for training and this can be easily performed while a device is charging or may be relegated to a moment when the phone is idle. A single evaluation of the remaining battery lifetime requires less than 100 μs on a modern CPU, which makes this model highly portable. Therefore, the proposed model can be used locally, on the users' smartphone, and does not need computational offloading on a remote server, confirming its privacy-oriented character.

3.3 Predicting future cell load

Cell load is one of the most studied factors in the literature, and there are several algorithms to predict it. The novelty of our work with respect to previous studies is that we consider machine learning techniques that exploit temporal and spatial data jointly: a cell's future load depends not only on its previous values, but also on the loads of neighboring cells. This joint approach can improve the prediction accuracy, especially in the noisiest and most challenging cases. We focus on medium-term prediction with a range of tens of minutes; such a range is still usable for network optimization, but is not as noisy and unpredictable as short-term cell load.[4]

3.3.1 State of the art

In the scientific literature, cell load prediction techniques are studied because of the potential gain they can provide to the performance of the network in a wide range of scenarios, such as energy efficient communications and dynamic network planning. In [101] the authors propose to use prediction techniques based on traffic matrices collected for groups of BSs under the

[4]The work presented in this section was presented at MOCAST 2017 and published in the conference proceedings [100].

same coordinator in order to optimize the sleeping time of network elements, while in [102] a classification and prediction method is applied to temporal information given by Call Data Records in order to decide when and where it is appropriate to deploy femtocells. The spatio-temporal relation between cells is analyzed in [103], where insights on the predictability of the traffic in a cellular network are given; however, the authors do not attempt to predict future values of the cell load, but use large-scale traffic patterns to examine the correlation. The study in [104] uses traffic variations in cell neighborhoods, using a Markov decision process model, in order to enable energy saving techniques. There are other studies that consider the spatio-temporal context in cellular networks, but their focus is on the prediction of mobility of users [105, 106]. These can be then exploited in association with some knowledge of the network topology, as done in [107].

3.3.2 Spatio-temporal prediction

All the techniques we present are based on the exploitation of spatio-temporal data, which was first proposed by Ohashi *et al.* [108]. In order to jointly consider the spatial and temporal data, we need to define the concept of *spatio-temporal neighborhood*. If a cell at a given instant is characterized by its position in space and time, given by the vector (x, y, t), we define the distance between two points as

$$d_{i,j} = \sqrt{\left(\frac{x_i - x_j}{d_0}\right)^2 + \left(\frac{y_i - y_j}{d_0}\right)^2 + \alpha\left(\frac{t_i - t_j}{T}\right)^2}, \tag{3.4}$$

where d_0 is the inter-cell distance and T is the time interval between measurements. Note that the spatio-temporal distance between different instants is non-zero even if the cell is the same, i.e., the spatial distance is 0. The parameter $\alpha \geq 0$ is a weighting factor to combine the spatial and temporal measures.

The spatio-temporal neighborhood of a point m can then be defined as the set of the discrete points in the dataset whose distance from m is smaller than some radius β:

$$N_m^\beta = \{p : d_{m,p} < \beta\} \,. \tag{3.5}$$

The points belonging to the spatio-temporal neighborhood are contained in an ellipsoid in space-time, and, given the same β, a smaller α includes in the neighborhood points which are further away in time. The cell load values z_p of the points within the neighborhood can be used in the prediction. In addition to the pure values, we also use as input a series of indicators that capture some of the most relevant dynamics of the cell load, as in [108].

We implemented three indicators, which are listed below:

- The *weighted mean* is an average of the cell load values in the neighborhood, weighted by their spatio-temporal distance, and is given by:

$$\omega(N_m^\beta) = \frac{1}{|N_m^\beta|} \sum_{p \in N_m^\beta} \frac{z_p}{d_{m,p}} \tag{3.6}$$

- The *spread* is the standard deviation of the cell load values in the spatio-temporal neighborhood:

$$\sigma(N_m^\beta) = \sqrt{\frac{1}{|N_m^\beta|} \sum_{p \in N_m^\beta} (z_p - \bar{z})^2}, \tag{3.7}$$

  where $\bar{z}$ is the arithmetic mean of the cell load of all the points in the neighborhood.

- The *weighted tendency* is given by the ratio between the weighted means with two radii $\beta_1 < \beta_2$ (following [108], we choose $\beta_2 = \beta = 2\beta_1$):

$$\tau(N_m^{\beta_1,\beta_2}) = \frac{\omega(N_m^{\beta_1})}{\omega(N_m^{\beta_2})}. \tag{3.8}$$

  This indicator summarizes the trend of the cell load as it approaches the target location. For example, if $\tau(N_m^{\beta_1,\beta_2}) > 1$, then the load on the closest points in time and space is larger than that of farther points.

While in [108] the indicators are added to a purely temporal prediction,

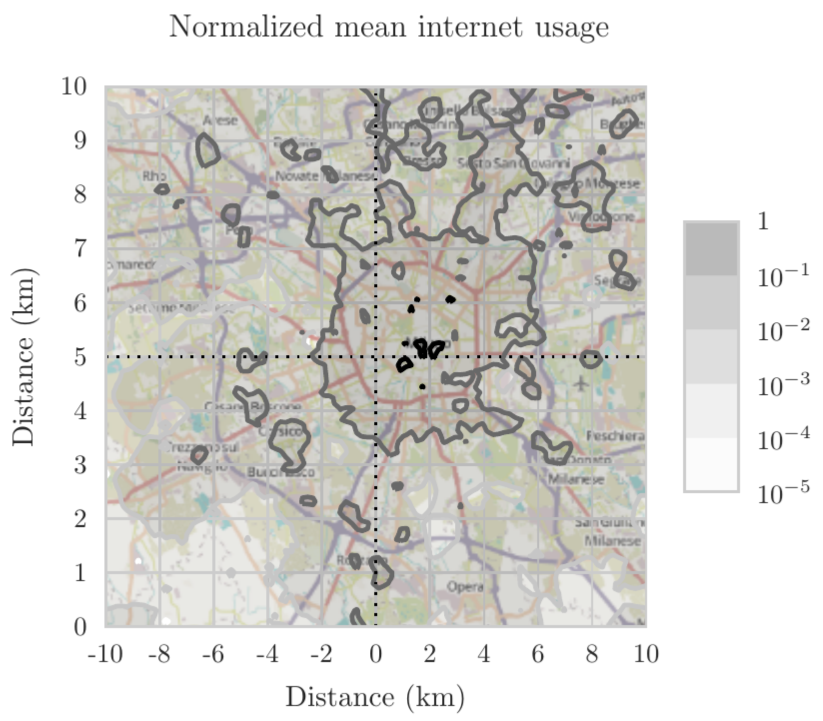


Fig. 3.12: Normalized average Internet usage map.

in our work we also use the cell load values of all the points in the spatio-temporal neighborhood as predictors.

In this example, we tested the performance of all the algorithms described in Section 2.1, except for Kalman filters: along with several regression techniques, we used SVMs, RF and NNs.

3.3.3 Parameter optimization and results

All the prediction methods we described above were trained and tested using the *Telecom Italia Big Data Challenge 2014* dataset,[5] which contains the records of the Internet usage for a grid of square cells with 200 m sides – which makes $d_0 = 200$ m in (3.4) – in the city of Milan, Italy, for the last two months of 2013. The data had a sampling period of 10 minutes, i.e., $T = 600$ s in (3.4). The normalized mean Internet usage is overlaid on a map

[5]`https://dandelion.eu/datamine/open-big-data/`

of the city in Figure 3.12.

For computational reasons, we only predicted the load of a small but representative subset of cells, namely, the cells with id 2583, 4241, 4856, 5060, 5091, 5259, 5262, 6065 and 7724. These cells were selected because they are placed in different areas of the city and they show different traffic patterns. In particular, cells 2583 and 4241 have an average traffic that is close to the average traffic for the whole city, cells 5060, 5091 and 7724 show very high peak usage, and cells 4856, 5259, 5262 and 6065 have a very high average traffic.

The metric we chose for the results was the coefficient of determination R^2 [109], which has been described in Section 3.2.2.

Parameter optimization All the parameters of the prediction algorithms were optimized by exhaustive search with 10-fold cross-validation, after dividing the dataset into training, validation and testing sets. The chosen values of the parameters are listed in Table 3.4.

The values of the spatio-temporal weighting factor α and of the neighborhood radius β were optimized for each cell and are listed in Table 3.5, for a number of neighbors from 27 to 46.

Parameter	Value	Description
λ_R	[1.637e-6, 0.074]*	Ridge regularization parameter
λ_L	[1e-06, 4.665e-6]*	Lasso regularization parameter
$\lambda_{R,E}$	[0, 1.105e-5]*	Ridge regularization (elastic net)
$\lambda_{L,E}$	[0, 4.665e-6]*	Lasso regularization (elastic net)
C	[0.22, 34.081]*	SVR linear kernel penalization term
N_t	200	Number of RF trees
γ	10^{-3}	NN learning rate
N_{iter}	10^4	Maximum NN iterations
ε	10^{-10}	NN convergence tolerance

*These parameters were optimized for each cell.

Table 3.4: Parameters used in the simulation.

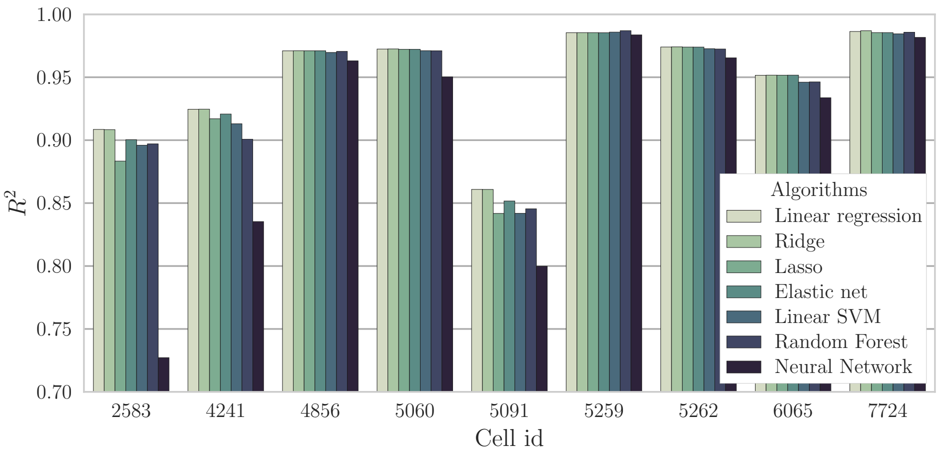


Fig. 3.13: Performance of the tested regression methods.

Prediction results Figure 3.13 shows the prediction accuracy on the test set for each regression method. The figure clearly shows that the NN is not an accurate method, probably due to an insufficient training set size, whereas the other algorithms often have a similar performance. The reason is that the cell load can be easily predicted in most cells, and therefore the differences among different algorithms are minimal. On the other hand, in cells with poor prediction accuracy different methods show some performance difference. This reveals that, when the behavior of the load in a cell is less predictable, the prediction performance can be improved using different algorithms and additional context information. Indeed, the simple linear regression and ridge regression have a slightly better performance in cells 2583, 4241 and 5091,

Cell id	α	β	Number of neighbors
2583	0.25	2	27
4241, 4856	2.25	3	25
5060	0.09	2	46
5091	0.19	2	28
5259, 5262, 6065	0.12	2	37
7724	0.19	2	28

Table 3.5: Optimal neighborhood definition for each cell.

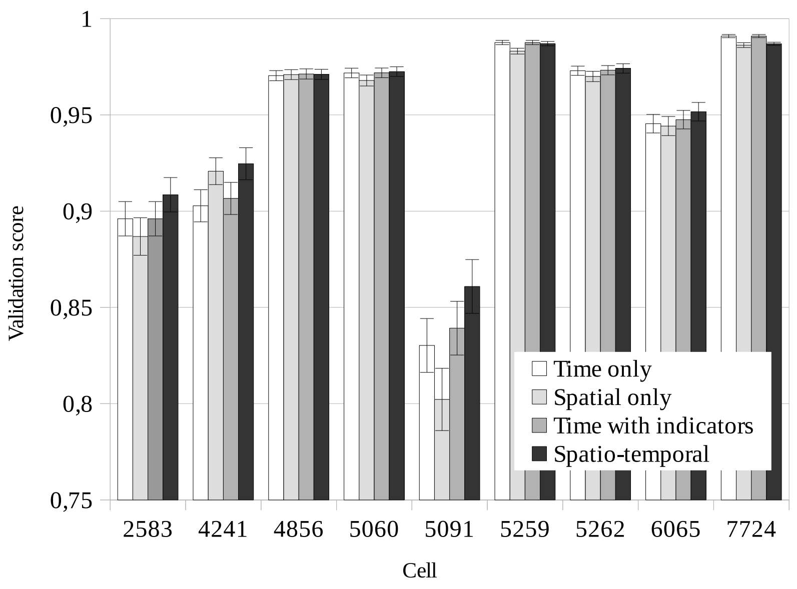


Fig. 3.14: Performance of the prediction algorithms for different neighborhood definitions.

which are all located in peripheral areas of the city, close to major traffic roads or hubs (Via Gianbellino for cell 2583, the A1 highway for cell 4241, and Linate airport for cell 5091). In locations like these, with high mobility and bursty traffic, the benefit of combining spatial and temporal information is intuitive, and the performance improvement can be seen in Figure 3.14. While only temporal or spatial data are sufficient in the highly predictable cells, the same 3 cells mentioned above show a marked improvement in the R^2 score when spatio-temporal data are considered jointly. Finally, the use of temporal indicators does not result in a significant improvement by itself, but only when combined with the spatio-temporal neighborhood data.

The most accurate prediction methods are also the simplest: both training and parameter optimization for the linear, ridge, lasso and elastic net algorithms were significantly faster than for RF, SVR and NN. This offsets the increased complexity due to the bigger size of the neighborhood caused by the inclusion of the spatial dimension in its definition.

Chapter 4

A predictive approach to providing Quality of Service

As we mentioned in Chapter 1, video content distribution is an application which currently faces difficult challenges in providing services to mobile users: live video conferencing already exists, but high resolution video services still suffer from glitches and frozen screens in wireless networks. Another rapidly developing application is AR, which is based on overlaying data and graphics over a live video of the real world, allowing users to use smartphones or headsets to access information seamlessly from the environment. Looking further ahead, the visions of the Tactile Internet [110] and the Internet of Skills [111] promise ultra-realistic immersive Virtual Reality (VR) telepresence using phones and headsets.

Vehicular networking is another field that might greatly benefit from faster, more reliable wireless connections: smart vehicles will soon need to send three-dimensional maps of their environment to remote servers for processing and receive the relevant safety commands with extremely tight delay constraints, and autonomous cars and drones will need to communicate even more to navigate the environment and coordinate with each other, e.g., using video from multiple vehicles to be aware of things happening behind corners and obstacles and avoid dangerous blind spots.

The problems these applications currently face are caused by three requirements common to all of them: they are throughput-intensive, they need an extremely low end-to-end latency, and they require the data delivery to be highly reliable. Although the network infrastructure will soon be able to support capacity requirements, thanks to the densification of cellular networks and the adoption of mmWave technology in 5G and to the deployment of WiGig [112], the usage of the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) in the transport layer of the network stack excludes the support of latency and reliability guarantees, even when using new congestion control schemes such as Google's Bottleneck Bandwidth and Round-trip propagation time (BBR). The reason is that standard (TCP) congestion control is almost always focused on maximizing throughput, with a high self-inflicted queuing delay by design.

Full QoS control, including both throughput and latency, can not be achieved without evolving from best-effort transmission of application data to a delivery with pre-defined application-specific targets, defined in terms of minimum throughput and maximum latency with a given reliability (e.g., receiving at least 95% of the packets with a delay of less than 50 ms). In order to do that, we need to be able to consistently and correctly predict the future conditions of an end-to-end channel, as well as exploiting multiple paths to increase the reliability and overall capacity.

The inadequacy of current transport protocols is clear from a review of the relevant literature: a recent measurement study on commercial video conferencing systems [113] has shown that latency in video calls is unacceptably high in most practical situations. A number of over the top solutions [114, 115] have tried to introduce coding and additional rate control at the application layer to limit it, but the limits of these workarounds show the urgent need for a transport layer solution designed to reliably provide both high throughput and bounded delay. Most of the literature on AR also shows the strong negative effect of delays over 200 ms on users' performance on simple tasks [116] and enjoyment of overlay applications [117]. The same considerations are common for both drones [118, 119] and smart and autonomous vehicles [120, 121], for whom the possibility to transmit potentially large amounts of data with very strict latency requirements would represent a significant safety improvement.

The ongoing virtualization of 4G cellular networks and the software-defined 5G architecture are making the deployment of new protocols technically possible [122]. Network operators can place server-side and user-side proxies that will implement such new transport protocols in a backward-compatible manner. For example, in [123], the authors show how a user-side proxy can be deployed in an unmodified user handset without any changes to the operating system kernel by using standard Virtual Private Network (VPN) framework. Such software-defined architectures enable flexible control of the trade-off between reliability, throughput and latency based on application/user-specific inputs. The trade-off consists in the fact that given a fixed latency, the requirement of a higher minimum rate implies a less reliable on-time data delivery, and vice versa. Increasing the latency target allows the protocol to increase the achievable data rate without compromising communication reliability.

In this context, we present the Latency-controlled End-to-End Aggregation Protocol (LEAP), a multi-path transport-layer protocol that provides end-to-end performance guarantees thanks to multiplexing parallel connections. LEAP is motivated by the fact that the multiple parallel connections can be theoretically used to create a logical connection whose capacity is the sum of the individual paths', with no negative effects on latency [124, 125], but the limitations of Multi-path TCP (MPTCP) (even with optimized congestion control schemes such as the Balanced Link Adaptation (BALIA) algorithm) make it unsuitable for this purpose. Standard smartphones are already equipped with at least LTE and WiFi interfaces, and new 5G and WiGig interfaces are expected to be added in the years to come. For backward compatibility reasons, LEAP can operate over a single link, in which case it still outperforms other latency-controlled protocols in terms of achievable capacity.

More specifically, LEAP exploits multi-connectivity and network coding to achieve this objective; a QoS-constrained data flow can be split into several subflows delivered over multiple parallel connections, e.g., WiFi and LTE, or WiGig and 5G in the near future. The send rate of each subflow is adaptively controlled to avoid congestion (long queuing delays, more specifically) and meet the required latency and reliability constraints. In addition, each subflow carries redundant encoded information generated from payload data of all parallel subflows: by adding the appropriate amount of redundancy, excessively delayed or lost data on one subflow can recovered without the need for retransmissions.

In order to determine the right amount of redundancy and avoid queuing delay, LEAP pro-actively predicts the future QoS-constrained capacity of each path. The forecasts are done in probabilistic terms to address the stochastic nature of the transmission capacity of network paths, which is particularly volatile in case of wireless links: the more accurate the forecasting algorithm, the higher the utilization of available channel capacity and the lower the redundancy overhead. Congestion control is performed separately on each path based on the forecast of future capacity, but the packets are coded across paths.

Extensive measurements in live networks and comparisons to the legacy protocols presented in Section 4.1 demonstrate that the proposed scheme can efficiently balance the fundamental trade-off between the latency, throughput and reliability requirements in a smooth and consistent manner over a wide range of QoS constraints, giving a large operational flexibility to applications.

LEAP can be set up to work at different points in the trade-off, depending on whether the first priority is throughput or latency control. In this way, it is possible to create the illusion of a reliable connection with a guaranteed bounded latency by combining several unreliable links. A similar scheme has been studied in [126] for UDP-based video streaming, in a simple scenario with a constant and known capacity and a known packet loss probability. Salsify [127] is another video protocol that integrates a video codec with a (single-path) transport protocol that guarantees a bounded latency in normal conditions; it is particularly significant because the integration between application and transport layer is similar to what we aim to achieve.[1]

[1]The results presented in this chapter were presented at IEEE INFOCOM 2018 and published in the conference proceedings [128], and an extended version of the conference paper has been submitted to the IEEE Transactions on Networking. There is a pending patent on the novel aspects of the protocol [129].

4.1 State of the art

Since the Internet's inception, the main design goal of network protocols has been throughput maximization, TCP being the prime example. Its congestion avoidance mechanism uses buffer overflows at the bottleneck to infer congestion and back off from its probing of the channel, exploiting most of the available capacity. However, as the only signal of congestion is a buffer overflow, TCP responds slowly to changes in the network and suffer from high self-inflicted queuing delays and instability [130].

For example, the default version of TCP on Linux hosts is Cubic [131]: its main benefit over previous iterations is that it achieves long-term fairness even between flows with different Round Trip Times (RTTs), as its congestion control function is not ACK-clocked. However, TCP Cubic suffers heavily from bufferbloat and self-inflicted queuing delays, as it is entirely buffer-based.

The inefficiency of loss-based congestion control has only been exacerbated by the progressive increase in the size of buffers in cellular networks. Bufferbloat [132] is a well-studied phenomenon, and TCP queuing delays can often be measured in seconds; naturally, this is not acceptable for the future applications with very strict delay constraints we discussed in the previous section. The presence of large buffers also leads to throughput instability [133], and the deployment of Active Queue Management (AQM) strategies, touted as the solution to the issue, has proven to be fraught with complications [134]; later studies have also expressed doubts about performance benefits [135].

4.1.1 Single-path latency-minimization protocols

One of the first attempts to enable TCP awareness to latency consists in TCP Vegas [136]: instead of inferring congestion only when a packet gets lost because of a buffer overflow, Vegas tries to keep its RTT close to the minimum measured value. The Vegas congestion control mechanism is more conservative than Cubic, as it infers congestion from increases in the RTT and reduces its transmission rate before it builds up a significant queue.

While Vegas arguably leads to a better QoS than Cubic or the traditional TCP Reno, which are both buffer-based and capacity-oriented, it has never been deployed on a large scale because of the unfairness problem [137]: since Vegas flows react far earlier to congestion than TCP Reno or Cubic, their throughput is disproportionately lower in shared channel. The fairness issue is almost always the biggest obstacle to the deployment of less aggressive protocols, as the ubiquity of buffer-based TCP versions gives any such scheme an inherent disadvantage. Another issue is the dependence of the protocol on the quality of the minimum RTT estimation [138]: if its estimate is incorrect, the Vegas sender will behave suboptimally.

The BBR congestion control algorithm [139], developed by Google, is another flavor of TCP that tries to keep right on the edge of congestion by using increased delay as a congestion signal, periodically emptying the queue and updating its estimate of the minimum RTT to mitigate the estimate quality issue. BBR is more aggressive than Vegas, and can fairly share a connection with Cubic flows. However, it cannot provide delay guarantees; it also exhibits most of the other limitations of Vegas, with which it shares the basic underlying principles.

Sprout [140] is a rate-based congestion control mechanism developed in 2013 by Winstein *et al.*; it uses a HMM to estimate the available capacity, modeled as the rate of a Poisson process. By counting the received and acknowledged bytes in a fixed slot, it can track the capacity of an unreliable channel and transmit as much as possible without increasing its own trans-

Protocol	Throughput	Latency control	Complexity	Multi-path gain
TCP Cubic	High	Poor	Low	Low
TCP Vegas	Low	Good	Low	Low
MPTCP BALIA	Medium	Poor	Low	Medium
Sprout	Low	Good	Medium	Low
Verus	High	Good	High	Medium
BBR	High	Good	Low	Low
LEAP	Adaptable	Good	Medium	High

Table 4.1: Quick comparison of the considered protocols

mission delay. Sprout can be tuned to be more throughput-oriented or more delay-oriented by changing the safety margin on the channel prediction. It provides a reliable performance with high throughput and low delay, but it requires a private buffer: if any other flow is queued in the same buffer as the Sprout flow, it will yield almost a [141]ll the capacity to the newcomer. Furthermore, based on our experiments, Sprout does not seem to be able to efficiently track the capacity of real 802.11 links, even without cross traffic. This limits its applicability with the ubiquitous WiFi technology.

Verus [142] is another low-delay congestion control protocol that explicitly takes into account the bursty nature of wireless networks. Verus maintains an estimate of the delay profile, i.e., the experienced relation between send rate and transmission delay, which is used to forecast the channel capacity. According to our empirical results, it can efficiently use the capacity of 802.11 channels, but it has two major issues: its strong reaction to packet losses, and its computational complexity. The load Verus imposes on a sender to build and maintain the delay profile at every packet acknowledgment is probably untenable in an uplink scenario, and can still be severely taxing in the downlink: our experiments, using a laptop with a 2014 Intel i7-4700QM processor, show that the protocol requires one full core at 20 Mb/s and two at 40 Mb/s, with a linear scaling factor. This makes Verus problematic for high-throughput scenarios, since the delay profile is critical to its adaptation mechanism.

Other congestion control algorithms have been developed over UDP, limiting the application layer rate in order to avoid errors and exploiting the well-known Real-time Transport Protocol (RTP) to send control information. In 2011, Tos and Ayav developed a rate control algorithm [143] that tries to keep the packet loss rate of an RTP flow constant using a Proportional-Integral-Derivative (PID) controller. A similar more recent protocol [144] based on source rate adaptation deals specifically with haptic feedback and control, and it uses a TCP-like algorithm to adapt to congestion and limit delay. However, these works do not explicitly consider latency, and they may incur the same problems as TCP Cubic in terms of queuing delay.

4.1.2 Multi-path aggregation protocols

In general, the above-mentioned protocols for single-path connectivity offer by design best-effort services, as the unpredictability of the capacity offered by network connections makes it difficult to support pre-defined QoS, particularly when wireless channels are involved. The multi-path scenario is even more challenging due to the head-of-line blocking phenomenon [145] - any outage or retransmission on any of the paths can negatively affect all the other paths.

As a result, most current multi-path protocols inherit the best-effort paradigm of their single-path predecessors (see Figure 4.13). Nevertheless, intelligent techniques can be used to compensate for the head-of-line blocking problem and other issues related to multi-flow fairness. For example, MPTCP [146] uses advanced schedulers such as BALIA [147] as well as coupled [148] and semi-coupled [149] congestion control mechanisms to address shared bottleneck scenarios.

To further improve latency, the problem of lengthy retransmissions must be addressed, together with the high unreliability of multi-path sessions due to wireless link heterogeneity [150]. MPTCP with Systematic Coding (SC-MPTCP) is a hybrid solution [124] that uses Forward Error Correction (FEC) to reconstruct missing packets in a MPTCP flow and avoid head-of-line blocking, operating on the multi-path level without changing the underlying TCP congestion control mechanism. Other recent solutions [125, 151] use different coding schemes to achieve the same objective. While these solutions are able to reduce the head-of-line blocking issue, they cannot change the fundamental nature of TCP: self-inflicted queuing can still cause an unacceptable delay in the data transmission.

For a more thorough survey of the research on multi-path congestion control and MPTCP, we refer the reader to [152]. A quick comparison of the protocols considered in this section and their main strengths and weaknesses is presented in Table 4.1.

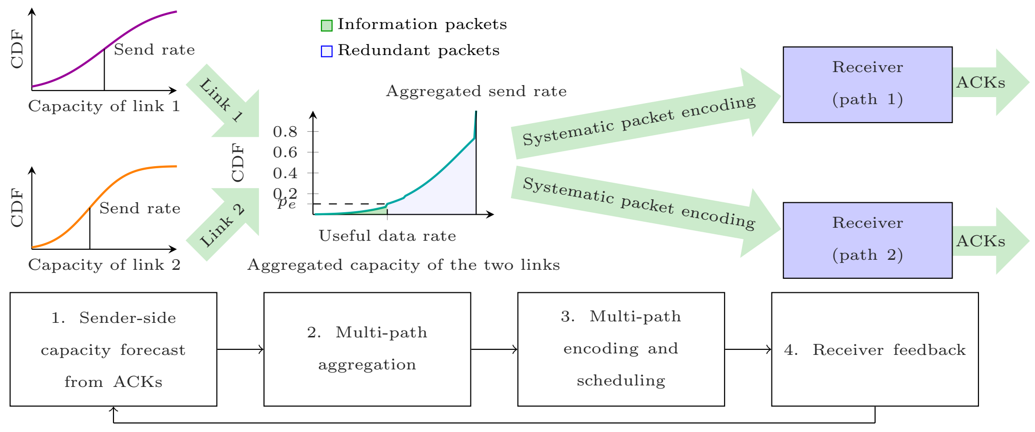


Fig. 4.1: Overview of the basic one-way cross-path coding (hosts can act both as a sender and a receiver)

4.2 The LEAP protocol

The proposed protocol exploits multi-path data delivery and a redundant data format to achieve reliable low-latency data transmission. More specifically, the protocol transmits over each path at the highest achievable rate associated with the pre-defined latency target. FECis used to compensate any violations of the latency constraints such as a packet delay beyond the application deadline due to channel fluctuations or estimation errors. The basic idea of FEC is to send extra redundant packets that contain the information necessary to retrieve any of the original packets that may be missing or late, in order to avoid a lengthy retransmission and immediately send the information to the application layer. In this work, we use packet-level systematic coding, i.e., a code that sends the original packets first and then appends a number of encoded ones to retrieve errors. Systematic codes are functionally equivalent to other linear codes, but have the added benefit of a lower complexity since most of the data will be retrieved without any decoding operations.

The basic assumption when designing the protocol is that an interactive application in quasi-real time, such as low-latency live video streaming or 360°AR with fast switching of viewing angles, will write data to the socket at regular intervals. The application needs to be adaptive, since the low latency threshold requires it not to exceed network bandwidth; applications with a fixed throughput can be downgraded to best-effort service. LEAP can tell the application what the predicted sustainable rate is, and the application can scale down its sending rate (e.g., by using video compression). An application that cannot adapt its sending rate will have to forgo its reliability constraint, as no protocol can guarantee to send a fixed throughput with a fixed maximum latency at a given probability in volatile network conditions; these conditions are similar to the ones applied in [127] on a single path.

LEAP can replace both TCP and UDP: in case the application needs in-order data delivery, a retransmission scheme can be added to cover the losses that the coding is not able to recover from. If the application can tolerate some missing information, LEAP simply delivers the information with the given reliability; as of today, rate control in UDP is left to applications, while our work would provide a reliable framework for them to work with, allowing them to set their constraints and simply follow the rate indications from the transport protocol. In both cases, the objective is to maximize the throughput without violating the latency and reliability constraints: all data packets should be delivered within the deadline T with a minimum probability p_s.

Although FEC has been explored as a way to reduce retransmissions on a single path, with unsatisfactory results [153], the concept becomes radically different on the multi-path level: encoding packets on a single paths only protects the flow from errors, and increases the risk on congestion. If the capacity of the path is unexpectedly low, the redundancy packets will do nothing but slow down the transmission further. However, when a block of packets is encoded across two or more paths the connection diversity factor comes into play: as long as the paths do not share a bottleneck, any loss of capacity on one flow can be recovered by a surplus on one of the others. The presence of FEC packets still increases the overhead of the flow, but it now serves the more important purpose of ensuring the timely delivery of the data, instead of just protecting the flow from random packet losses. However only time-critical data should be protected with FEC, in order to avoid unnecessary congestion; best-effort traffic from activities such as web browsing and bulk data transfer, which do not require strict latency bounds, can be sent using a standard retransmission mechanism.

The logic of the protocol is schematically exemplified in Figure 4.1 for a simpler two-path case:

1. After sending a batch of packets on each of the available paths in slot $n-1$, the sender receives acknowledgments from the receiver and uses the feedback to estimate the *unconstrained capacity* of each end-to-end channel in the next slot n; the details of this estimation process will be clarified in Section 4.2.1, as LEAP operates in discrete time slots, unlike most congestion control mechanisms. Because of the requirements of block coding, packets are sent in discrete batches, which share a common deadline (e.g., a video frame or a HyperText Transfer Protocol (HTTP) object). The natural way to estimate capacity is then on a time slot basis, where the duration of a slot is equivalent to the deadline T. This requirement makes it hard to reuse existing congestion control mechanisms, which are based on a fluid model and react poorly to intermittent source traffic; for this reason, and because we need an estimate not just of the instantaneous capacity but also of its distribution, we will design a new congestion control mechanism that fits our requirements. The size of the next batch of packets on each path is then determined, tracking the evolution of the unconstrained capacity and estimating the probability distribution of the *latency-constrained capacity*, i.e., the number of packets that can be received within the deadline. This distribution is then used in step 2.

2. The distributions of all the available paths are aggregated into a multipath Cumulative Distribution Function (CDF), which can be used to find the correct coding rate. The K original packets are encoded into N packets, where N is the total batch size for all the aggregated paths. The coding rate $R_c = \frac{K}{N}$ represents the efficiency of the channel utilization, so that $1 - R_c$ is a measure of the coding overhead. R_c also determines the strength of the coding, i.e., how many losses can be tolerated while still recovering the original K packets. In our case, the

coding rate is determined by the required reliability threshold p_s (set to 90% in Figure 4.1) or, equivalently, by the error probability $p_e = 1 - p_s$.

3. In the third step, the packets are encoded using a systematic packet-level code, and the coded packets are scheduled and sent over the available paths. The inter-packet interval is computed as a function of the batch size in order to ensure capacity allocation fairness between different hosts.

4. Finally, the receiver gets the packets and periodically sends acknowledgment packets, which are then used by the sender to refine the path distribution estimates and begin the cycle anew. If the receiver suspects that any losses might be due to causes other than congestion, it can report that to the sender explicitly.

The four phases can be implemented independently, as long as their functions are the ones stated above; it is even possible to recycle the TCP decision-making logic and signaling in phases 3 and 4. We now present a more detailed description of our implementation, with a discussion of each phase.

4.2.1 Congestion control on a single path

In the first phase of the protocol logic, we need an estimate of the capacity of each path, along with a congestion control mechanism that can avoid overshooting the path capacity and maximize the packets delivered within the deadline T. In our discrete-time model, packets are sent in batches; the sender transmits $s_i(n)$ packets on path i during slot n[2]. If we assume a minimum RTT $\tau < T$ and that the slot begins at time t, the sender will receive feedback starting from time $t + \tau$. Since FEC works well for tight deadlines but it adds redundancy, if the RTT is lower than $T/2$, we may simply fall back to retransmissions.

[2]From here on, we will omit the slot number from the notation whenever possible to improve readability

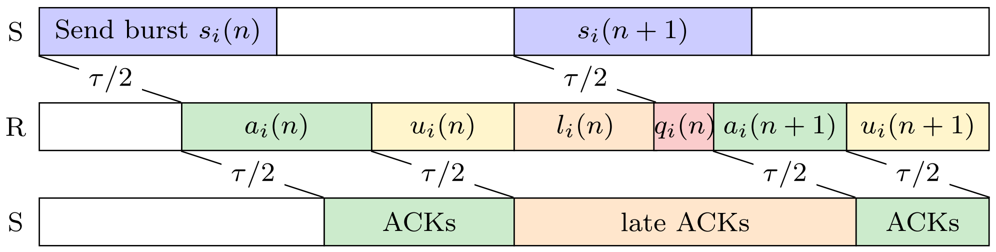


Fig. 4.2: Schematic of the reception of a packet batch

We denote by $d_i \leq s_i$ the number of packets delivered within the deadline, i.e., before time $t+T$. Out of these d_i packets, we indicate by a_i and u_i the number of packets that are acknowledged and unacknowledged, respectively, by the time $t+T$, so that $d_i = a_i + u_i$. The reception and acknowledgment of packets is shown in Figure 4.2, where a_i is indicated in green and u_i in yellow.

A number $l_i = s_i - d_i$ of packets will be delivered to the receiver in the next time interval $\tau/2$ (orange in Figure 4.2), but a certain number q_i may still be in flight when the next batch of packets arrives to the bottleneck (red in Figure 4.2), thus causing a delay its delivery.

Most versions of TCP, including New Reno, Cubic and Vegas, react very strongly to losses [154], as it considers them to be always due to congestion; the consequent decrease in the sending rate makes it underestimate the capacity of links with high losses at the lower layers. LEAP does not explicitly consider lost packets, as it works on a tight deadline: in most practical scenarios, the deadline is too short to completely fill the buffer at the bottleneck and experience losses because of congestion. For this reason, if we have a loss in the middle of the flow, with packets before and after it arriving before the deadline, lower layer factors are the most likely culprits: if routers implement sensible queuing policy, we would expect to see a significant increase in the delay of the packets before any congestion loss, and any packets after the loss would definitely arrive after our deadline. Since we do not want lower layer losses to impact our sending rate, we can disregard any losses in the middle of the flow (i.e., with packets before and after the lost one arriving within the deadline) to be, considering only losses at the end of the flow (i.e., with no packets with a higher index arriving within the deadline). The number of masked losses $e_i(n)$ is included in the ACK packets, and the sender can decide what to do with them: aggressive senders can mask them entirely, while more conservative ones can mask them partially, weighting the number of masked packets by a factor $\lambda \in [0, 1]$. The value $\lambda = 0$ corresponds to the standard TCP practice of considering every loss as a congestion event. Each ACK packet contains three fields: the number of acknowledged packets in the current batch $a_i(n)$, the number of late or missing packets $e_i(n)$, and a timestamp. The capacity estimate $C_i(n)$ is then given by:

$$C_i(n) = \frac{a_i(n) + \lambda e_i(n)}{T - \tau}. \tag{4.1}$$

This instantaneous estimate of the capacity can be extremely noisy, since quick fluctuations in the channel can have strong effect on it; in order to get a reliable estimate of the capacity, we need a filter to remove noise and track the capacity evolution effectively. For this purpose, LEAP uses an adaptive Kalman filter [52] (see Section 2.1.6), which is updated after each deadline T and is based on the feedback $C_i(n)$ returned by the receiver at the end of the slot. The Kalman filter is not necessarily the most precise way to track the path capacity, but it has three important properties that make it a good fit for the purpose: (i) it is computationally light, avoiding the issues of more complex estimators like the Verus delay profile, and fully general and independent of the features of each link in the path; (ii), it returns a probabilistic estimate by design, and that estimate can be used directly in the multi-path block; and (iii), it is entirely transparent to the lower layers, requiring no cross-layer information. Cross-layer mechanisms would improve performance, but they would need to be adapted to each individual technology and, as such, are reserved for future study. In order to apply the Kalman theory, we consider the following update equations for the estimate $\mu_{c,i}(n)$ of the path capacity and for the feedback process:

$$\mu_{c,i}(n+1) = \mu_{c,i}(n) + w_i(n) \tag{4.2}$$

$$C_i(n) = \frac{\mu_{c,i}(n) + v_i(n)}{T}, \tag{4.3}$$

where w and v are two independent white Gaussian processes with zero mean and variance Q_i and R_i, respectively. The process noise w_i tracks permanent changes such as user mobility and long-term cross-traffic, while the measurement noise v models temporary disturbances such as short-lived cross-traffic flows and the fast fading of the wireless channel. The input to the Congestion Kalman Filter (CKF) is $C_i(n)$, as defined in (4.1), multiplied by the slot duration T to obtain an estimate of the distribution of $r_i(n)$. The output of the CKF is a probability distribution, described by its mean $\mu_{c,i}(n)$ and its standard deviation $\sigma_{c,i}(n)$ (the variance of the output of the filter is simply $Q_i + R_i$).

The traditional Kalman filter assumes perfect knowledge of Q_i and R_i, while we track and adapt them dynamically. Since they are not known *a priori* in our case, and they may vary over time as the network environment changes, we use the Least Squares method to estimate them from the estimate error signal's autocorrelation as in [55].

In order for the bottleneck buffer to be stable, the sender cannot transmit at a rate higher than the capacity of the channel. The batch size $s_i(n+1)$ is then effectively limited by the mean $\mu_{c,i}(n)$, as it is the best estimate of the capacity. In order to control the aggressiveness and fluctuations of the send batch size around the estimated link capacity mean, the actual batch size can be set to a smaller value:

$$s_i(n+1) = \mu_{c,i}(n) - \alpha\sigma_{c,i}(n), \tag{4.4}$$

where α is a parameter that controls the aggressiveness of the protocol and can be adjusted dynamically. The use of a non-zero α can help avoid unstable behavior caused by frequently overshooting the available capacity, and future work will include a thorough examination of its effects in different network scenarios.

Since the estimation mechanism needs at least a few packets to avoid excessive fluctuations, a minimum of s_{min} packets will always be transmitted in each slot, irrespective of the estimated capacity; this is equivalent to a baseline link probing in order to get an accurate estimate of the capacity.

The use of a Kalman filter and cumulative acknowledgments is not mandatory for the protocol, but its performance in a real setting was satisfactory, as discussed in Section 4.3.2. A possible alternative is to simply recycle TCP's congestion control logic and signaling: packets are sent over multiple single-path TCP connections, and the coding rate is calculated according to the estimated distribution of the available constrained capacity. In this case, the benefit would be limited, as TCP's queuing delay issues would still limit LEAP's ability to deliver data within the deadline, but its implementation is possible even when a user-side proxy [123] is not available.

4.2.2 Integrating single-path congestion control and multi-path coding

In order to avoid fluctuations and separate the estimate of the capacity $C_i(n)$ from the estimate of the number of delivered packets $d_i(n)$, LEAP uses a second Kalman filter, called Delivery Kalman Filter (DKF), to estimate the path's delivery CDF. The value of $d_i(n)$ is still unknown to the sender at the end of slot n, as Figure 4.2 shows: only $a_i(n)$ packets have been acknowledged, while $d_i(n) = a_i(n) + u_i(n)$ packets were delivered on time. For this reason, the DKF is temporarily updated with an estimated value, given by:

$$\hat{d}_i(n) = a_i(n) + \frac{\tau}{2}\mu_{c,i}(n). \tag{4.5}$$

The mean $\mu_{d,i}(n)$ and standard deviation $\sigma_{d,i}(n)$ of the DKF after this temporary update are used in (4.9) to decide the coding rate for slot $n+1$, as they are the best available estimate of the CDF of the number of packets that can be transmitted successfully on path i at that moment.

The state of the DKF before the temporary update is saved and updated $\frac{\tau}{2}$ seconds after the end of slot n, i.e., when a new ACK containing the value of $u_i(n)$ has also been received. As all the ACKs for the $d_i(n)$ packets have arrived by that time, the real value of $d_i(n)$ is used to update the state of the DKF for the next slot.

There is another trade-off to consider in the design of the acknowledgment

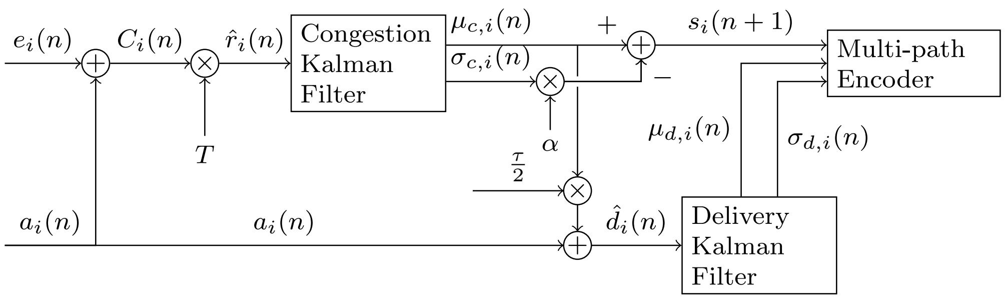


Fig. 4.3: Block diagram of the sender congestion control logic, implementing (4.2), (4.3) and (4.5)

mechanism: while retransmission-based protocol need an explicit confirmation of the reception of each packet, LEAP can just transmit one ACK packet very T_{ACK} seconds. A higher ACK frequency will reduce the error in the estimate of $\hat{d}_i(n)$, but with a higher impact on the uplink channel, although still lower than TCP's. Acknowledgment packets might even be aggregated and sent on a single path (if possible, one with a scheduled medium access scheme), but we leave this to future research.

A complete schematic of the congestion control mechanism for LEAP is depicted in Figure 4.3; the delayed update of the DKF is not shown for simplicity.

4.2.3 Aggregating flows through coding

In our design, the multi-path scheduler decides the coding rate R_c for the given channel capacity estimates to ensure the reliable delivery with a maximum failure rate of p_e. LEAP models the capacity of each path as a random variable.

If the sender transmits s_i packets on path i in slot n, and the path's capacity is modeled as a random value r_i with a given probability distribution with mean $\mu_{d,i}$ and variance $\sigma^2_{d,i}$, the number d_i of packets delivered within the deadline is given by:

$$d_i = \max(0, \min(s_i, r_i)),\ i \in \{1, 2, \dots, m\}. \tag{4.6}$$

Notice that the distribution of the capacity might admit negative capacity values, which are clearly unrealistic in practical settings. For ease of modeling, we overlook this incongruence (which generally has negligible probability to occur). However, according to (4.6), d_i is always non negative. The distribution of r_i is depicted in step 1 of Figure 4.1. The probability of delivering d_i packets on path i given that the transmitted batch size was of $s_i > 0$ packets is then given by:

$$P(d_i|s_i) = \begin{cases} 0 & d_i < 0 \\ 0 & d_i > s_i \\ P_{d,i}\left(1\right) & d_i = 0 \\ 1 - P_{d,i}\left(s_i + 1\right) & d_i = s_i \\ P_{d,i}\left(d_i + 1\right) - P_{d,i}\left(d_i\right) & \text{otherwise} \end{cases}, \tag{4.7}$$

where $P_{d,i}(x)$ is the CDF of the capacity distribution with mean $\mu_{d,i}$ and variance $\sigma^2_{d,i}$ [155].

If we assume that the receiver is able to decode the batch if it receives at least K of the $N = \sum_{i=1}^{m} s_i$ packets, and that the capacity of each path is independent from the others', the probability p_s of successfully decoding the batch within the deadline for a multi-path transmission with m paths is given by:

$$p_s = P\left(\sum_{i=1}^{m} d_i \geq K \middle| \sum_{i=1}^{m} s_i = N\right) \tag{4.8}$$

$$p_s = \sum_{d_1=0}^{s_1} \cdots \sum_{d_m=0}^{s_m} \prod_{i=1}^{m} P(d_i|s_i)\mathcal{H}\left(\sum_{i=1}^{m} d_i - K\right), \tag{4.9}$$

where $\mathcal{H}(x)$ is the Heaviside step function. This calculation is step 2 of the protocol schematic in Figure 4.1.

The value of p_s is hard to compute in the general case, but numerical solutions can be computed efficiently [156, 157] if we assume that capacity is Gaussian; since the output of the Kalman filter is a mean and variance, it would completely describe a normally distributed capacity. This channel model is not perfectly realistic, as discussed in Section 4.3.2, but it is a good enough approximation which simplifies our calculations significantly. In Section 4.2.5, we provide an efficient method for calculating it in the two-path Gaussian case, which can be extended to three or more paths.

4.2.4 Scheduling and retransmission

In order to receive as many packets as possible within the deadline, the packets are sent at the beginning of each slot with a constant rate R_s. The sending rate is calculated as:

$$R_s = \frac{T}{2s_i(n)}\left(1 - \frac{1}{\log(s_i(n))}\right). \tag{4.10}$$

If we consider a minimum sending window of 10 packets, $1-\frac{1}{\log(s_i(n))}$ is always positive; setting a different minimum rate would require setting a minimum value or changing the base of the logarithm. The packets are sent in the first half of the slot, i.e., over a total time of $T/2$, and the logarithmic factor helps new clients increase their sending rate and get a fair share of the channel capacity: the packets of the new flows will likely fill in the space between the packets of the on-going flows that, in turn, will estimate a lower capacity and make room for the newcomers in the next slots.

Finally, we consider the failure case of the multi-path coding: retransmission. Unlike traditional TCP-based protocols, LEAP is not designed to retransmit packets often: since the coding rate adaptation takes them into account, most of the losses should be recovered using the redundant packets, and even late batches are just transmitted with a higher latency most of the time.

However, in cases of sudden and total link outage (caused by, e.g., a handover or a burst of cross-traffic), the coding might not be enough to recover all the lost packets; if batch n (transmitted at time t) suffers from an unrecoverable loss, the sender needs to wait until the end of the next slot (i.e, time $t + 2T$) to avoid useless retransmissions. If batch n still has not been acknowledged by this time, the sender should include the missing packets at the beginning of batch $n + 2$. In this way, any information packet whose reception is delayed more than $2T - \frac{\tau}{2}$ is considered lost, and retransmitted if it cannot be recovered using the redundancy. As the retransmission is part of a standard batch, the retransmitted packets have all the protection of the redundant multi-path transmission, and there is no requirement to retransmit packets on the same path they were originally sent through.

LEAP's congestion control mechanism does not inherently require retransmissions, and applications that can deal with missing data in the stream (e.g., UDP-based applications which discard any late data and just rely on new measurements without filling the gap) can simply not use this protocol option.

4.2.5 Computation of the combined capacity in the two-path Gaussian case

As we mentioned in Section 4.2.3, the combined capacity in the two-path case can be computed more efficiently under some assumptions; we now derive its CDF exploiting some properties of Gaussian random variables. We assume that we have two paths, whose capacity is distributed as two doubly censored Gaussian random variables. The CDF of a doubly censored Gaussian random variable, identified by the mean μ and variance σ of its uncensored version and by its limits a and b is expressed as:

$$P(X \leq x) = \begin{cases} 0 & x < a \\ Q\left(\frac{x-\mu}{\sigma}\right) & a \leq x < b \\ 1 & x \geq b \end{cases} . \tag{4.11}$$

In order to compute the sum of two doubly censored Gaussian random variables, we need to introduce the doubly truncated Gaussian random variable with limits a and b, whose CDF is given by:

$$P(X \leq x) = \begin{cases} 0 & x < a \\ \frac{Q\left(\frac{x-\mu}{\sigma}\right)-Q\left(\frac{a-\mu}{\sigma}\right)}{Q\left(\frac{b-\mu}{\sigma}\right)-Q\left(\frac{a-\mu}{\sigma}\right)} & a \leq x < b \; . \\ 1 & x \geq b \end{cases} \tag{4.12}$$

In our case, the inferior limit of both distributions is 0, as capacity is inherently non-negative, and the superior limit for path i is the amount of sent packets s_i. From now on, we indicate the value of $Q\left(\frac{x-\mu_i}{\sigma_i}\right)$ as $Q_i(x)$ to make the notation lighter. The probability distribution of the overall capacity D is then given by the sum of the capacities of the two paths, which can be computed as a piecewise function. To reduce the number of cases, we assume that $s_1 \leq s_2$, but the computation in the other case is identical. We define the two random variables as D_1 and D_2.

Sum of two doubly truncated Gaussian random variables The Probability Distribution Function (PDF) of the sum of two doubly truncated Gaussian random variables T_1 and T_2 is not simple, but it can be computed in this way. First, we define $\varphi_a(x)$ as the aggregated PDF of the non-truncated Gaussian variables for the two flows:

$$\varphi_a(x) = \frac{e^{-\frac{(x-\mu_1-\mu_2)^2}{2(\sigma_1^2+\sigma_2^2)}}}{\sqrt{2\pi(\sigma_1^2+\sigma_2^2)}}. \tag{4.13}$$

Then, we define the scaling factor $F([s_1, s_2])$ that rescales the two doubly truncated Gaussian variables:

$$F([s_1, s_2]) = (Q_1(s_1) - Q_1(0))(Q_2(s_2) - Q_2(0)) \tag{4.14}$$

We performed a symbolic integration to find the actual PDF, which we will define with the aid of four auxiliary functions:

$$g_1(d) = Q\left(\frac{(d-\mu_2)\sigma_1^2 + \mu_1\sigma_2^2}{\sigma_1\sigma_2\sqrt{\sigma_1^2+\sigma_2^2}}\right) \tag{4.15}$$

$$g_2(d) = Q\left(\frac{(\mu_1-d)\sigma_2^2 - \mu_2\sigma_1^2}{\sigma_1\sigma_2\sqrt{\sigma_1^2+\sigma_2^2}}\right) \tag{4.16}$$

$$g_3(d) = Q\left(\frac{(d-s_1-\mu_2)\sigma_1^2 - (\mu_1-s_1)\sigma_2^2}{\sigma_1\sigma_2\sqrt{\sigma_1^2+\sigma_2^2}}\right) \tag{4.17}$$

$$g_4(d) = Q\left(\frac{(s_2-\mu_2)\sigma_1^2 + (-d+s_2+\mu_1)\sigma_2^2}{\sigma_1\sigma_2\sqrt{\sigma_1^2+\sigma_2^2}}\right). \tag{4.18}$$

Then, we can distinguish three cases. If the value d is between 0 and s_1, the PDF is given by:

$$r(d,[s_1,s_2]) = \frac{2\phi_a(d)}{F([s_1,s_2])}\left[g_1(d) - g_2(d)\right] \tag{4.19}$$

If the value d is between s_1 and s_2, the PDF is given by:

$$r(d,[s_1,s_2]) = \frac{2\phi_a(d)}{F([s_1,s_2])}\left[g_3(d) - g_2(d)\right] \tag{4.20}$$

If the value d is larger than s_2, the PDF is given by:

$$r(d,[s_1,s_2]) = \frac{2\phi_a(d)}{F([s_1,s_2])}\left[g_3(d) - g_4(d)\right] \tag{4.21}$$

The computation of the CDF $R(d,[s_1,s_2])$, which is needed to calculate the CDF of the capacity, can only be performed numerically, but it is considerably more efficient than computing the whole capacity distribution numerically or by Monte Carlo.

Computation of the combined capacity CDF We can compute the combined CDF by distinguishing between three different intervals of d.

Case 1: $0 \leq d \leq s_1$ If the examined value d is lower than s_1, we can distinguish two cases:

- One of the two paths (or both) has a capacity $D_i \geq d$
- None of the two paths have a capacity higher than d, and neither of the paths has a capacity $D_i = 0$

The probability of the first case is simply given by the joint probability of the two events $D_1 > d$ and $D_2 > d$:

$$P_a = Q_1(d) + Q_2(d) - Q_1(d)Q_2(d). \tag{4.22}$$

The second case is slightly more complex, as the probability can be considered as the sum of two doubly truncated Gaussians, both limited to the interval $[0, d]$, rescaled to ensure that the overall probability sums to one.

$$P_b = Q_1(0)Q_2(0)(1 - Q_1(d))(1 - Q_2(d))R(d, [d, d]). \tag{4.23}$$

The overall probability is given by the joint probability of the two disjoint cases, so it is simply:

$$P(D \leq d) = P_a + P_b. \tag{4.24}$$

Case 2: $s_1 < d \leq s_2$ If the examined value d is between s_1 and s_2, the two cases change slightly:

$$P_a = Q_2(d) - Q_1(s_1)Q_2(d) \tag{4.25}$$
$$P_b = Q_1(s_1)Q_2(d - s_1) + Q_1(0)Q_2(0)(1 - Q_1(s_1))(1 - Q_2(d))R(d, [s_1, d]). \tag{4.26}$$

In both cases, the limit of the first path is no longer d, but s_1, and the gap must be covered by the second path, hence the additional term in P_b; however, the derivation is mostly the same.

Case 3: $s_1 \leq s_2 < d$ In this case, d is larger than s_1 and s_2, and one single path cannot cover the whole required capacity any longer:

$$P_a = 0 \tag{4.27}$$

$$\begin{aligned} P_b &= Q_1(0)Q_2(0)(1 - Q_1(s_1))(1 - Q_2(s_2))R(d, [s_1, s_2]) \\ &\quad + Q_1(s_1)Q_2(d - s_1) + Q_1(d - s_2)Q_2(s_2) - Q_1(s_1)Q_2(s_2). \end{aligned} \tag{4.28}$$

4.2.6 Implementation considerations

The ossification of the transport layer is largely due to deployment issues: since protocols need to be implemented in the OS kernel, the adoption of new protocols requires a very broad consensus, and middleboxes often discard any packet that does not match any of the well-known transport protocols. However, recent proposals such as the Quick UDP Internet Connections (QUIC) protocol [153] avoided this problem by working entirely in user-space and tunneling over a UDP socket. Our implementation of LEAP exploits the same principle, tunneling traffic through a local VPN [123] and adding its headers to the payload of a UDP packet, which is then sent over the appropriate connection. The Linux implementation in user space can exploit optimized libraries for the encoding and decoding, and its computational load is very light.

The necessary fields in a LEAP header are limited to 12 bytes, as shown in Figure 4.4: the maximum size of a batch in this version of the protocol is 65536 packets, which corresponds to a maximum supported data rate of 7.65 Gb/s with a 100 ms deadline and a maximum packet size of 1500 bytes. The "options" field can be used to extend the header, but this would increase the header size and, consequently, the protocol overhead. At the moment the total size of the header, combined with the UDP standard header, is 20 bytes, which is exactly the same size as a standard TCP header.

0 8 16 24 32

Batch index i (32 bits)	
Batch size N_i(16 bits)	Packet index j (16 bits)
Information packets K_i (16 bits)	Options (16 bits)

Fig. 4.4: Contents of a LEAP header

4.3 Experimental results

The LEAP protocol was evaluated against and compared to Sprout, Versus, TCP Vegas and TCP Cubic, which are described in Section 4.1, both in terms of single-path congestion control performance and of multi-path reliability and throughput. The protocols were tested with one and two clients connecting through the downlink to a Linux server, which acted as a source of traffic. The server was a Debian x64 system running the standard 3.16 kernel, with TCP Vegas, Cubic and BBR; the implementations of Sprout and Verus were downloaded from the repositories provided by the authors. We set up live experiments in the following three scenarios.

Ethernet In the first scenario, the shared bottleneck connection was limited to 30 Mb/s using the `NetEm` [158] Linux utility, which was also used to set the minimum RTT τ to 50 ms. The cross-traffic had an average rate of 10 Mb/s, and was sent over UDP following a packet trace. The sending rate variations were Gaussian, with zero mean and a standard deviation of 2 Mb/s every 50 ms. There was an additional noise that was changed every 20 ms, with a standard deviation of 5 Mb/s.

802.11n The client was connected to the server through an 802.11n connection; the client was set at approximately 5 meters from an Apple Airport Plus access point, connected to the server by an Ethernet 100 Mb/s cable. In this scenario, there was no controlled cross-traffic, since the abundance of cross-traffic from more than a dozen other interfering networks on the 2.4 GHz band in the office environment proved to be challenging enough. Like in the first scenario, `NetEm` was used to bring the minimum RTT to 25 ms.

LTE A single client was connected to the server through a commercial LTE service in Ireland. Since the minimum measured RTT was 55 ms, we increased the deadline for this scenario to 100 ms for all protocols. In this case, we did not inject any cross-traffic, since the normal background traffic on the LTE access network proved to be challenging enough.

4.3.1 Combining the traces

In order to be independent of the scheduler, which can significantly affect the throughput and latency of MPTCP flows [159], we evaluated most of the protocols using an idealized scheduler over single-path traces. First, we gathered packet traces for all protocols running in single-path mode. Then, we simulated a multi-path flow combining two of the traces. By simply looking at the send times of the packet and the corresponding acknowledgments, we were able to measure the equivalent delay of each packet and calculate the aggregated throughput.

This procedure is equivalent to using a simple round-robin scheduler and an uncoupled congestion control, but with an infinite receiver buffer: by joining two separate single-path traces, we avoid any head-of-line blocking issues. For this reason, the multi-path performance of the protocols is an idealized upper bound to the actual performance that can be achieved, except for LEAP; since LEAP is free from the head-of-line blocking issue, its performance is the same in a real multi-path environment, and this was confirmed empirically.

However, comparing idealized versions of the protocols would not be a meaningful performance comparison; for this reason, we added MPTCP Cubic (with uncoupled congestion control) and MPTCP BALIA [147] to the multi-path results. As the results show, both real MPTCP implementations perform worse than the ideal version.

4.3.2 Single-path congestion control

Over a single path, the difference between capacity- and latency-oriented protocols are evident: while TCP Cubic has a high throughput but a poor

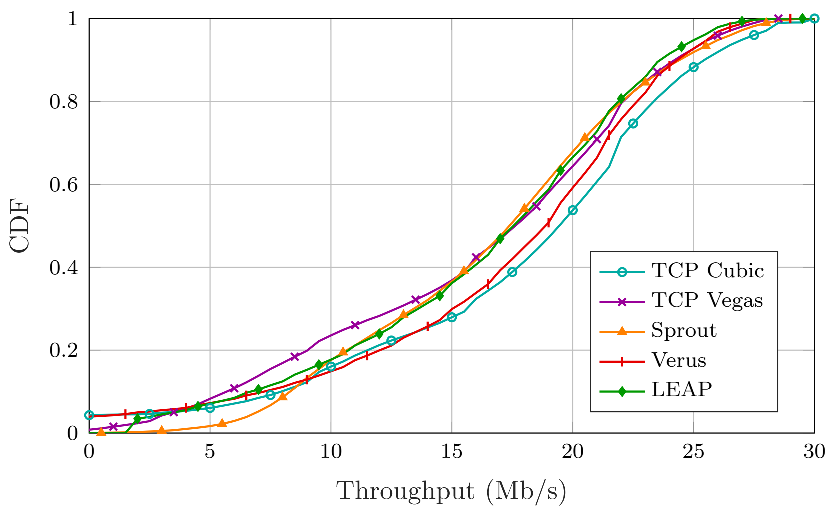


Fig. 4.5: Single-path congestion control evaluation: CDF of the raw unconstrained sending rate [Ethernet scenario with controlled UDP cross-traffic]

latency performance, latency-oriented protocols have a slightly lower raw throughput but a far lower latency. Without the multi-path aggregation, LEAP behaves much like other latency-oriented protocols such as TCP Vegas.

Figure 4.5 shows the CDF of the raw throughput of each protocol in the controlled Ethernet experiment. Somewhat expectedly, TCP Cubic has the largest raw throughput, but the difference between the protocols is barely

Algorithm	Parameter	Value
Sprout	Tick duration	20 ms
	Deadline	100 ms
	HMM states	256
	Maximum throughput	40 Mb/s
Verus	K	2
LEAP	α	0.5
	λ	1
	T	100 ms
	T_{ACK}	5 ms
	s_{min}	10

Table 4.2: Adaptation algorithm parameters

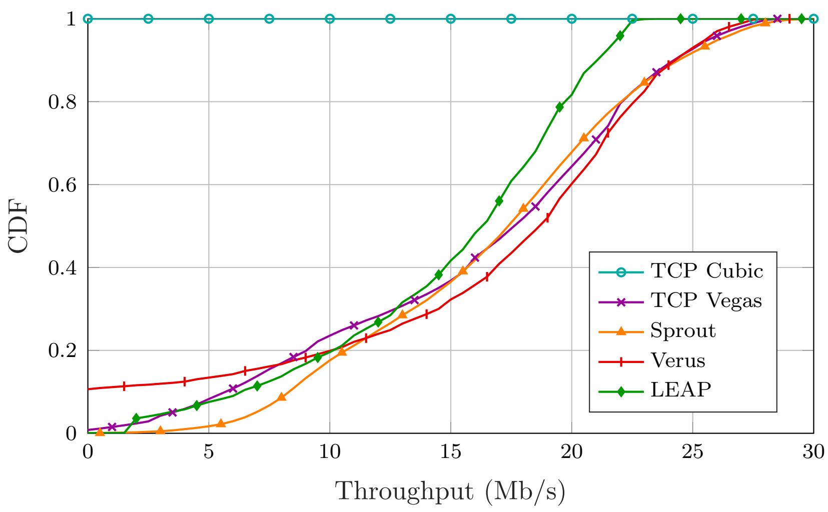


Fig. 4.6: Single-path congestion control evaluation: CDF of the latency-constrained sending rate [Ethernet scenario with controlled UDP cross-traffic]

perceivable: in this relatively controlled scenario, all the protocols manage to exploit the available capacity fairly well. When we consider the low-latency throughput, i.e., the throughput delivered within the deadline of 50 ms, whose CDF is shown in Figure 4.6, Cubic's queuing problem becomes evident: almost all packets are delivered beyond the deadline. Sprout shows the best performance in this setting, but all the other delay-based algorithms show little difference: Verus tends to have a higher capacity but with a higher variance, while LEAP's CDF curve is steeper, which indicates a more constant throughput.

Fairness is another key performance index for a transport protocol, which

Algorithm	Average Jain Fairness Index (JFI)	Relative goodput
LEAP	0.996	1.010
TCP Cubic	0.818	0.997
TCP Vegas	0.894	0.965
Verus	0.889	0.897
Sprout	0.514	0.006

Table 4.3: Performance of the protocols in the Ethernet scenario with two competing clients

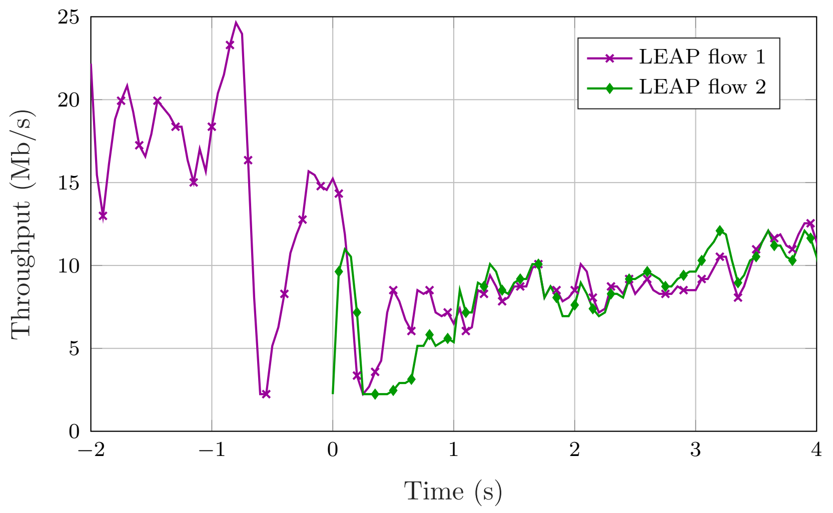


Fig. 4.7: Single-path congestion control evaluation: convergence of the bandwidth allocation between two flows

should be able to guarantee that two flows share a bottleneck link fairly. In order to test this, we performed a test in the Ethernet scenario, comparing all protocols and calculating the JFI after convergence, as well as the relative goodput. The latter is simply the total throughput of the two competing flows, normalized by the throughput of a single flow operating alone in the same channel conditions; in order to be efficient, a protocol's performance with multiple flows should not be far worse than with a single one. Table 4.3 lists the results of the experiments; LEAP achieves almost perfect fairness without any decrease in throughput; TCP Cubic, TCP Vegas and Verus experience a slightly lower fairness, along with a small throughput decrease, but still have acceptable performance. As its authors themselves remarked, and as we noted in Section 4.1, Sprout's performance when sharing a bottleneck buffer is extremely bad, as the throughput of the two competing flows is reduced to a trickle. LEAP is the only protocol that preserves both fairness and channel utilization; our first experiments confirm that this property is also verified for a higher number of clients.

Another important factor is the rate of convergence to the fair allocation; the steady-state capacity fairness means nothing if the flows take too long to reach it. As Figure 4.7 shows, LEAP has a relatively quick convergence;

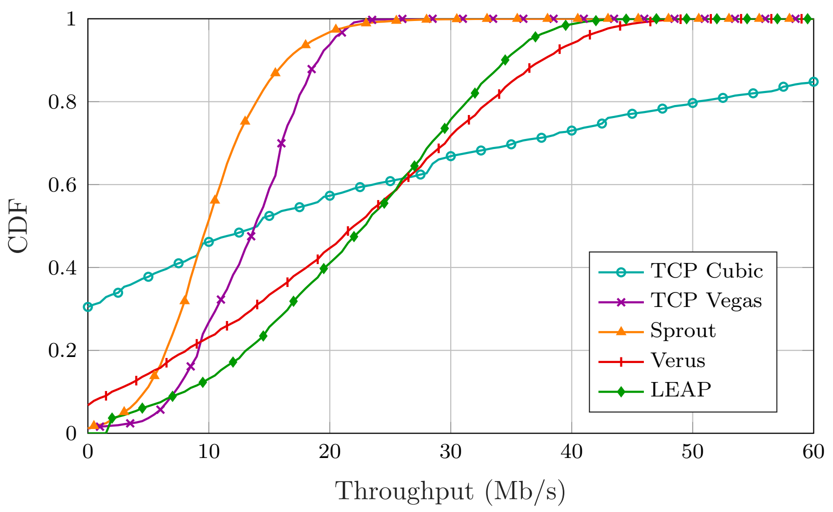


Fig. 4.8: Single-path congestion control evaluation: CDF of the raw unconstrained sending rate [802.11 scenario with live office traffic]

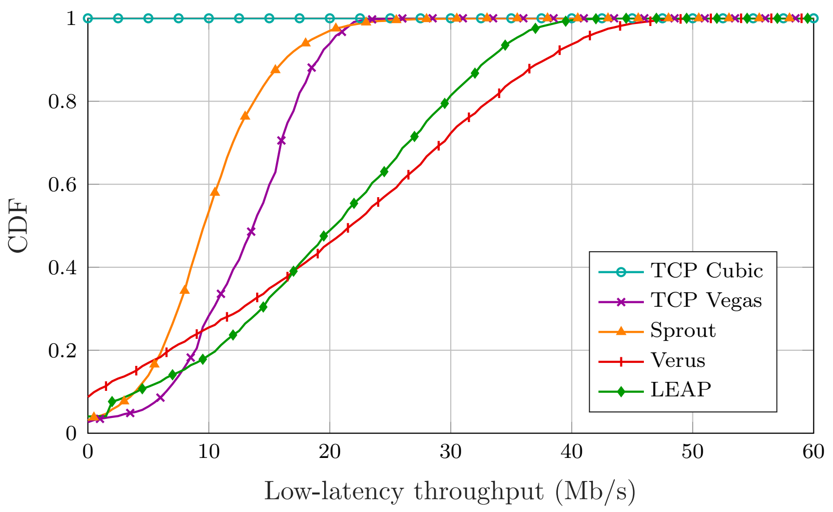


Fig. 4.9: Single-path congestion control evaluation: CDF of the latency-constrained sending rate [802.11 scenario with live office traffic]

even with the presence of fast-varying UDP cross-traffic, the two flows reach almost perfect fairness in about a second (20 adaptation steps) and proceed to maintain a stable, fair and efficient capacity allocation.

In the 802.11 scenario, Verus and LEAP adapt much better than the other protocols to the wireless channel: as Figure 4.8 shows, the two protocols have

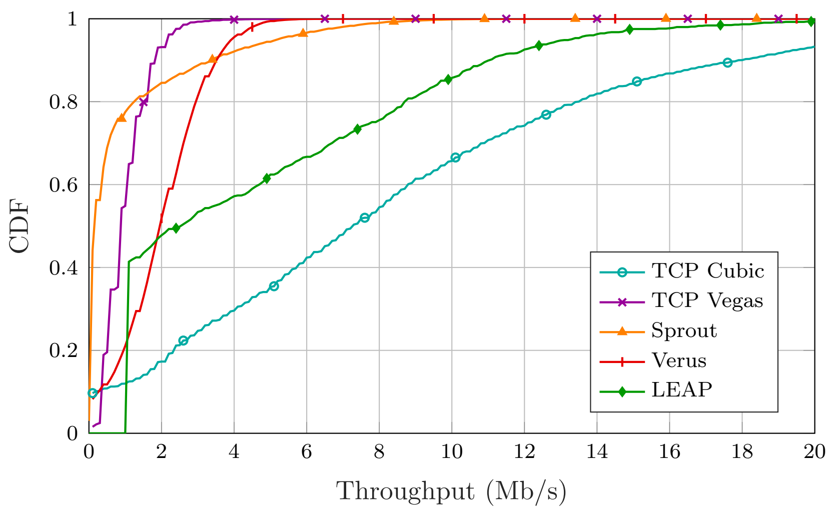


Fig. 4.10: Single-path congestion control evaluation: CDF of the raw unconstrained sending rate [LTE scenario]

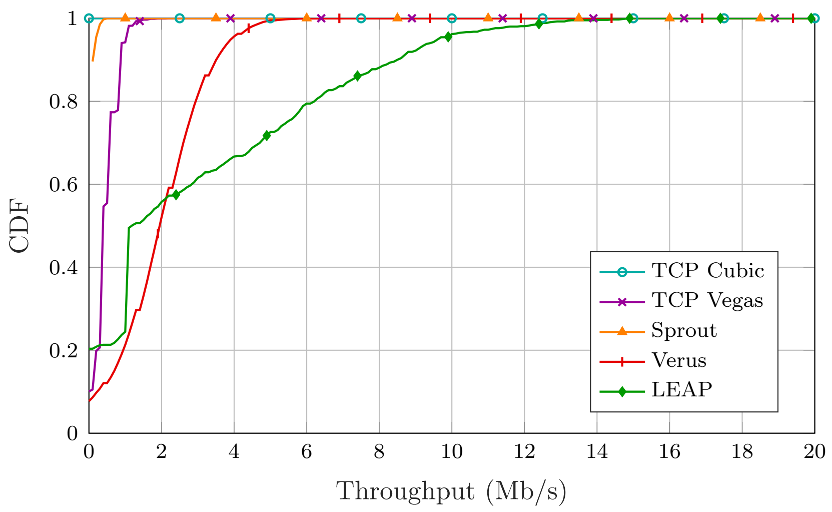


Fig. 4.11: Single-path congestion control evaluation: CDF of the latency-constrained sending rate [LTE scenario]

a significantly higher capacity than either TCP Vegas or Sprout (which is penalized by its underlying assumptions, which do not consider the impact of acknowledgment traffic on the half-duplex channel). TCP Cubic has a similar throughput, but with a higher variance.

Figure 4.9 shows the same pattern for the low-latency throughput: LEAP

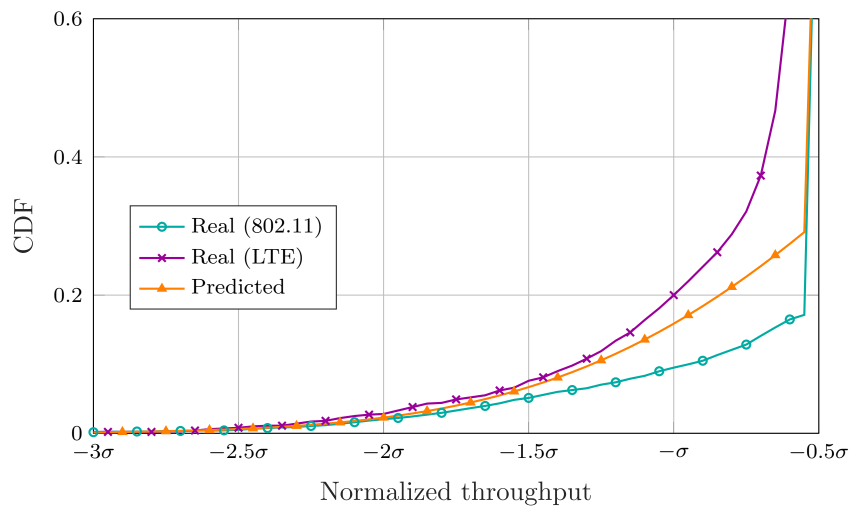


Fig. 4.12: CDF of the number of delivered packets, normalized and compared to the estimated capacity

shows the best performance, with a slight edge over Verus given by its steeper CDF, which makes the throughput more stable and predictable, while TCP Vegas and Sprout have a significantly lower goodput. TCP Cubic does not manage to transmit almost any packets with a low latency.

The results for the LTE scenario are shown in Figure 4.10 and 4.11: LEAP outstrips all other protocols in terms of low-latency throughput, and even comes close to Cubic's performance in terms of raw capacity.

Finally, Figure 4.12 shows the quality of the Gaussian prediction by the CKF: the filter slightly underestimates the variance of the LTE channel, and overestimates the variance of the WiFi channel, but the quality of the approximation is more than satisfactory to perform the coding calculations. Future works may employ more realistic distributions and even tailor them for each specific connection, increasing the efficiency of the adaptive coding scheme.

4.3.3 Combining multiple paths

The benefits of LEAP become clear in the multi-path scenario: as previously discussed in Section 4, LEAP can achieve a far higher reliability (up to 99.7%)

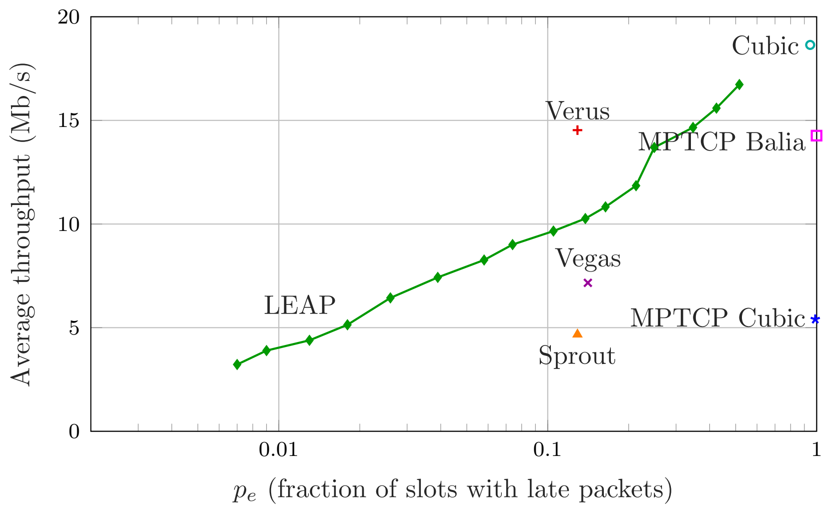


Fig. 4.13: Position of the considered protocols in the throughput/reliability trade-off in the multi-path scenario [802.11+LTE]

over a multi-path scenario than any of the state of the art protocols we consider.

Figure 4.13 shows that LEAP can effectively control the throughput-latency trade-off, achieving a higher throughput than either TCP Vegas or Sprout at the same reliability level and managing to reduce the missed deadlines by a factor of 30 , albeit with a lower throughput, over two parallel paths over the 802.11n and LTE interfaces. LEAP can maintain a given latency deadline of 100 ms with up to 99.3% reliability (green line) while other protocols fail in the best case at least 10% of the time. In the studied scenario, LEAP achieves a goodput of 8 Mb/s for a 95% reliability target and 4 Mb/s with a 99% reliability target. Reliability is defined as the number of slots in which all packets are successfully delivered within the user-defined deadline; some of the other protocols achieve somewhat higher throughput than LEAP but have no control over the throughput-vs-latency trade-off. LEAP can be set up to work at different points in the trade-off, depending on whether the first priority is throughput or latency control. Verus manages to get a higher throughput with the same amount of late packets, but its reliability is not adjustable, and it is unsuited for applications which require 90% or more packets to be delivered on time. Furthermore, late packets often come one

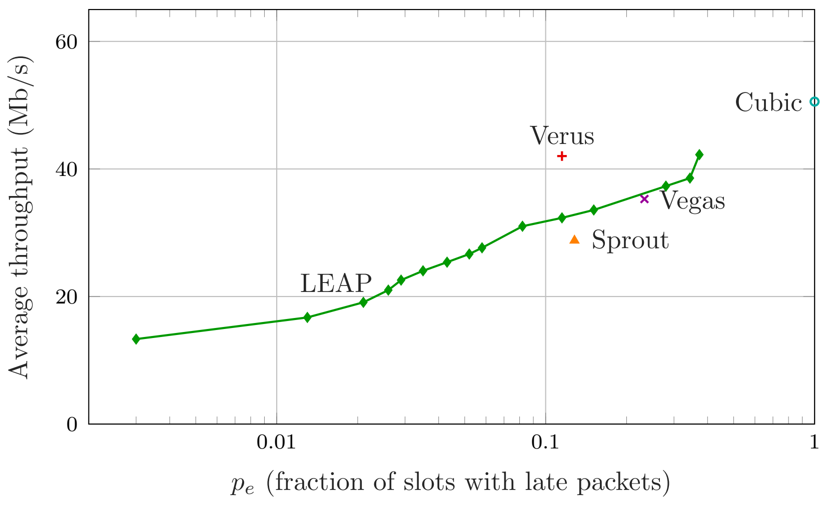


Fig. 4.14: Position of the considered protocols in the throughput/reliability trade-off in the multi-path scenario [802.11+Ethernet]

in bursts, and a transmission failure often means that more than half of the packets in a slot are lost.

In order to highlight the benefits of LEAP, we provide both results from combining two separate TCP traces, without any head-of-line blocking, and from actual MPTCP sessions. As the figure shows, the real throughput of an MPTCP session is far lower than the expected one, decreasing by almost 25% if we use the BALIA semi-coupled congestion control mechanism and over 70% using the Cubic protocol on both paths. The LEAP protocol can achieve a higher throughput than MPTCP, even while delivering most packets with a controlled latency: while MPTCP Cubic delivers only 1% of packets within 100 ms, with an average throughput of 5.4 Mb/s, LEAP achieves the same throughput with a 98% reliability on its latency control.

Figure 4.14 shows the results for the mixed scenario, when an 802.11 link is aggregated with an Ethernet link. The relative predictability of the Ethernet link plays in LEAP's favor: when privileging reliability over throughput, the protocol manages to get 13.3 Mb/s with 99.7% reliability. Verus also halves its failure rate, but its reliability is still not controllable.

Finally, Figure 4.15 shows the evolution of LEAP's throughput in the mixed scenario: the application-layer data rate (with a reliability level of

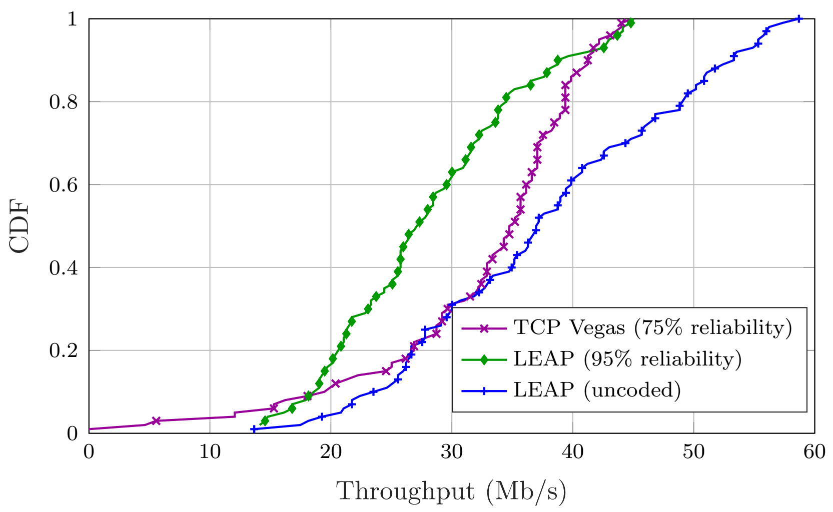


Fig. 4.15: Evolution of the capacity over time in the multi-path scenario [802.11+Ethernet]

95%) is represented in green, while the actual measured throughput is plotted in blue. The low-latency throughput of TCP Vegas in the same scenario is shown in violet for comparison. The throughput of the two protocols is essentially similar, with an average of 28 Mb/s for LEAP and 32 for Vegas, but LEAP achieves a far higher reliability. The throughput of LEAP is also more stable, with no total outages. This is a result of the combination of multiple paths through coding: the raw uncoded send rate of LEAP is far higher than Vegas's throughput, and the FEC is what makes the overall flow reach a 95% reliability.

Chapter 5

Providing Quality of Experience guarantees with Reinforcement Learning

Video streaming has been the dominant source of Internet traffic for the last few years; right now, videos make up 55% of all mobile traffic, and this figure is predicted to rise to 75% in the next five years [10]. Its importance in the Internet ecosystem and the difficulty of providing QoE guarantees make it an ideal setting for intelligent techniques, as its strict requirements can strain even the increased capabilities of new generation networks.

Since its inception in 2011, Dynamic Adaptive Streaming over HTTP (DASH) [160] has become the dominant standard for video transmission, as it relies on the existing HTTP server and Content Delivery Network (CDN) infrastructure and is not affected by firewalls and Network Address Translation (NAT). The DASH standard leaves complete freedom to the client in the choice of the adaptation policy: videos are divided into short segments (usually a few seconds long), which are encoded at different compression levels to generate an *adaptation set* of *representations* at different bitrates. The server makes all the segments in the adaptation set available to the client as HTTP resources, as well as a Media Presentation Description (MPD) file containing all the information about the video segments and their locations.

The client sequentially downloads each segment, switching between representations according to its adaptation logic in an attempt to optimize the QoE for the current video and network conditions.

The research on DASH adaptation algorithms is still ongoing, and most commercial systems employ very basic heuristic approaches, leading to annoying quality variations [161, 162] and to an inefficient use of network resources. In order to maximize the user QoE, the adaptation logic needs to take into account both the video content, which affects the perceived quality of the downloaded representations, and the playout buffer state. In fact, a major factor in video QoE is rebuffering [163], i.e., the temporary freezes in the video playout as the client waits until the next segment is downloaded [164].

In this challenging scenario, RL has emerged as an elegant and viable solution to the video adaptation problem. As we explained in Section 2.2, RL-based algorithms learn from past experience by trial-and-error, and gradually converge to the optimal policy [15]. The biggest design issue in these systems is that the state space of the corresponding MDP is very large. On the other hand, the number of states needs to be small to ensure quick convergence and make online solutions react timely to changes in the environment statistics.

In this chapter, we advocate the use of deep Q-learning, which we presented in Section 2.2.3, to learn excellent video adaptation strategies, while compactly and effectively capturing the experience acquired from the environment. The use of deep neural networks leads to several advantages, such as: i) the ability to deal with very large state spaces efficiently, effectively coping with the curse of dimensionality issue of RL algorithms, ii) the possibility of compactly representing the acquired experience through a set of weights, iii) the attainment of much better QoE performance, that is here quantified in terms of both the instantaneous visual quality of each segment and the quality variations across video segments, as well as the frequency of freezing/rebuffering events. We hence propose D-DASH, a framework for DASH video streaming that, combining deep neural networks with a carefully designed reinforcement learning mechanism, yields better QoE than prominent state-of-the-art rate-adaptation algorithms. More specifically, we formulate the DASH video streaming problem within a Deep Q-learning framework, detailing our design choices, that include: the choice of a reward function that effectively takes into account video quality variations and freezing/rebuffering events, the use of a learning architecture featuring two twin neural networks and a replay memory to get an improved stability and a faster convergence, and the use of a pretraining phase (on synthetic traces) to speed up the convergence of D-DASH when confronted with real traces.

Our results shed some light on the importance of tracking a certain amount of channel memory in the learning architecture, especially for complex network scenarios, and the superiority of the proposed deep Q-learning designs: in all our experiments, these have shown a higher average quality with smaller fluctuations across video segments, and a much lower percentage of rebuffering events.[1]

5.1 State of the art

In this section, we provide an overview of the most relevant works on DASH client-side adaptation strategies in the literature. As measuring QoE itself is a subject of intensive research in the field, there is not a single set of reference performance metrics [166]. In-depth reviews of the factors that impact QoE and the way they are measured in different streaming systems can be found in [167, 168]. Among such elements, in this work we focus on the following three factors. The first, and most obvious, is the *instantaneous picture quality*, which can be assessed through different techniques. Notable ones are: objective metrics such as the bitrate, no-reference metrics, which gauge the video distortion solely from the received frames (i.e., no external quality reference is provided) [169], or full-reference metrics such as Structural Similarity Index (SSIM) [170], a perception-based metric which measures the distortion between the input and output of the video encoder at the transmitter side. These approaches are adopted by various DASH adaptation algorithms in the literature [168, 171, 172]. The second factor, which is often the most

[1]The work presented in this chapter was published in the IEEE Transactions on Cognitive Networking [165].

pressing concern in adaptive video streaming, is represented by *rebuffering events*: their length and frequency strongly affect user QoE, as demonstrated by Hoßfeld *et al.* in [163]. The third factor is *video quality stability*: users notice frequent transitions in the video quality and might be annoyed by them [173]; in DASH video streaming, quality changes are not due to packet loss, but to switches between different adaptations (i.e., sudden changes in the quality of the video).

The literature on adaptation logics for DASH is vast, and we refer the reader to [174] for a comprehensive overview of existing techniques. For the purpose of this work, adaptation logics can be divided into two broad categories: 1) heuristics and 2) dynamic programming based. One of the earliest adaptation algorithms to go beyond simple buffer-based control loops was the Fair, Efficient, and Stable adapTIVE algorithm (FESTIVE) [175]: this scheme does not address QoE explicitly, but rather uses a stability cost function and a limit on the frequency of bitrate increases to privilege stability over instantaneous video quality. FESTIVE has the added advantage of being fairer than commercial protocols in terms of QoE, as well as of reducing the number of rebuffering events because of its conservative approach when increasing the bitrate.

Probe and Adapt (PANDA) [176] is another landmark algorithm in the DASH adaptation literature. It uses a proactive probing strategy to evaluate the channel capacity and changes its rate estimate according to an additive-increase approach to prevent fluctuations due to non-persistent cross-traffic and on-off effects. In stationary conditions, its bitrate estimate equals the TCP throughput. PANDA then uses a hysteresis threshold to avoid frequent switches between adjacent representations. A similar but simpler heuristic, called QoE-driven Rate Adaptation Heuristic for Adaptive video Streaming (QoE-RAHAS), was presented by Petrangeli *et al.* in 2014 [177].

Both PANDA and FESTIVE tend to be extremely conservative in their decisions, and they often end up underutilizing the available capacity unless the network conditions are extremely stable. Heuristics are hard-pressed to efficiently exploit the available capacity and still limit rebuffering events and quality switches, unless they are designed with knowledge of the network statistics, thus enabling the adoption of a dynamic programming approach to optimally solve the adaptation problem.

Li *et al.* use finite-horizon dynamic programming [176], modeling the adaptation problem as a fairness problem between future segments, with an added penalty for quality switches. Adding the adaptation logic on top of PANDA's switching strategy, the authors manage to significantly improve video quality without losing the algorithm's buffer and quality stability properties. In order to reduce the computational load, the authors assume that the available capacity remains constant throughout the optimization horizon. However, this assumption could prove to be unrealistic in wireless channels, which can suffer from outages and quick capacity variations due to mobility, fading and cross-traffic.

Yin *et al.* developed a system [178] that uses Model Predictive Control (MPC) to make decisions based on a look-ahead approach, with a QoE model similar to that considered in this work. More specifically, they rely on a throughput predictor to make optimal choices for a finite time horizon of five future segments. This approach is equivalent to dynamic programming and has the same critical dependency on the quality of the throughput predictor: if the predicted channel statistics are inaccurate, the adaptation logic will make suboptimal decisions.

An interesting paper by Bokani *et al.* [179] uses a Markov Decision Process (MDP) model to determine the optimal adaptation policy with dynamic programming. The main issue of this solution is the computational load: the model is too complex to be solved at runtime. The authors propose several solutions to mitigate the complexity issue, but the performance of these heuristics is either unsatisfactory or requires to store a large amount of offline computational results in the device's memory. A more recent work by Zhou *et al.* [180] proposes a pseudo-greedy heuristic to tackle the same problem, with the same kind of limitations. Other works use MDPs in more specific situations, such as streaming for multi-homed hosts [181] and cloud-assisted adaptive streaming [182].

5.1.1 Reinforcement Learning and DASH

There are several works in the literature that use RL to overcome dynamic programming's two biggest drawbacks: computational load, and the need to know the statistics of the network and video content in advance. RL-based algorithms learn the network statistics from experience, and their computational complexity is low. The main limitation of RL is the amount of experience it needs to make good decisions: as the number of states of the MDP grows, so does the necessary training time. Specifically, adaptive RL algorithms need to have a very coarse state granularity to effectively react to changes in the environment. Hence, there is a fundamental trade-off between adaptability and descriptive power: to be adaptable, the algorithm will need to visit and update each state frequently, i.e., to have a small number of states. However, having fewer states means that the knowledge of the environment is inevitably poorer (and, probably, so will be the algorithm's choices).

Two works by Claeys *et al.* represent the two opposite extremes in this trade-off: the algorithm in [183] has a complex reward function and a 6-dimensional state definition, and its training requirements are daunting, while the algorithm presented in [184] is lean and converges very quickly, but has a very rough state definition and a suboptimal performance. Another lean Q-learning algorithm that uses a similar MDP model was presented by Martín *et al.* in 2015 [185].

In 2015, Van Der Hooft *et al.* presented an interesting hybrid between RL and standard algorithms [186]: their system is based on Microsoft Smooth Streaming (MSS), but adapts the parameters of the heuristic to reflect the network conditions, improving its performance. The algorithm is still not QoE-aware, and the simple buffer-based MSS heuristic is suboptimal, but hybrid solutions such as this might be an interesting approach to overcome some of RL's main drawbacks.

In another recent work [187], the authors try to overcome the basic trade-off between efficiency and adaptability by parallelizing the learning process: since the problem has a well-known structure, experience in one situation can be used to learn how to act in others. This ad-hoc generalization scheme relies on the specific structure of the problem, but it is a step in the right direction.

As it will become apparent in the next sections, our D-DASH framework makes it possible to better exploit the observed data, learning from experience faster than other algorithms in the literature and quickly adapting the video encoding policy to the current working context.

5.2 System model

We consider a DASH client that downloads a video sequence segment by segment. The decision-making process follows a slotted time model, where $t = 1, 2, \ldots$ is the video segment number. For any given segment t, the client is free to choose which representation to download from a given adaptation set A. Each representation, in turn, is uniquely associated with a certain video quality level of the segment, which we indicate by q_t.

In order to apply the RL approach to the problem, we need to model our system as an MDP; this is possible if we model the video content as a sequence of scenes with exponentially distributed duration, like in [188]. We consider the download of each segment as a step of the MDP, and the action space of the problem can be intuitively mapped to the adaptation set A; by a slight abuse of notation, but for the sake of a compact notation, we use q_t to also indicate the action of downloading a segment t with visual quality q_t.

The solution of the video adaptation MDP is the policy that maximizes the QoE perceived by the user; we now describe our model of the problem as an MDP, presenting our assumptions regarding the video streaming service and then introducing the reward function that accounts for the QoE factors described in Section 5.1.

5.2.1 Video streaming model

Each video segment is characterized by a certain quality-vs-rate trend, which we describe by means of the function $F_t(q_t)$ that gives the size of the segment

t with quality q_t. We assume $F_t(\cdot)$ to be known before downloading segment t, as it can be made available to the client as part of the MDP or predicted from previous scenes, since the correlation between subsequent segments is usually high.

Denoting by C_t the average channel capacity experienced during the downloading of the segment t, we easily obtain the total downloading time τ_t as

$$\tau_t = \frac{F_t(q_t)}{C_t}\,. \tag{5.1}$$

We indicate by T the (constant) playout duration of a video segment. Moreover, we define the *buffer* (time) for segment t, denoted by B_t, as the lapse of time between the starting of the download of segment t and the instant the segment is due to start its playout at the client. Rebuffering events (during which the video playout freezes) occur whenever the playout buffer empties before the next segment has been completely downloaded, i.e., when $\tau_t > B_t$. Conversely, when $\tau_t < B_t \leq T$ the download of the segment t is completed before its planned playout time and the download of the next segment can start immediately, thus adding an extra $B_t - \tau_t$ time budget to the buffer of segment $t+1$. Accordingly, the rebuffering time for a generic segment t is given by:

$$\phi_t = \max(0, \tau_t - B_t)\,, \tag{5.2}$$

while the buffer for the next segment is computed as

$$B_{t+1} = T + \max(0, B_t - \tau_t)\,. \tag{5.3}$$

The buffer is limited to 20 seconds, as most commercial video streaming systems limit video buffering to save memory and network capacity.

5.2.2 Reward function

As discussed in Section 5.1, the QoE of a video client depends on the visual quality of the current segment, the quality variation between segments, and the playout freezing events due to rebuffering. In the following, we introduce a reward function that captures these aspects and, in turn, can be used to derive policies that maximize the QoE of video streaming customers.

In this work, we chose SSIM as the instantaneous quality metric of a video sequence, so that q_t is the average SSIM of the video frames in segment t. SSIM is one of the most common objective metrics in the literature and has been shown to correlate well with the perceived QoE [189]. As it is a full-reference metric (i.e., its computation requires a full knowledge of the uncompressed segment [190]), it cannot be calculated by the streaming client, but its values when varying the video representation can be conveniently pre-computed, stored on the video server, and included as a field in the MPD. This puts the computational burden for calculating the SSIM onto the server side, even though, as reported in [191] the $F_t(q_t)$ characteristic of a video sequence can be automatically estimated from the size of the *encoded frames* by using a properly-trained deep neural network, thus making it possible to skip the computationally intensive frame-by-frame comparison with the original video sequence at the server. Other approaches are possible, for example computing the quality q_t of video frames at the client through a no-reference metric [169]. Although we do not consider this possibility in our model, it is also viable and would require the implementation of an additional software entity at the client to estimate the quality of the received frames. The impact of inaccurate quality estimates, via no-reference metrics, is left to future investigations.

We finally define the reward function for segment t as follows:

$$r(q_{t-1}, q_t, \phi_t, B_{t+1}) = q_t - \beta\|q_t - q_{t-1}\| - \gamma\phi_t - \delta[\max(0, B_{\text{thr}} - B_{t+1})]^2 \,. \quad (5.4)$$

The first term on the right-hand side accounts for the benefit of a higher quality q_t of the video, while the following two negative terms are penalty factors due to *quality variations* in consecutive frames and *rebuffering events*, respectively. The right-most term is a further penalty that is applied whenever the buffer level is below a (preset) threshold B_{thr} and it has been introduced to further reduce the chance of highly-damaging rebuffering events. The coefficients β, γ, and δ are weighting factors that regulate the relative importance of the three penalty terms. Note that, neglecting the right-most penalty term (i.e., setting δ to zero), the reward function (5.4) is the same used in [187] and [178], and similar to that proposed and validated by De Vriendt *et al.* in [173].

The weights β, γ and δ are here used to select different points in the trade-off between a high instantaneous quality, a constant quality level, and a smooth playback. The desired operational point might depend on several factors, including user preferences and video content, and tuning these parameters is outside the scope of this work.

5.2.3 Defining the Markov Decision Process

As we mentioned above, the action space clearly corresponds to the set A of possible representations of the video segments that can be chosen by the client. Accordingly, the action at step t corresponds to the choice of the quality q_t of the next segment to be downloaded. Therefore, from a state $s_t \in S$, an agent implementing a policy $\Pi(\cdot)$ will require the downloading of the segment t with quality $q_t = \Pi(s_t)$. The state space should be as small as possible, to reduce the complexity of the MDP, but at the same time each state should contain enough information to permit a precise evaluation of the utility function for each possible action $q_t \in A$. Considering the reward function (5.4) as a natural option for the utility function of the MDP, the state should hence include the video quality q_{t-1} of the last (the $(t-1)$-th) downloaded segment, the current buffer B_t state, the quality-rate characteristic $F_t(q_t)$ of the next segment (the t-th), and the future channel capacity C_t from which, given the chosen action q_t, it is possible to determine the download time τ_t of the next segment, the rebuffering time f_t and the next buffer state B_{t+1} using (5.1), (5.2), and (5.3), respectively. However, the future capacity C_t of the channel can only be known in a statistical sense, from the past observations. We then define the vector $\mathbf{C}_t = [C_{t-n}, C_{t-n+1}, \dots, C_{t-1}]$ of the previous n channel capacity samples, where n is assumed to be larger than the coherence time of the channel. In this case, the process $\mathbf{C}_t$ exhibits the Markov property, since the knowledge of samples that are further away in the past do not add any information to that contained in the current channel vector $\mathbf{C}_t$. Summing up, the state of the MPD at step t can be described by the 4-tuple $s_t = (q_{t-1}, \mathbf{C}_t, \phi_t, B_t)$.

Given the state s_t and the action q_t, we can finally define the utility function of our MDP as

$$\rho(s_t, s_{t+1}, q_t) = r(q_t, q_{t-1}, \phi_t, B_{t+1}). \tag{5.5}$$

5.3 Deep Q-learning for DASH adaptation

Depending on the definition of the video adaptation MDP, standard Q-learning algorithms can either be fast to converge and adaptable in an online setting [184, 185] or efficient after convergence [183]: the number of states necessary to accurately represent the environment makes Q-learning slow and unwieldy.

Our objective is to use an RL algorithm that converges quickly and that is capable of generalizing based on experience, i.e., that can cope with *previously unseen* channel/quality patterns, and that approximate well the optimal policies that would be obtained by solving the MDP. To achieve this, referring to s' as the state of the system upon taking action q and by r the actual reward accrued from that action, we advocate using the 4-tuple (s, a, r, s') to update $Q(s, a)$ and, along the same lines of [187], to concurrently improve the Q-values associated with other states and actions. We obtain this through deep Q-learning, as it provides a natural and effective way to generalize the knowledge acquired during specific transitions and reuse it for other states and actions.

Two different types of deep network architectures have been tested in this study (see Figure 5.1): **1.** a fully connected feed-forward network, and in particular an MLP network with one and two hidden layers, respectively

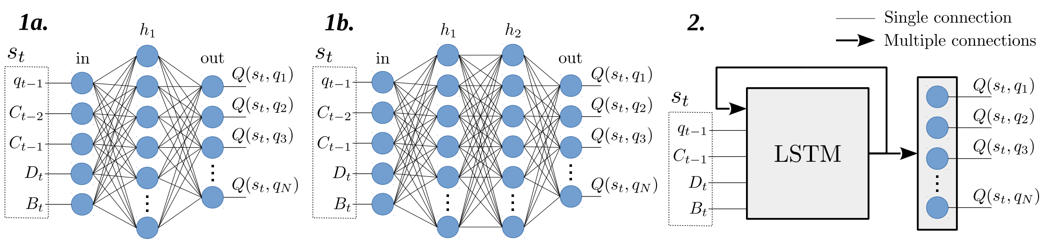


Fig. 5.1: Network architectures considered in D-DASH: an MLP network with one (***1a***) and two (***1b***) hidden layers; a RNN based on an LSTM cell (***2***). For this plot, the channel memory was assumed $n = 2$, i.e., $\mathbf{C}_t = [C_{t-2}, C_{t-1}]$.

referred to as MLP$_1$ and MLP$_2$ in the following, and **2.** a recurrent neural network based on a Long-Short Term Memory (LSTM) cell [192]. An LSTM with deep hidden-to-output function has been implemented [193]. Accordingly, the LSTM output is processed by another fully connected layer to provide the final Q-values. We remark that the number of previous components considered in the capacity vector $\mathbf{C}_t$ of the input state differs in the two cases: feed-forward networks (MLP$_1$ and MLP$_2$) require to be fed with the whole vector $\mathbf{C}_t$, whereas recurrent networks only need the last channel capacity sample C_{t-1} as input, as the feedback loop inside the LSTM cell keeps track of the channel memory. The dimension of vector $\mathbf{C}_t$ has to be fixed beforehand for the feed-forward networks, and cannot be adapted after the definition of the network. In this work, we consider 2 history samples for MLP$_1$ and MLP$_2$, and 5 history samples for a *"long history"* (lh) version of MLP$_2$, referred to as MLP$_{\text{lh}}$.

The LSTM has the advantage of automatically learning what to store inside the memory cell, and what to forget. Here, in addition to standard LSTM, we also consider an LSTM cell with peephole connections, referred to as LSTM$_{\text{ph}}$.

Due to the continuous changes of the target function and training data, we do not expect serious overfitting issues. However, a dropout regularization technique has been applied to MLP$_2$ (termed MLP$_{\text{do}}$) to verify this assumption. Dropout is a well-known and simple method to prevent neural networks from overfitting [194]. It amounts to randomly dropping neurons and their connections, with a certain probability. A 20% dropout probability

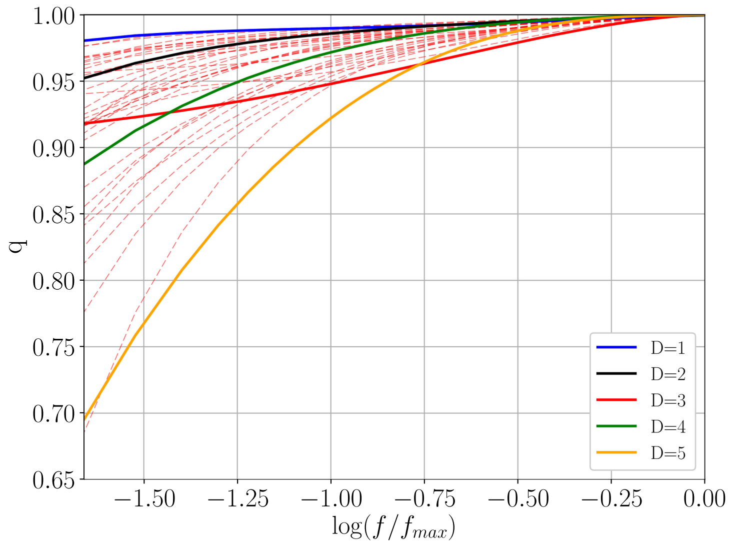


Fig. 5.2: Reference quality-rate curves.

is considered in the training of $\mathrm{MLP}_{\mathrm{do}}$.

The network's weights are updated in order to approximate the optimal action-value function $Q^*(s,q)$ using typical gradient descent optimization methods: the numerical results shown in this work have been obtained by using the Adam method [69], in conjunction with back-propagation.

5.4 Simulation and results

In this work, we assume segments belonging to the same representation set have a constant size and variable quality; this is a common assumption in the relevant literature. In real systems, the encoding parameters of the video are often constant, with segments of varying size [195]; however, the performance of the learning algorithms should not be significantly affected by this factor, and the model presented in Section 5.2 is fully general.

To evaluate the performance of the proposed algorithms, we carried out extensive trace-based simulations, where the different components of the system have been modeled in a realistic manner from real data traces. More specifically, the video model was derived from real videos from the EvalVid database.[2] The video traces were characterized in terms of their quality-rate function where, as previously explained, we used SSIM [170] as the instantaneous video quality metric. Then, we derived the SSIM-rate characteristics of a large number of videos in the dataset, from which we extracted a limited number of reference SSIM-rate curves. Following [196], such curves have been represented as 4th-degree polynomials function of the bitrate, so that the quality (SSIM) of a given video sequence encoded at rate $f/f_{max} \in (0,1]$ is given by

$$q \simeq d_0 + \sum_{k=1}^{4} d_k (\log(f/f_{\max}))^k, \tag{5.6}$$

where $f_{\max}$ is the full-quality video segment size, and $f \leq f_{\max}$ is the actual segment size. The vector $\mathbf{d} = [d_0, \ldots, d_4]$ offers then a synthetic representation of the complexity of a video scene. The chosen reference curves are shown in Figure 5.2, and the values of each reference curve's vector $\mathbf{d}$ are listed in Table 5.1. These reference curves were combined into scenes with exponentially distributed duration, a model validated by Rose in [188], with an average of 10 seconds (i.e., 5 segments). In this way, we generated a large number of realistic video traces for our tests. In our numerical results, we picked 8 different values for the segment size f: if q is the segment quality, f is defined as $f = F_t(q)$, where $F_t(\cdot)$ is the inverse of (5.6), see also Table 5.2 for the considered adaptation set. In the learning system, each video segment is characterized by a vector $\mathbf{d}_t$, which is summarized by an index D_t, as shown in Table 5.2.

To model the transport capacity of the HTTP connection we considered three datasets: a) *real* traces of HTTP video streaming sessions over LTE [197]; b) *synthetic* traces obtained through the network simulation platform `ns--3`; c) *Markovian* traces obtained by means of a simple Markov model. The learning algorithms were first trained on the *Markovian* channel model (*pretrain* phase), then shown a small number of realistic traces (either *real* or *synthetic*) as an actual training (*train* phase). The neural networks' weights

[2] `http://www2.tkn.tu-berlin.de/research/evalvid/cif.html`

were then frozen and their exploration parameter was set to zero to carry out a fair comparison against state of the art algorithms from the literature using other traces from the same dataset (*test* phase). The *pretrain* phase lasted for 500 synthetic videos, while the *train* phase only lasted for 40 actual videos. The final *test* phase was performed over 100 actual video episodes; each video episode in all phases lasted 400 segments of $T = 2$ s each. The purpose of the *pretrain* phase is to teach the learning agent the basic mechanics of video streaming in a highly variable and realistic network scenario. The rationale is that the solution so obtained should represent a sensible starting point for the learning process when confronted with real traces, leading to a much shorter training time.

The Markov model for the *pretrain* phase used the same settings as in [187]; the capacity evolution was controlled by a random walk model among a set of levels, with a state change probability of 0.5. The possible capacity levels C_t are listed in Table 5.1. We remark that the Markovian traces were only used in the pre-training phase, while the performance evaluation was carried out by considering realistic traces, either from real HTTP measurements or generated by ns–3.

Each component of the reward function in (5.4) was scaled into the range between 0 and 1 to avoid numerical convergence issues.

Parameter	Value
T	2 s
$f_{\max}$	20 Mb
$F_t(\cdot)/T$	$\{0.25, 0.5, 1, 2, 3, 4, 6, 10\}$ Mb/s
C_t (pretrain)	$\{0.4, 0.75, 1.5, 2.5, 3.5, 4.5, 5.75, 7.25, 9,\ 12.5\}$ Mb/s
$D_t = 1$ (Akiyo)	$\mathbf{d}_t = [0.99947, -0.01015, -0.02888, -0.02427, 0.00415]$
$D_t = 2$ (News)	$\mathbf{d}_t = [0.99970, -0.01064, -0.02291, -0.02531, 0.00074]$
$D_t = 3$ (Bridge - far)	$\mathbf{d}_t = [1.00033, -0.01051, -0.05385, -0.08211, 0.01361]$
$D_t = 4$ (Harbor)	$\mathbf{d}_t = [0.99977, -0.00505, 0.00554, -0.01726, 0.00022]$
$D_t = 5$ (Husky)	$\mathbf{d}_t = [0.99984, 0.00998, 0.07590, -0.01138, 0.00040]$

Table 5.1: Simulation parameters

5.4.1 Algorithm settings

The algorithms from the literature selected as benchmarks were the parallel Q-learning and the simple rate-based heuristic from [187], MPC [178], which is a dynamic programming-like solution, and FESTIVE [175], which is among the most conservative heuristic-based algorithms, thus generally guaranteeing good stability and a low number of rebuffering events. One of FESTIVE's aims is providing a stable quality for multiple clients, which is beyond the scope of this work, so its performance should be evaluated accordingly. Another point that bears consideration is the computational complexity of MPC: the authors themselves state that a real-time implementation would require pre-computed tables, which leads to memory occupation. We also implemented the PANDA [176] and QoE-RAHAS [177] algorithms, but their performance was significantly worse than all the other algorithms' for any configuration of their parameters in the simulation environment, so we did not show them in any of the plots.

The standard Q-learning algorithm used a Softmax policy; the other learning algorithms were tested with both policies, and the best-performing one was chosen in each case. The policy for each algorithm is listed in Table 5.2.

The hyperparameters of the learning algorithms were optimized by performing an exhaustive search and selecting the combinations that performed best in the *pretrain* phase; the values of the parameters for all algorithms are listed in Table 5.2, except for the exploration parameter (either ε or ξ, depending on the exploration mode), which followed the profile shown in Figure 5.3 over the three phases. The settings and model for FESTIVE and the standard Q-learning algorithm are the same used in [175] and [187] respectively, except where otherwise stated. In Figure 5.3, each video episode has $M = 400$ segments, the standard Q-learning algorithm uses preset thresholds to quantize the video quality and capacity measurements, which are listed in Table 5.2.

We adapted the MPC parameters to reflect our definition of QoE and provide a fair comparison, as its basic model is very similar to ours and

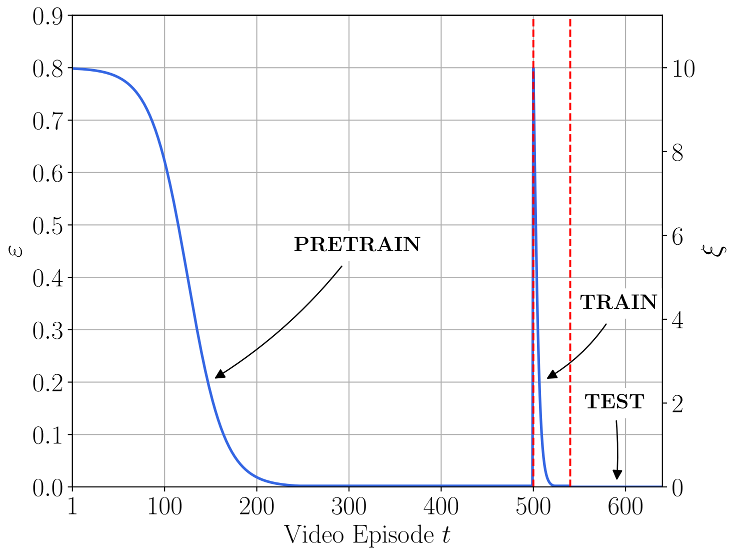


Fig. 5.3: Profile of the exploration rate during the training and testing phases.

using the same QoE function provides a more meaningful comparison.

5.4.2 Results: real traces

Our simulations aimed to assess the performance of the D-DASH framework with regard to the three major video quality metrics, i.e., instantaneous quality, its stability and the frequency of rebuffering events. We compared D-DASH based algorithms against existing approaches from the literature, while also investigating their convergence speed.

Figure 5.4 shows a smoothed version of the reward over the three phases of the simulation for the MLP$_1$ and MLP$_2$, using standard Q-learning and the rate-based heuristic as a comparison. The tests were performed by freezing the learner state and setting a greedy policy (i.e., always taking the action with the highest expected reward) after each video episode. The benefits of the Deep-Q approach appear evident at a first glance; standard Q-learning gets a lower reward at convergence, as well as taking more than 200 episodes to overtake the rate-based heuristic and failing to adapt quickly enough to the real traces in the *train* phase.

The two D-DASH algorithms achieve higher rewards after just a few

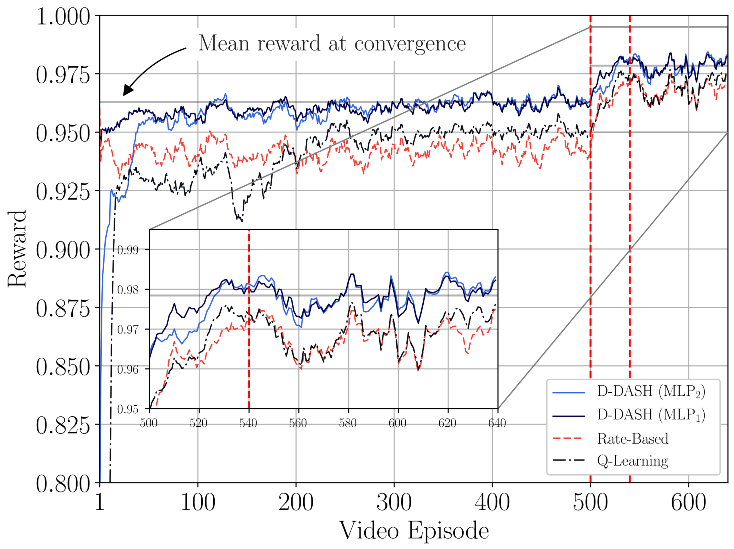


Fig. 5.4: Reward for the standard Q-learning, D-DASH and rate-based algorithm (using real traces as test set). The pretrain phase takes the first 500 video episodes, the training phase is enclosed within the two vertical lines and the test phase takes the remaining points, after the second line.

videos: MLP$_2$ converges in the first 50 videos, while MLP$_1$ already reaches its peak performance after the first video episode. The same trend can be seen in the *train* phase, where both schemes maintain a significant advantage on the simple heuristic and standard Q-learning.

One of the main advantages of smart QoE-aware adaptation schemes is a better buffer management: while most classical algorithms try to keep the buffer as stable as possible, the D-DASH algorithms can use the buffered segments when the available capacity drops and build up the buffer when it is convenient to do so. Figure 5.5 shows an example taken from the *test* phase: while FESTIVE keeps an extremely stable buffer (except for a sudden drop after about 200 segments, due to a sharp change in the capacity), the MLP$_2$ algorithm has a large dynamic range, building up the buffer when capacity is high and using it up to cover temporary outages. The other D-DASH algorithms make a similar use of the buffer, as well as standard Q-learning. MPC also maintains a high and stable quality throughout the video, but it incurs in several rebuffering events because of its optimistic throughput prediction, as Figure 5.5 shows.

The benefits of the smarter buffer use are clearly visible in the quality

Algorithm	Parameter	Value
FESTIVE	Target buffer	15 s
	Buffer randomness	0.25 s
	α	10
	k_S (switch memory)	10
	k_C (throughput memory)	5
MPC	Maximum buffer	30 s
	γ	2
	μ	50
	K	5
All learners	β	2
	γ	50
	δ	0.001
	λ (where not specified)	0.9
	Policy (where not specified)	Softmax
	Maximum buffer	20 s
	B_{thr} (safe buffer)	10 s
Q-learning	α	0.1
	C_t quantization thresholds (Mb/s)	$\{0.5, 1, 2, 3, 4, 5, 6, 8, 10\}$
	q_t quant. thresholds (SSIM×100)	$\{84, 87, 9, 92, 94, 96, 98, 99, 99.5\}$
MLP$_1$	Hidden neurons N_h	256
	Learning rate	0.001
	Batch size M	1000
	K	20
MLP$_2$	Hidden neurons N_{h1}, N_{h2}	128, 256
MLP$_{\text{do}}$	Learning rate	10^{-4}
MLP$_{\text{lh}}$	Batch size M	1000
	K	20
	Policy	ε-greedy
LSTM	Number of units N_c	128
LSTM$_{\text{ph}}$	Learning rate	0.001
	Batch size M	100
	K	200

Table 5.2: Adaptation algorithm parameters

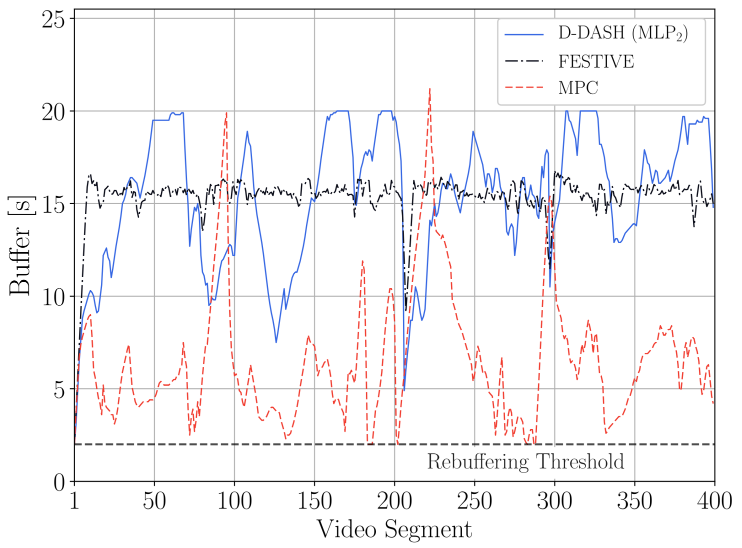


Fig. 5.5: Evolution of the buffer during a video episode in the *test* phase.

graph of Figure 5.6: MLP$_2$ can avoid sharp drops in the video quality without triggering rebuffering events.

In the next figures we compare the performance of the different algorithms in the *test* phase, in terms of image quality (SSIM), rebuffering, and quality stability.

Figure 5.7 shows the achieved SSIM: all the learning solutions, including standard Q-learning, achieve a higher SSIM than FESTIVE and the rate-based heuristic; aside from the higher median, the bottom 5% of the videos still have an average SSIM of over 0.97, while FESTIVE and the rate-based heuristic go below that threshold for a significant fraction of the videos. It should be noted that FESTIVE has a lower average SSIM than the rate-based heuristic, since it privileges stability over instantaneous quality. MPC performs about as well as the learning-based algorithms, but with a slightly larger variance.

Figure 5.8 shows the average difference between the SSIM of one segment and the next. As expected, FESTIVE keeps the quality more stable than the rate-based heuristic. Similar performance is obtained by standard Q-learning, which also keeps a higher average SSIM. The two Deep-Q algorithms outperform FESTIVE and Q-learning, but the best quality stability is

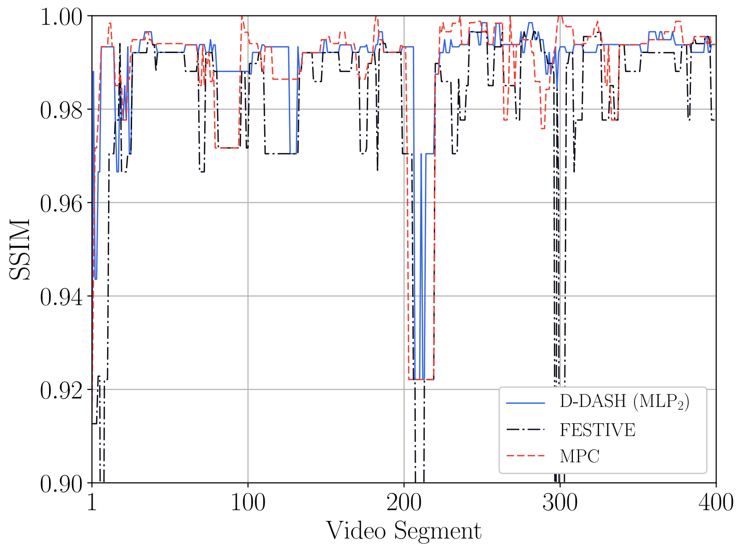


Fig. 5.6: Evolution of the SSIM during a video episode in the *test* phase.

achieved by MPC, with an average SSIM variation always lower than 0.006, while that of the other algorithms reaches 0.008 for at least one video.

Finally, Figure 5.9 shows the frequency of rebuffering events for each algorithm: since FESTIVE is extremely conservative, there were no rebuffering events over the whole *test* phase. However, even the most aggressive learning algorithms (standard Q-learning and MLP$_{\mathrm{do}}$) do not experience rebuffering events often. The LSTM and LSTM$_{\mathrm{ph}}$ algorithms experience at least one rebuffering in 25% of the videos, but they never have more than three, and one rebuffering over a whole 400 segment video episode can be considered acceptable. MPC pays for its higher stability by having an average of 5 rebuffering events per video, far higher than any of the other algorithms'. MPC's reliance on an imperfect throughput predictor shows its limits when dealing with highly variable environments.

We also included another version of MLP$_2$, called MLP$_{\mathrm{lh}}$, which uses the last 5 throughput samples as input instead of the last 2. Its aim is to show the benefits of a longer memory in the presence of long-term correlation. We observe that, with real channel capacity traces, the performance of MLP$_{\mathrm{lh}}$ are basically the same of LSTM, probably because the correlation of the empirical

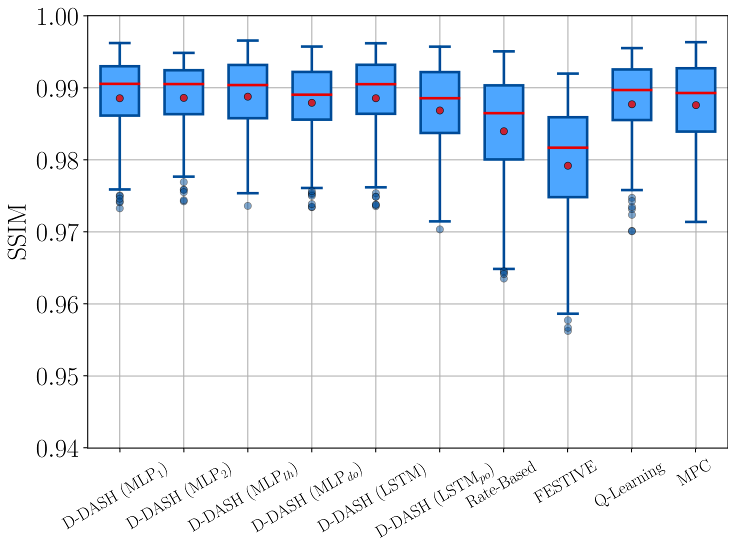


Fig. 5.7: Boxplot of the average SSIM for the 100 video episodes in the *test* phase (real traces).

channel data considered in this study is low, so that the extra memory in the LSTM is not necessary. Finally, the use of dropout techniques (MLP$_{\text{do}}$) to avoid overfitting does not provide any improvement, as expected.

Its relative simplicity, low rebuffering rate and quick convergence arguably make MLP$_1$ the best adaptation algorithm in this scenario: aside from some very rare rebuffering events, it is an improvement over the state of the art in all the considered QoE metrics over real capacity traces. Nevertheless, as shown in the following section, any network with a limited memory shows its limitations when a more complicated scenario is analyzed, and the presence of long-term correlation in the channel capacity makes a more complex approach necessary.

5.4.3 Results: synthetic traces

After running the adaptation algorithms over real capacity traces, we generated a set of traces with the *ns–3* simulator to gauge the effects of TCP cross-traffic on the algorithms' performance. The traces were generated by measuring the throughput of a saturated TCP flow sharing a bottleneck with a capacity of 10 Mb/s and a latency of 50 ms with 19 other TCP clients. Each

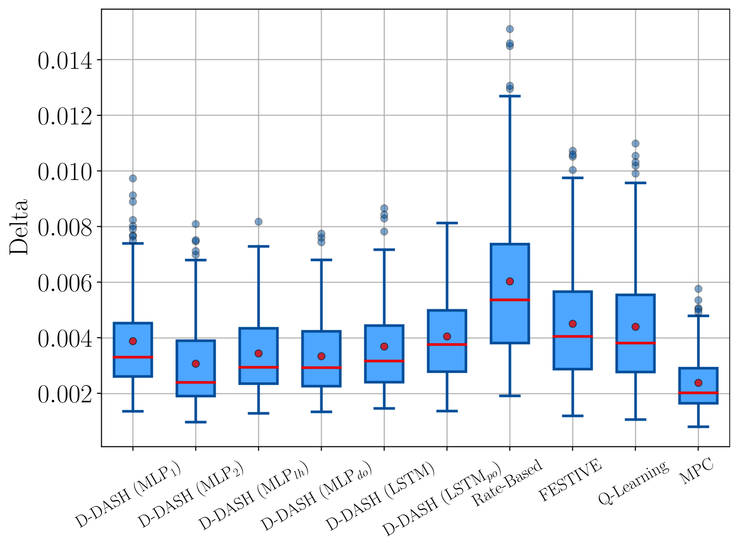


Fig. 5.8: Boxplot of the average SSIM variation for the 100 video episodes in the *test* phase (real traces).

of the cross-traffic clients generated TCP traffic as an on/off source: the off period was exponentially distributed with a mean of 2 seconds, while the on time was set to 4 seconds. This traffic model was designed to introduce long-term correlations in the available capacity [198], which are notoriously hard to handle for adaptation algorithms. In this part of the study, we only tested the MLP$_2$ (both with the short and long memory) and LSTM algorithms for clarity, since the performance of the other variants was similar.

Figure 5.10 shows the average SSIM over the *test* phase using the synthetic traces. In this situation, the advantage of the D-DASH algorithms is more marked: the MLP$_2$ and LSTM are able to maintain an average SSIM above 0.98 for all the considered videos; standard Q-learning and MPC also perform better than the Rate-Based heuristic and FESTIVE.

Figure 5.11 shows the net advantage of a long memory when dealing with long-term correlations: although the average SSIM of LSTM and MLP$_{\mathrm{lh}}$ is the same as that of MLP$_2$ they can achieve it with half of the quality variation. Even if MLP$_{\mathrm{lh}}$ performs slightly better than LSTM, it considers a fixed amount of memory. The LSTM network has the additional capability to automatically adapt to different memory requirements, without increasing

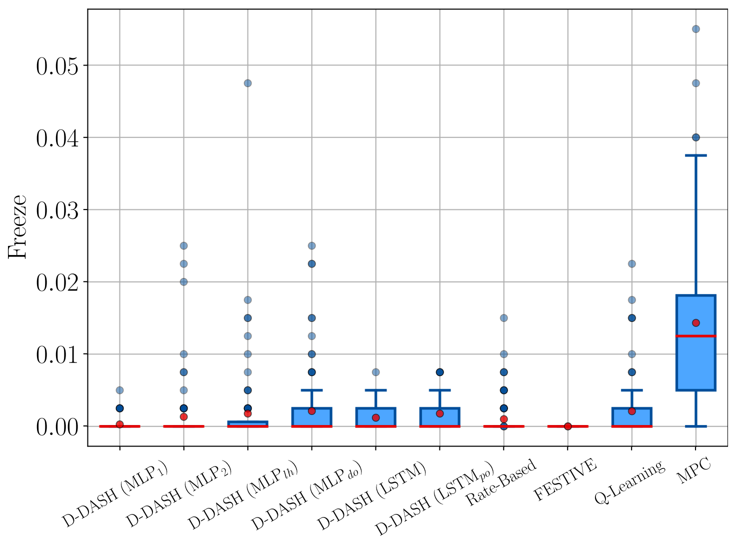


Fig. 5.9: Boxplot of the frequency of rebuffering events for the 100 video episodes in the *test* phase (real traces).

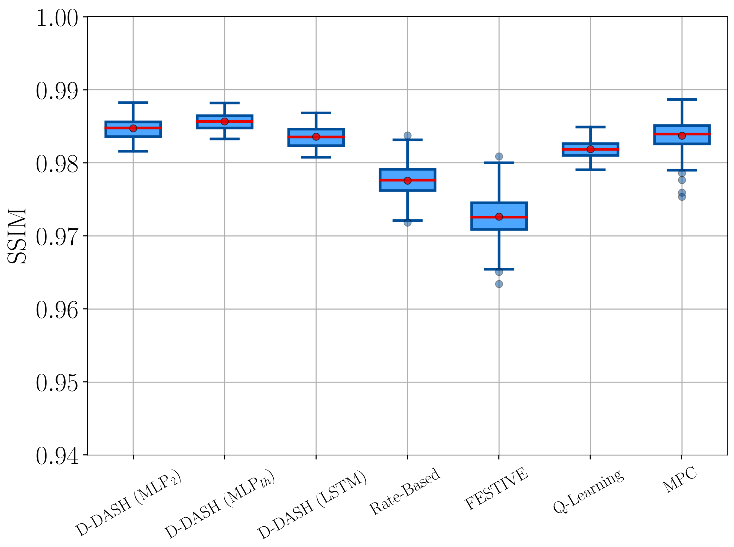


Fig. 5.10: Boxplot of the average SSIM for the 100 video episodes in the *test* phase (synthetic traces).

the state space dimension. The D-DASH algorithms and standard Q-learning all perform significantly better than FESTIVE and the Rate-Based heuristic on this metric. As with the real traces, MPC is the best at keeping a stable quality among the considered algorithms.

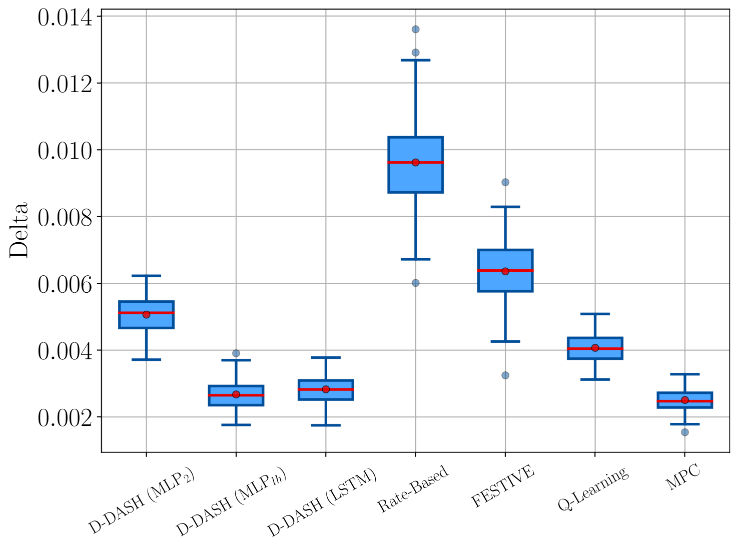


Fig. 5.11: Boxplot of the average SSIM variation for the 100 video episodes in the *test* phase (synthetic traces).

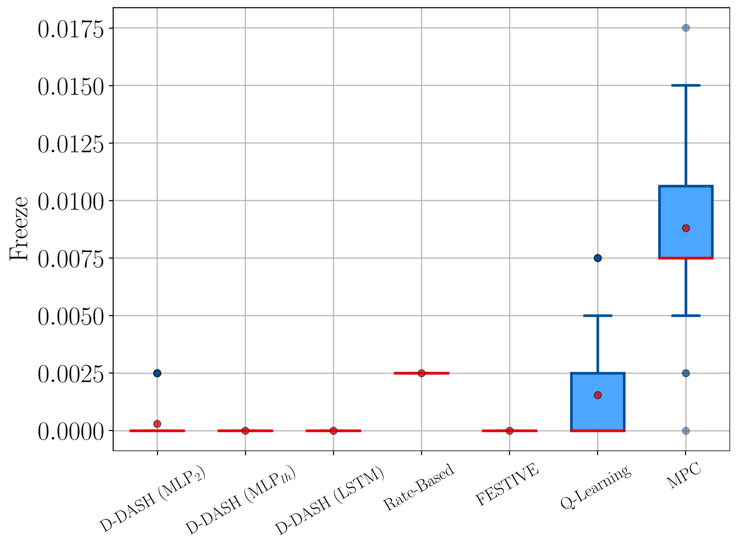


Fig. 5.12: Boxplot of the frequency of rebuffering events for the 100 video episodes in the *test* phase (synthetic traces).

Figure 5.12 shows the real advantage of the D-DASH framework in this scenario: while standard Q-learning has almost the same average SSIM as LSTM, MLP$_{\text{lh}}$ and MLP$_2$, and actually has smaller quality fluctuations than the latter, it can only achieve this at the cost of relatively frequent rebuffer-

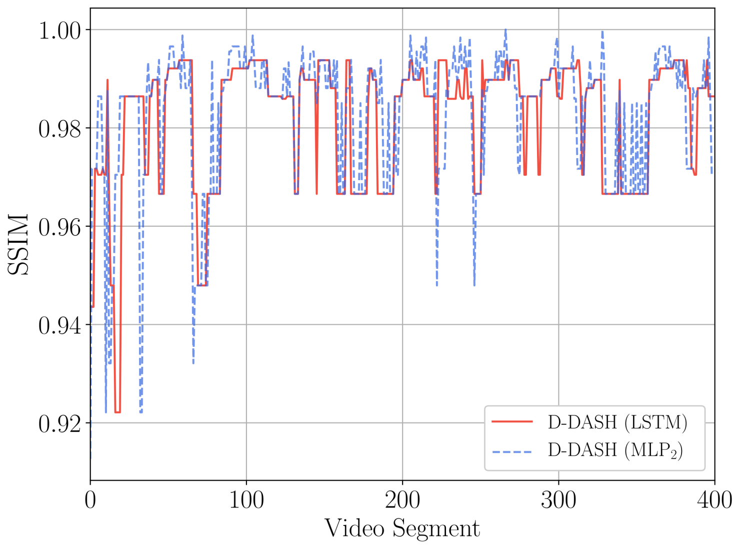


Fig. 5.13: Evolution of the SSIM during a video episode in the *test* phase, using the synthetic traces.

ings: a quarter of the video episodes have at least one rebuffering event, while this only happens for a few episodes with MLP$_2$ and never for LSTM. FESTIVE is also able to avoid rebuffering events, while the Rate-based heuristic is not. MPC still suffers from a high number of rebuffering events, totaling an average of 3 events per video episode.

Figure 5.13 shows the higher stability of LSTM with respect to MLP$_2$: for the sake of a higher stability LSTM avoids some of the quality increases. For this complex channel, LSTM is able to predict more accurately when a certain quality increase is likely to be sustainable, i.e., without having to shortly move back to the previous quality setting. Finally, a comparison of the convergence speed between standard Q-learning and D-DASH, performed during the pretrain phase, is provided in Figure 5.14: D-DASH algorithms converge much faster, making a better use of the video examples that are supplied during the learning phase.

Memory allocation

Since adaptation algorithms are client-side, their memory footprint is an important consideration. The number of variables required for D-DASH,

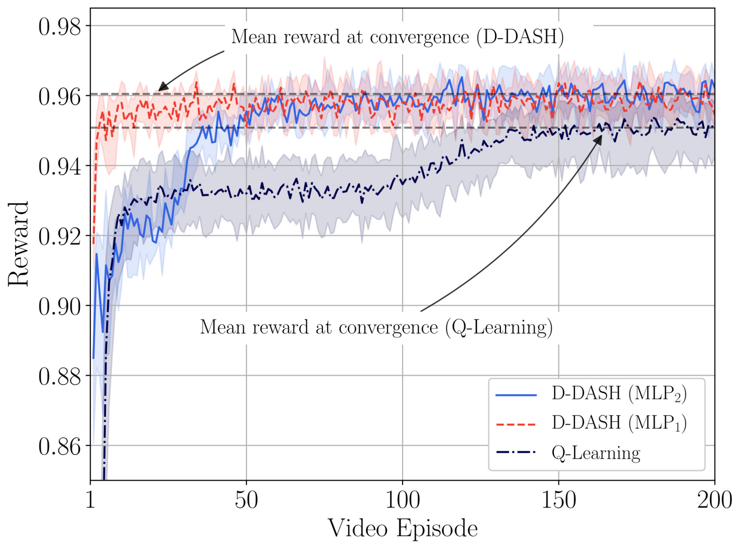


Fig. 5.14: Comparison of the convergence speed between standard Q-learning and D-DASH implementations (MLP$_1$ and MLP$_2$). The shaded area represents the interquartile range.

and also for the classic Q-learning (N_Q), are as follows:

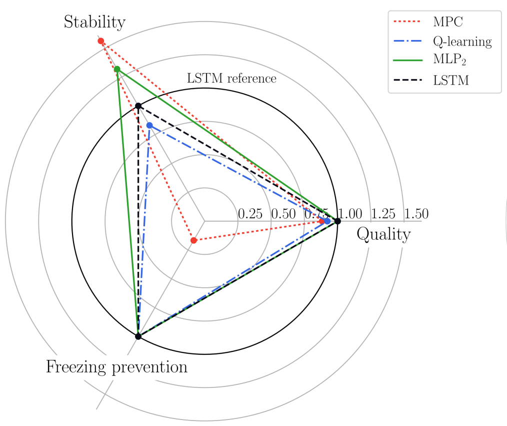
$$N_Q = N_s N_a \tag{5.7}$$

$$N_{MLP_1} = (V_s + 1)N_h + (N_h + 1)N_a \tag{5.8}$$

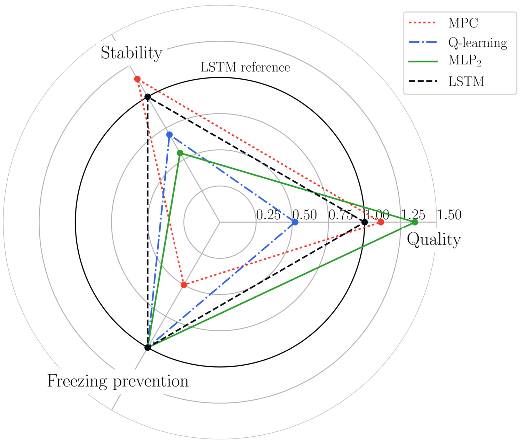
$$N_{MLP_2} = (V_s + 1)N_{h1} + (N_{h1} + 1)N_{h2} + (N_{h2} + 1)N_a \tag{5.9}$$

$$N_{LSTM} = 4(V_s + N_c + 1)N_c + (N_c + 1)N_a \tag{5.10}$$

where N_s is the cardinality of the state set, V_s is the number of state variables and N_a is the cardinality of the action set. Note that V_s and N_a also corresponds to the number of inputs and outputs of the neural networks, respectively (see also Figure 5.1). N_h, N_{h1} and N_{h2} are the number of hidden layer's neurons in the MLP networks, and N_c are the number of units in the LSTM cell. According to (5.8), (5.9) and (5.10), considering the values in Table 5.2 and 32 bit floating-point representation for storing real variables, the memory space required for MLP$_1$, MLP$_2$ and LSTM is about 14.4 kB, 143.4 kB and 276.5 kB, respectively. We remark that these values are fixed given a specific network implementation, in the sense that they do not de-



(a) real traces



(b) synthetic traces

Fig. 5.15: Summary of performance of video adaptation algorithms: (a) real traces, (b) synthetic traces (exhibit long-term correlation).

pend on the number of states. On the other hand, classic Q-learning space requirement directly depends on the cardinality of the state set, which is closely related to the quantization granularity of continuous state variables, as discussed in Section 2.2.2. The granularity that was used for the results in this work leads to a total memory space of 32 kB for Q-learning. The memory footprint needed by each of the presented methods appears reasonable considering the hardware of modern client devices. Note that, even if the number of variables for Q-learning is lower than those required by the D-DASH algorithms, their generalization capabilities and the concurrent update of the network's weights allow for a more efficient utilization of the experience acquired in the learning phase. This behavior can be seen in Figure 5.14, where the convergence speed of D-DASH is considerably lower.

5.4.4 Summary of performance

A summary of the performance of video adaptation techniques is shown in Figure 5.15. For a convenient visual comparison, the three metrics have been scaled and normalized with respect to the LSTM performance, according to the following criteria (the normalization term has been omitted for simplicity):

$$\begin{aligned} \text{Quality} &= 0.98 - \text{SSIM} \quad &(5.11)\\ \text{Stability} &= 1 \,/\, \text{Quality Variation} \quad &(5.12)\\ \text{Freezing Prevention} &= 0.015 - \text{Frequency of rebuffering} \quad &(5.13) \end{aligned}$$

The proposed deep-Q learning based schemes significantly outperform existing Q-learning and standard techniques from the literature. As Figure 5.15a shows, with real traces MPC achieves a more stable, albeit slightly lower, quality than either MLP_2 or LSTM, but it fails to avoid rebuffering events and results in a far worse QoE for the user. Other state of the art algorithms, such as FESTIVE, which is not shown in the plot for readability, manage to avoid rebuffering events but perform much worse in the other two metrics. Our algorithms are the only ones reaching high scores on all the three considered metrics, i.e., video quality, stability and rebuffering avoidance. Furthermore, the D-DASH algorithms converge faster compared to standard Q-learning schemes, which require hundreds of video episodes to reach an acceptable performance. In fact, they approach optimal policies after just a few videos, or even just a couple in the case of the MLP_1 scheme. This also makes it possible to consider an online version that adapts to video and network conditions, learning how to deal with each new situation as it arises and memorizing it for future use. The longer memory of the LSTM algorithm proved to be very valuable on channel traces with long-term correlation: when the channel correlation stretches to over 10 seconds, LSTM shows significantly better performance than other schemes in all the three metrics, as Figure 5.15b shows. Also in this case, MPC obtains a higher stability than LSTM, but it pays for it by having several rebuffering events per episode.

Chapter 6

Optimizing Smart City services with data-driven techniques

The Smart City [199] paradigm extends the IoT vision [200] by focusing on an urban scenario in which the data collected from sensors are exploited to optimize services delivered to citizens, improve city management and quickly react to events within the city. Today, the idea is getting a lot of attention from the research community, the industry and city institutions [199, 201]. This popularity is due to the huge expectations for its vision, which promises to offer better services to citizens with lower costs for public administrations.[1]

Communications networks play a crucial role in this framework [202], as shown in Figure 6.1, and they are already used as the underlying infrastructure for several modern city services; this is only going to increase with the integration of the IoT into the 5G paradigm, and we will discuss the Smart City scenario from a network perspective in Chapter 7. For example, pressure bands and induction coils are often placed underneath the tarmac of the main city roads to monitor the intensity of the traffic entering and exiting the city; "dome" cameras are used for surveillance and traffic monitoring of intersections or critical areas; weather stations collect environmental data in different parts of the city to monitor temperature, humidity, rain intensity, and air pollution; road signs and smartphone applications provide real-time

[1] Pike Research on Smart Cities: `http://www.pikeresearch.com/research/smart-cities`.

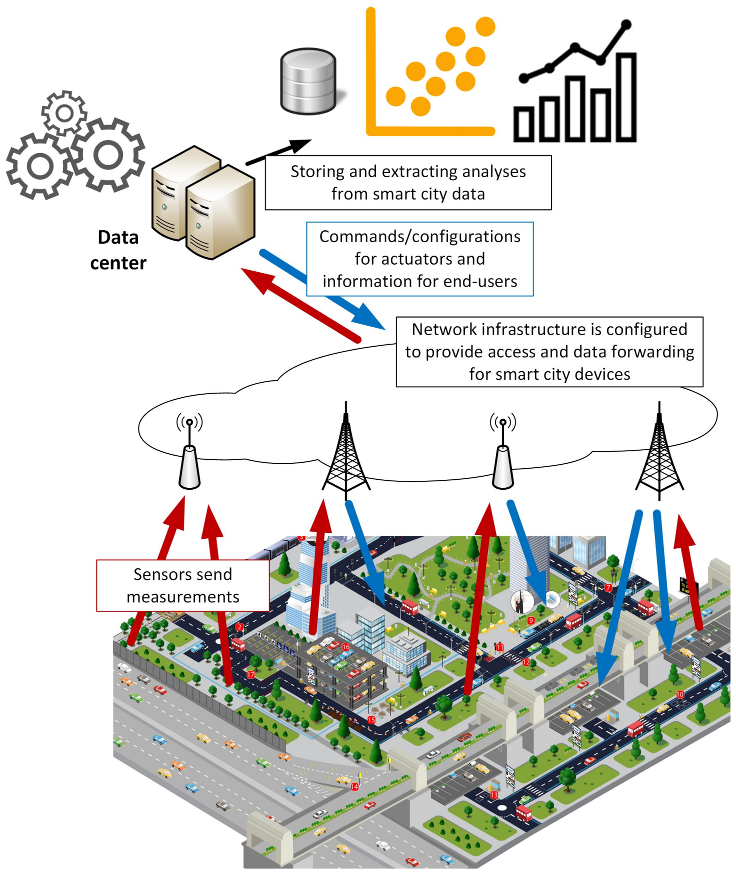

Storing and extracting analyses from smart city data
Data center
Commands/configurations for actuators and information for end-users
Network infrastructure is configured to provide access and data forwarding for smart city devices
Sensors send measurements

Fig. 6.1: A representation of a Smart City environment which highlights the role of the communication infrastructure in providing connectivity to Smart City devices.

information about available parking places and public transport services, while other apps are used for car and bike sharing services.

The services enabled by the Smart City range from human-oriented services targeting a better quality of life, to machine-oriented services handling the management of the Smart City itself [203]. The final goal of Smart Cities is to build smart communities [204] consisting of humans and machines interacting both in the physical and virtual realms, but the significant complexity of such a vision makes data-driven and intelligent techniques a necessity to optimize services.

6.1 Bike sharing in Smart Cities

Bike sharing is a key component of the emerging Smart City paradigm [203]: bike sharing services offer more flexibility than standard public transportation, while also reducing both traffic and its impact on the environment [205] and improving public health [206].

Bike sharing is a perfect example of a Smart City-enabled service. Although bike sharing schemes have existed since the 1960s [207], issues with theft, vandalism and wrongful use prevented their widespread adoption [208] until the arrival of smart biking systems. With the possibility of electronically unlocking bicycles and identifying users, smart bike sharing systems solved or mitigated these issues: by requesting users' credit card information before lending the bikes, bike sharing companies can charge users for damages or theft, with a strong deterrent effect. The use of technology also allowed cities to provide a better service, embedding sensors to obtain real-time data, which can be used to plan and manage the bike sharing docking stations and adapt to the needs of the users. Many services provide live maps of the available bikes, and ways to detect broken bikes remotely to quickly repair or substitute them are being studied [209]. Over the past few years, bike sharing schemes were implemented in most major world cities, with almost universal success.

Such services leverage technology to identify users, track the paths they cover, and provide real-time information on the current bike availability through, e.g., smartphone applications. Sensors embedded in the bikes enable the collection of real-time data that can be used in the short term to dynamically adapt the service to the current needs of users, and, in the long-term, to enhance the system by adding bikes or stations where they are needed most and plan new bike lanes to cover the most common routes [210].

Despite the great success of bike sharing all across the world, smart sharing systems are not perfect, and improving their efficiency is an active research goal. Finding a way to maintain availability during rush hour is one of the most important issues faced by modern bike sharing systems: traffic is not homogeneous across stations and time, and there may be empty and full stations, preventing users from either taking or depositing a bike, respectively. This dishomogeneity requires the development of smart techniques to manage large bike sharing systems, so that bikes are available *where* and *when* needed, and users are not dissatisfied with the service. An effective way to deal with this problem consists in relocating bikes from overcrowded stations to those with a shortage of bikes. This technique is commonly known as *rebalancing*. Operators can, e.g., deploy a fleet of trucks to move the bikes across stations and reach a configuration that minimizes the probability of stations being either empty or completely full. The design of an efficient rebalancing scheme requires addressing two major challenges: identifying where bikes need to be at a given moment to meet the demand, and selecting the most efficient path to relocate bikes within the required time [211].

However, exploiting a fleet of vehicles to rebalance the network entails a high economic cost for fuel, drivers, and truck cost and upkeep. An emerging strategy to reduce such cost is to give incentives to the users themselves, so that they are encouraged to walk for part of their path to reach a less congested station, thus reducing the heterogeneity in the traffic demand. Such incentives may be provided through a smartphone application, and may consist in cash prizes or privileges for service utilization, like reserving a bike, premium tickets for other transportation modes, etc. However, this is out of the scope of this thesis, and here we are solely interested in the effects that such incentives bring to the system. For a more in-depth review of the various types of incentives and their effects, we refer the readers to the vast body of literature on gamification [212, 213].

The basic concept behind the two paradigms is simple, and we show it in Figure 6.2: since the demand of bikes tends to drive the system towards an unacceptable state with empty stations where the demand for bikes is higher and full stations where users deposit bikes, as shown in the left panel, there are two ways to restore the system and improve its availability. Standard rebalancing takes the system back to a more stable state by moving bikes, i.e., altering the state directly as shown in the middle panel; incentive-based traffic shaping acts on the demand without changing the state, so that the bike sharing system is steered towards acceptable states, as shown in the right panel.

In this section, we introduce an analytical model for bike sharing network rebalancing, which includes the effect of both user incentives and traditional truck-based rebalancing. First of all, we model the occupancy of each station as a Birth-Death Process (BDP), with time varying birth and death rates (i.e., arrivals and departures, respectively); this model is fitted with data from the New York City's CitiBike bike sharing system,[2] which is publicly available and contains historical data since 2013. This allows us to infer the target network configuration, i.e., the one that minimizes the probability of a stations being unusable. We then consider a combined approach to reach the desired status of the bike sharing network and limit the user dissatisfaction: the users of the service are given incentives to choose routes that help moving the network state away from dangerous states, and, when the incentives are not themselves sufficient, trucks are deployed to move bikes across stations and restore system availability. The rebalancing scheme we propose is dynamic, so that bike reallocation is adaptive rather than performed at predetermined times. Therefore, it requires to determine whether the gain obtained with rebalancing overcomes the cost of employing the trucks, and the path they should cover.

The results of our numerical evaluation show that combining incentives and traditional rebalancing can yield great benefits in terms of performance, while at the same time reducing operating costs. By combining strong incentives and a dynamic rebalancing strategy, we were able to almost halve both the downtime and the number of failed trips due to empty or full stations.[3]

Notation. Matrices are represented with bold characters. The i-th row of a matrix $\boldsymbol{A}$ is denoted as $\boldsymbol{A}_i$, and the j-th element of the i-th row is denoted as $A_{i,j}$.

[2] `https://www.citibikenyc.com/system-data`

[3] The work in this section is the combination of several published papers: the data analysis was presented at MOCAST 2018 and published in the conference proceedings [214], the model was first published in MDPI Sensors [211], and a paper describing the incentive scheme has been submitted for publication to the IEEE Transactions on Intelligent Transport Systems.

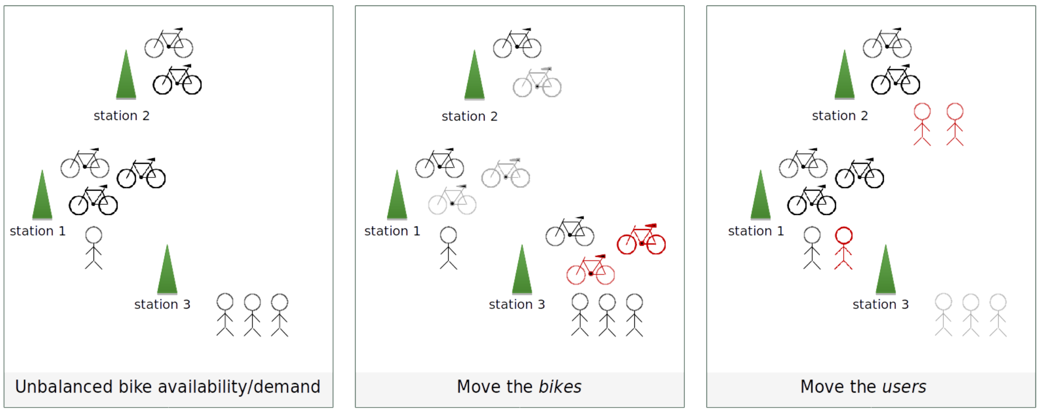


Fig. 6.2: Diagram showing the rebalancing and user incentives paradigms in optimizing bike sharing systems

6.2 State of the art

The widespread implementation of bike sharing systems raised an increasing research interest, bolstered by the availability of real data from publicly open bike sharing services in many cities. The analysis of historical data enables the design of models and frameworks that give insights on how and when the service is used, and help improve the system. Several works propose analytical schemes, which are generally based on Markov chains [211] or queuing theory [215]. Other studies use a different approach, exploiting machine learning tools to learn the bike traffic pattern directly from the observed data [216]. Whilst the former approaches provide parameterized models that can be readily adapted to the various peculiarities of different bike sharing systems, the latter ones allow to include information that is troublesome to consider in an analytical model but has a significant effect on the service usage, such as weather conditions [214]. Traffic analysis is also useful to construct clusters of stations with similar demand patterns [217], which may help the rebalancing optimization.

The historical data analysis is necessary to understand the needs of the users, but the heterogeneous user behaviors and unexpected or rare events (e.g., a snow storm, a bus strike, or a city event that attracts many tourists) make it challenging to accurately predict the traffic pattern [218]. To counteract the uneven bikes usage across stations and time, external intervention is necessary to prevent or limit system failures and ensure bike availability. As we explained above, this can be achieved by either incentivizing users to modify their behavior or employing trucks to move the bikes across stations. In the following, we provide a high-level overview of the vast body of literature about rebalancing and pricing schemes. Interested readers can refer to [219] and [220] for comprehensive surveys on the state of the art literature about bike sharing systems.

6.2.1 Rebalancing bike sharing systems

The most common rebalancing approaches can be classified as *static*, where the rebalancing is performed according to a predetermined schedule, likely when the system use is minimum (e.g., at night); or *dynamic*, in which the rebalancing occurs during the daytime, when needed.

In the literature, static rebalancing problems are often tackled through Mixed Integer Programming (MIP) techniques. For example, Ref. [221] uses a branch-and-cut algorithm on a relaxed MIP model together with a taboo search to obtain upper bounds to the NP-hard problem of redistributing bikes among stations with the minimum traveling distance [222]. A similar method is used by Ho and Szeto in [223] inside an iterative procedure, resulting in a better solution. Several MIP techniques are used in [224], where a convex penalty objective function aims at minimizing both the user dissatisfaction and the costs of moving the vehicle fleet. The numerical experiment shows significant service quality improvements for networks with up to 100 stations with two repositioning vehicles. The authors also take into account the time needed to load and unload bikes. Constraint programming is used in [225], where a large neighborhood search approach is used to tackle the problem of balancing the bike sharing system. In [226], the authors propose a three-step mathematical programming based heuristic, which consists of clustering stations based on geographical and inventory considerations, and then constructing an appropriate traversal route. Differently from other approaches, rebalancing is not limited within clusters, but the routing vehicles can travel through a sequence of clusters. Similarly, [215] proposes a static rebalancing scheme where each vehicle is assigned a 'self-sufficient' cluster of stations. Stations are in fact grouped in such a way that the target network configuration can be achieved by performing only within-cluster pickups and deliveries. Clusterization is useful to organize rebalancing paths more easily, but adopting such a technique in a dynamic rebalancing context is not trivial, since each rebalancing operation may involve different stations and clusters should continuously be identified. In [227], Dell'Amico et al. propose a metaheuristic that uses a destroy and repair approach to solve the static rebalancing problem, improving on the simple branch-and-cut approach that the same authors presented in [228], both in the quality of the solution and in the convergence time. These works are particularly interesting, as they consider the variability of the demand for bikes during the day and the different features of the stations, distinguishing between three different types of stations with similar traffic patterns.

Static approaches, however, may not be sufficient to avoid network failures during the day. To overcome this issue, dynamic rebalancing aims at redistributing bikes throughout the day according to the current network state. Clearly, this is much more challenging, since it includes a scheduling component based on the users' activity during the rebalancing operation, and also a routing problem.

In [229], upper and lower bounds for the solution of a pickup-and-delivery problem are obtained by using Dantzig–Wolfe [230] and Benders [231] decompositions, respectively, but the study does not deal with a time-varying demand and, even in this simple scenario, the optimality gap of the approximation is large. An exact solution to the static rebalancing problem using Benders' cuts for given target levels is instead provided in [232].

In [233], the loading instructions for the rebalancing vehicles are derived from an optimization function that weighs the unfulfilled user demand with the target filling level. Furthermore, each truck is assigned a capacity so that there is a maximum number of bikes it can load. An accurate simulation model is proposed in [216]: it simulates the bike sharing system in space and time to determine the optimal repositioning flows and time intervals between relocation operations, by explicitly considering the route choice for trucks among the stations. Some studies also determine the optimal size of the rebalancing fleet [234]. An MDP is used in [235] to solve a stochastic inventory routing problem for bike sharing systems. The objective is to minimize the number of expected violations of *due dates*, where a due date is the deadline within which a station has to be served by a rebalancing vehicle in order to satisfy the requests and, hence, avoid failures. The paper shows the importance of using both long-term and short-term relocation strategies. The former tries to estimate the target level that can deal with predicted demand, while the latter assigns a priority level to each station based on its urgency of being rebalanced. The short-term approach is similar to the one proposed in this paper, since we dynamically serve only those stations that are going to experience a failure in a short time horizon. An interesting framework is presented in [236], where rebalancing comprises two different strategies to tackle the bike imbalance during night and day, respectively: (i) static rebalancing overnight to move the network to an optimal configuration that minimizes the probability of stations becoming either empty or full in the following day, and (ii) clustering optimization to handle rush-hour usage and ensure that users are never too far from an available bike or dock. The joint use of the two strategies brings a larger improvement than considering the nightly rebalancing only. This result encouraged us in choosing a dynamic approach.

For a more detailed discussion on the rebalancing problem and other open research issues in bike sharing systems, we refer the reader to [219] and [220].

6.2.2 User incentives and pricing

The main cause of service failures in bike sharing systems is the unevenness of traffic patterns: depending on the type of district a station is in (e.g., residential, industrial, or commercial), and on the facilities it is close to (e.g., subway stations, shops, university buildings), its peak hours and the balance between pickups and drop-offs will be significantly different. Rebalancing, i.e., redistributing bikes across the network, can mitigate the problem by bringing the system back to a state with a lower risk of service failures. However, moving bikes around has a high operating cost; there are no academic papers estimating the daily rebalancing cost, but a calculation by Better Bike Share[4] based on public budget data from Denver's B-Cycle system estimates a rebalancing cost of about \$1.50 per ride.

An emerging strategy to reduce the rebalancing trips and still improve the bike sharing service consists in developing pricing strategies to incentivize users to return bikes at neighboring stations so as to strategically minimize the number of imbalanced stations. As Smart Cities often seek to be also green cities, such a solution has been gaining increasing interest thanks to its low cost and impact on the environment.

An analytical model for a pricing heuristic was proposed in [237]: when a user arrives at a station to take a bike, it randomly select two possible destinations and the system directs it toward the least loaded one. This study, however, considers an homogeneous scenario, in which all stations have the same parameters, which is very far from a real deployment. A more realistic scenario is used in [238], which models the incentive problem as a Stackelberg game that relies on the key idea of price discrimination. Users are in fact divided into leisure travelers and commuters, and the former are stimulated to redistribute the bikes as needed to limit the system failures, while the latter category is encouraged to avoid traveling during peak periods. In [239], vehicle-enabled rebalancing is followed by a dynamic pricing mechanism; this framework is similar to ours, although the two schemes are performed in the reverse order and [239] assumes deterministic customer arrivals and linearized customer reaction to incentives. Interestingly, the results show that incentives are much more effective during weekends, when users are fewer and likely do not have timing constraints, while an external intervention is needed during weekdays, especially when the commuting rush hour is more prominent.

A field trial of a pricing mechanism has been deployed for 30 days in Mainz, Germany, based on the architecture proposed in [240]. Users are

[4]`http://betterbikeshare.org/2016/08/16/much-bike-share-ride-cost-system-lets-math/`

engaged in the bike repositioning process through a smartphone application, and the incentives they are given are dynamically adapted based on the current system state through an online learning framework. New York City's CitiBike system, which we analyze in this work, has also implemented an experimental opt-in incentive scheme called Bike Angels[5], which mixes cash and service-related rewards; the system operates over a very simple heuristic point-based model which divides the station into 4 categories, but it has already reduced the number of rebalancing trips by 10%, according to a recent statement by the company that operates CitiBike [6]. The success of the program confirms the potential of the idea, and it has encouraged further studies on data-driven approaches to incentive programs, some specifically targeting Bike Angels and New York [241].

6.3 System model

Rebalancing schemes are necessary to move the bikes across stations and reach a configuration that minimizes the probability of stations being either empty or completely full. The ultimate goal is to reduce the user dissatisfaction due to the impossibility of using the bike sharing service. In this work, we try to achieve this goal by minimizing either the expected downtime or the number of expected failures.

In the following, we describe how we model the bike sharing system and how to derive the expected downtime and the number of expected failures.

6.3.1 The bike sharing system as a network

We model the bike sharing system as a fully connected directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of stations and the set of edges $\mathcal{E} = \mathcal{V} \times \mathcal{V}$ contains all possible routes between stations. Each node $v \in \mathcal{V}$ has a capacity $M_v \in \mathbb{N}$, corresponding to the maximum number of bikes that can be docked at the station; naturally, the occupancy $m_v(t)$, i.e., the number of bikes that are

[5]`https://www.citibikenyc.com/bikeangels/`

[6]`https://www.motivateco.com/in-the-news-slate-on-citi-bikes-bike-angels-program/`

docked at the station at time t, can only take a value in the set $\mathcal{M}_v \triangleq \{0, 1, \ldots, M_v\}$. Each edge is associated to a distance metric, which might depend on the edge's direction to account for traffic, one-way streets, and different routes between the two nodes. It is then possible to define a distance matrix $\boldsymbol{d}$ whose element $d_{u,v}$ represents the distance from station u to station v.

For the sake of mathematical tractability, we assume that the operating day is divided into discrete time frames of length T_r, and that rebalancing can be done only at the beginning of such frames. Hence, for each time frame k, we need to derive the desired state $m_v^\star(kT_r)$ for each station $v \in \mathcal{V}$, defined as the state that minimizes the expected downtime of the station, i.e., the fraction of time the service will be unavailable at that station.

To avoid scalability issues, we model the process of departures and arrivals at each station as an independent Markov-Modulated Poisson Process (MMPP) [242], whose arrival and departure rates follow historic patterns. Consequently, the occupancy of station v follows a finite Markov BDP $m_v(t)$ limited by the station's capacity M_v. To simplify the analysis of the historical data and reduce the computational effort, we consider the BDP to be discrete with time step T_p [243, 244]. The birth and death processes are Poisson distributed with time-varying rates, namely $\lambda_v(t)$ and $\mu_v(t)$, that are assumed to be independent of the current state $m_v(t)$. According to the discrete model, the birth and death rates are piecewise constant during time frames of duration T_r, which is chosen as a whole multiple of the slot duration T_p used for the dynamics of the BDP. In practice, the continuous time t is mapped into a couple (k, n) where $k = \lfloor t/T_r \rfloor \in \{0, 1, \ldots\}$ corresponds to the frame index, and $n = \lfloor (t - kT_r)/T_p \rfloor \in \{0, 1, \ldots, T_r/T_p\}$ is the slot index within frame k. In our previous work [211], we verified this model against real data from the New York CitiBike system, which proved to follow the discrete time BDP with Poisson birth and death rates extremely well.

6.3.2 Downtime at a station

In this section, we derive the expected downtime at a station, which is the time during which users can neither pick up nor drop bikes at that station. More specifically, we define the expected downtime T of a station v over a finite time horizon NT_p as the total time period during which the station is either in state $m_v(t) = 0$ or $m_v(t) = M_v$. We can then consider two matrices $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$, with size $|\mathcal{V}| \times N$, that contain the arrival and departure rates of all stations during the considered time window, respectively. The v-th row $\boldsymbol{\lambda_v}$ of matrix $\boldsymbol{\lambda}$ contains the arrival rates of station v; row $\boldsymbol{\mu_v}$ is defined similarly for the departures.

Since the two conditions of empty and full stations are mutually exclusive, T can be expressed as the weighed sum of the expected time T_e that the station will spend in the empty state plus the expected time T_f such station will spend in the full state:

$$T(m_v; \boldsymbol{\lambda_v}, \boldsymbol{\mu_v}) = \alpha_e T_e(m_v; \boldsymbol{\lambda_v}, \boldsymbol{\mu_v}) + \alpha_f T_f(m_v; \boldsymbol{\lambda_v}, \boldsymbol{\mu_v}), \tag{6.1}$$

where α_e and α_f are weighting constants that define the relative importance of each failure mode, and m_v is the current station's occupancy. Although, rigorously, both T_e and T_f, and thus T, depend on the birth and death rates of the BDP during the considered period, we will omit such dependency from the notation for the sake of readability (unless misleading). T_e and T_f are defined as follows:

$$T_e(m_v) = \sum_{\ell=0}^{N-1} p(m_v(\ell T_p) = 0 | m_v(0)) \tag{6.2}$$

$$T_f(m_v) = \sum_{\ell=0}^{N-1} p(m_v(\ell T_p) = M_v | m_v(0)). \tag{6.3}$$

To compute T_e and T_f, we thus need to calculate the probability that station v will be empty or full at any given time. Since we model the occupancy of a station as a discrete-time BDP, this can be done using the ℓ-step transition probability. We denote as $\boldsymbol{P}(\ell T_p; \{\boldsymbol{\lambda_v}\}_1^\ell, \{\boldsymbol{\mu_v}\}_1^\ell)$ the transition matrix of a station at time ℓT_p, and as $P_{i,j}(\ell T_p; \boldsymbol{\lambda_v}, \boldsymbol{\mu_v})$the corresponding element in the i-th row and j-th column, which is the probability that the considered station transitions from state i to state j. Vector $\{\lambda_v\}_1^\ell$ contains the elements from 1 to ℓ of the arrival rate vectors, which are the arrival rates of the first ℓ time slots; analogously for $\{\mu_v\}_1^\ell$. The Markov property then allows us to write (6.2), (6.3) as:

$$T_e(m_v) = \sum_{\ell=1}^{N-1} T_p P_{m_v,0}(\ell T_p; \{\boldsymbol{\lambda_v}\}_1^\ell, \{\boldsymbol{\mu_v}\}_1^\ell) \tag{6.4}$$

$$T_f(m_v) = \sum_{\ell=1}^{N-1} T_p P_{m_v,M_v}(\ell T_p; \{\boldsymbol{\lambda_v}\}_1^\ell, \{\boldsymbol{\mu_v}\}_1^\ell) \tag{6.5}$$

Note that the transition matrix depends on the birth and death rates, which, as explained in Sec. 6.3, are not constant but piecewise constant with a step T_r. For each time window ℓT_p with $\ell > 1$, applying the Markov property results in:

$$P_{i,j}(\ell T_p; \{\boldsymbol{\lambda_v}\}_1^\ell, \{\boldsymbol{\mu_v}\}_1^\ell) = \\ \sum_{h \in \mathcal{M}_v} \left(P_{i,h}((\ell-1)T_p; \{\boldsymbol{\lambda_v}\}_1^{\ell-1}, \{\boldsymbol{\mu_v}\}_1^{\ell-1}) \cdot P_{h,j}(T_p; \{\boldsymbol{\lambda_v}\}_\ell^\ell, \{\boldsymbol{\mu_v}\}_\ell^\ell \right), \tag{6.6}$$

Note that $\{\boldsymbol{\lambda_v}\}_\ell^\ell$ contains only the ℓ-th element of vector $\boldsymbol{\lambda_v}$, which corresponds to element $\lambda_{v,\ell}$ of the arrival rate matrix; analogously for $\{\boldsymbol{\mu_v}\}_\ell^\ell$. It follows that, to compute (6.4) and (6.5), it is sufficient to derive the single-step transition matrix in one time slot and then apply (6.6) recursively.

Therefore, we now proceed to compute the one-step transition probability $P_{i,j}(T_p; \lambda_{v,\ell}, \mu_{v,\ell})$ in an arbitrary slot ℓ of length T_p from state i to state j. We assume that the state of station v in the next slot only depends on the total number of departures and arrivals, and not on their order. This is not rigorously true, because some orderings could yield a system failure, with the station being empty or full, but the probability of this happening is negligible. The number of arrivals and departures at station v during the considered subframe are modeled as two random variables, A_v and D_v, respectively. We model the demand for bikes (i.e., potential departures) and empty stalls (i.e., potential arrivals) as Poisson processes, with means $\mu_{v,\ell}T_p$ and $\lambda_{v,\ell}T_p$, respectively. The one-step transition probability from state i to state j is then equal to the probability that the difference between arrivals and departures is $j-i$:

$$\begin{aligned}P_{i,j}(T_p;\lambda_{v,\ell},\mu_{v,\ell}) &= \Pr(A_v - D_v = j-i)\\ &= \sum_{h=-\infty}^{\infty} \Pr(A_v = j-i+h)\Pr(D_v = h)\\ &= \sum_{h=\max\{0,j-i\}}^{+\infty} \frac{(\lambda_{v,\ell}T_p)^{-(j-i+h)}\,(\mu_{v,\ell}T_p)^{-h}}{h!\,(j-i+h)!}.\end{aligned} \tag{6.7}$$

The last sum in (6.7) follows a truncated Skellam distribution $p_{\text{Sk}}(\cdot)$ [245], which depends on two hyperparameters. In our previous work [211], we explain how to deal with the truncation in the range $[0, M_v]$, which allows to explicitly write the one-step transition probabilities:

$$\begin{aligned}P_{i,j}(T_p;\lambda_{v,\ell},\mu_{v,\ell}) &= p_{\text{Sk}}(j-i;\lambda_{v,\ell}T_p,\mu_{v,\ell}T_p),\, 0<j<M_{v,\ell}\\ P_{i,0}(T_p;\lambda_{v,\ell},\mu_{v,\ell}) &= \sum_{k=-\infty}^{0} p_{\text{Sk}}(k-i;\lambda_{v,\ell}T_p,\mu_{v,\ell}T_p)\\ P_{i,M_{v,\ell}}(T_p;\lambda_{v,\ell},\mu_{v,\ell}) &= \sum_{k=M_{v,\ell}}^{\infty} p_{\text{Sk}}(k-i;\lambda_{v,\ell}T_p,\mu_{v,\ell}T_p).\end{aligned} \tag{6.8}$$

The one-step transition matrix at the beginning of a frame with birth and death rates $\lambda_{v,\ell}$ and $\mu_{v,\ell}$ is defined in (6.8), which yields the transition probabilities at each slot ℓ within that frame. Finally, the Markov property is used to take into account previous slots and compute the desired probability as in (6.6). This is then used to calculate T_e (see (6.4)) and T_f (see (6.5)), and therefore the expected downtime of the station.

6.3.3 Expected number of system failures

We also consider a model where the relevant variable to be minimized is not the downtime, but the number of expected failures, which corresponds to the downtime multiplied by the expected traffic rate. Failures in fact happen during a station's downtime and represent the number of users that cannot use the service. In this case, the objective is not to keep the system available as long as possible, but to make it work effectively at rush hour, even if its availability during off hours is slightly reduced. The number of expected failures F is defined as the sum of the number of trips F_e that are canceled because the station the user is at is empty, and the number of times F_f that a user is forced to deposit their bike at a station that is not their preferred one because it is full. This can be expressed as:

$$F(m_v;\boldsymbol{\lambda_v},\boldsymbol{\mu_v}) = \alpha_e F_e(m_v;\boldsymbol{\lambda_v},\boldsymbol{\mu_v}) + \alpha_f F_f(m_v;\boldsymbol{\lambda_v},\boldsymbol{\mu_v})) \tag{6.9}$$

$$F_e(m_v;\boldsymbol{\lambda_v},\boldsymbol{\mu_v}) = \sum_{n=1}^{N} \mu_{v,n} T_p P_{m_v,0}(n,\boldsymbol{\lambda_v},\boldsymbol{\mu_v}) \tag{6.10}$$

$$F_f(m_v;\boldsymbol{\lambda_v},\boldsymbol{\mu_v}) = \sum_{n=1}^{N} \lambda_{v,n} T_p P_{m_v,M_v}(n,\boldsymbol{\lambda_v},\boldsymbol{\mu_v}). \tag{6.11}$$

where, as for the downtime T, α_e and α_f are weights that can be tuned to change the relative importance of each type of failure. The derivation of the transition probabilities is the same as described in Section 6.3.2, and the computation of F_e and F_f follows the same steps as that of T_e and T_f.

6.3.4 The incentive problem

The basic idea behind the incentives is that some users would be willing to walk to a nearby (and less congested) station, or deposit their bike at a station slightly farther from their destination, in exchange for a small reward.

In the following, we introduce our incentive model to characterize the contribution given to the system by its own users, and then explain how to solve it.

6.3.5 The incentive model

As stated above, we assume that we can convince users to slightly change their route by introducing an incentive. This behavior changes the arrival and departure rate at each station, so that the original matrices $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ become $\boldsymbol{\lambda'}$ and $\boldsymbol{\mu'}$. The incentive optimization problem consists in determining the new arrival and departure rate matrices, which minimize the expected downtime T at a station (see Section 6.3.2 and (6.6)). The optimization objective can be stated as

$$\boldsymbol{\lambda}^*, \boldsymbol{\mu}^* = \underset{\boldsymbol{\lambda'}, \boldsymbol{\mu'}}{\arg\min} \sum_{v \in \mathcal{V}} T(m_v, \boldsymbol{\lambda'_v}, \boldsymbol{\mu'_v}). \tag{6.12}$$

where we explicitly highlighted the dependence of T on the arrival and departure rates.

Naturally, we need to set constraints to keep the estimated effects of the incentives realistic, as well as to prevent outcomes that are negative to the system. It is reasonable to assume that using positive incentives will not damp the demand, as users who do not care about the reward will simply ignore the incentive and take the trip they originally planned. In this work, we also assume the incentives to be small enough that the number of additional users who take rides they would not have just to get the reward is negligible. In this case, the total demand of the network will not change, but it will simply be shifted from one place to another. We can express this as a constraint to the optimization problem:

$$\sum_{v=1}^{V} \lambda_{v,n} = \sum_{v=1}^{V} \lambda'_{v,n} \ \forall n \tag{6.13}$$

$$\sum_{v=1}^{V} \mu_{v,n} = \sum_{v=1}^{V} \mu'_{v,n} \ \forall n. \tag{6.14}$$

where n is the time slot index.

Secondly, we can assume that only a fraction of users will be swayed by the rewards: the effectiveness of the incentive $\eta \in [0, 1]$ is defined as the maximum fraction of users who will choose to accept the incentive and walk to a close-by station instead of taking the bike from their preferred one. Hence, the larger η, the greater the impact of the incentives. Clearly, the demand should never become negative at any station, so additional constraints should be added:

$$\left|\frac{\lambda'_{v,n} - \lambda_{v,n}}{\lambda_{v,n}}\right| \leq \eta \; \forall v, n \tag{6.15}$$

$$\left|\frac{\mu'_{v,n} - \mu_{v,n}}{\mu_{v,n}}\right| \leq \eta \; \forall v, n \tag{6.16}$$

$$\lambda'_{v,n} \geq 0 \; \forall v, n \tag{6.17}$$

$$\mu'_{v,n} \geq 0 \; \forall v, n \tag{6.18}$$

Finally, the incentives will never convince users to walk very far, and they are not meant to do so: our objective is to improve the efficiency of the system with minimal disruption to users' habits and mobility, so we need to limit the distance we expect them to go. For this reason, we assume we can convince a user to walk to another station only if it is within a radius d_{thr} from the position of their preferred station. This corresponds to a set of conditions limiting the "shift" in the traffic to the neighborhood set $U_v = \{u \in \mathcal{V} : d_{u,v} \leq d_{\text{thr}}\}$. This can be translated to the following set of constraints:

$$\sum_{u \in U_v} \lambda'_{u,n} \geq \lambda_{v,n} \; \forall v, n \tag{6.19}$$

$$\sum_{u \in U_v} \lambda_{u,n} \geq \lambda'_{v,n} \; \forall v, n \tag{6.20}$$

$$\sum_{u \in U_v} \mu'_{u,n} \geq \mu_{v,n} \; \forall v, n \tag{6.21}$$

$$\sum_{u \in U_v} \mu_{u,n} \geq \mu'_{v,n} \; \forall v, n. \tag{6.22}$$

The four constraints above express a simple concept: none of the original demand at station v should be outside of the neighborhood U_v, i.e., the sum of the demand in the neighborhood after the incentives should be larger than the initial demand at station v (see (6.19) and (6.21)), and none of the

Algorithm 6.1 Incentive optimization algorithm

```
1: function SETINCENTIVES($\boldsymbol{\lambda}, \boldsymbol{\mu}, \eta, \boldsymbol{d}, d_{\text{thr}}, \varepsilon, \boldsymbol{m}, \boldsymbol{M}$)
2:     $\boldsymbol{\lambda}' = \boldsymbol{\lambda}$;                          ▷ Initialize arrival rate matrix
3:     $\boldsymbol{\mu}' = \boldsymbol{\mu}$;                          ▷ Initialize departure rate matrix
4:     for $v \in \mathcal{V}$ do
5:         $T'_v \leftarrow T(m_v, \boldsymbol{\lambda}'_{\boldsymbol{v}}, \boldsymbol{\mu}'_{\boldsymbol{v}})$;                    ▷ Expected downtime
6:     end for
7:     i = 1;                                  ▷ Initialize station counter
8:     while $i < V$ do                  ▷ Visit all stations (at least once)
9:         if $i == 1$ then
10:            $\boldsymbol{v} \leftarrow$ Sort by decreasing $\boldsymbol{T'}$;
11:        end if
12:        $U_{v_i} \leftarrow$ FINDNEIGHBORHOOD($v_i, \boldsymbol{d}, d_{\text{thr}}$);
13:        [$\boldsymbol{\lambda}', \boldsymbol{\mu}', \boldsymbol{T}'$, changed] $\leftarrow$ SHIFTTRAFFIC($v_i, U_{v_i}$,
               $\boldsymbol{\lambda}', \boldsymbol{\mu}', \boldsymbol{\lambda}, \boldsymbol{\mu}, \eta, \boldsymbol{d}, d_{\text{thr}}, \varepsilon, \boldsymbol{m}, \boldsymbol{M}$);
14:        if changed then
15:            $i \leftarrow 1$;                          ▷ Repeat the optimization
16:        else
17:            $i++$;                       ▷ Go on with the next station
18:        end if
19:    end while
20:    return $\boldsymbol{\lambda}', \boldsymbol{\mu}'$;
```

demand at station v after the incentives are applied should have come from outside the neighborhood (see (6.20) and (6.22)). If the four constraints are applied to all neighborhoods (one for each station v), they ensure that a node's traffic is never shifted outside its neighborhood, and that traffic from outside is never shifted to it.

Then, the objective function of the incentive problem is (6.12), and the complete set of constraints is given by (6.13)-(6.22).

If we want to minimize expected failures instead of downtime, the solution becomes:

$$\boldsymbol{\lambda}^*, \boldsymbol{\mu}^* = \arg\min_{\boldsymbol{\lambda}', \boldsymbol{\mu}'} \sum_{v \in \mathcal{V}} F(m_v, \boldsymbol{\lambda}'_{\boldsymbol{v}}, \boldsymbol{\mu}'_{\boldsymbol{v}}), \tag{6.23}$$

with the same set of constraints (6.13)-(6.22).

6.3.6 Solving the incentive problem

The method we use to solve the incentive problem is reported in Algorithm 6.1, and assumes that we can only affect the rate for the next frame (of duration T_r), without looking farther ahead. It takes as input the original arrival and departure rates matrices $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$, the incentive effectiveness η, a matrix $\boldsymbol{d}$ containing the distances between all the station pairs, the coverage radius d_{thr} that defines a neighborhood, the step ϵ used to compute the traffic shifts among neighbor stations, a vector $\boldsymbol{m}$ with the current state of all stations, and a vector $\boldsymbol{M}$ with the capacity of all stations. The algorithm returns the new arrival and departure rates matrices, $\boldsymbol{\lambda}'$ and $\boldsymbol{\mu}'$, after the incentives have been applied. First of all (Line 5), we compute the time T'_v that any station $v \in \mathcal{V}$ spends in the empty state; we then proceed with the incentive optimization. We sort the stations by decreasing order of expected downtime, obtaining a sorted array $\boldsymbol{v}$ (Line 10). Starting from the first station in $\boldsymbol{v}$, we apply function shiftTraffic (Line 13), which determines the optimal traffic demand within the neighborhood of the considered station. If any traffic change happened (i.e., the boolean variable 'changed' is set to 1), then the optimization is repeated from scratch (Line 15 in the *while* loop), because the change may influence the traffic rates of stations whose neighborhoods have already been optimized. Otherwise, if no shift happened, the station counter i is increased by one, so that function shiftTraffic is applied to the next station in vector v (Line 17). This is repeated until all stations have been visited and no traffic shifts happened (Line 8).

The shiftTraffic function is in Algorithm 6.2: it is called every time a neighborhood needs to be optimized, and its basic idea is equivalent to gradient descent. The function shifts small portions of traffic to or from the central node, checking every possible configuration and selecting the one that results in the lowest total expected downtime. The operation is then repeated until a local minimum is reached or a constraint is violated. It is based on an iterative algorithm, which is a variation on the gradient method [246]. The rationale behind our algorithm is to analyze possible traffic shifts in the neighborhood U_v of the target station v and identify the best possible shift,

Algorithm 6.2 Shift traffic function

```
function shiftTraffic($v, U_v, \boldsymbol{\lambda}', \boldsymbol{\mu}', \boldsymbol{\lambda}, \boldsymbol{\mu},$
    $\eta, \boldsymbol{d}, d_{\text{thr}}, \varepsilon, \boldsymbol{m}, \boldsymbol{M}$)
done, changed $\leftarrow$ false;
while !done do                                              ▷ Optimization cycle
    for $u \in U_v$ do                                      ▷ All the nodes in the neighborhood
        $T'_u \leftarrow T_e(\lambda'_u, \mu'_u, m_u, M_u)$;          ▷ Time in empty state
    end for
    $T^* \leftarrow$ sum($\boldsymbol{T}'$);                             ▷ Total time in empty state
    $\Delta\lambda^*, \Delta\mu^*, u^*, w^* \leftarrow 0$;
    for $\Delta\lambda \in \{-\varepsilon, 0, \epsilon\}$ do                    ▷ Possible variations on $\lambda$
        for $\Delta\mu \in \{-\varepsilon, 0, \epsilon\}$ do                ▷ Possible variations on $\mu$
            $\lambda''_v \leftarrow \lambda'_v + \Delta\lambda$;                     ▷ New arrival rate
            $\mu''_v \leftarrow \mu'_v + \Delta\mu$;                     ▷ New departure rate
            $\boldsymbol{T}'' \leftarrow \boldsymbol{T}'$;                         ▷ Time in empty state (vector)
            for $u \in U_v$ do                              ▷ All nodes in neighborhood
                $\lambda''_u \leftarrow \lambda'_u - \Delta\lambda$;                 ▷ Update neighbor arrivals
                for $w \in U_v$ do
                    $\mu''_w \leftarrow \mu'_w - \Delta\mu$;             ▷ Update neighbor dep.s
                    if checkConstraints($\boldsymbol{\lambda}'', \boldsymbol{\mu}'',$
                            $\boldsymbol{\lambda}, \boldsymbol{\mu}, \eta, \boldsymbol{d}, d_{\text{thr}}$) then
                        $T''_u \leftarrow T_e(\lambda''_u, \mu''_u, m_u, M_u)$;
                        $T''_v \leftarrow T_e(\lambda''_v, \mu''_v, m_v, M_u)$;
                        $T''_w \leftarrow T_e(\lambda''_w, \mu''_w, m_w, M_u)$;
                        if sum($\boldsymbol{T}''$)$< T^*$ then              ▷ Less failures
                            changed $\leftarrow$ true;              ▷ Update config.
                            $T^* \leftarrow$ sum($\boldsymbol{T}''$);
                            $\Delta\lambda^* \leftarrow \Delta\lambda$;
                            $\Delta\mu^* \leftarrow \Delta\mu$;
                            $u^* \leftarrow u$;
                            $w^* \leftarrow w$;
                        end if
                    end if
                end for
            end for
        end for
    end for
    if !changed then                                        ▷ Reached a minimum
        done = true;                                        ▷ Finished
```

```
38:     else
39:         $\lambda'_v = \lambda'_v + \Delta\lambda^*$;
40:         $\lambda'_{u^*} = \lambda'_{u^*} - \Delta\lambda^*$;
41:         $\mu'_v = \mu'_v + \Delta\mu^*$;
42:         $\mu'_{w^*} = \mu'_{w^*} - \Delta\mu^*$;
43:         $T''_{u^*} = T_e(\lambda'_{u^*}, \mu'_{u^*}, m_{u^*}, M_{u^*})$;
44:         $T''_v = T_e(\lambda'_v, \mu'_v, m_v, M_u)$;
45:         $T''_{w^*} = T_e(\lambda'_{w^*}, \mu'_{w^*}, m_{w^*}, M_{w^*})$;
46:     end if
47: end while
48: return $\boldsymbol{\lambda}', \boldsymbol{\mu}', \boldsymbol{T}'$, changed;
```

i.e., the change in the traffic pattern that leads to a reduced aggregate time in the empty state for the stations in U_v.

To this aim, we introduce ϵ, which is the step size for variations in the arrival and departures rates. Since the total aggregated traffic within the neighborhood should remain constant, any increase in the arrival (departure) state of station v corresponds to an equivalent decrease in the arrival (departure) state of a neighboring station. For every possible configuration analyzed by the algorithm, we check whether the constraints (6.13)-(6.22) are satisfied. Algorithm 6.3 includes the checkConstraints function, which is used to check the validity of problem solutions; its code corresponds to the constraints in (6.17)-(6.22).

We would like to remark that the iterative algorithm in function shiftTraffic is guaranteed to converge to a *local* minimum, so its output might not be the globally optimal solution. In general, T is not a convex function of λ and μ (see (6.1)), but the non-convexities are outside of the range of values we examine, and the function always has a unique global minimum if $\eta < 1$, so the local minimum corresponds to the global minimum. Implementations in other systems or with perfect incentives should consider this non-convexity.

Algorithm 6.3 Incentive optimization constraint check

```
1: function CHECKCONSTRAINTS($\boldsymbol{\lambda}', \boldsymbol{\mu}', \boldsymbol{\lambda}, \boldsymbol{\mu}, \eta, \boldsymbol{d}, d_{\text{thr}}$)
2:    for $v \in \mathcal{V}$ do
3:       $U_v$ =FINDNEIGHBORHOOD($v, \boldsymbol{d}, d_{\text{thr}}$);
4:       if $\lambda'_v < 0$ or $\mu'_v < 0$ then
5:          return false;                                ▷ Constraints (6.17), (6.18)
6:       end if
7:       if $|\lambda'_v - \lambda_v| > \eta\lambda_v$ or $|\mu'_v - \mu_v| > \eta\mu_v$ then
8:          return false;                                ▷ Constraints (6.15), (6.16)
9:       end if
10:      $l, m, l', m' \leftarrow 0$;                    ▷ Initialize counters
11:      for $u \in U_v$ do                              ▷ All nodes in the neighborhood
12:         $l \leftarrow l + \lambda_u$;                ▷ Update total old arrival rates
13:         $l' \leftarrow l' + \lambda'_u$;             ▷ Update total new arrival rates
14:         $m \leftarrow m + \mu_u$;                    ▷ Update total old departure rates
15:         $m' \leftarrow m' + \mu'_u$;                 ▷ Update total new departure rates
16:      end for
17:      if $\lambda'_v > l$ or $\lambda_v > l'$ then
18:         return false;                                ▷ Constraints (6.19), (6.20)
19:      end if
20:      if $\mu'_v > m$ or $\mu_v > m'$ then
21:         return false;                                ▷ Constraints (6.21), (6.22)
22:      end if
23:   end for
24:   return true;                                       ▷ All constraints satisfied
```

6.4 Dynamic rebalancing

Our network optimization aims at minimizing the system downtime, computed as described in Section 6.3.2. Part of this optimization is achieved by giving incentives to users, as modeled in Section 6.3.4, but may not be in itself sufficient to reach the target network configuration, i.e., the one that minimizes either the downtime or the expected number of failures. Dynamic rebalancing consists in moving bikes across stations using a fleet of trucks, changing the current network state in order to minimize the system failures. This is performed after the incentives have been applied, and requires to decide *when* and *how* to do the rebalancing. In fact, using a truck fleet has a monetary cost (fuel, drivers, etc.), so that the gain obtained with it has to overcome such cost. It is also necessary to identify the stations to rebalance, and the corresponding path to visit them. Clearly, the when and how aspects are intertwined.

In the following, we define the mathematical problem and solve it for two distinct scenarios: we in fact consider both the benchmark case of a single vehicle and the more realistic case of multiple vehicles with a limited capacity.

6.4.1 Preliminaries

Dynamic rebalancing schemes do not change the system at preset times every day, but rather rebalance the network whenever the expected system-wide downtime is larger than $p_{\text{reb}}NT_p$, i.e., when the system is expected to be unavailable for a fraction p_{reb} of the considered time horizon NT_p:

$$\sum_{v\in\mathcal{V}} T(m_v, \boldsymbol{\lambda'_v}, \boldsymbol{\mu'_v}) > \frac{p_{\text{reb}}}{NT_p}, \tag{6.24}$$

where we explicitly wrote the dependence of the expected downtime T on the arrival and departure rates $\boldsymbol{\lambda'_v}$ and $\boldsymbol{\mu'_v}$ to highlight that they are those modified by the incentives.

The rebalancing fleet consists of R trucks, which all start from a deposit node z, so that we can define a new graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$, where $\mathcal{V}' = \mathcal{V} \cup \{z\}$ and $\mathcal{E}' = \mathcal{V}' \times \mathcal{V}'$. We assume each truck to have a capacity C limiting the number of bikes it can load; since stations should not be visited twice, we cannot change their state by more than C bikes. The rebalancing should shift the network from its current state to the unique optimal one, which, as described in Section 6.3.2, is the one in which every station has the minimal expected downtime. Let Δm_v^* be the difference between the optimal and the current state of station v:

$$\Delta m_v^* = \underset{\delta\in\{(m_v-C)^\dagger,(m_v-C+1)^\dagger\ldots,(m_v+C)^\dagger\}}{\arg\min} T(m_v+\delta) \tag{6.25}$$

where $(x)^\dagger \triangleq \max(0, \min(M_v, x))$ is used to ensure that, after the truck takes or deposits bikes, the number of bikes at station v neither is negative nor exceeds the station capacity M_v. The difference in the state is also limited by the capacity of the trucks: if the current state is more than C bikes away from the optimum, the optimal state is the one closest to it. The difference Δm_v^* yields the best achievable state.

6.4.2 System-wide rebalancing problem

We can now define the system-wide rebalancing problem. The objective is to minimize the total distance that the trucks have to cover:

$$\boldsymbol{e}^\star = \arg\min_{\boldsymbol{e}} \sum_{u \in \mathcal{V}'} \sum_{v \in \mathcal{V}'} e_{u,v} d_{u,v}, \tag{6.26}$$

where $\boldsymbol{e}^\star$ is a matrix over $\mathcal{E}'$ whose values are either 0 or 1:

$$e_{u,v} \in \{0, 1\} \; \forall (u, v) \in \mathcal{E}'. \tag{6.27}$$

If $e_{u,v} = 1$, then the edge from station u to v is covered by the rebalancing trucks, otherwise it is not. Note that there might be multiple optimal solutions $\boldsymbol{e}^*$, and if the graph distances are not directed (i.e., $d_{u,v} \equiv d_{v,u} \, \forall u, v \in \mathcal{V}'$) there are always at least two optimal solutions, which correspond to following the same route in opposite directions.

Since the solution $\boldsymbol{e}^\star$ represents a path, or a series of paths, each outgoing edge should be balanced by an incoming edge; in this way, the trucks all start and end at the same point, i.e., the deposit node z. This corresponds to the following constraints:

$$\sum_{u \in \mathcal{V}'} e_{u,v} = \sum_{u \in \mathcal{V}'} e_{v,u} \; \forall v \in \mathcal{V}' \tag{6.28}$$

$$e_{v,v} = 0 \; \forall v \in \mathcal{V}'. \tag{6.29}$$

The constraint in (6.29) prohibits edges from a node to itself to be part of the solution, for obvious reasons. Furthermore, each node (except for the deposit) should be visited at most once by a rebalancing truck, in order to avoid loops and inefficient routes:

$$\sum_{u \in \mathcal{V}'} e_{u,v} \leq 1 \ \forall v \in \mathcal{V}. \tag{6.30}$$

As regards the deposit node, its outgoing degree is equal to the number of used trucks R. The route of each truck corresponds to a series of successive connected nodes. For each truck $i \in \{1, \dots, R\}$, we define the successor $s_v(i)$ of node v as the next node visited by truck i, so that $e_{v,s_v(i)} = 1$. Note that there exists at most a unique $s_v(i)$ for each truck i and node $v \in \mathcal{V}'$. Naturally, nodes that are not successors of any other do not have any successors, as a consequence of (6.28). We define the route r_i of the i-th truck as a vector of length $l(r_i)$:

$$r_i(0) = z \tag{6.31}$$
$$r_i(1) = s_z(i) \tag{6.32}$$
$$r_i(j) = s_{r_i}(j-1) \quad \forall\, j > 1 \tag{6.33}$$
$$r_i(l(r_i)) = z. \tag{6.34}$$

The last node in the route is the deposit station itself; (6.34) makes it a closed loop. No station that is not part of a route should be visited, i.e., all stations in the solution should be connected to the starting point z. We express this constraint as:

$$\sum_{u \in \mathcal{V}'} e_{u,v} = 0 \quad \forall\, v \in \mathcal{V} : v \notin \bigcup_{i=1}^{R} r_i. \tag{6.35}$$

Since the number of bikes on a truck cannot exceed its capacity at any point in its route, we formulate the load of truck i after visiting the j-th node in its route as:

$$c_{i,j} = c_{i,0} + \sum_{k=1}^{j} \Delta m^*_{r_i(k)} \tag{6.36}$$

where $c_{i,0}$ is the initial load as the truck leaves the deposit. We can formulate an additional constraint to ensure that no trucks have an overload or a negative load at any point in their route, following the work in [227]:

$$0 \leq c_{i,j} \leq C \quad \forall i, \forall j \in \{0, \ldots, l(r_i)\}. \tag{6.37}$$

In order to adhere to a more realistic model, we can also limit the maximum distance a single truck can cover before the rebalancing operation is too late:

$$\sum_{j \in \{1,\ldots,l(r_i)\}} d_{r_i(j-1),r_i(j)} \leq d_{\max} \; \forall i \in \{1, \ldots, R\}. \tag{6.38}$$

Finally, we set a constraint to only consider solutions that reduce the expected downtime to a fraction p_{thr} of the total running time of the system:

$$\sum_{v \in \mathcal{V}} T \left(m_v(t) + \sum_{u \in \mathcal{V}'} e_{u,v} \Delta m_v^* \right) \leq p_{\text{thr}}. \tag{6.39}$$

where Δm_v^* represents the difference between the optimal and the current state of station v, as defined in (6.25).

The rebalancing optimization problem is then defined by the objective function in (6.26). The constraints are given by (6.27)-(6.30), (6.35), and (6.37)-(6.39). In the following, we solve it for two distinct scenarios: the simpler but less realistic case of a single truck ($R = 1$), and the complicated but more general case of multiple vehicles ($R > 1$).

6.4.3 Single-vehicle optimization

The first version of the problem we consider is simpler and slightly less realistic, and corresponds to the scenario in our previous work [211]: in this case, $R = 1$, $C = \infty$, $d_{\max} = \infty$. Since there is a single truck, Constraint (6.30) is also applied to the deposit node z. We use Algorithm 6.4 to solve this problem.

Algorithm 6.4 Single-vehicle rebalancing algorithm

```
1: function REBALANCE($\boldsymbol{T}, \boldsymbol{m}, \boldsymbol{\Delta m}^*, \boldsymbol{T}^*, \boldsymbol{d}, p_{\text{reb}}, p_{\text{thr}}$)
2:    $\boldsymbol{S} \leftarrow [z]$;                          ▷ Start with deposit node
3:    $T \leftarrow \text{sum}(\boldsymbol{T})$;                      ▷ Total downtime time
4:    $l \leftarrow 0$;                            ▷ Total distance covered by the truck
5:    $\boldsymbol{m}' \leftarrow \boldsymbol{m}$;                          ▷ Initialize new network state
6:    if $T \leq p_{\text{reb}}$, then                   ▷ Check Constraint (6.24)
7:       return $l, \boldsymbol{m}$;                       ▷ Do not rebalance
8:    end if
9:    for $u \in \mathcal{V}$ do
10:      for $v \in \mathcal{V}$ do
11:         $e_{u,v} \leftarrow 0$;                   ▷ Initialize path matrix
12:      end for
13:   end for
14:   while $|\boldsymbol{S}| < |\boldsymbol{T}|$ and $T > p_{\text{thr}}$ do
15:      $v, r_v, d_v \leftarrow 0$;
16:      for $u \in (\mathcal{V} \setminus \boldsymbol{S})$ do                ▷ Find node to rebalance
17:         $l_u \leftarrow$GREEDYROUTE$(\boldsymbol{S}, \boldsymbol{d}) - l$;
18:         $r_u \leftarrow \frac{T_u - T_u^*}{d_u}$;               ▷ Gain of rebalancing $u$
19:         if $r_u > r_v$ then          ▷ More gain than previous option
20:            $v \leftarrow u$;                ▷ Update node to rebalance
21:            $r_v \leftarrow r_u$;            ▷ Update rebalancing reward
22:            $l_v \leftarrow l_u$;                  ▷ Update path length
23:         end if
24:      end for
25:      if $r_v > 0$ then                   ▷ Reward is not a cost
26:         $S \leftarrow [S; v]$;           ▷ Add node $v$ to rebalancing set
27:         $m'_v \leftarrow m_v + \Delta m^*_v$;                 ▷ Update state of $v$
28:         $T_v \leftarrow T^*_v$;               ▷ Update downtime of $v$
29:         $T \leftarrow \text{sum}(\boldsymbol{T})$;             ▷ Update total downtime
30:         $l \leftarrow l + l_v$;          ▷ Update total covered distance
31:         $e_{v_{|S|-1},v} \leftarrow 1$;               ▷ Update path matrix
32:      else
33:         break;
34:      end if
35:   end while
36:   return $\boldsymbol{e}, l, \boldsymbol{m'}$;
```

Algorithm 6.5 Single-vehicle greedy routing algorithm

```
1: function GREEDYROUTE($\boldsymbol{S}, \boldsymbol{d}$)
2:     $R \leftarrow [z]$;                                   ▷ Start with deposit node
3:     $r \leftarrow z$;                                     ▷ Last node visited
4:     $l \leftarrow 0$;                                     ▷ Length of rebalancing path
5:     while $|R| < |S|$ do
6:         $v^* \leftarrow \arg\min_{v \in (S \setminus R)} d_{v,z} + d_{r,v} - d_{r,z}$;
7:         $R \leftarrow [R; v^*]$;                          ▷ Add node $v^*$ to route
8:         $l \leftarrow l + d_{v^*,z} + d_{r,v^*} - d_{r,z}$;
9:         $r \leftarrow v^*$;
10:    end while
11:    return $l$
```

It takes as input the vector $\boldsymbol{T}$ with the downtimes of all stations, the current network state $\boldsymbol{m}$, the vector $\boldsymbol{\Delta m^\star}$ containing the desired state shift for each possible station (see (6.25)), a vector $\boldsymbol{T}^\star$ with the target downtimes for all stations, the distance matrix $\boldsymbol{d}$, the threshold p_{reb} that triggers the rebalancing (see (6.24)), and the minimum fraction of downtime reduction p_{thr} (see (6.39)). It computes the path matrix $\boldsymbol{e}$ defined in (6.26), the length l of the rebalancing path, and the new system state $\boldsymbol{m'}$.

The algorithm works by iteratively updating the set $\boldsymbol{S}$ of stations to rebalance until either all stations have been visited or the total downtime drops below the threshold p_{thr} (Line 14). Each iteration determines the station u (among the unvisited ones) to add to S in a greedy way. Such station is the one which yields the largest reward among all unvisited stations (Lines 18 and 19 in the *for* cycle). The station, however, is added to S only if it brings a positive reward (Lines 25-31); otherwise, the algorithm stops.

An auxiliary function GREEDYROUTE is used (Line 17 calls Algorithm 6.5) to compute the length of the rebalancing path when a station is added. It determines a path that starts from the deposit node z (Line 2) and visits all stations in S (Line 5) selecting, at each iteration, the next station as the closest one to the last node visited (Line 6).

For a more detailed description of the algorithm, we refer the reader to our previous work [211]. We highlight that, although this scenario is not realistic, it has a very low computational complexity when compared to the multi-vehicle optimization, and represents a useful benchmark for performance comparison.

6.4.4 Multi-vehicle optimization

In this case, we consider the full problem as defined in Section 6.4.2, and use a variation on Dell'Amico *et al.*'s *Destroy and Repair* procedure [227]. If we constrain the possible solutions to those including all nodes in the graph, the model exactly matches the one defined in [227]; however, the metaheuristic they define is flexible enough to be used even if not all nodes are included. We now recap the steps the algorithm follows, with a discussion of the differences we implemented. A more complete discussion of each step can be found in the original paper proposing it.

1. Nodes are added to the solution iteratively using Clarke and Wright's *Savings* algorithm [247]. Basically, each node is added as a new separate route, so the edges $e_{z,v}$ and $e_{v,z}$ are added to the solution. In our case, instead of stopping after all nodes are included in the solution, the algorithm stops when the relative downtime drops below p_{thr}. After that, the *Savings* algorithm tries to merge routes, evaluating both the gain in terms of total covered distance and the loss of flexibility due to the merging. Flexibility is a measure of the slack in a route, which becomes less flexible when the truck is almost empty or almost full at any point.

2. A local search is performed using several heuristics in succession: nodes are swapped, added or removed to routes, trying to find a solution with a lower cost. Our local search includes seven heuristics, performed in sequence as in [227]. The seven heuristics basically try to explore the solution space by swapping nodes, joining routes, or crossing them over so that the first and second parts of two routes are mixed. The idea behind the local search is to explore possible solutions that are close to the current one.

3. A *Random removal* procedure is performed, as described by Ropke and Pisinger [248], taking some of the nodes out of the solution. This partly randomizes the solution, allowing the algorithm to explore other points in the solution space and avoiding getting stuck in local minima.

4. Dell'Amico *et al.*'s *Repair* procedure is performed, adding random nodes until p_{thr} is crossed again. Each node is added to the solution by iteratively trying to place it in all possible positions in all routes, then selecting the best possible one. If the solution is worse than the one before the removal of nodes and repair operation, the older one is kept in memory as the best solution.

5. Steps 2 to 4 are repeated until convergence.

If no acceptable solution exists (i.e., rebalancing all nodes still results in a downtime higher than p_{thr}), we still rebalance the system in order to minimize the expected downtime: this corresponds to a rebalancing operation involving all nodes, which can be calculated with Dell'Amico *et al.*'s unmodified *Destroy and Repair* algorithm.

Notice that the solutions proposed for both for the single- and the multi-vehicle rebalancing problems can be straightforwardly applied to the number of expected failures F as defined in (6.9) rather than to the downtime T.

6.4.5 Simulation settings and analysis

In this work, we used the publicly available dataset from New York City's CitiBike network, which we briefly described in Section 6.1. A map of the city, with the positions of the docking stations, is shown in Figure 6.3.

The service publishes monthly reports with all the recorded rides, the start and destination times and stations, the unique identifiers of the bikes, and some records about the service subscribers, such as year of birth and gender. In this study, we use the data from July 2013 (the earliest available period) to July 2017. Since several stations were added during the considered time frame, we limit ourselves to the 280 stations that were present for a sufficiently long time to extract the demand data.

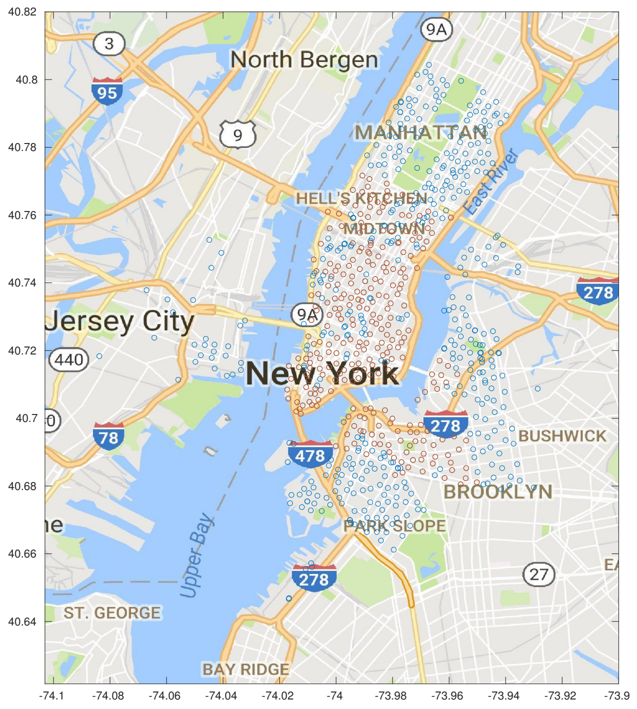


Fig. 6.3: Map of the bike-sharing stations: red points identify the stations considered in our study.

The traffic pattern intuitively depends on several factors, such as the time of the year (fewer people tend to use bikes in winter than in summer), whether it is a weekday or weekend (people may tend to use bikes at more regular times on weekdays and for shorter periods), and the position of the considered station (and the facilities nearby), as shown in Figure 6.4, which reports the traffic pattern for a station close to CUNY Baruch College during the month of July 2015.

On weekdays, the rush hours around 8 a.m. and 5 p.m. can be identified by the spikes in the demand for bikes; since the station is probably used by many students and workers, there are more arrivals than departures in the morning and more departures than arrivals in the evening. On Fridays, there is a more homogeneous pattern in the afternoon, likely because many people leave work early, and the demand is much lower during the weekend. These patterns highlight that both the day of the week and the hour need to be considered when calculating the arrival and departure rates.

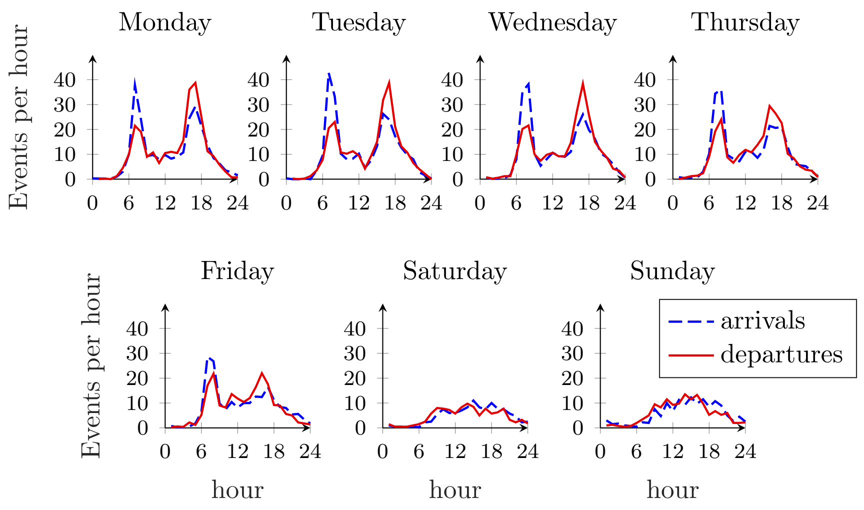


Fig. 6.4: Traffic patterns for the month of July 2015 at station 537 (Lexington Ave. and East 24th St.)

Another important factor which is more difficult to predict is the weather, as rain and snow may significantly affect the use of the bike sharing system: predictions of future bike traffic can become more accurate by including weather patterns [249]. Table 6.1 lists the effect of temperature, rain, and snow on the daily bike traffic citywide. The cross-correlation between temperature and traffic is particularly strong, as users tend to bike more during the warmer months. Rain does not affect the system too heavily, as the demand only drops by 10% on rainy days; snow has a stronger deterrent effect on bikers, as it makes it more difficult to ride.

Weather pattern	R^2	Demand drop
Temperature	0.45	–
Rainfall	0.06	10%
Snowfall*	0.13	49%
Snow on the ground*	0.10	36%

* In the case of snow and snowfall, the drop in the demand is calculated only on the winter months (from December to March)

Table 6.1: Effects of weather variables on the daily bike demand

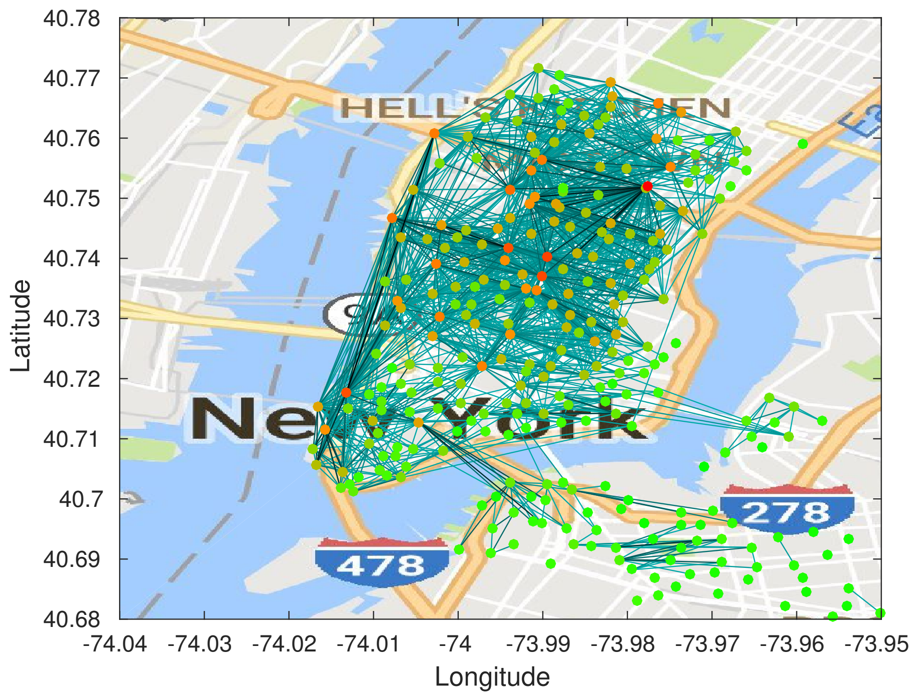


Fig. 6.5: Map of the demand for the month of May 2017 in the Lower Manhattan and Brooklyn area, color-coded from green (low demand) to red (high demand). The most common routes are plotted in teal, with a darker shade representing a more frequently taken route.

In order to account for the combined effect of snow and low temperature, we only considered the winter months (from December to March), and the demand still dropped by almost half on snowy days and by more than a third when there was snow on the ground. However, this aspect is difficult to predict and its integration in an analytical model is challenging.

This fits with the simplifying assumptions in Section 6.3. We assume that the Poisson rates $\lambda_v(\cdot)$ and $\mu_v(\cdot)$ for the arrival and departure process, respectively, are piecewise constant at steps of duration T_r equal to one hour. Hence, we can extract the value of the arrival and departure rates for each period Ω, defined as the hour, day, and month of the year, for all the stations in the network.

There are some other issues that need to be mentioned. Stations that are empty or full prevent customers from taking or returning bikes, so that the observed trip data may actually differ from the true demand and rather represent a lower bound for it, causing a censoring problem. In addition, the bike demand may increase thanks to the rebalancing operations, as people would see that the system is becoming more efficient. Unfortunately, we have no way to gauge the real demand from the available data, and the effect of the current rebalancing operations prevents the extrapolation of the real state of the system at any given time. However, the framework we propose is general and can be easily adapted to different datasets. Thus, although the numerical evaluation could be affected by the censoring issue, the model and the proposed solution have general validity.

To obtain the data for the performance evaluation, we first fitted the real data from the CitiBike dataset to our model (see Section 6.3), obtaining the parameters for the BDP that models each station. The frame duration that dictates the changes in the birth and death rates was chosen as $T_r = 1$ hr, while the slot duration was taken as $T_p = 15$ min. The demand map for the month of May 2017 is shown in Figure 6.5. While in our previous work [211,214] we directly used the real data to gauge our model, here we need to take into account that incentives affect the arrival and departure rates, which should be consistent with the application of the incentives. Since real data does not satisfy this requirement, we generated artificial traces based

Parameter	Value
η	{0,0.05,0.1,0.2}
d_{thr}	250 m
ε	0.01
p_{reb}	0.2
p_{thr}	0.05
d_{max}	40 km
R	{25,50}
T_r	1 h
T_p	15 min

Table 6.2: Incentive and rebalancing algorithm parameters

on the model derived from the real dataset. Note, however, that the model resembles the real data with extreme accuracy.

In general, the rebalancing strategies we consider are still not entirely realistic; in particular, the rebalancing operation is assumed to be instantaneous, while a real route covering multiple stations and a distance of several kilometers would take most of an hour. However, this assumption is extremely common in the literature, and we plan to remove it in future works.

6.5 Results

In our simulations, we used the traffic patterns from the first two weeks of May 2017 to simulate traffic across the whole system as described above. The parameters of the simulation are listed in Table 6.2; we considered both the idealized version of the rebalancing algorithm, which we will call *Single* from here on, and Dell'Amico *et al.*'s *Destroy and Repair* algorithm, or *D&R* for short. We also used two different settings for the latter algorithm, comparing its performance with trucks whose capacity is 25 and 50 bikes. All the versions of the algorithm were tested using both a *Static* and a *Dynamic* strategy; in the static case, rebalancing is performed on all stations that are not in the optimal state twice a day, at 3 a.m. and 3 p.m.. We considered every possible combination of the rebalancing and incentive algorithm settings for a total of 28 possible configurations (including the baseline options with no incentives or no rebalancing).

6.5.1 Performance

In this section, we consider two measures of a bike sharing optimization algorithm's performance: namely, the fraction of time a station spends either empty or full and the number of resulting failed trips. Although empty and full stations and the resulting failures might have different effects, we do not show them separately, as in all scenarios the fraction of downtime and failures due to empty stations was close to 0.6. For the same reason, we only show the performance of the Static and Dynamic strategies without showing each

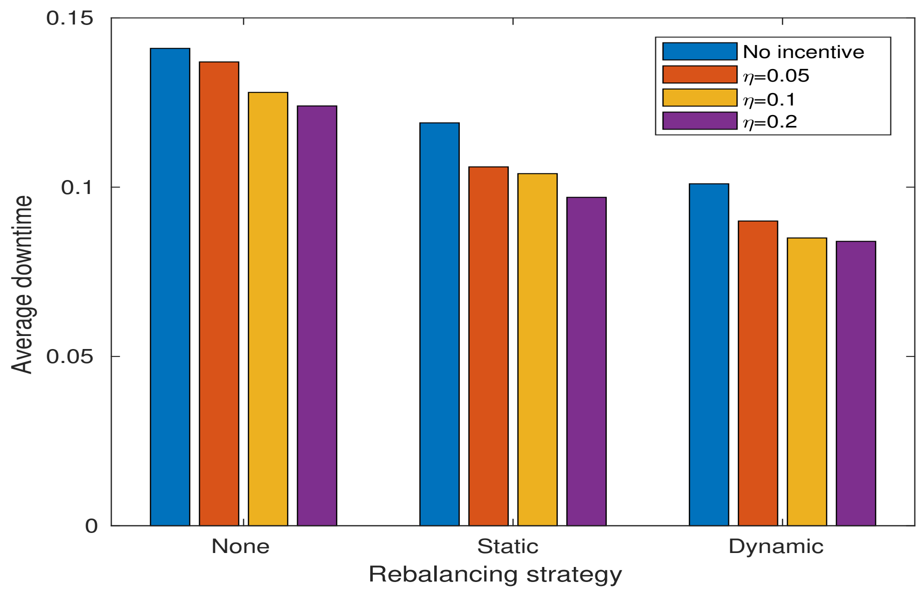


Fig. 6.6: Bar plot showing the fraction of downtime for the various rebalancing and incentive schemes

algorithm, as the performances were extremely similar for all versions.

Figure 6.6 shows the average fraction of downtime for the different configurations: the dynamic strategy clearly has an edge over the static one, although naturally both perform better than leaving the system without any rebalancing. Interestingly, the strongest incentives can effectively substitute the static strategy without incentives, and they allow the static strategy to outperform the dynamic one without incentives. Another important trend is the diminishing effect of incentives: with both rebalancing strategies, going from no incentives to $\eta = 0.05$ results in a larger performance improvement than going from $\eta = 0.05$ to $\eta = 0.2$.

Figure 6.7 shows the average fraction of failed trips for the different configurations; in this case, the effectiveness of incentives is even clearer, as using them can provide a significant performance boost to both rebalancing strategies. However, using incentives without rebalancing is not very effective, since the system will often be in "dangerous" states, with multiple stations that are either almost full or almost empty, and the effect of slightly chang-

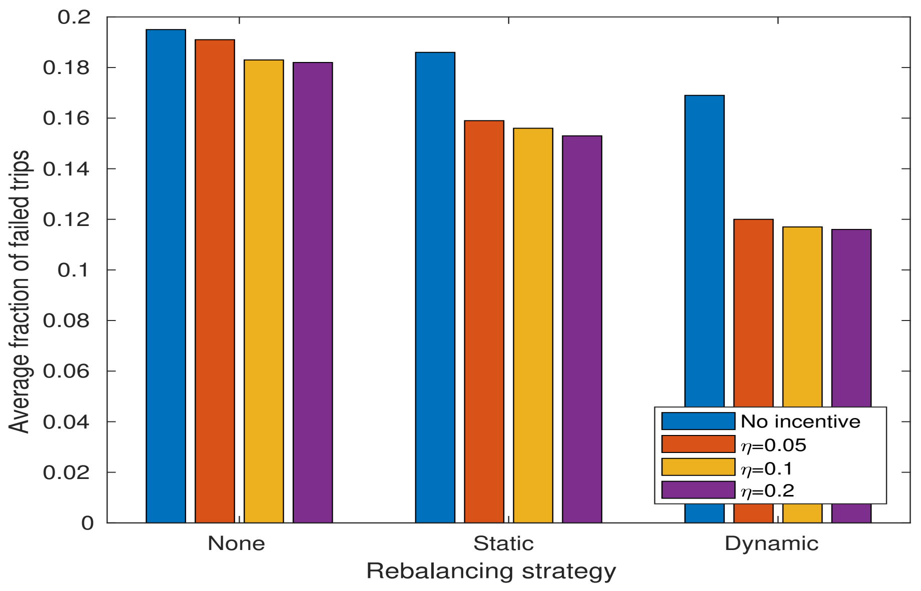


Fig. 6.7: Bar plot showing the fraction of failed trips for the various rebalancing and incentive schemes

ing the traffic pattern is not strong enough to compensate. The diminishing returns of incentives are also even stronger here, since increasing η past 0.05 only brings a negligible performance improvement.

6.5.2 Rebalancing effort

Naturally, the dynamic rebalancing strategy has its costs: as Figure 6.8 shows, the dynamic strategy always results in more rebalancing actions than the static one. However, the incentives also have a strong positive effect here: aside from improving performance, they also significantly reduce the number of rebalancing actions by making the system more stable and keeping it below the rebalancing threshold.

This effect is also evident in the reduction of the total distance that the rebalancing trucks cover every day, shown in Figure 6.9: although they have almost no effect on the static system, incentives significantly reduce the distance for the dynamic strategy, although with sharply diminishing returns. This figure also shows the price of using a realistic scheme such

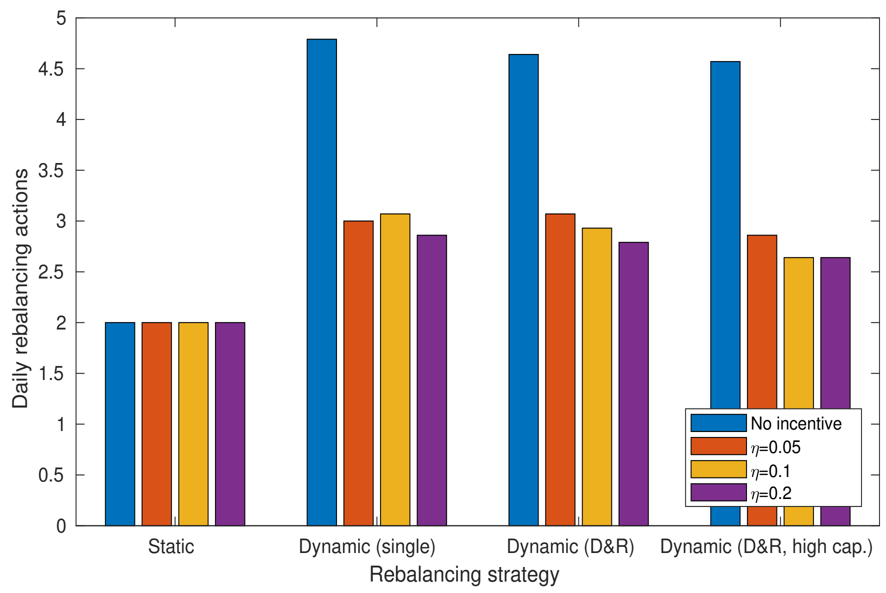


Fig. 6.8: Average number of rebalancing actions every day

as D&R, with trucks with finite capacity covering routes of limited lengths: the distance the trucks have to cover to rebalance the system increases by an order of magnitude with respect to the idealized Single algorithm. It is interesting to note that increasing the capacity R of the rebalancing trucks has a negligible effect on the total distance for the Static D&R scheme, while it significantly reduces it for Dynamic D&R. This is probably due to the fact that the dynamic scheme deploys the truck at high-congestion times, when the system is badly out of balance and some stations need a lot of bikes, straining the capacity of trucks.

Figure 6.10 confirms this analysis, as it shows that the number of deployed trucks per rebalancing action slightly decreases for the Static D&R scheme when capacity is doubled, while the effect on the dynamic scheme is more significant; interestingly, the incentives seem to have almost no effect on this.

6.5.3 Cost analysis

As mentioned above, the best estimate we have for the cost of rebalancing operation is 1.50$ per user ride; after multiplying the cost by the number

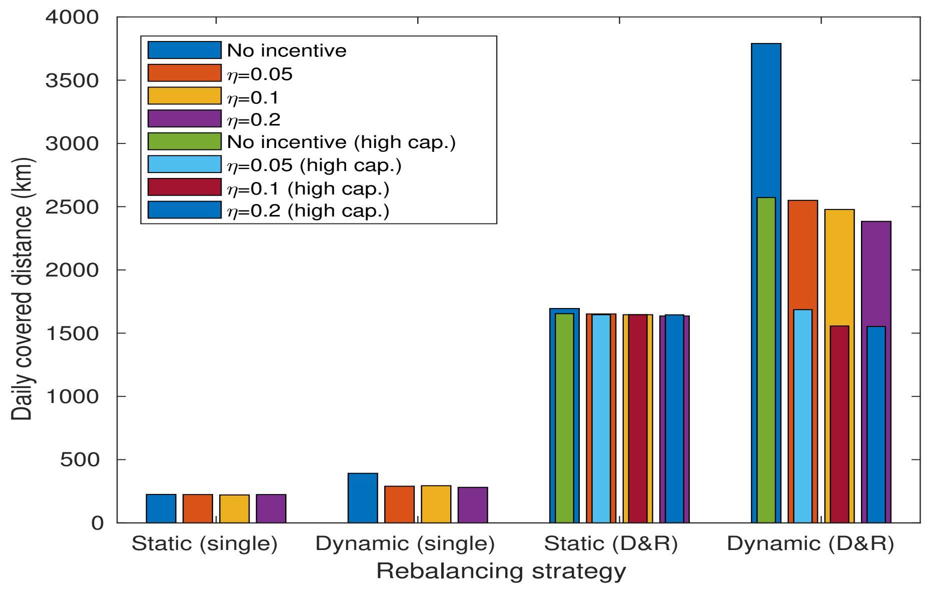


Fig. 6.9: Average distance covered daily by the rebalancing trucks

of daily rides, we get an average daily cost of 75000\$ for static rebalancing. We evaluate the cost of rebalancing the system dynamically by comparing the total number of deployed trucks. The daily cost of static rebalancing slightly decreases to 68800\$ if the capacity of the trucks is increased to 50 bikes, but the cost of dynamic rebalancing is significantly higher, since the improved performance of the system does not come for free: dynamic rebalancing would cost 170800\$ a day. Larger trucks would reduce the cost of dynamic rebalancing significantly, as in that scenario the rebalancing costs decrease to 105000\$ per day.

The cost analysis for the incentive scheme is based on the value of the prizes awarded by the CitiBike Angels program[7]: the rewards cost the company between 0.05 and 50 cents per incentivized ride, so we consider a 0.30\$ average cost, using a conservative estimate of users' prize choices. The average daily incentive cost is just 450\$ if $\eta = 0.05$, 1500\$ if $\eta = 0.1$, and 2300\$ if $\eta = 0.2$. Since the cost is two orders of magnitude smaller than rebalancing, even raising incentives significantly would still be significantly more

[7]https://www.citibikenyc.com/bikeangels/rewards

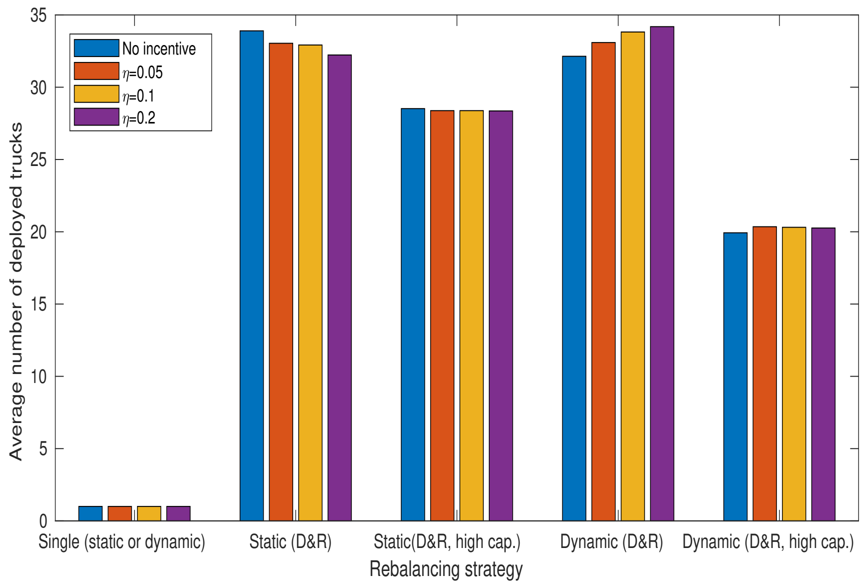


Fig. 6.10: Average number of deployed trucks in a rebalancing action

cost-effective. We remind the reader that, unlike the Angels program, the incentives we model do not need users to take trips that go in the opposite direction of the main flow of traffic during rush hour (targeting the so-called reverse commuters), but simply encourage users to walk a short distance to a less congested station instead of riding, and they could be extremely enticing to users even without the promise of higher rewards.

The combination between rebalancing and incentives gives interesting results: Figures 6.11 and 6.12 show the effectiveness of the optimization as a function of its daily operating cost. While using incentives always increases the effectiveness of the optimization, it has no effect on the cost of the schemes that entail no rebalancing or a static rebalancing strategy. However, their effect on the costs of the dynamic strategy is striking: by significantly reducing the number of rebalancing actions the system needs to perform, the cost of the optimization is reduced by more than a third.

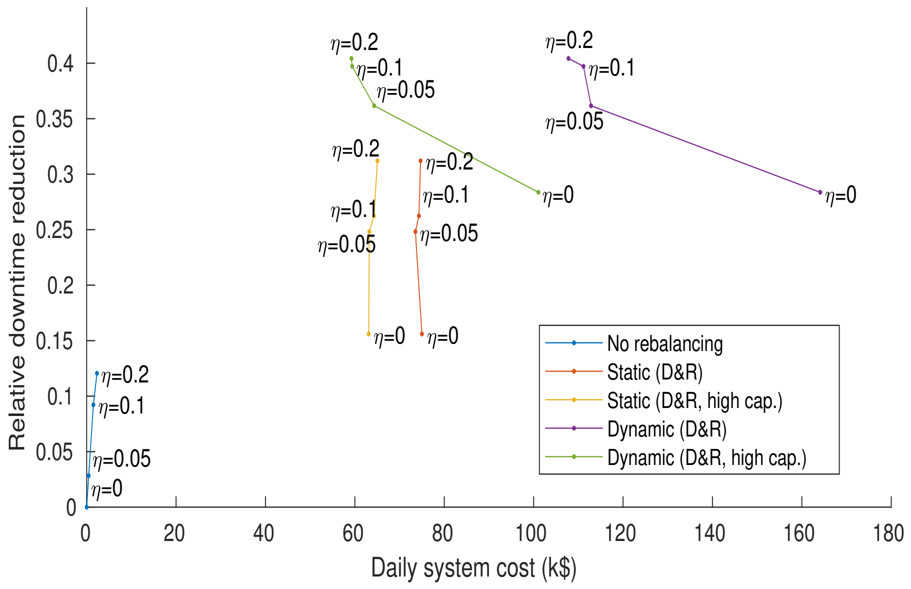


Fig. 6.11: Cost effectiveness of different configurations in terms of downtime reduction

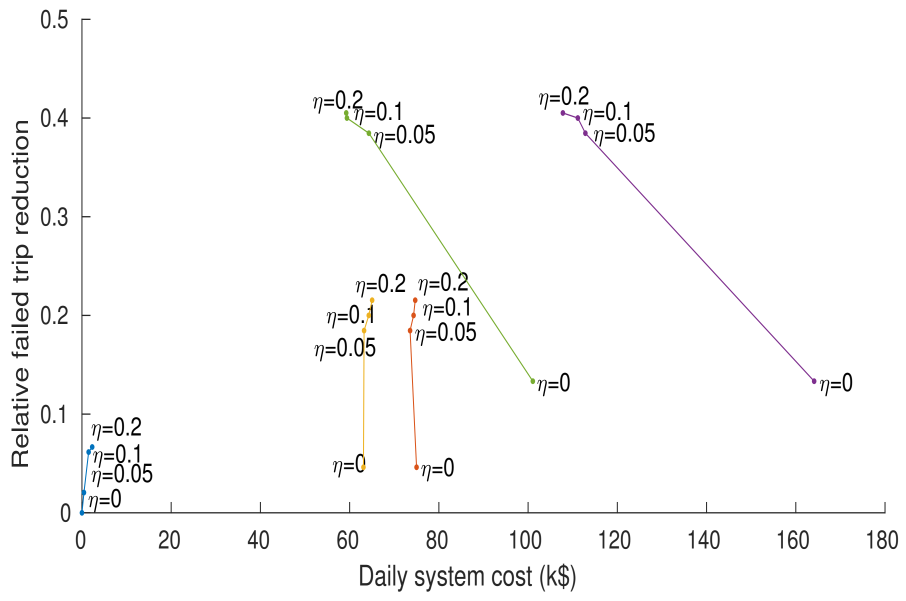


Fig. 6.12: Cost effectiveness of different configurations in terms of failed trip reduction

Chapter 7

Exploiting Smart City data to optimize the network

As we described in Chapter 6, Smart City services can be significantly improved by applying machine learning and data-driven techniques; however, the data that Smart City sensors provide are also a valuable source of information for network optimization.

If Smart City data were used to expand the awareness of the network, the relationship between the network and the Smart City sensors would truly become symbiotic: the sensors would rely on the 5G network to relay their information to the management servers and be the backbone of the city services, while the network itself would be able to process a considerable wealth of information from the Smart City and adapt itself more intelligently to its environment. In this chapter, we describe the SymbioCity concept we first advocated for in [250], presenting two case studies that exploit the traffic data from the Transport for London (TfL) Urban Traffic Control (UTC) network [251] in order to dynamically optimize network parameters such as (i) the handover range expansion bias for Heterogeneous Networks (HetNets) and (ii) the number of virtualized Mobility Management Entities (MMEs) deployed city-wide.

Since handovers will be one of the major issues in 5G ultra-dense networks, the techniques we propose will reduce the handover completion time and the well-known ping-pong effect [252, 253] without losing the benefits of micro cell offloading. The ability to choose the point in the trade-off between handover frequency and offloading capability is going to be a key element in the design of self-organizing 5G networks.[1]

7.1 Smart Cities and networks

In order for the Smart City to function, data need to flow from the sensors to the central controller [202] and from the controller to the actuators and smartphone apps that provide the services to citizens [256]. This communication largely relies on existing communication infrastructures, which have been deployed for other uses and will have to make space for these new services and applications [70]. As of today, the wireless access to sensor nodes scattered over the city is mostly provided by cellular systems, which were originally deployed to offer traditional human-based voice and data services. This puts a significant strains on these networks [257], and efforts to separate the services, either virtually by using network slicing or multi-tenancy [258] or physically by deploying dedicated technologies, are under way.

The access technologies that are currently used in the Smart City context can be roughly divided into the following three main families:

- *Cellular systems.* The almost ubiquitous service coverage and the commercial and technological maturity of cellular systems make them a natural solution to provide connectivity to IoT end-devices. Indeed, many telecom operators include bundles for M2M data traffic in their commercial offer, aiming to corner the market on this new type of services. However, current cellular network technologies have been designed for traditional human-initiated wideband services, which are significantly different in terms of traffic demands from Smart City services, and as

[1]The work in this section is the combination of several published papers: the concept of a SymbioCity was published in the Transactions on Emerging Telecommunications Technologies [250], while the first case study was presented at the IEEE International Symposium on Wireless Communication Systems (ISWCS) 2017 and published in the conference proceedings [254] and both case studies were published together in the IEEE Internet of Things Journal [255].

the volume of M2M traffic grows the issues caused by this are becoming evident [11].

- *Short-range multi-hop technologies.* Short-range technologies have almost complementary characteristics with respect to cellular technologies: they operate in the unlicensed 2.4 GHz frequency band and are explicitly designed for communications with nearby devices, offering a very limited coverage area (a few meters). Examples of standards in this category are IEEE 802.15.4 [259], Bluetooth Low Energy (BLE) [260], and Z-Wave [261]. Most short-range technologies actually support multi-hop packet delivery, but the management of these so-called mesh networks can be tricky and appears particularly impractical when extended to a wide urban area. Nonetheless, short-range technologies can still be used to provide local coverage (e.g., within buildings) or access to the main delivery network by means of opportunistic point-to-point relaying.

- *Low-Power Wide-Area (LPWA).* LPWA technologies have been recently proposed as the ultimate solution to provide data access to the IoT peripheral nodes. Specifically designed for M2M connectivity, they generally provide very low bitrates, low energy consumption, and wide geographical coverage. Some relevant LPWA technologies are LoRaWAN, Sigfox, and Ingenu [262, 263]. The evolution of such technologies has so far been entirely separate from mainstream cellular systems, although one of the objectives of 5G is the convergence of basically all services into a common platform. The Narrowband IoT (NB-IoT) standard is a possible way to achieve this convergence;it can take advantage of the widespread presence of the existing cellular infrastructure to support an LPWA network. As of today, LPWA networks are growing in popularity and commercial interest, although their transmission capacity remains limited and no single technology can claim to be the *de facto* standard for machine-type communications.

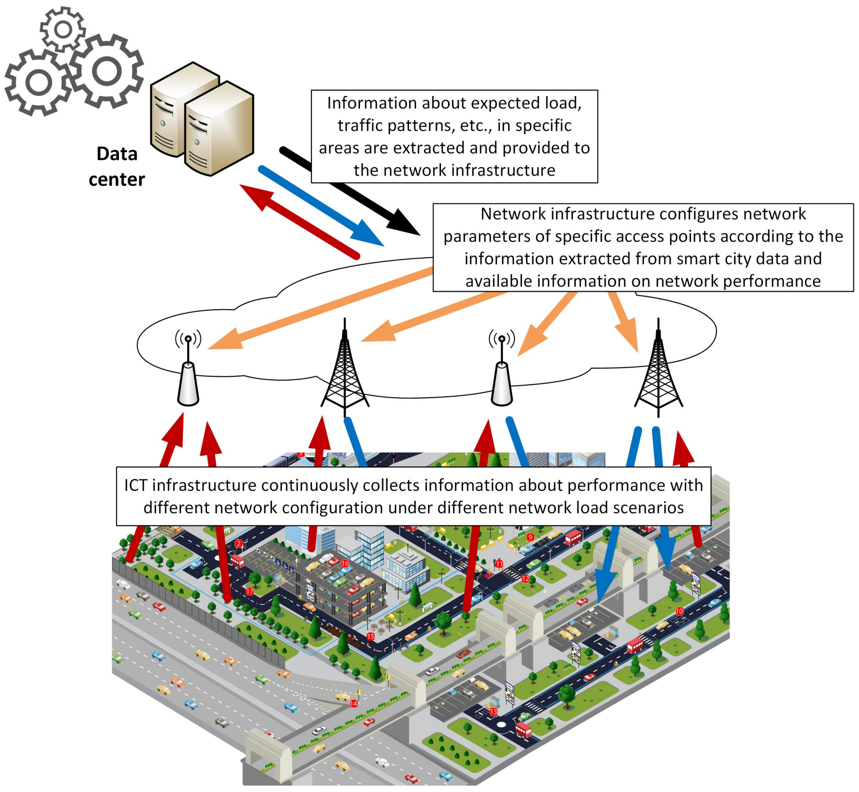

Data center
Information about expected load, traffic patterns, etc., in specific areas are extracted and provided to the network infrastructure
Network infrastructure configures network parameters of specific access points according to the information extracted from smart city data and available information on network performance
ICT infrastructure continuously collects information about performance with different network configuration under different network load scenarios

Fig. 7.1: The SymbioCity vision: knowledge extracted from Smart City data is exploited by the network to configure itself and improve the efficiency of SON techniques.

7.2 The SymbioCity concept

Smart Cities can provide useful services and information to both city managers and citizens [203]; however, services are still developed in a vertical and *ad hoc* fashion, lacking interoperability and often relying on dedicated servers and communication technologies [264]. This unstandardized and unsystematic design strategy hinders the development of the Smart City and does not fully exploit the IoT paradigm [200], making it harder to manage the existing services and to design and implement new ones.

As we explained in Section 7.1, Smart City sensors communicate using either dedicated low power networks or standard cellular networks [265]. Both solutions have their advantages and drawbacks; using cellular networks requires no additional infrastructure investment (place & play concept), but dedicated networks separate the M2M traffic from traditional human communications [11], making network management easier.

In SON [266–268], the core and access network monitor some key performance metrics (such as inter-cell interference, traffic load, packet error rate, etc.) and automatically optimize the network configuration. any SON scheme depends on the capability of the system to dynamically adapt its behavior to the variations of the operational context. Unfortunately, inferring the operational context is a complex process, in particular when observations are limited to the state of the communication system itself, with no outside information. Since one of the 5G design guidelines is the usage of big-data-driven optimization [9, 269] at various scales (e.g., fog computing [270]), any additional source of information would significantly improve the efficiency of SON schemes. In our vision, the flow of information and support between the network and the Smart City sensors goes both ways: while the network allows the Smart City to collect data and perform actions remotely, the information provided by the Smart City sensors can be used to improve the performance of communication systems, as shown in Figure 7.1. In a nutshell, the additional context information that can be obtained from the Smart City services can be exploited by network operators to improve the QoS for all customers, thus increasing the efficiency of the communication system. We called this new paradigm *SymbioCity*, to emphasize the symbiotic relationship between Smart City services and the underlying enabling technologies [271].

Despite the number and variety of access technologies that are currently available to support Smart City services, they will still be hard-pressed to sustain the massive deployment of the envisioned services. Many of their shortcomings, however, can be avoided or mitigated by employing different kinds of network optimization techniques. We now list the ones that we believe would most benefit from the integration of Smart City data:

- *Cell breathing.* Green communications and networking have become an important topic of research in the past few years; for example, cell breathing, i.e., shutting down cells with low load, is an effective scheme to reduce the environmental footprint of the network is . Furthermore, adapting the cell coverage is a way to control the interference and trade connection speed for cell capacity.

- *HetNets.* Future networks can provide high-speed wireless access by exploiting densification: instead of having a few powerful base stations, network operators can place several pico and/or femto base stations closer to the users, complementing the macro cell and exploiting the spectrum more efficiently. This kind of network is called a HetNet. However, the deployment of HetNets raises new problems. As networks tend to become more dense, handovers become more frequent and need to be carefully managed to avoid load imbalances or resource starvation [141]; avoiding inter-cell interference [272] is also crucial. Therefore, the HetNet paradigm has the potential to increase the capacity of the communication system in a certain area, but it requires more sophisticated management mechanisms [273, 274] to fully exploit its potential.

- *Context-aware content distribution.* The massive diffusion of smartphones and tablets has contributed to the increase of the demand for mobile multimedia content. This demand can be further exacerbated by the diffusion of video surveillance services at the urban level. Among the different techniques proposed to address this challenge, a promising approach consists in proactively caching the most popular content in different locations, closer to the final users, thus reducing the latency and the traffic over the core network. However, content caching should account for the users' mobility and the nature of the events that generate the traffic demand.

- *Vehicular networking* The strictly hierarchical structure of traditional wireless networks can be augmented by direct communications between cars and other IoT objects or traditional Internet devices. Accurate and up-to-date knowledge about the traffic patterns [275] is essential to make these approaches effective.

- *Access & scheduling protocols.* Massive access is going to be a problem for the current cellular networks because of the predicted surge in M2M traffic [11,276]. This problem calls for new access schemes and scheduling mechanisms that need to predict, or at least quickly react to, the waves of access requests from multiple devices. A possible approach is to dynamically adapt the settings of some protocol parameters to avoid network collapse. Other 5G applications, such as low-latency transmissions, also need special solutions due to the very restrictive constraints they impose [277]. These advanced access and scheduling protocols need to be enhanced with context awareness and predictive capabilities. *Multi-homing.* In most mobility scenarios, handovers and cell load variations can make the capacity experienced by a mobile user suddenly change; the integration of different access technologies and the appropriate protocols can provide users with a smoother experience while allowing the network to rebalance the load [278], as we described in Chapter 4.

Network Slicing. Different services within the Smart City ecosystem would also require different solutions in terms of user mobility support, handover management, multimedia management (proactive content caching), storing, routing, multi-homing, broadcasting, etc. Network slicing [279,280] is a possible way to provide multiple features over a single network is n. Through network slicing, functionalities such as specific mobility functions or anchor point migration will be configured according to different types of provided services. In addition, other aspects such as path configuration and load balancing need to be carefully set in order to handle Smart City services while maintaining a high degree of freedom in handling mobility issues. Through slicing, the network will be able to differentiate the service offered to M2M traffic, which typically requires low delay and jitter and high packet delivery ratio, and human-type traffic, more sensitive to bandwidth and delay. This technique can also be used to adapt the network's response to changing requirements, giving a higher priority to emergency services and redistributing the load across the backhaul links [281,282].

Table 7.1 shows a summary of the possible benefits of a fully developed

SymbioCity for various network optimization schemes. The networking applications of the Smart City data range from SON to service resilience in stressful situations such as big events and natural disasters.

The first and foremost type of Smart City data is *location*: Smart City services can help the network gauge the position of users, allowing it to optimize cell breathing, coordinate interference and handovers in HetNets, and support M2M traffic and low-delay applications at the same time. On a longer timescale, the city's event calendar can make the network aware of large concentrations of people in advance, allowing service providers to take measures to mitigate the interruption of service that has plagued such events since mobile phones first became popular.

Another interesting development is the integration of location, traffic, parking and public transport data to enable a true city-wide vehicular network. The high speed and unpredictability of cars have always made long-distance vehicular communication extremely complex, but recent efforts to integrate V2V and V2I communication have had some success, and a Smart City-based approach to the problem might be the last piece of the puzzle. Parked cars, buses, and strategically placed access points can form the backbone of such a vehicular network, and Smart City data are necessary to keep it from collapsing. City infrastructure, public transport, taxis and car sharing services can also be exploited to give users multiple paths to the Internet, giving them more resilience to channel outages and a more robust transmission [278].

Some of these factors are already taken into account by network planners, but the lack of a systematic infrastructure makes these optimizations hap-

Smart City Services	Smart network techniques								
	SON	Cell breathing	HetNet management	Context-aware content distr.	Device to Device (D2D), Vehicle to Vehicle (V2V), Vehicle to Infrastructure (V2I), Infrastructure to Vehicle (I2V)	Access & *scheduling*	Multi-homing	Service resilience	Network Slicing
Crowd mapping		strong			strong	strong	strong	strong	
Traffic map			strong	medium	strong	medium	medium	medium	medium
Parking monitoring	medium	weak			strong		medium	medium	
Building automation	medium	weak		medium		medium			strong
Logistics	strong		strong	strong					
Public transport/taxis	medium		medium	medium	medium		strong	strong	
Patrolling (dome cameras)		medium	medium						strong
Bike/car sharing	medium				medium		medium	medium	
City event calendar	strong	strong		strong				strong	medium

Table 7.1: Table of the possible usefulness of Smart City parameters for network optimization.

hazard and extremely dependent on human intervention. In our vision, the SymbioCity should be able to integrate far more diverse data, in real time and without human intervention. While the actual optimization logic will be completely automated, its goals will not: in an environment as complex as a city, network and city planners can have very different objectives and even change them dynamically. For example, in a very polluted city, it might be wise to optimize the network to be more eco-friendly and reduce power usage at night, while using resources to improve vehicular communication during rush hour and mitigate traffic jams. A modular, fully interoperable structure should also be able to integrate new data and applications with ease: since the city expands and adapts to its population, the SymbioCity should grow with it, adapting its functions to the changing environment.

7.3 Analyzing traffic data

In the two case studies we present, we use traffic data from the city of London, provided by TfL. The TfL UTC network is composed of more than 10000 road sensors, placed at all critical crossings around the city. The Split Cycle Offset Optimization Technique (SCOOT) optimizer uses the traffic flow data from the sensors to adapt the traffic light times to the traffic situation in real time. TfL released the raw sensor data of the first three months of 2015 for the North and Central regions of London, and we use those data in our optimization.

The sensors are actually very basic presence-detectors: every $T_s = 250$ ms, each sensor returns a 1 if it detects a vehicle in close proximity, and a 0 otherwise. The resulting binary signal (see Figure 7.2) is packetized and sent to a central collector through different types of technologies.

These values are not directly provided by TfL. However, they can be roughly estimated using the binary signals generated by the detectors. Indeed, when a vehicle of length L moving at speed v passes over a sensor, the detector will generate a run of about $n = \frac{L}{vT_s}$ ones, followed by a few zeros corresponding to the inter-vehicle spacing.

It is then possible to estimate the speed by counting n and assuming a

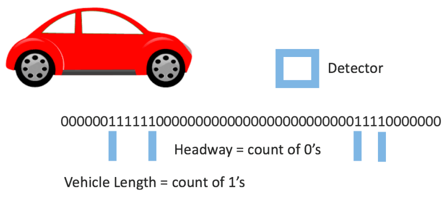


Fig. 7.2: Scheme of a traffic detector. Source: TfL.

reference vehicle length of $L = 4$ m:

$$v = \frac{L}{nT_s} \tag{7.1}$$

Figure 7.3 shows the evolution of the average speed measured by a single sensor over a whole day (namely, January 23, 2015): as expected, the speed of the vehicles is higher at night because of the lighter traffic, while during rush hour (from 8 a.m. to 9 a.m. and from 5 p.m. to 6 p.m.) the average speed drastically decreases. The spatial distribution of traffic is shown in Figure 7.4.

For the second part of our data analysis, we assume that the Macro eNBs are placed using a standard regular hexagonal tiling, with sides of 100 m.

We associate the detection of a car by a sensor in a cell with a handover, and, given a time interval T_{per} equal to 1 hour, we estimate the number of handovers H_m as the total number of detections from the different sensors in cell m during T_{per} . Since the timescale is long and each vehicle is likely detected only once when crossing the area (because of the relatively low density of sensors), the number of vehicles counted in the area in the period T_{per} is roughly equal to the number of cell handovers performed by the vehicles crossing that area in the considered time interval. This assumption is not necessarily realistic for a single cell, but it is a valid approximation on the city-wide scale and for timescales of minutes or hours. Moreover, we assume that on average each vehicle carries an LTE device. This is a working assumption based on the available data, and the integration of additional data such as bus position and usage can be easily accommodated by the framework.

After computing H_m for all eNBs, the cells are partitioned into N areas,

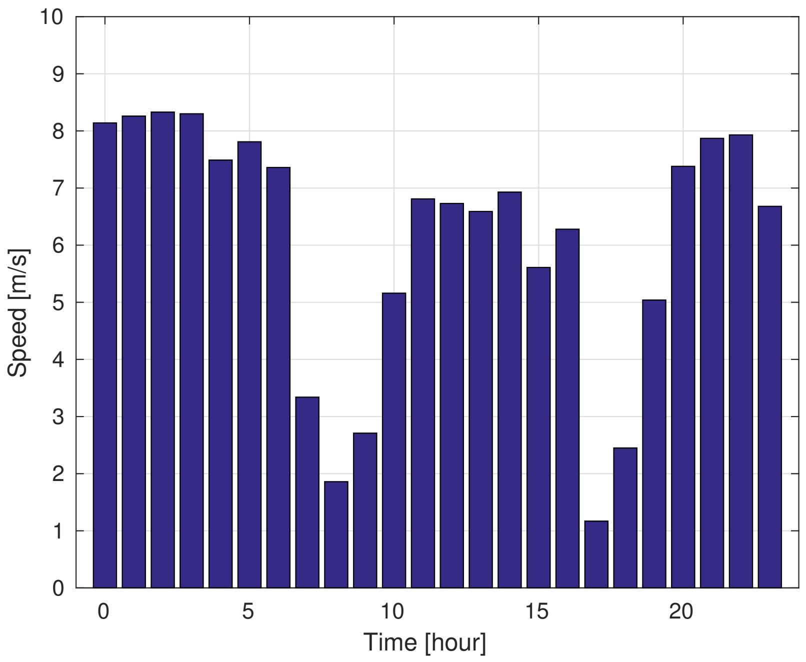


Fig. 7.3: Hourly average speed for 23 January 2015 at the intersection between Homerton High St. and Daubeney Rd.

with $N \in \{1, 2, 3, 4\}$, each controlled by a different Virtual MME (vMME). Given the estimated number of handovers at peak hours, 4 vMMEs should be enough to maintain network stability. The results in Section 7.6 confirm this hypothesis. These groups are obtained using a clustering algorithm that divides the cells among N vMMEs so that each vMME handles approximately the same number of handovers. An example of this is shown in Figure 7.5.

We define I_i as the total number of handovers for vMME i, and $S_{i,j}$ as the number of handovers from vMME i to vMME j. I_i is given by

$$I_i = \sum_{m \in A_i} H_m \tag{7.2}$$

where A_i is the set of cells controlled by vMME i. $S_{i,j}$ can be approximated with this formula:

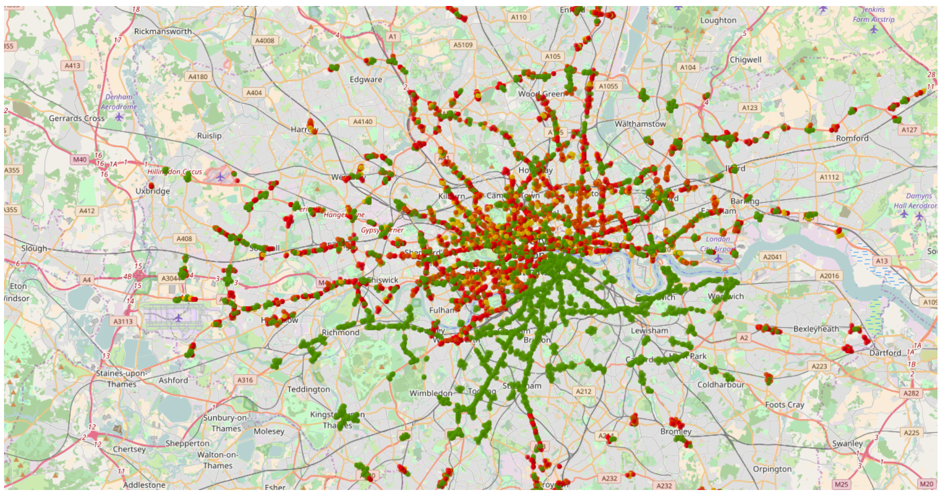
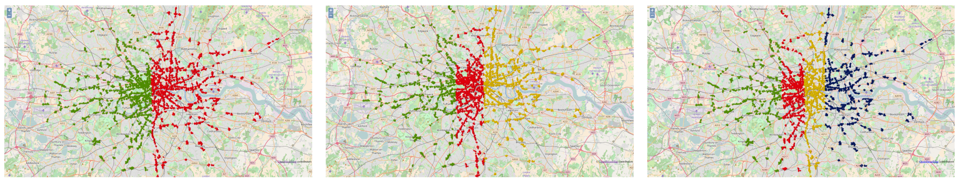


Fig. 7.4: Map of traffic in London from 12 p.m. to 1 p.m. of 23 January 2015. Free intersections are shown in green, heavily congested ones in red.



(a) $N = 2$ (b) $N = 3$ (c) $N = 4$

Fig. 7.5: Partition for a different number N of vMMEs. The colors indicate the areas controlled by each vMME.

$$S_{i,j} = \sum_{m \in A_i} \sum_{n \in A_j} \frac{I_m}{6} e_{m,n} \tag{7.3}$$

where the variable $e_{m,n} \in \{0, 1\}$ indicates the number of sides that cells m and n have in common.

7.4 State of the art

Mobility patterns are some of the most valuable pieces of information that the Smart City can provide to the network. Mobility affects ultra-dense network performance significantly, as suboptimal handover strategies both (i) increase the Radio Access Network (RAN) and the Core Network (CN) signaling and (ii) reduce the overall throughput. The research on mobility models [283, 284] and their integration in communication protocols (e.g., Medium Access [285] or interference coordination [286]) is already ongoing, and using real Smart City data as input for these techniques would reduce the uncertainty compared to purely statistical approaches.

7.4.1 Handover in HetNets

In order to support the ever growing traffic demand within the limits of the available spectrum, cellular networks are becoming denser and denser. Micro-, femto- and pico cells have been a hot topic of research for the last ten years [287] and are now being deployed all over the world. The main challenges that the network densification is causing are (i) interference coordination and (ii) cell association and mobility management. The SON approach is one of the most promising candidates for addressing these complex issues [288].

We now concentrate on handover management; while handover algorithms are well-studied and several decision criteria have been proposed in the literature [289], the most common ones are based on Received Signal Strength (RSS). 3GPP defines a baseline handover procedure for LTE in [290], and most studies concentrate on optimizing its parameters. The handover is triggered if the difference between the serving and the neighbor cell RSS is larger than a threshold value for at least one Time-to-Trigger (TTT). This parameter is meant to avoid unnecessary handovers due to fluctuations caused by fast fading, but it introduces a delay in the association with the optimal eNB, whose impact becomes more significant as the User Equipment (UE) speed increases [291, 292]. An analytical model for optimizing the TTT is introduced in [253].

Using the TTT to reduce ping-pong effect inevitably leads to a higher handover delay. In order to overcome this trade-off, we need to exploit other parameters, such as the hysteresis threshold. Biasing this threshold towards femto cells is already a standard practice to favor offloading from the Macro tier [287], and it is possible to adapt the bias based on the user mobility to reduce the handover delay problem caused by the TTT. In [293], the authors present a heuristic that reacts to late or early handovers and adapts the bias for each pair of neighboring cells. Another work jointly adapts the TTT and bias in a reactive manner [294].

7.4.2 Virtual MME

One of the main architectural trends in the evolution towards 5G is Network Function Virtualization (NFV): instead of using specialized and costly hardware in both the core and the access network, most of the processing is virtualized and run on general-purpose machines in the cloud [70]. This allows a larger flexibility and adaptability to the instantaneous load of the control and user planes. The initialization cost of a new Virtual Machine (VM) is orders of magnitude smaller than the cost of the equivalent worst-case dimensioned hardware. A broad overview of the issues and other potential benefits of NFV is presented in [295]. Although the research is still ongoing, preliminary results [296] show that it is possible to increase the energy efficiency of the network without significant performance losses.

In the second part of this work, we focus on handover management in virtualized MMEs. A first model of the performance of the different virtualized CN functions is presented in [297], and the MME is identified as a critical element for scalability of control plane functionalities. Virtualization can also enable distributed MME designs [298].

An optimized design of a virtualized MME is given in [299], where the number of vMME instances is adapted to the traffic load in an M2M scenario, using a traffic model for CN-related events.

7.5 Asymmetrical Handover Bias Optimization in HetNets

In this simulation we provide a technique to dynamically set the handover range expansion bias of Femto eNBs (FeNBs) in order to improve the capacity provided to the UE by the only Macro eNB (MeNB).

We focus on a scenario consisting of a MeNB with transmission power P_{TX}^M and a FeNB with transmission power P_{TX}^F placed at a distance d_{MF} from each other. The two tiers transmit at different carrier frequencies (off-band HetNets) to avoid cross-tier interference [300]: f_0^M for the MeNB and f_0^F for the FeNB. Both tiers use the same bandwidth B. The parameters we used in the simulation are summarized in Table 7.2.

We consider the following channel model, with Friis path loss and log-normal shadowing:

$$P_{RX}(t) = P_{TX}(t)\frac{\Psi_{SH}\alpha(t)}{P_L(f_0, \beta, d)}, \tag{7.4}$$

where Ψ_{SH} is the shadowing gain, which is distributed as $\mathcal{N}(0, \sigma)$ when measured in dB, $\alpha(t)$ is the multi-path fading gain, and $P_L(f_0, \beta, d)$ is the path loss attenuation with exponent β.

The Signal to Noise Ratio (SNR) at time t is denoted by $\gamma_F(t)$ and $\gamma_M(t)$ for the FeNB and the MeNB respectively, and it is defined as:

$$\gamma_M(t) = \frac{P_{RX}^H}{N_0 B} \qquad H \in \{M, F\}, \tag{7.5}$$

where N_0 is the noise power spectral density.

Parameter	Value	Description
P_{TX}^M	46 dBm	MeNB transmission power
P_{TX}^F	26 dBm	FeNB transmission power
N_0	-143.82 dBW/MHz	Noise power spectral density
f_0^M	900 MHz	MeNB carrier frequency
f_0^F	1800 MHz	FeNB carrier frequency
B	20 MHz	Bandwidth [MHz]
d_{M-F}	40 m	Distance between MeNB and FeNB
d_{F-UE}	10 m	Distance between MeNB and FeNB
σ_M^2	8 dB	MeNB log-normal shadowing variance
σ_F^2	4 dB	FeNB log-normal shadowing variance
β_M^2	4.28	MeNB path loss exponent
β_F^2	3.76	FeNB path loss exponent
TTT	256 ms	Handover Time-to-Trigger

Table 7.2: Parameters used in the simulation (taken from [290, 301]).

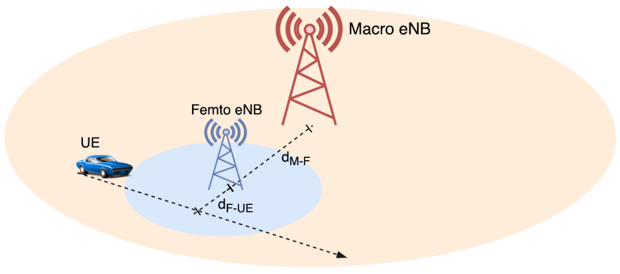


Fig. 7.6: UE trajectory in the considered scenario.

For the sake of simplicity, we assume that one UE is attached in the MeNB, moving as in Figure 7.6 with constant speed v. The UE speed at any time is derived from the TfL data as explained in Section 7.3; the average speed over the whole day is shown in Figure 7.3. We consider the UE to move at the average speed of the traffic around it.

The SNR that the UE can perceive while moving depends on its distance from the Macro and Femto eNBs. As we can see in Figure 7.7, the SNR from the FeNB is higher than that from the MeNB when the UE is close to the FeNB. The coverage area of the FeNB is defined as the area in which its SNR is higher than any other cell's.

In this scenario, the UE has to start a handover procedure towards the FeNB when the condition

$$P^F_{RX}(t) + \gamma_{th} > P^M_{RX}(t) \tag{7.6}$$

holds for a period of time equal to the TTT, as specified in [302]. Note that in the simulation we have assume $\gamma_{th} = 0$ for the sake of simplicity. We hence set TTT = 256 ms [290], which is high enough to avoid the ping-pong effect but small enough to minimize the handover delay.

This TTT value improves the performance of the system considerably when the traffic is moving slowly, but reduces the Theoretical Spectral Efficiency $\nu = log_2(1 + \gamma)$ when the UE speed is too high. This is because a fast-moving UE exploits the advantages of the FeNB for just a short time, while it remains in the FeNB for TTT seconds after the condition (7.6) is reversed.

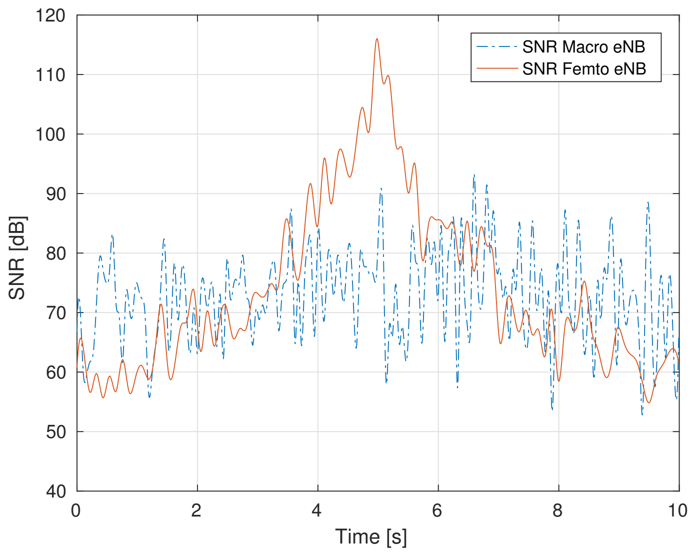


Fig. 7.7: $\gamma_M(t)$ and $\gamma_F(t)$ with a UE speed of 10 m/s. Multi-path fading is not considered in this figure for reasons of visual clarity.

To make sure that the UE starts the handover towards the FeNB as soon as (7.6) is verified, an asymmetrical handover bias can be applied to P^F_{RX}. When the handover is towards the FeNB, the bias needs to be positive to anticipate the beginning of the procedure, while when the handover is from the FeNB to the MeNB, the bias must be negative. We define the SNR difference in position x as

$$\Delta(x) = \bar{\gamma}_F(x) - \bar{\gamma}_M(x). \tag{7.7}$$

where $\bar{\gamma}_F(x)$ and $\bar{\gamma}_M(x)$ are the average SNRs of the two eNBs in position x. Moreover, the trajectory of the UE draws a chord within the coverage area of the FeNB, with linear coordinates $-r$ and r with respect to the central point of the chord, as shown in Figure 7.6. The optimal value of the bias is then given by

$$B_1 = \Delta(-r - vTTT) \tag{7.8}$$

$$B_2 = -\Delta(r - vTTT). \tag{7.9}$$

If the FeNB uses the optimal bias, the handover will be performed exactly

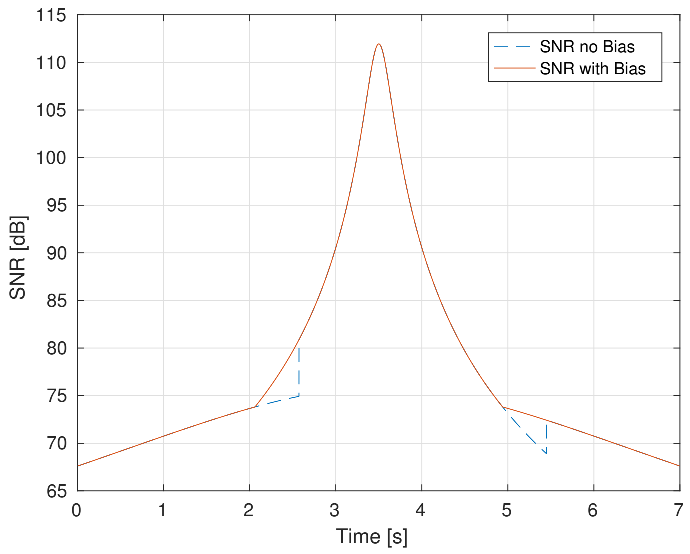


Fig. 7.8: $\gamma_M(t)$ and $\gamma_F(t)$ with a UE speed of 16 m/s. Multi-path fading and shadowing are not considered in this figure for reasons of visual clarity.

on the edge of its coverage area. Note that the handover bias is linearly proportional to the UE's speed. By applying B_1 and B_2 to P_{RX}^F, (7.6) becomes

$$P_{RX}^F(t) + B_1 > P_{RX}^M(t) \tag{7.10}$$

while the condition to leave the FeNB is

$$P_{RX}^M(t) + B_2 > P_{RX}^F(t) \tag{7.11}$$

The difference between $\bar{\gamma}(x)$ with or without bias can be viewed in Figure 7.8. Since the Theoretical Spectral Efficiency ν depends logarithmically on $\bar{\gamma}(x)$, using the handover bias will increase ν, fully exploiting the FeNB.

However, the bias from (7.10) and (7.11) does not take shadowing and fading into account: while this is optimal in an ideal situation, real channels often experience deep fading, and a bias value tailored to the path loss difference between the two base stations does not protect the UE from them. In order to avoid resetting the timer every time the fading envelope exceeds the path loss-based bias, we can add an additional bias term B_f, which does not depend on the speed of the UE.

$$B_f = 10\log_{10}\left(\min\left\{B : p\left(\frac{\psi_M \alpha_M}{\psi_F \alpha_F} \geq B\right) \leq 1 - p_{\text{thr}}\right\}\right) \tag{7.12}$$

$$B'_i = B_i + B_f, \quad i \in \{1, 2\} \tag{7.13}$$

The parameter p_{thr} in (7.12) represents the amount of protection against deep fading offered by the extra bias term B_f: a higher value of p_{thr} will reset the TTT timer less often, but a higher bias will lead to stronger ping-pong effects. For this reason, we limit the total handover bias $B'_i, i \in \{1, 2\}$ to a maximum of 7 dB. The value of B_f is shown in Table 7.3 for different p_{thr}, which correspond to different percentiles. In the performance evaluation we used p_{thr}=0.68, which is equivalent to one standard deviation in the normal approximation.

The improvement obtained by setting an asymmetric handover bias can be seen in Figure 7.9. This figure is obtained calculating the average ν over 100 Monte Carlo simulations with independent shadowing and fading for a UE speed from 4 m/s to 20 m/s.

In the simplest case, in which there is no FeNB and the UE is always attached to the MeNB, ν_{MeNB} is essentially independent of the UE speed. The second case is a legacy handover with no bias: as the plot shows, ν_{noBias} decreases drastically as speed increases, as the delay in the handover caused by the TTT wastes most of the performance improvement from the FeNB. If the UE speed is higher than 6 m/s, the handover is so late that the UE would do better to disregard the existence of the FeNB completely: as soon as the UE finishes the handover process, it has to start it again since it has already moved outside of the FeNB coverage area. The improvement given by the

p_{thr}	0.5	0.68 (σ)	0.75	0.95 (2σ)	0.99 (3σ)
B_f [dB]	0	3.8	5.4	13.7	21.4

Table 7.3: Values of B_f for different threshold probabilities.

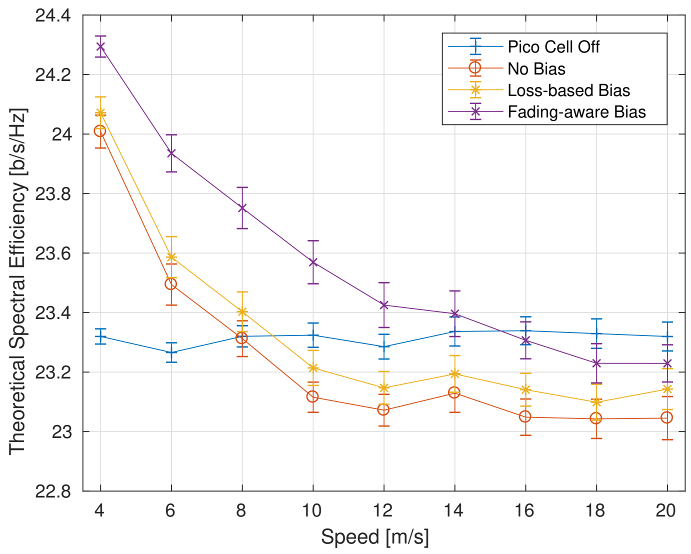


Fig. 7.9: Theoretical Spectral Efficiency as a function of vehicular traffic speed v.

fading-aware bias is clear: the FeNB can be exploited if the speed is lower than 16 m/s, and there is a clear performance gain compared to the legacy scheme. The path loss-based bias shows a smaller performance improvement with respect to the legacy scheme, and handing over to the FeNB is already detrimental to the UE at 10 m/s. This is due to the handover happening too late, as even a large bias is not enough to balance the variations of the channel due to fading. In general, ν_{Bias} decreases when the speed increases, since the time in the FeNB coverage area gets shorter, but the FeNB is always fully exploited. Note that the effect of the 7 dB cap is only relevant at a speed of 20 m/s.

The presence of the FeNB is detrimental to vehicular UEs in the legacy scenario (no handover bias) if the speed of traffic passes 6 m/s, since $\nu_{\text{noBias}} \leq \nu_{\text{MeNB}}$. However, setting the optimal asymmetrical handover bias allows network operators to keep the FeNB switched on until the speed reaches 16 m/s, benefiting both pedestrian and vehicular UEs, in any situation, since $\nu_{\text{Bias}} \geq \nu_{\text{MeNB}}$ at any speed.

We also performed a sensitivity analysis by adding a normally distributed

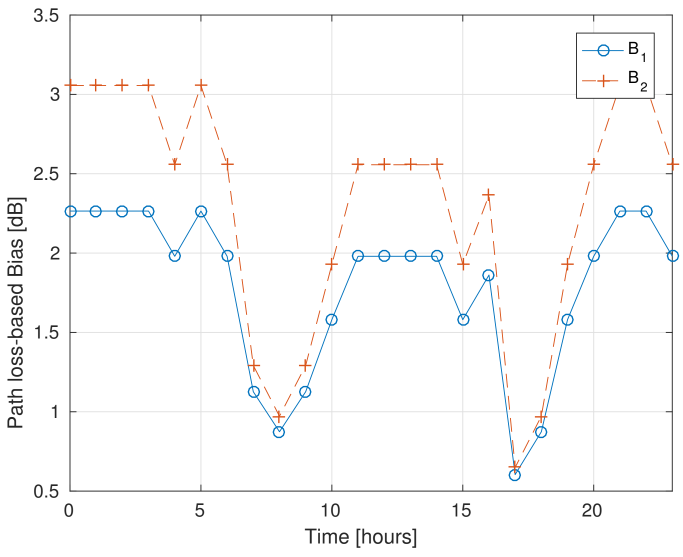


Fig. 7.10: Optimal path-loss handover bias throughout the day for 23 January 2015.

error with standard deviation σ_v to the velocity estimate used to determine the bias and performing multiple independent simulations. The metric we consider is the maximum value $\Delta\nu$ of the difference in the spectral efficiency for all the considered velocities. As shown in Table 7.4, the effects of the errors in the speed estimation are negligible when compared to the randomness of the channel (represented by the standard deviation $\hat{\sigma}$ in the table). This makes the system robust to small variations of the speed of the flow of traffic, as well as protecting it from imprecisions due to vehicles of different lengths, i.e., the parameter L in (7.1).

The optimal asymmetrical handover bias over the course of a day for a specific intersection can be calculated from the TfL data as explained in

σ_v/v	0.1	0.2	0.3
$\Delta\nu$	$0.05\hat{\sigma}$	$0.07\hat{\sigma}$	$0.09\hat{\sigma}$

Table 7.4: Effect of errors in the speed estimate on the system performance. $\hat{\sigma}$ is the standard deviation of ν across independent channel realizations.

Section 7.3; the speed evolution shown in Figure 7.3 results in the bias shown in Figure 7.10. As expected, the handover bias is higher at nighttime, as the average speed of traffic is far higher than during the day. For this reason we can fix a threshold for the handover bias beyond which FeNB can be shut down in order to save energy, leaving all traffic to the MeNB. If we fix this threshold to 3 dB, then the FeNB will only turn off in the middle of the night, when the load on the MeNB is very light.

7.6 Adaptive vMME Allocation

As already mentioned in Section 7.4, NFV allows to dynamically allocate the resources needed by a cellular network. In traditional mobile networks a single dedicated MME is usually used to manage millions of end users, as those in the London metropolitan area [299]. With the NFV approach, instead, it is possible to change the number of vMME instances on the fly, adapting to the number of handovers that are expected to happen in a certain interval.

In this application, we use data processed as in Section 7.3 to determine the number of handovers that happen in the London area during a typical day. We distinguish between the two kind of handovers that may happen in LTE networks [303], i.e., intra-MME (X2–based) and inter-MME (S1–based) handovers, since they require different procedures and different interactions with the MMEs. The X2–based handover happens when the UE remains in an area managed by the same MME and changes the eNB to which it is attached. The S1–based procedure, instead, is used when the UE performs a handover between two eNBs managed by different MMEs. The two procedures are described in detail in [303]. In this paper, we consider the duration of handover procedure as the interval from the instant in which the source eNB (SeNB) triggers the handover to the instant in which SeNB receives the `RELEASE_RESOURCES` command. During this period the UE firstly experiences a degraded channel, then receives packets with an increased latency, thus the Quality of Service perceived by the final user decreases. The goal of this application is to minimize the duration of these intervals, while using as few vMME instances as possible.

In particular, we model the duration of an X2–based handover handled by vMME i as a function of the number of vMMEs N and of the total number of handovers I_i that involve that vMME during an interval T_{per}:

$$t_{HO}^{X2}(N, I_i) = 3t_{S_e - T_e} + 2t_{T_e - S_M}(N) + t_{HR} + \tau(I_i) \tag{7.14}$$

while the time required to complete an S1–based handover that involves vMMEs i and j also depends on the number of handovers I_j that are served by the target vMME j:

$$\begin{aligned} t_{HO}^{S1}(N, I_i, I_j) = &\tau_1(I_i) + 3\tau_2(I_j) + \\ &4t_{S_e - S_M}(N) + 4t_{T_e - T_M}(N) + \\ &2t_{S_M - T_M}(N) + \max\{t_{T_M - S_M}(N) + \\ &t_{S_M - S_e}(N) + t_{HR}, t_{T_M - S_e}(N)\} + \\ &\max\{t_{T_M - S_M}(N) + \tau_1(I_i), t_{T_M - S_e}\} \end{aligned} \tag{7.15}$$

In (7.14) and (7.15), $t_{A-B}(N)$ with $A, B \in \{T_e, S_e, S_M, T_M\}$[2] is the latency between the element A and the element B of the network. Unless both A and B represent eNBs, then

$$t_{A-B}(N) = t_{tx} + \frac{d_N(A, B)}{v_f}, \tag{7.16}$$

with $t_{tx} = 5$ ms a factor that models the time spent in middleboxes and $t_{PROP} = d_N(A, B)/v_f$ is the propagation delay, given by the ratio of the distance between the two devices and the speed of light inside optical fibers[3] (i.e., $v_f = 2 \cdot 10^8$ m/s). The dependence on the number of vMMEs N is in the distance $d_N(A, B)$ between two network elements, that changes according to the allocation of eNBs to the vMMEs. Instead, $t_{T_e - S_e}$ is the latency time between two adjacent eNBs and does not depend on the relative position

[2] T_e stands for Target eNB, S_e stands for Source eNB, T_M stands for Target MME and S_M stands for Source MME

[3] We assume that the backhaul network uses fiber-optic links.

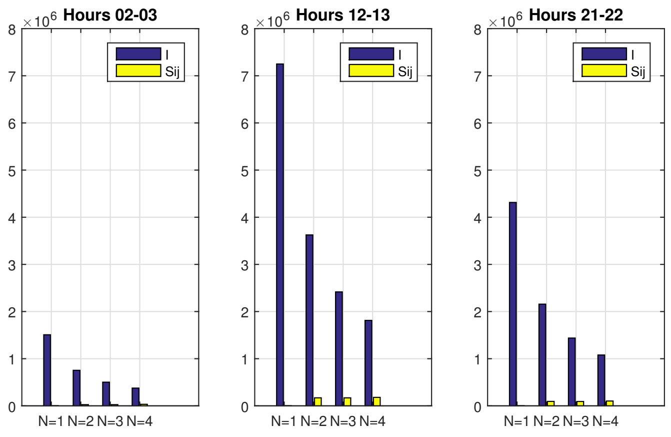


Fig. 7.11: Average number of X2–based and S1–based handovers per vMME instance, for a different N and different time slots, during 01/23/2015.

between the eNBs and the MMEs, therefore, as in [304], it is modeled as a constant latency $t_{T_e-S_e} = 2.5$ ms. t_{HR} is the duration of the interval from which the UE actually disconnects from the SeNB and connects to the TeNB. In [305], t_{HR} is estimated to be in the order of 50 ms.

Finally, $\tau(I_i)$ is the time that a vMME takes to elaborate the received command. In [299] the process of handover requests is modeled as a Markov process. We adopt the same approach and in particular we model the vMME as an M/D/1 queue, assuming a Poisson arrival process with *arrival rate* $\lambda = I_i/T_{per}$ and a deterministic service time T_s. Given these assumptions, it is possible to compute the value of τ as the *system time* of a M/D/1 queue:

$$\tau = \frac{1}{\mu} + \frac{\rho}{2 \cdot \mu \cdot (1-\rho)}, \tag{7.17}$$

where $\mu = 1/T_s$ and $\rho = \lambda T$ are the *service rate* and the *loading factor* of the vMME. The study in [297] uses the value $T_s = 110$ μs as *service time* of a vMME, requiring considerable computational resources. Since our work only considers vehicular UEs, and the adaptive nature of our system, over-dimensioning each vMME would be a waste of resources: a number of slow vMMEs can provide the same performance as a single powerful vMME during rush hour, and the additional vMMEs can be turned off at less congested

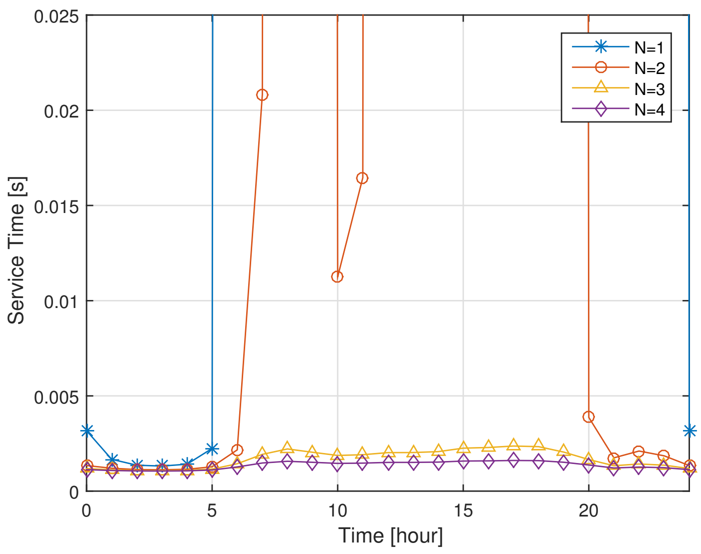


Fig. 7.12: Average service time τ for different N, during 01/23/2015.

times, with a substantial reduction in server management costs and energetic requirements. For this reason, we limit the processing power of our vMMEs dedicated to vehicular handovers to the value of $\mu = 1000$ handovers per second.

Since our goal is to find the optimal number of vMMEs N that minimizes the total duration of the handovers, we consider the objective function

$$J_{T_{per}}(N) = \sum_{i=1}^{N}(I_i - \sum_{j\neq i} S_{i,j})t_{HO}^{X2}(N, I_i) + \sum_{i=1}^{N}\sum_{j=1, j\neq i}^{N} S_{i,j}t_{HO}^{S1}(N, I_i, I_j) + C(N), \tag{7.18}$$

where the sums consider all the handovers in a time slot T_{per} of one hour, and $C(N)$ is a penalty function representing the operational cost of N vMMEs. We consider it to be a linear function of the number of vMMEs N, i.e., $C(N) = kN$.

The optimization problem uses the vehicular traffic data elaborated in Section 7.3 to compute the value of I_i, $S_{i,j}$ and $\lambda(I_i) = I_i/T_{per}$ for each vMME $i, j \in \{1, \cdots, N\}$ and computes

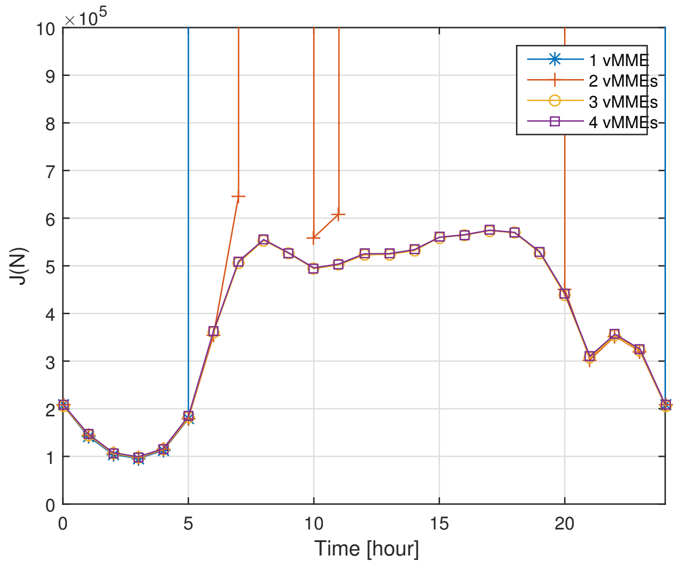


Fig. 7.13: Objective function $J(N)$, for $N \in \{1, 2, 3, 4\}$, during 01/23/2015.

$$N_{opt} = \min_{N} J_{T_{per}}(N) \tag{7.19}$$

for each interval T_{per} during a certain day.

In the following results we consider the data of January 23rd, 2015. Figure 7.11 shows the average number of handovers inside a single vMME in different time slots. Notice that since we consider only the inter MME handovers for the London area MMEs, then S_{ij} is zero for $N = 1$. The number of handovers in different time slots changes greatly, from $1.5 \cdot 10^6$ per hour during the night to more than $7 \cdot 10^6$ at midday. This justifies a dynamic allocation of resources; a single and dedicated MME that targets the worst case scenario at midday would be wasted during the night. Instead the adaptive approach allows to use less powerful vMMEs, which are able to serve a smaller number of handover requests, and have lower operational expenses than dedicated hardware [295], but can be instantiated on the fly according to the control traffic intensity.

In Figure 7.12, the average service time of the vMME instances is shown for different values of N. It can be seen that during the night the values have a small difference, but one or two vMME instances are not enough to

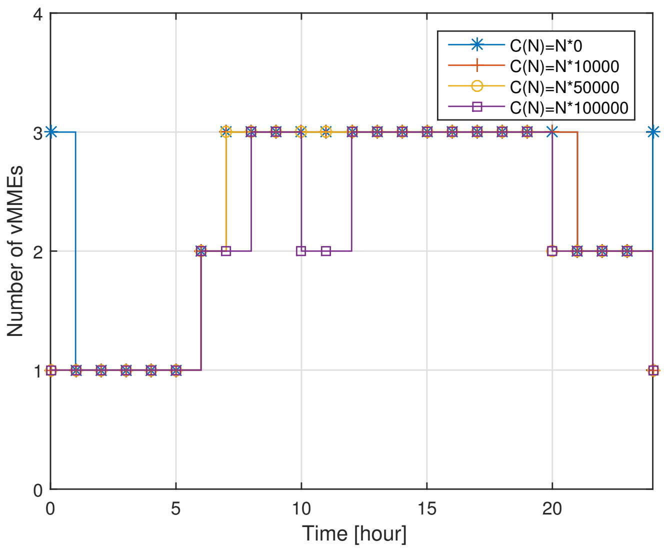


Fig. 7.14: N_{opt} for different costs $C(N)$, during 01/23/2015.

handle the load during the day. Figure 7.13, instead, shows the value of the objective function $J(N)$ throughout the whole day, assuming a cost factor $k = 0$. In this case, one vMME instance is enough only from midnight to 5 a.m., and more instances (up to 3) must be allocated during the day to meet the vehicular handover traffic load.

If we increase the value of k, as shown in Figure 7.14, the optimal number of vMMEs changes. At certain times using a lower number of vMMEs becomes more convenient, because of the operational cost which is now accounted for.

The adaptation of the number of vMMEs significantly improves the efficiency of the system: while a worst-case dimensioned system would need 3 vMMEs at all times, the average number of active vMME instances for the most aggressive adaptive system ($k = 0$) is 2.42, while a more conservative system ($k = 100000$) only uses an average of 2.17 vMMEs.

Chapter 8

Conclusion

In this thesis, we presented an overview of how intelligence can be applied to the networking stack. First, we defined intelligence as “an agent’s ability to achieve goals in a wide range of environments” [1]; then, we applied this definition in a communications context and described the necessary steps to apply it.

The first step towards intelligence was prediction: by generating a prediction of the future environment, a system can make more informed choices, as we demonstrated in Chapter 4. Secondly, we present how RL can be a general solution to several problems in the field; all our considerations are supported by several case studies at different layers of the protocol stack. Thirdly, we presented examples of how data integration in a Smart City environment can enhance the solutions we discuss, benefiting both the Smart City services and the network. Finally, we presented how intelligence is fully general, and the concepts and techniques we described can be applied to improve networks of all kinds, not just in communications.

Naturally, the topic of intelligent networking is too complex and vast to tackle at once, but we provide a framework to guide the discussion of future intelligent networking protocol and make some tentative steps towards a truly intelligent network. Over the next few years, we hope to extend it and move towards an intelligent network that can support new applications and technologies and fulfill the promises of 5G.

8.1 Published works

The work presented in this thesis has led to the following publications, divided by chapter and presented in chronological order:

Chapter 3

- F. Chiariotti, D. Del Testa, M. Polese, A. Zanella, G. M. Di Nunzio, and M. Zorzi, "Learning Methods for Long-term Channel Gain Prediction in Wireless Networks," in *International Conference on Computing, Networking and Communications (ICNC)*, pp. 162-166, IEEE, Jan. 2017.
- E. Lovisotto, E. Vianello, D. Cazzaro, M. Polese, F. Chiariotti, D. Zucchetto, A. Zanella, and M. Zorzi, "Cell Traffic Prediction Using Joint Spatio-temporal Information," in *6th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pp. 1-4, IEEE, Apr. 2017.
- M. Gentil, A. Galeazzi, F. Chiariotti, M. Polese, A. Zanella, and M. Zorzi, "A Deep Neural Network Approach for Customized Prediction of Mobile Devices Discharging Time," in *IEEE Global Communications Conference (GLOBECOM)*, pp. 1-6, IEEE, Dec. 2017.

Chapter 4

- F. Chiariotti, S. Kucera, and A. Zanella, "Method for Explicit Quality-of-Service Support in 5G Networks," Patent Application PCT/EP2017/082279 (pending), Dec. 2017.
- F. Chiariotti, S. Kucera, A. Zanella, and H. Claussen, "LEAP: A Latency Control Protocol for Multi-Path Data Delivery with Pre-Defined QoS Guarantees," in *IEEE International Conference on Computer Communications (INFOCOM) Workshop on Cloud Computing Systems, Networks, and Applications (CCSNA)*, pp. 166-171, IEEE, Apr. 2018.
- F. Chiariotti, S. Kucera, A. Zanella, and H. Claussen, "LEAP: A Latency Control Protocol for Multi-Path Data Delivery with Pre-Defined QoS Guarantees," submitted to *IEEE/ACM Transactions on Networking.*

Chapter 5

- F. Chiariotti, S. D'Aronco, L. Toni, and P. Frossard, "Online Learning Adaptation Strategy for DASH Clients," in *7th International Conference on Multimedia Systems (MMSys)*, pp. 8:1-8:12, ACM, May 2016.

- M. Gadaleta, F. Chiariotti, M. Rossi and A. Zanella, "D-DASH: A Deep Q-Learning Framework for DASH Video Streaming," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 703-718, Dec. 2017.

Chapter 6

- F. Chiariotti, C. Pielli, A. Zanella, and M. Zorzi, "A Dynamic Approach to Rebalancing Bike-Sharing Systems," *MDPI Sensors*, vol. 18, no. 2, art. 512, Jan. 2018.
- F. Chiariotti, C. Pielli, A. Cenedese, A. Zanella, and M. Zorzi, "Bike Sharing as a Key Smart City Service: State of the Art and Future Developments," in *7th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pp. 1-6, IEEE, May 2018.
- F. Chiariotti, C. Pielli, A. Zanella, and M. Zorzi, "A Dynamic Approach to Rebalancing Bike-Sharing Systems," submitted to *IEEE Transactions on Intelligent Transport Systems.*

Chapter 7

- M. Dalla Cia, F. Mason, D. Peron, F. Chiariotti, M. Polese, T. Mahmoodi, M. Zorzi, and A. Zanella, "Mobility-aware Handover Strategies in Smart Cities," in *International Symposium on Wireless Communication Systems (ISWCS)*, pp. 438-443, Aug. 2017.
- F. Chiariotti, M. Condoluci, T. Mahmoodi, and A. Zanella, "SymbioCity: Smart Cities for Smarter Networks". *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 1, pp. 1-16, Jan. 2018.
- M. Dalla Cia, F. Mason, D. Peron, F. Chiariotti, M. Polese, T. Mahmoodi, M. Zorzi, and A. Zanella, "Using Smart City Data in 5G Self-Organizing Networks," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 645-654, Apr. 2018.

Additional works not presented in this thesis

- F. Chiariotti, C. Pielli, N. Laurenti, A. Zanella, and M. Zorzi, "A Game-theoretic Analysis of Energy-depleting Jamming Attacks," in *International Conference on Computing, Networking and Communications (ICNC)*, pp. 100-104, IEEE, Jan. 2017.
- R. Coutinho, F. Chiariotti, D. Zucchetto, and A. Zanella, "Just-in-time Proactive Caching for DASH Video Streaming," in *17th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, pp. 1-6, IEEE, June 2018.

- F. Chiariotti, C. Pielli, N. Laurenti, A. Zanella, and M. Zorzi, “A Game-Theoretic Analysis of Energy-Depleting Jamming Attacks with a Learning Counterstrategy,” submitted to *Hindawi Wireless Communications and Mobile Computing*.
- D. Talon, L. Attanasio, F. Chiariotti, M. Gadaleta, A. Zanella, and M. Rossi, “Comparing DASH Adaptation Algorithms in a Real Network Environment,” submitted to *IEEE Wireless Communications and Networks Conference (WCNC)*, IEEE, Apr. 2019.
- R. Zanol, F. Chiariotti, and A. Zanella, “Drone Mapping Through Multi-agent Reinforcement Learning,” submitted to *IEEE Wireless Communications and Networks Conference (WCNC)*, IEEE, Apr. 2019.
- M. Polese, F. Chiariotti, A. Zanella, and M. Zorzi, “A Survey of Recent Advances in Transport Layer Protocols,” submitted to *IEEE Communications Surveys & Tutorials*.

Bibliography

[1] S. Legg and M. Hutter, "A collection of definitions of intelligence," *Frontiers in Artificial Intelligence and applications*, vol. 157, pp. 17–24, June 2007.

[2] S. Legg and M. Hutter, "Universal intelligence: A definition of machine intelligence," *Minds and Machines*, vol. 17, pp. 391–444, Dec. 2007.

[3] C. Jiang, H. Zhang, Y. Ren, Z. Han, K.-C. Chen, and L. Hanzo, "Machine learning paradigms for next-generation wireless networks," *IEEE Wireless Communications*, vol. 24, pp. 98–105, Apr. 2017.

[4] S. Glisic, "Self-Organizing Networks," in *Advanced Wireless Networks: Technology and Business Models, Third Edition*, pp. 478–485, Wiley Online Library, May 2016.

[5] O. Iacoboaiea, B. Sayrac, S. B. Jemaa, and P. Bianchi, "SON conflict diagnosis in heterogeneous networks," in *26th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pp. 1459–1463, IEEE, Aug. 2015.

[6] O. C. Iacoboaiea, B. Sayrac, S. Ben Jemaa, and P. Bianchi, "SON conflict resolution using reinforcement learning with state aggregation," in *4th Workshop on All Things Cellular: Operations, Applications, & Challenges*, pp. 15–20, ACM, Aug. 2014.

[7] N. Zia, S. S. Mwanje, and A. Mitschele-Thiel, "A policy based conflict resolution mechanism for MLB and MRO in LTE self-optimizing networks," in *Symposium on Computers and Communication (ISCC)*, pp. 1–6, IEEE, June 2014.

[8] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. Soong, and J. C. Zhang, "What will 5G be?," *IEEE Journal on Selected Areas in Communications*, vol. 32, pp. 1065–1082, June 2014.

[9] A. Imran, A. Zoha, and A. Abu-Dayya, "Challenges in 5G: how to empower SON with big data for enabling 5G," *IEEE Network*, vol. 28, pp. 27–33, Nov. 2014.

[10] Cisco Visual Networking Index, "Global mobile data traffic forecast update, 2015–2020," *Cisco White Paper*, Feb. 2016.

[11] A. Biral, M. Centenaro, A. Zanella, L. Vangelista, and M. Zorzi, "The challenges of M2M massive access in wireless cellular networks," *Digital Communications and Networks*, vol. 1, pp. 1–19, Feb. 2015.

[12] J. Hoadley and P. Maveddat, "Enabling small cell deployment with HetNet," *IEEE Wireless Communications*, vol. 19, pp. 4–5, Apr. 2012.

[13] Y. Niu, Y. Li, D. Jin, L. Su, and A. V. Vasilakos, "A survey of millimeter wave communications (mmWave) for 5G: opportunities and challenges," *Wireless Networks*, vol. 21, pp. 2657–2676, Nov. 2015.

[14] M. Zorzi, A. Zanella, A. Testolin, M. D. F. D. Grazia, and M. Zorzi, "Cognition-based networks: A new perspective on network optimization using learning and distributed intelligence," *IEEE Access*, vol. 3, pp. 1512–1530, Aug. 2015.

[15] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press Cambridge, Sep. 1998.

[16] Y. Dong, N. V. Chawla, and A. Swami, "Metapath2vec: Scalable representation learning for heterogeneous networks," in *23rd ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pp. 135–144, ACM, Aug. 2017.

[17] D. Laselva, M. Mattina, T. E. Kolding, J. Hui, L. Liu, and A. Weber, "Advancements of QoE assessment and optimization in mobile networks in the machine era," in *Wireless Communications and Networking Conference Workshops (WCNCW)*, pp. 101–106, IEEE, Apr. 2018.

[18] D. Sabella, P. Serrano, G. Stea, A. Virdis, I. Tinnirello, F. Giuliano, D. Garlisi, P. Vlacheas, P. Demestichas, V. Foteinos, *et al.*, "A flexible and reconfigurable 5G networking architecture based on context and content information," in *26th European Conference on Networks and Communications (EuCNC)*, pp. 1–6, IEEE, June 2017.

[19] J. Ma and J. C. Cheng, "Estimation of the building energy use intensity in the urban scale by integrating GIS and big data technology," *Applied Energy*, vol. 183, pp. 182–192, Dec. 2016.

[20] F. Herrema, V. Treve, R. Curran, and H. Visser, "Evaluation of feasible machine learning techniques for predicting the time to fly and aircraft speed profile on final approach," in *7th International Conference for Research in Air Transportation*, pp. 1–8, June 2016.

[21] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, and F. L. Lewis, "Optimal and autonomous control using reinforcement learning: A survey," *IEEE transactions on neural networks and learning systems*, vol. 29, pp. 2042–2062, June 2018.

[22] M. A. Fischler and O. Firschein, *Intelligence: the eye, the brain, and the computer.* Addison-Wesley, Jan. 1987.

[23] A. Clark, "Expecting the world: Perception, prediction, and the origins of human knowledge," *The Journal of Philosophy*, vol. 110, pp. 469–496, Sep. 2013.

[24] N. Bui, M. Cesana, S. A. Hosseini, Q. Liao, I. Malanchini, and J. Widmer, "A survey of anticipatory mobile networking: Context-based classification, prediction methodologies, and optimization techniques," *IEEE Communications Surveys & Tutorials*, vol. 19, pp. 1790–1821, July 2017.

[25] L. Guang, J. Plosila, and H. Tenhunen, "From self-aware building blocks to self-organizing systems with hierarchical agent-based adaptation," in *International Conference on Hardware/Software Codesign and System Synthesis*, p. 23, ACM, Oct. 2014.

[26] P. S. Churchland and T. J. Sejnowski, *The computational brain.* MIT press, 2016.

[27] D. Geiger and D. Heckerman, “A characterization of the Dirichlet distribution with application to Learning Bayesian Networks,” in *11th Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 196–207, ACM, Aug. 1995.

[28] F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller, and B. V. Saunders, “Beta function,” in *NIST Digital Library of Mathematical Functions*.

[29] V. N. Vapnik, *The Nature of Statistical Learning Theory*. Springer Science & Business Media, June 2013.

[30] H. Drucker, C. J. C. Burges, L. Kaufman, A. J. Smola, and V. Vapnik, “Support Vector Regression Machines,” *Advances in Neural Information Processing Systems*, pp. 155–161, Sep. 1997.

[31] A. J. Smola and B. Schölkopf, “A tutorial on Support Vector Regression,” *Statistics and computing*, vol. 14, pp. 199–222, Aug. 2004.

[32] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine learning*, vol. 20, pp. 273–297, Sep. 1995.

[33] W. Karush, “Minima of functions of several variables with inequalities as side constraints,” Master’s thesis, Dept. of Mathematics, Univ. of Chicago, 1939.

[34] H. W. Kuhn and A. W. Tucker, “Nonlinear programming,” in *Traces and Emergence of Nonlinear Programming*, pp. 247–258, Springer Basel, Nov. 2013.

[35] D. F. Andrews, “A robust method for multiple linear regression,” *Technometrics*, vol. 16, pp. 523–531, Nov. 1974.

[36] S. Weisberg, *Applied linear regression*. John Wiley & Sons, 2005.

[37] A. E. Hoerl and R. W. Kennard, “Ridge regression: Biased estimation for nonorthogonal problems,” *Technometrics*, vol. 12, pp. 55–67, Feb. 1970.

[38] R. Tibshirani, “Regression shrinkage and selection via the lasso,” *Journal of the Royal Statistical Society, Series B (Methodological)*, vol. 58, pp. 267–288, Jan. 1996.

[39] H. Zou and T. Hastie, “Regularization and variable selection via the elastic net,” *Journal of the Royal Statistical Society, Series B (Methodological)*, vol. 67, pp. 301–320, Apr. 2005.

[40] T. K. Ho, “The random subspace method for constructing decision forests,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, pp. 832–844, Aug. 1998.

[41] W. N. Venables and B. D. Ripley, “Tree-based methods,” in *Modern Applied Statistics with S*, pp. 251–269, Springer, Mar. 2002.

[42] N. S. Altman, “An introduction to kernel and nearest-neighbor nonparametric regression,” *The American Statistician*, vol. 46, pp. 175–185, Aug. 1992.

[43] Y. Lin and Y. Jeon, “Random forests and adaptive nearest neighbors,” *Journal of the American Statistical Association*, vol. 101, pp. 578–590, June 2006.

[44] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain.,” *Psychological review*, vol. 65, p. 386, Nov. 1958.

[45] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, p. 533, Oct. 1986.

[46] B. Widrow and M. A. Lehr, "30 years of adaptive neural networks: perceptron, madaline, and backpropagation," *Proceedings of the IEEE*, vol. 78, pp. 1415–1442, Sep. 1990.

[47] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, Mar. 2015.

[48] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015.

[49] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, Nov. 2016.

[50] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with LSTM recurrent networks," *Journal of Machine Learning Research*, vol. 3, pp. 115–143, Aug. 2002.

[51] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, pp. 1550–1560, Oct. 1990.

[52] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, pp. 35–45, Mar. 1960.

[53] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, pp. 257–286, Feb. 1989.

[54] R. Mehra, "On the identification of variances and adaptive Kalman filtering," *IEEE Transactions on Automatic Control*, vol. 15, pp. 175–184, Apr. 1970.

[55] M. R. Rajamani and J. B. Rawlings, "Estimation of the disturbance structure from data using semidefinite programming and optimal weighting," *Automatica*, vol. 45, pp. 142–148, Jan. 2009.

[56] E. A. Wan and R. Van Der Merwe, "The unscented kalman filter," *Kalman filtering and neural networks*, pp. 221–280, Oct. 2001.

[57] C. J. C. H. Watkins, *Learning from delayed rewards.* PhD thesis, King's College, Cambridge, 1989.

[58] G. Tesauro, "Temporal difference learning and td-gammon," *Communications of the ACM*, vol. 38, pp. 58–68, Mar. 1995.

[59] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu, "Deep blue," *Artificial intelligence*, vol. 134, pp. 57–83, Jan. 2002.

[60] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[61] J. X. Chen, "The evolution of computing: AlphaGo," *Computing in Science & Engineering*, vol. 18, pp. 4–7, July 2016.

[62] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, Dec. 2017.

[63] I. Bratko, “AlphaZero–what’s missing?,” *Informatica*, vol. 42, Mar. 2018.

[64] J. Boyan and M. Littman, “A distributed reinforcement learning scheme for network routing,” in *International Workshop on Applications of Neural Networks to Telecommunications*, pp. 55–61, Psychology Press, June 2013.

[65] R. Bellman, “A Markovian decision process,” *Indiana University Mathematics Journal*, vol. 6, pp. 679–684, Jan. 1957.

[66] C. J. Watkins and P. Dayan, “Q-learning,” *Machine Learning*, vol. 8, pp. 279–292, May 1992.

[67] M. Kearns and S. Singh, “Finite-sample convergence rates for Q-learning and indirect algorithms,” *Advances in Neural Information Processing Systems*, vol. 11, pp. 996–1002, Apr. 1999.

[68] L.-J. Lin, “Reinforcement learning for robots using neural networks,” tech. rep., DTIC Document, 1993.

[69] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *International Conference on Learning Representations (ICLR)*, vol. 5, pp. 1–15, May 2015.

[70] P. K. Agyapong, M. Iwamura, D. Staehle, W. Kiess, and A. Benjebbour, “Design considerations for a 5G network architecture,” *IEEE Communications Magazine*, vol. 52, pp. 65–75, Nov. 2014.

[71] S. Mekki and S. Valentin, “Anticipatory quality adaptation for mobile streaming: Fluent video by channel prediction,” in *16th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pp. 1–3, IEEE, June 2015.

[72] G. P. Perrucci, F. H. P. Fitzek, and J. Widmer, “Survey on energy consumption entities on the smartphone platform,” in *IEEE 73rd Vehicular Technology Conference (VTC Spring)*, pp. 1–6, May 2011.

[73] X. Zhao, Y. Guo, Q. Feng, and X. Chen, “A system context-aware approach for battery lifetime prediction in smart phones,” in *ACM Symposium on Applied Computing (SAC)*, pp. 641–646, Mar. 2011.

[74] S. Zhou and G. B. Giannakis, “How accurate channel prediction needs to be for transmit-beamforming with adaptive modulation over Rayleigh MIMO channels?,” *IEEE Transactions on Wireless Communications*, vol. 3, pp. 1285–1294, July 2004.

[75] Q. Liao, S. Valentin, and S. Stanczak, “Channel gain prediction in wireless networks based on spatial-temporal correlation,” in *16th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pp. 400–404, IEEE, June 2015.

[76] F. Chiariotti, D. Del Testa, M. Polese, A. Zanella, G. M. Di Nunzio, and M. Zorzi, “Learning methods for long-term channel gain prediction in wireless networks,” in *International Conference on Computing, Networking and Communications (ICNC)*, pp. 162–166, IEEE, Jan. 2017.

[77] L. Dong, G. Xu, and H. Ling, “Prediction of fast fading mobile radio channels in wideband communication systems,” in *Global Telecommunications Conference (GLOBECOM)*, vol. 6, pp. 3287–3291, IEEE, Nov. 2001.

[78] Z. Shen, J. G. Andrews, and B. L. Evans, "Short range wireless channel prediction using local information," in *37th Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 1147–1151, IEEE, Nov. 2003.

[79] I. C. Wong, A. Forenza, R. W. Heath, and B. L. Evans, "Long range channel prediction for adaptive OFDM systems," in *38th Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 732–736, IEEE, Nov. 2004.

[80] D. Jarinová, "On autoregressive model order for long-range prediction of fast fading wireless channel," *Telecommunication Systems*, vol. 52, pp. 1533–1539, Mar. 2013.

[81] S. Ramanan and J. M. Walsh, "Distributed estimation of channel gains in Wireless Sensor Networks," *IEEE Transactions on Signal Processing*, vol. 58, pp. 3097–3107, June 2010.

[82] P. Demestichas, A. Katidiotis, K. A. Tsagkaris, E. F. Adamopoulou, and K. P. Demestichas, "Enhancing channel estimation in Cognitive Radio systems by means of Bayesian networks," *Wireless personal communications*, vol. 49, pp. 87–105, Apr. 2009.

[83] E. F. Flushing, J. Nagi, and G. A. Di Caro, "A mobility-assisted protocol for supervised learning of link quality estimates in wireless networks," in *International Conference on Computing, Networking and Communications (ICNC)*, pp. 137–143, IEEE, Jan. 2012.

[84] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and Tools for Network Simulation*, pp. 15–34, Springer, June 2010.

[85] N. Baldo, M. Miozzo, M. Requena-Esteso, and J. Nin-Guerrero, "An open source product-oriented LTE network simulator based on ns-3," in *14th International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pp. 293–298, ACM, Oct. 2011.

[86] 3GPP Technical Specification 36.104, "Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) radio transmission and reception," *LTE ETSI TS*, Mar. 2018.

[87] J. Chon and H. Cha, "LifeMap: A smartphone-based context provider for location-based services," *IEEE Pervasive Computing*, vol. 10, pp. 58–67, Apr. 2011.

[88] M. Gentil, A. Galeazzi, F. Chiariotti, M. Polese, A. Zanella, and M. Zorzi, "A deep neural network approach for customized prediction of mobile devices discharging time," in *Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, Dec. 2017.

[89] O. Gérard, J.-N. Patillon, and F. D'Alché-Buc, "Discharge prediction of rechargeable batteries with neural networks," *Integrated Computer-Aided Engineering*, vol. 6, pp. 41–52, Jan. 1999.

[90] Y. Wen, R. Wolski, and C. Krintz, "Online prediction of battery lifetime for embedded and mobile devices," in *International Workshop on Power-Aware Computer Systems*, pp. 57–72, Springer, Dec. 2003.

[91] E. A. Oliver and S. Keshav, "An empirical approach to smartphone energy level prediction," in *13th International Conference on Ubiquitous Computing (UbiComp)*, pp. 345–354, Sep. 2011.

[92] D. N. Rakhmatov and S. B. Vrudhula, “An analytical high-level battery model for use in energy management of portable electronic systems,” in *IEEE/ACM international conference on Computer-aided design*, pp. 488–493, IEEE, Nov. 2001.

[93] S. B. Kotsiantis, D. Kanellopoulos, and P. E. Pintelas, “Data preprocessing for supervised leaning,” *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, vol. 1, pp. 4091 – 4096, Dec. 2007.

[94] R. Anderson-Sprecher, “Model comparisons and R^2,” *The American Statistician*, vol. 48, pp. 113–117, Apr. 1994.

[95] G. Andrew and J. Gao, “Scalable training of L1-regularized log-linear models,” in *24th International Conference on Machine Learning*, pp. 33–40, ACM, June 2007.

[96] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, Oct. 2011.

[97] Y. Chon, H. Shin, E. Talipov, and H. Cha, “Evaluating mobility models for temporal prediction with high-granularity mobility data,” in *International Conference on Pervasive Computing and Communications*, pp. 206–212, IEEE, Mar. 2012.

[98] A. Acquisti, L. Brandimarte, and G. Loewenstein, “Privacy and human behavior in the age of information,” *Science*, vol. 347, pp. 509–514, January 2015.

[99] H. Almuhimedi, F. Schaub, N. Sadeh, I. Adjerid, A. Acquisti, J. Gluck, L. F. Cranor, and Y. Agarwal, “Your location has been shared 5,398 times!: A field study on mobile app privacy nudging,” in *33rd Annual ACM Conference on Human Factors in Computing Systems*, pp. 787–796, ACM, Apr. 2015.

[100] E. Lovisotto, E. Vianello, D. Cazzaro, M. Polese, F. Chiariotti, D. Zucchetto, A. Zanella, and M. Zorzi, “Cell traffic prediction using joint spatio-temporal information,” in *6th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pp. 1–4, IEEE, Apr. 2017.

[101] R. Li, Z. Zhao, X. Zhou, and H. Zhang, “Energy savings scheme in Radio Access Networks via compressive sensing-based traffic load prediction,” *Transactions on Emerging Telecommunications Technologies*, vol. 25, pp. 468–478, Nov. 2012.

[102] S. E. Hammami, H. Afifi, M. Marot, and V. Gauthier, “Network planning tool based on network classification and load prediction,” in *Wireless Communications and Networking Conference (WCNC)*, pp. 1–6, IEEE, Apr. 2016.

[103] U. Paul, A. P. Subramanian, M. M. Buddhikot, and S. R. Das, “Understanding traffic dynamics in cellular data networks,” in *30th International Conference on Computer Communications (INFOCOM)*, pp. 882–890, IEEE, Apr. 2011.

[104] R. Li, Z. Zhao, X. Chen, J. Palicot, and H. Zhang, “TACT: a transfer actor-critic learning framework for energy saving in cellular Radio Access Networks,” *IEEE Transactions on Wireless Communications*, vol. 13, pp. 2000–2011, Apr. 2014.

[105] S. Scellato, M. Musolesi, C. Mascolo, V. Latora, and A. T. Campbell, “NextPlace: a spatio-temporal prediction framework for pervasive systems,” in *International Conference on Pervasive Computing*, pp. 152–169, Springer, May 2011.

[106] H. Gao, J. Tang, and H. Liu, "Mobile location prediction in spatio-temporal context," in *Nokia Mobile Data Challenge Workshop*, vol. 41, pp. 1–4, June 2012.

[107] W.-S. Soh and H. S. Kim, "QoS provisioning in cellular networks based on mobility prediction techniques," *IEEE Communications Magazine*, vol. 41, pp. 86–92, Jan. 2003.

[108] O. Ohashi and L. Torgo, "Wind speed forecasting using spatio-temporal indicators," in *20th European Conference on Artificial Intelligence (ECAI)*, pp. 975–980, IOS Press, Aug. 2012.

[109] N. R. Draper and H. Smith, *Applied regression analysis*. John Wiley & Sons, May 1998.

[110] M. Simsek, A. Aijaz, M. Dohler, J. Sachs, and G. Fettweis, "5G-enabled tactile internet," *IEEE Journal on Selected Areas in Communications*, vol. 34, pp. 460–473, Mar. 2016.

[111] M. Dohler, T. Mahmoodi, M. A. Lema, M. Condoluci, F. Sardis, K. Antonakoglou, and H. Aghvami, "Internet of Skills, where Robotics meets AI, 5G and the Tactile Internet," in *26th European Conference on Networks and Communications (EuCNC)*, pp. 1–5, IEEE, June 2017.

[112] C. J. Hansen, "WiGiG: Multi-gigabit wireless communications in the 60 GHz band," *IEEE Wireless Communications*, vol. 18, pp. 6–7, Dec. 2011.

[113] C. Yu, Y. Xu, B. Liu, and Y. Liu, ""Can you SEE me now?" A measurement study of mobile video calls," in *International Conference on Computer Communications (INFOCOM)*, pp. 1456–1464, IEEE, Apr. 2014.

[114] J. Wu, B. Cheng, C. Yuen, N.-M. Cheung, and J. Chen, "Trading delay for distortion in one-way video communication over the internet," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, pp. 711–723, Apr. 2016.

[115] A. A. Khalek, C. Caramanis, and R. W. Heath, "Delay-constrained video transmission: Quality-driven resource allocation and scheduling," *IEEE Journal of Selected Topics in Signal Processing*, vol. 9, pp. 60–75, Feb. 2015.

[116] M. Li, K. Arning, L. Vervier, M. Ziefle, and L. Kobbelt, "Influence of temporal delay and display update rate in an augmented reality application scenario," in *14th International Conference on Mobile and Ubiquitous Multimedia*, pp. 278–286, ACM, Nov. 2015.

[117] N. Narendra, P. K. Reddy, K. Kumar, A. Varghese, P. Swamy, G. Chandra, and P. Balamuralidhar, "Mobicostream: Real-time collaborative video upstream for mobile augmented reality applications," in *International Conference on Advanced Networks and Telecommuncations Systems (ANTS)*, pp. 1–6, IEEE, Dec. 2014.

[118] S. Shailendra, K. Aniruddh, B. Panigrahi, and A. Simha, "Multipath TCP path scheduler for drones: A segregation of control and user data," in *18th International Symposium on Mobile Ad Hoc Networking and Computing*, p. 40, ACM, July 2017.

[119] P. Millan, L. Orihuela, I. Jurado, and F. R. Rubio, "Formation control of autonomous underwater vehicles subject to communication delays," *IEEE Transactions on Control Systems Technology*, vol. 22, pp. 770–777, Mar. 2014.

[120] P. Fernandes and U. Nunes, “Platooning with IVC-enabled autonomous vehicles: Strategies to mitigate communication delays, improve safety and traffic flow,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, pp. 91–106, Mar. 2012.

[121] M. di Bernardo, A. Salvi, and S. Santini, “Distributed consensus strategy for platooning of vehicles in the presence of time-varying heterogeneous communication delays,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, pp. 102–112, Feb. 2015.

[122] C. Liang and F. R. Yu, “Wireless network virtualization: A survey, some research issues and challenges,” *IEEE Communications Surveys Tutorials*, vol. 17, pp. 358–380, Mar. 2015.

[123] K. Fahmi and S. Kucera, “Multiple path transmission of data,” Sep. 2016. EU Patent EP16306173.2.

[124] M. Li, A. Lukyanenko, S. Tarkoma, Y. Cui, and A. Ylä-Jääski, “Tolerating path heterogeneity in multipath TCP with bounded receive buffers,” *Computer Networks*, vol. 64, pp. 1–14, May 2014.

[125] Y. Cui, L. Wang, X. Wang, H. Wang, and Y. Wang, “FMTCP: A fountain code-based Multipath Transmission Control Protocol,” *IEEE/ACM Transactions on Networking*, vol. 23, pp. 465–478, Apr. 2015.

[126] D. Jurca, P. Frossard, and A. Jovanovic, “Forward error correction for multipath media streaming,” *IEEE Transactions on circuits and systems for video technology*, vol. 19, pp. 1315–1326, Sep. 2009.

[127] S. Fouladi, J. Emmons, E. Orbay, C. Wu, R. S. Wahby, and K. Winstein, “Salsify: Low-latency network video through tighter integration between a video codec and a transport protocol,” in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, USENIX Association, 2018.

[128] F. Chiariotti, S. Kucera, A. Zanella, and H. Claussen, “LEAP: A latency control protocol for multi-path data delivery with pre-defined QoS guarantees,” in *International Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 166–171, IEEE, Apr. 2018.

[129] F. Chiariotti, S. Kucera, and A. Zanella, “Method for explicit Quality-of-Service support in 5G networks,” Dec. 2017. EU Patent request PCT/EP2017/082279.

[130] A. Veres and M. Boda, “The chaotic nature of TCP congestion control,” in *19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, vol. 3, pp. 1715–1723, IEEE, Mar. 2000.

[131] S. Ha, I. Rhee, and L. Xu, “CUBIC: a new TCP-friendly high-speed TCP variant,” *ACM SIGOPS Operating Systems Review*, vol. 42, pp. 64–74, July 2008.

[132] J. Gettys and K. Nichols, “Bufferbloat: Dark buffers in the internet,” *Queue*, vol. 9, p. 40, Nov. 2011.

[133] G. Raina and D. Wischik, “Buffer sizes for large multiplexers: TCP queueing theory and instability analysis,” in *Next Generation Internet Networks*, pp. 173–180, IEEE, Apr. 2005.

[134] Y. Gong, D. Rossi, C. Testa, S. Valenti, and M. D. Täht, "Fighting the bufferbloat: on the coexistence of AQM and low priority congestion control," *Computer Networks*, vol. 65, pp. 255–267, June 2014.

[135] N. Khademi, D. Ros, and M. Welzl, "The new AQM kids on the block: An experimental evaluation of CoDel and PIE," in *International Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 85–90, IEEE, Apr. 2014.

[136] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, pp. 1465–1480, Oct. 1995.

[137] J. Mo, R. J. La, V. Anantharam, and J. Walrand, "Analysis and comparison of TCP Reno and Vegas," in *18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, vol. 3, pp. 1556–1563, IEEE, Mar. 1999.

[138] Y. Lei, R. Zhu, and W. Wang, "A survey on tcp protocol and rtt estimation," in *6th World Congress on Intelligent Control and Automation*, vol. 1, pp. 4410–4414, June 2006.

[139] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Queue*, vol. 14, p. 50, Oct. 2016.

[140] K. Winstein, A. Sivaraman, and H. Balakrishnan, "Stochastic forecasts achieve high throughput and low delay over cellular networks," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 459–471, Apr. 2013.

[141] F. Guidolin, I. Pappalardo, A. Zanella, and M. Zorzi, "A Markov-based framework for handover optimization in HetNets," in *13th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, pp. 134–139, IEEE, 2014.

[142] Y. Zaki, T. Pötsch, J. Chen, L. Subramanian, and C. Görg, "Adaptive congestion control for unpredictable cellular networks," in *ACM SIGCOMM Computer Communication Review*, vol. 45, pp. 509–522, ACM, Aug. 2015.

[143] U. Tos and T. Ayav, "Adaptive RTP rate control method," in *35th Annual Computer Software and Applications Conference Workshops (COMPSACW)*, pp. 7–12, IEEE, July 2011.

[144] V. Gokhale, J. Nair, and S. Chaudhuri, "Congestion control for network-aware telehaptic communication," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 13, p. 17, May 2017.

[145] S. Ferlin, T. Dreibholz, and Ö. Alay, "Multi-path transport over heterogeneous wireless networks: Does it really pay off?," in *IEEE Global Communications Conference (GLOBECOM)*, pp. 4807–4813, IEEE, 2014.

[146] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP extensions for multipath operation with multiple addresses," *IETF RFC 6824*, Jan. 2013.

[147] A. Walid, J. Hwang, Q. Peng, and S. Low, "Balia (Balanced Linked Adaptation)–A New MPTCP Congestion Control Algorithm," in *90th IETF Meeting*, July 2014.

[148] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant, and D. Towsley, "Multi-path TCP: a joint congestion control and routing scheme to exploit path diversity in the internet," *IEEE/ACM Transactions on Networking*, vol. 14, pp. 1260–1271, Dec. 2006.

[149] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for Multipath TCP," in *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, vol. 11, pp. 8–8, Mar. 2011.

[150] Y.-C. Chen, Y.-s. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, and D. Towsley, "A measurement-based study of Multipath TCP performance over wireless networks," in *ACM Conference on Internet Measurement*, pp. 455–468, ACM, Oct. 2013.

[151] A. Garcia-Saavedra, M. Karzand, and D. J. Leith, "Low delay random linear coding and scheduling over multiple interfaces," *IEEE Transactions on Mobile Computing*, Nov. 2017.

[152] C. Xu, J. Zhao, and G.-M. Muntean, "Congestion control design for multipath transport protocols: a survey," *IEEE Communications Surveys & Tutorials*, vol. 18, pp. 2948–2969, Apr. 2016.

[153] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, *et al.*, "The QUIC transport protocol: Design and Internet-scale deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 183–196, ACM, Aug. 2017.

[154] A. A. Abouzeid, S. Roy, and M. Azizoglu, "Stochastic modeling of TCP over lossy links," in *19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, vol. 3, pp. 1724–1733, IEEE, Mar. 2000.

[155] J. L. Folks and R. S. Chhikara, "The inverse Gaussian distribution and its statistical application–a review," *Journal of the Royal Statistical Society, Series B (Methodological)*, pp. 263–289, Jan. 1978.

[156] W. C. Horrace, "Some results on the multivariate truncated normal distribution," *Journal of Multivariate Analysis*, vol. 94, pp. 209–221, May 2005.

[157] V. Francis, "On the distribution of the sum of n sample values drawn from a truncated normal population," *Supplement to the Journal of the Royal Statistical Society*, vol. 8, pp. 223–232, Jan. 1946.

[158] S. Hemminger, "Network emulation with NetEm," in *Linux Conf*, pp. 18–23, Apr. 2005.

[159] H. Shi, Y. Cui, X. Wang, Y. Hu, M. Dai, F. Wang, and K. Zheng, "STMS: Improving MPTCP throughput under heterogeneous networks," in *USENIX Annual Technical Conference (ATC)*, USENIX Association, 2018.

[160] ISO/IEC 23009-1:2014, "Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats," *International Organization for Standardization*, May 2014.

[161] R. K. Mok, E. W. Chan, and R. K. Chang, "Measuring the quality of experience of HTTP video streaming," in *12th IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 485–492, IEEE, May 2011.

[162] T. Hoßfeld, M. Seufert, C. Sieber, and T. Zinner, "Assessing effect sizes of influence factors towards a QoE model for HTTP adaptive streaming," in *6th International Workshop on Quality of Multimedia Experience (QoMEX)*, pp. 111–116, IEEE, Sep. 2014.

[163] T. Hoßfeld, S. Egger, R. Schatz, M. Fiedler, K. Masuch, and C. Lorentzen, "Initial delay vs. interruptions: between the devil and the deep blue sea," in *4th International Workshop on Quality of Multimedia Experience (QoMEX)*, pp. 1–6, IEEE, Jul. 2012.

[164] Z. Li, A. C. Begen, J. Gahm, Y. Shan, B. Osler, and D. Oran, "Streaming video over HTTP with consistent quality," in *5th ACM Multimedia Systems Conference (MMSys)*, pp. 248–258, ACM, Mar. 2014.

[165] M. Gadaleta, F. Chiariotti, M. Rossi, and A. Zanella, "D-DASH: A deep Q-learning framework for DASH video streaming," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, pp. 703–718, Dec. 2017.

[166] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang, "Developing a predictive model of quality of experience for internet video," *SIGCOMM Computer Communication Review (CCR)*, vol. 43, pp. 339–350, Aug. 2013.

[167] P. Juluri, V. Tamarapalli, and D. Medhi, "Measurement of quality of experience of video-on-demand services: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, pp. 401–418, Jan. 2016.

[168] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia, "A survey on quality of experience of HTTP Adaptive Streaming," *IEEE Communications Surveys & Tutorials*, vol. 17, pp. 469–492, Mar. 2015.

[169] M. Shahid, A. Rossholm, B. Lövström, and H.-J. Zepernick, "No-reference image and video quality assessment: a classification and review of recent approaches," *EURASIP Journal on Image and Video Processing*, vol. 2014, pp. 1–32, Dec. 2014.

[170] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, pp. 600–612, Apr. 2004.

[171] C. Kreuzberger, B. Rainer, H. Hellwagner, L. Toni, and P. Frossard, "A comparative study of DASH representation sets using real user characteristics," in *26th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, p. 4, ACM, May 2016.

[172] S. Cicalo, N. Changuel, R. Miller, B. Sayadi, and V. Tralli, "Quality-fair HTTP adaptive streaming over LTE network," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 714–718, IEEE, May 2014.

[173] J. De Vriendt, D. De Vleeschauwer, and D. Robinson, "Model for estimating QoE of video delivered using HTTP adaptive streaming," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 1288–1293, IEEE, May 2013.

[174] J. Kua, G. Armitage, and P. Branch, "A survey of rate adaptation techniques for Dynamic Adaptive Streaming over HTTP," *IEEE Communications Surveys & Tutorials*, Mar. 2017.

[175] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with FESTIVE," in *8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pp. 97–108, ACM, Dec. 2012.

[176] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran, "Probe and adapt: Rate adaptation for HTTP video streaming at scale," *IEEE Journal on Selected Areas in Communications*, vol. 32, pp. 719–733, Apr. 2014.

[177] S. Petrangeli, N. Bouten, E. Dejonghe, J. Famaey, P. Leroux, and F. De Turck, "Design and evaluation of a dash-compliant second screen video player for live events in mobile scenarios," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pp. 894–897, IEEE, May 2015.

[178] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over HTTP," *ACM SIGCOMM Computer Communication Review*, vol. 45, pp. 325–338, Aug. 2015.

[179] A. Bokani, M. Hassan, and S. Kanhere, "HTTP-based adaptive streaming for mobile clients using Markov Decision Process," in *20th International Packet Video Workshop*, pp. 1–8, IEEE, Dec. 2013.

[180] C. Zhou, C.-W. Lin, and Z. Guo, "mDASH: A Markov Decision-based rate adaptation approach for dynamic HTTP streaming," *IEEE Transactions on Multimedia*, vol. 18, pp. 738–751, Apr. 2016.

[181] J. Lee and S. Bahk, "On the MDP-based cost minimization for video-on-demand services in a heterogeneous wireless network with multihomed terminals," *IEEE Transactions on Mobile Computing*, vol. 12, pp. 1737–1749, Sep. 2013.

[182] S. Colonnese, F. Cuomo, T. Melodia, and R. Guida, "Cloud-assisted buffer management for HTTP-based mobile video streaming," in *10th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks*, pp. 1–8, ACM, Nov. 2013.

[183] M. Claeys, S. Latré, J. Famaey, T. Wu, W. Van Leekwijck, and F. De Turck, "Design of a Q-learning-based client quality selection algorithm for HTTP adaptive video streaming," in *Adaptive and Learning Agents Workshop (ALA)*, pp. 30–37, May 2013.

[184] M. Claeys, S. Latré, J. Famaey, T. Wu, W. Van Leekwijck, and F. De Turck, "Design and optimisation of a (FA) Q-learning-based HTTP adaptive streaming client," *Connection Science*, vol. 26, pp. 25–43, Jan. 2014.

[185] V. Martín, J. Cabrera, and N. García, "Q-learning based control algorithm for HTTP adaptive streaming," in *International Conference on Visual Communications and Image Processing (VCIP)*, pp. 1–4, IEEE, Dec. 2015.

[186] J. van der Hooft, S. Petrangeli, M. Claeys, J. Famaey, and F. De Turck, "A learning-based algorithm for improved bandwidth-awareness of adaptive streaming clients," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 131–138, IEEE, May 2015.

[187] F. Chiariotti, S. D'Aronco, L. Toni, and P. Frossard, "Online learning adaptation strategy for DASH clients," in *7th International Conference on Multimedia Systems (MMSys)*, pp. 8:1–8:12, ACM, May 2016.

[188] O. Rose, "Statistical properties of MPEG video traffic and their impact on traffic modeling in ATM systems," in *20th Conference on Local Computer Networks*, pp. 397–406, IEEE, Oct. 1995.

[189] S. Chikkerur, V. Sundaram, M. Reisslein, and L. J. Karam, “Objective video quality assessment methods: A classification, review, and performance comparison,” *IEEE transactions on broadcasting*, vol. 57, pp. 165–182, June 2011.

[190] H. R. Sheikh, M. F. Sabir, and A. C. Bovik, “A statistical evaluation of recent full reference image quality assessment algorithms,” *IEEE Transactions on image processing*, vol. 15, pp. 3440–3451, Nov. 2006.

[191] M. Zorzi, A. Zanella, A. Testolin, M. D. F. De Grazia, and M. Zorzi, “Cognition-based networks: a new perspective on network optimization using learning and distributed intelligence,” *IEEE Access*, vol. 3, pp. 1512–1530, Aug. 2015.

[192] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, pp. 1735–1780, Nov. 1997.

[193] X. Li and X. Wu, “Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition,” in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4520–4524, IEEE, Apr. 2015.

[194] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting.,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, Jan. 2014.

[195] P. Juluri, V. Tamarapalli, and D. Medhi, “QoE management in DASH systems using the segment aware rate adaptation algorithm,” in *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, pp. 129–136, IEEE, Apr. 2016.

[196] A. Testolin, M. Zanforlin, M. D. F. D. Grazia, D. Munaretto, A. Zanella, M. Zorzi, and M. Zorzi, “A machine learning approach to QoE-based video admission control and resource allocation in wireless systems,” in *13th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, pp. 31–38, June 2014.

[197] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. De Turck, “HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks,” *IEEE Communications Letters*, vol. 20, pp. 2177–2180, Nov. 2016.

[198] H. X. Nguyen, P. Thiran, and C. Barakat, “On the correlation of TCP traffic in backbone networks,” in *IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 5, pp. V–481, IEEE, May 2004.

[199] M. Batty, K. W. Axhausen, F. Giannotti, A. Pozdnoukhov, A. Bazzani, M. Wachowicz, G. Ouzounis, and Y. Portugali, “Smart Cities of the future,” *The European Physical Journal Special Topics*, vol. 214, pp. 481–518, Nov. 2012.

[200] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Sensing as a service model for Smart Cities supported by Internet of Things,” *Transactions on Emerging Telecommunications Technologies*, vol. 25, pp. 81–93, Jan. 2014.

[201] P. Neirotti, A. De Marco, A. C. Cagliano, G. Mangano, and F. Scorrano, “Current trends in Smart City initiatives: Some stylised facts,” *Cities*, vol. 38, pp. 25–36, June 2014.

[202] L. Filipponi, A. Vitaletti, G. Landi, V. Memeo, G. Laura, and P. Pucci, “Smart City: An event driven architecture for monitoring public spaces with heterogeneous sensors,” in *4th International Conference on Sensor Technologies and Applications (SENSORCOMM)*, pp. 281–286, IEEE, July 2010.

[203] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of Things for Smart Cities," *IEEE Internet of Things Journal*, vol. 1, pp. 22–32, Feb. 2014.

[204] F. Xia and J. Ma, "Building smart communities with cyber-physical systems," in *Proceedings of 1st international symposium on From digital footprints to social and community intelligence*, pp. 1–6, ACM, Sep. 2011.

[205] E. Fishman, S. Washington, and N. Haworth, "Bike share's impact on car use: evidence from the United States, Great Britain, and Australia," *Transportation Research Part D: Transport and Environment*, vol. 31, pp. 13–20, Aug. 2014.

[206] D. Rojas-Rueda, A. de Nazelle, M. Tainio, and M. J. Nieuwenhuijsen, "The health risks and benefits of cycling in urban environments compared with car use: health impact assessment study," *British Medical Journal*, vol. 343, p. d4521, Aug. 2011.

[207] S. Shaheen, S. Guzman, and H. Zhang, "Bikesharing in Europe, the Americas, and Asia: past, present, and future," *Transportation Research Record: Journal of the Transportation Research Board*, pp. 159–167, Oct. 2010.

[208] P. Midgley, "The role of smart bike-sharing systems in urban mobility," *Journeys*, vol. 2, pp. 23–31, May 2009.

[209] M. Kaspi, T. Raviv, and M. Tzur, "Detection of unusable bicycles in bike-sharing systems," *Omega*, vol. 65, pp. 10–16, 2016.

[210] J. Bao, T. He, S. Ruan, Y. Li, and Y. Zheng, "Planning bike lanes based on sharing-bikes' trajectories," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1377–1386, ACM, 2017.

[211] F. Chiariotti, C. Pielli, A. Zanella, and M. Zorzi, "A dynamic approach to rebalancing bike-sharing systems," *Sensors*, vol. 18, p. 512, Feb. 2018.

[212] R. Kazhamiakin, A. Marconi, M. Perillo, M. Pistore, G. Valetto, L. Piras, F. Avesani, and N. Perri, "Using gamification to incentivize sustainable urban mobility," in *1st International Smart Cities Conference (ISC2)*, pp. 1–6, IEEE, Oct. 2015.

[213] J. Hamari, J. Koivisto, and H. Sarsa, "Does gamification work?–a literature review of empirical studies on gamification," in *47th Hawaii International Conference on System Sciences (HICSS)*, pp. 3025–3034, IEEE, Jan. 2014.

[214] F. Chiariotti, C. Pielli, A. Cenedese, A. Zanella, and M. Zorzi, "Bike sharing as a key smart city service: State of the art and future developments," in *7th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pp. 1–6, IEEE, May 2018.

[215] J. Schuijbroek, R. C. Hampshire, and W.-J. Van Hoeve, "Inventory rebalancing and vehicle routing in bike sharing systems," *European Journal of Operational Research*, vol. 257, pp. 992–1004, Mar. 2017.

[216] L. Caggiani and M. Ottomanelli, "A dynamic simulation based model for optimal fleet repositioning in bike-sharing systems," *Procedia-Social and Behavioral Sciences*, vol. 87, pp. 203–210, Oct. 2013.

[217] P. Vogel, T. Greiser, and D. C. Mattfeld, "Understanding bike-sharing systems using data mining: Exploring activity patterns," *Procedia-Social and Behavioral Sciences*, vol. 20, pp. 514–523, Jan. 2011.

[218] A. Faghih-Imani and N. Eluru, "Incorporating the impact of spatio-temporal interactions on bicycle sharing system demand: A case study of new york citibike system," *Journal of Transport Geography*, vol. 54, pp. 218–227, June 2016.

[219] G. Laporte, F. Meunier, and R. W. Calvo, "Shared mobility systems," *4OR*, vol. 13, pp. 341–360, Dec. 2015.

[220] H. M. Espegren, J. Kristianslund, H. Andersson, and K. Fagerholt, "The static bicycle repositioning problem-literature survey and new formulation," in *International Conference on Computational Logistics*, pp. 337–351, Springer, Sep. 2016.

[221] D. Chemla, F. Meunier, and R. W. Calvo, "Bike sharing systems: Solving the static rebalancing problem," *Discrete Optimization*, vol. 10, pp. 120–146, May 2013.

[222] P. Toth and D. Vigo, "Exact solution of the vehicle routing problem," in *Fleet management and logistics*, pp. 1–31, Springer, Dec. 2012.

[223] S. C. Ho and W. Szeto, "Solving a static repositioning problem in bike-sharing systems using iterated tabu search," *Transportation Research Part E: Logistics and Transportation Review*, vol. 69, pp. 180–198, Sep. 2014.

[224] T. Raviv, M. Tzur, and I. A. Forma, "Static repositioning in a bike-sharing system: models and solution approaches," *EURO Journal on Transportation and Logistics*, vol. 2, pp. 187–229, Aug. 2013.

[225] L. Di Gaspero, A. Rendl, and T. Urli, "Balancing bike sharing systems with constraint programming.," *Constraints*, vol. 21, pp. 318–348, Apr. 2016.

[226] I. A. Forma, T. Raviv, and M. Tzur, "A 3-step math heuristic for the static repositioning problem in bike-sharing systems," *Transportation research part B: methodological*, vol. 71, pp. 230–247, Jan. 2015.

[227] M. Dell'Amico, M. Iori, S. Novellani, and T. Stützle, "A destroy and repair algorithm for the bike sharing rebalancing problem," *Computers & Operations Research*, vol. 71, pp. 149–162, July 2016.

[228] M. Dell'Amico, E. Hadjicostantinou, M. Iori, and S. Novellani, "The bike sharing rebalancing problem: Mathematical formulations and benchmark instances," *Omega*, vol. 45, pp. 7–19, June 2014.

[229] C. Contardo, C. Morency, and L.-M. Rousseau, *Balancing a dynamic public bike-sharing system.* Cirrelt Montreal, Mar. 2012.

[230] G. B. Dantzig and P. Wolfe, "Decomposition principle for linear programs," *Operations research*, vol. 8, pp. 101–111, Feb. 1960.

[231] J. F. Benders, "Partitioning procedures for solving mixed-variables programming problems," *Numerische mathematik*, vol. 4, pp. 238–252, Dec. 1962.

[232] G. Erdoğan, M. Battarra, and R. W. Calvo, "An exact algorithm for the static rebalancing problem arising in bicycle sharing systems," *European Journal of Operational Research*, vol. 245, pp. 667–679, Sep. 2015.

[233] C. Kloimüllner, P. Papazek, B. Hu, and G. R. Raidl, "Balancing bicycle sharing systems: an approach for the dynamic case," in *European Conference on Evolutionary Computation in Combinatorial Optimization*, pp. 73–84, Springer, Apr. 2014.

[234] C.-C. Lu, "Robust multi-period fleet allocation models for bike-sharing systems," *Networks and Spatial Economics*, vol. 16, pp. 61–82, Mar. 2016.

[235] J. Brinkmann, M. W. Ulmer, and D. C. Mattfeld, "Short-term strategies for stochastic inventory routing in bike sharing systems," *Transportation Research Procedia*, vol. 10, pp. 364–373, Jan. 2015.

[236] E. O'Mahony and D. B. Shmoys, "Data Analysis and Optimization for (Citi) Bike Sharing.," in *29th AAAI Conference on Artificial Intelligence*, pp. 687–694, Jan. 2015.

[237] C. Fricker and N. Gast, "Incentives and redistribution in homogeneous bike-sharing systems with stations of finite capacity," *EURO journal on transportation and logistics*, vol. 5, pp. 261–291, Aug. 2016.

[238] L. Li and M. Shan, "Bidirectional incentive model for bicycle redistribution of a bicycle sharing system during rush hour," *Sustainability*, vol. 8, p. 1299, Dec. 2016.

[239] J. Pfrommer, J. Warrington, G. Schildbach, and M. Morari, "Dynamic vehicle redistribution and online price incentives in shared mobility systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, pp. 1567–1578, Aug. 2014.

[240] A. Singla, M. Santoni, G. Bartók, P. Mukerji, M. Meenen, and A. Krause, "Incentivizing users for balancing bike sharing systems.," in *29th AAAI Conference on Artificial Intelligence*, pp. 723–729, Jan. 2015.

[241] H. Chung, D. Freund, and D. B. Shmoys, "Bike angels: An analysis of citi bike's incentive program," in *1st ACM SIGCAS Conference on Computing and Sustainable Societies (COMPASS)*, pp. 5:1–5:9, ACM, June 2018.

[242] W. Fischer and K. Meier-Hellstern, "The markov-modulated poisson process (mmpp) cookbook," *Performance evaluation*, vol. 18, pp. 149–171, 1993.

[243] A. Krinik and C. Mortensen, "Transient probability functions of finite birth-death processes with catastrophes," *Journal of Statistical Planning and Inference*, vol. 137, pp. 1530–1543, May 2007.

[244] F. W. Crawford and M. A. Suchard, "Transition probabilities for general birth-death processes with applications in ecology, genetics, and evolution," *Journal of mathematical biology*, vol. 65, pp. 553–580, Sep. 2012.

[245] J. G. Skellam, "The frequency distribution of the difference between two Poisson variates belonging to different populations," *Journal of the Royal Statistical Society, Series A*, vol. 109, p. 296, Jan. 1946.

[246] S. Boyd and L. Vandenberghe, *Convex optimization.* Cambridge University Press, Mar. 2004.

[247] G. Clarke and J. W. Wright, "Scheduling of vehicles from a central depot to a number of delivery points," *Operations research*, vol. 12, pp. 568–581, Aug. 1964.

[248] S. Ropke and D. Pisinger, "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows," *Transportation science*, vol. 40, pp. 455–472, Nov. 2006.

[249] T. H. T. Nguyen, T. T. Le, T. P. D. Chu, L. T. Nguyen, and V. H. Le, "Multi-source data analysis for bike sharing systems," in *International Conference on Advanced Technologies for Communications (ATC)*, Oct. 2017.

[250] F. Chiariotti, M. Condoluci, T. Mahmoodi, and A. Zanella, "Symbiocity: Smart cities for smarter networks," *Transactions on Emerging Telecommunications Technologies*, vol. 29, p. e3206, Jan. 2018.

[251] P. Hunt, D. Robertson, R. Bretherton, and M. C. Royle, "The SCOOT on-line traffic signal optimisation technique," *Traffic Engineering & Control*, vol. 23, Apr. 1982.

[252] P. Muñoz, R. Barco, and I. de la Bandera, "On the potential of handover parameter optimization for self-organizing networks," *IEEE Transactions on Vehicular Technology*, vol. 62, pp. 1895–1905, Feb. 2013.

[253] F. Guidolin, I. Pappalardo, A. Zanella, and M. Zorzi, "Context-aware handover policies in HetNets," *IEEE Transactions on Wireless Communications*, vol. 15, pp. 1895–1906, Mar. 2016.

[254] M. Dalla Cia, F. Mason, D. Peron, F. Chiariotti, M. Polese, T. Mahmoodi, M. Zorzi, and A. Zanella, "Mobility-aware handover strategies in Smart Cities," in *International Symposium on Wireless Communication Systems (ISWCS)*, pp. 438–443, IEEE, Aug. 2017.

[255] M. Dalla Cia, F. Mason, D. Peron, F. Chiariotti, M. Polese, T. Mahmoodi, M. Zorzi, and A. Zanella, "Using Smart City data in 5G Self-Organizing Networks," *IEEE Internet of Things Journal*, vol. 5, pp. 645–654, Apr. 2018.

[256] C. G. Cassandras, "Smart cities as cyber-physical social systems," *Engineering*, vol. 2, pp. 156–158, June 2016.

[257] M. Nakamura and L. Du Bousquet, "Constructing execution and life-cycle models for Smart City services with self-aware IoT," in *International Conference on Autonomic Computing (ICAC)*, pp. 289–294, IEEE, July 2015.

[258] M. Condoluci, F. Sardis, and T. Mahmoodi, "Softwarization and virtualization in 5G networks for Smart Cities," in *International Conference on Cyber physical systems, IoT and sensors Networks (CYCLONE)*, pp. 179–186, EAI, Oct. 2015.

[259] I. 802.15.4-2011, "IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)," *IEEE*, June 2011.

[260] Bluetooth SIG, "Bluetooth Core Specification 4.2," Dec. 2014.

[261] ITU-T G.9959, "Short range narrow-band digital radiocommunication transceivers - PHY, MAC, SAR and LLC layer specifications," *International Telecommunication Union*, Jan. 2015.

[262] R. Sanchez-Iborra and M.-D. Cano, "State of the art in LP-WAN solutions for industrial IoT services," *Sensors*, vol. 16, p. 708, May 2016.

[263] M. Centenaro, L. Vangelista, A. Zanella, and M. Zorzi, "Long-range communications in unlicensed bands: The rising stars in the IoT and smart city scenarios," *IEEE Wireless Communications*, vol. 23, pp. 60–67, Oct. 2016.

[264] M. Condoluci, G. Aaniti, T. Mahmoodi, and M. Dohler, "Enabling the IoT machine age with 5G: Machine-type multicast services for innovative real-time applications," *IEEE Access*, vol. 4, pp. 5555 – 5569, May 2016.

[265] M. Amani, T. Mahmoodi, M. Tatipamula, and H. Aghvami, "Programmable policies for Data Offloading in LTE Network," in *International Conference on Communications (ICC)*, pp. 3154–3159, June 2014.

[266] H. Silk, M. Homer, and T. Gross, "Design of Self-Organizing Networks: creating specified degree distributions," *IEEE Transactions on Network Science and Engineering*, vol. 3, pp. 147–158, July 2016.

[267] P. P. Patel and R. H. Jhaveri, "Soft computing techniques to address various issues in wireless sensor networks: A survey," in *International Conference on Computing, Communication and Automation (ICCCA)*, pp. 399–404, Apr. 2016.

[268] O. K. Tonguz and W. Viriyasitavat, "A Self-Organizing Network approach to priority management at intersections," *IEEE Communications Magazine*, vol. 54, pp. 119–127, June 2016.

[269] K. Zheng, Z. Yang, K. Zhang, P. Chatzimisios, K. Yang, and W. Xiang, "Big data-driven optimization for mobile networks toward 5G," *IEEE Network*, vol. 30, pp. 44–51, Jan. 2016.

[270] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *ACM 1st MCC workshop on Mobile cloud computing*, pp. 13–16, Aug. 2012.

[271] J. Al-Jaroodi and N. Mohamed, "Service-oriented architecture for big data analytics in smart cities," in *18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 633–640, May.

[272] R. Irmer, H. Droste, P. Marsch, M. Grieger, G. Fettweis, S. Brueck, H.-P. Mayer, L. Thiele, and V. Jungnickel, "Coordinated multipoint: Concepts, performance, and field trial results," *IEEE Communications Magazine*, vol. 49, pp. 102–111, Feb. 2011.

[273] A. Checko, H. L. Christiansen, Y. Yan, L. Scolari, G. Kardaras, M. S. Berger, and L. Dittmann, "Cloud RAN for mobile networks—a technology overview," *IEEE Communications surveys & tutorials*, vol. 17, pp. 405–426, Mar. 2015.

[274] M. A. Lema, T. Mahmoodi, and M. Dohler, "On the performance evaluation of enabling architectures for uplink and downlink decoupled networks," in *IEEE GLOBECOM Workshops*, pp. 1–6, Dec. 2016.

[275] I. Pappalardo, G. Quer, B. D. Rao, and M. Zorzi, "Caching strategies in heterogeneous networks with D2D, small BS and macro BS communications," in *International Conference on Communications (ICC)*, pp. 1–6.

[276] M. Polese, M. Centenaro, A. Zanella, and M. Zorzi, "M2M massive access in LTE: RACH performance evaluation in a Smart City scenario," in *International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2016.

[277] G. P. Fettweis, "The tactile internet: applications and challenges," *IEEE Vehicular Technology Magazine*, vol. 9, pp. 64–70, Mar. 2014.

[278] A. Gladisch, R. Daher, and D. Tavangarian, "Survey on mobility and multihoming in future internet," *Wireless personal communications*, vol. 74, pp. 45–81, Jan. 2014.

[279] "5G White Paper." White Paper, Feb. 2015.

[280] M. Jiang, M. Condoluci, and T. Mahmoodi, "Network slicing management & prioritization in 5G mobile systems," in *European Wireless*, pp. 1–6, May 2016.

[281] T. Mahmoodi and S. Seetharaman, "Traffic jam: Handling the increasing volume of mobile data traffic," *IEEE Vehicular Technology Magazine*, vol. 9, pp. 56–62, Sep. 2014.

[282] T. Mahmoodi and S. Seetharaman, "On using a SDN-based Control Plane in 5G mobile networks," in *Wireless World Research Forum, 32nd Meeting*, pp. 1–6, May 2014.

[283] P. I. Bratanov and E. Bonek, "Mobility model of vehicle-borne terminals in urban cellular systems," *IEEE Transactions on Vehicular Technology*, vol. 52, pp. 947–952, July 2003.

[284] A. S. Hassani, A. R. Momen, and P. Azmi, "Mobility model of vehicular terminals in cellular networks," in *2nd International Conference on Information Communication Technologies*, vol. 2, pp. 2434–2437, 2006.

[285] Q. Dong and W. Dargie, "A survey on mobility and mobility-aware MAC protocols in Wireless Sensor Networks," *IEEE Communications Surveys & Tutorials*, vol. 15, pp. 88–100, Feb. 2013.

[286] S. Vasudevan, R. N. Pupala, and K. Sivanesan, "Dynamic eICIC – a proactive strategy for improving spectral efficiencies of heterogeneous LTE cellular networks by leveraging user mobility and traffic dynamics," *IEEE Transactions on Wireless Communications*, vol. 12, pp. 4956–4969, Oct. 2013.

[287] J. G. Andrews, H. Claussen, M. Dohler, S. Rangan, and M. C. Reed, "Femtocells: Past, present, and future," *IEEE Journal on Selected Areas in Communications*, vol. 30, pp. 497–508, Mar. 2012.

[288] O. G. Aliu, A. Imran, M. A. Imran, and B. Evans, "A survey of self organisation in future cellular networks," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 1, pp. 336–361, 2013.

[289] D. Xenakis, N. Passas, L. Merakos, and C. Verikoukis, "Mobility management for femtocells in LTE-advanced: key aspects and survey of handover decision algorithms," *IEEE Communications Surveys & Tutorials*, vol. 16, pp. 64–91, July 2014.

[290] 3GPP Technical Specification 36.331, "Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Resource Control (RRC); Protocol specification," *LTE ETSI TS*, July 2014.

[291] Y. Lee, B. Shin, J. Lim, and D. Hong, "Effects of time-to-trigger parameter on handover performance in SON-based LTE systems," in *IEEE 16th Asia-Pacific Conference on Communications (APCC)*, pp. 492–496, 2010.

[292] I. Pappalardo, A. Zanella, and M. Zorzi, "Upper bound analysis of the handover performance in HetNets," *IEEE Communications Letters*, vol. 21, pp. 418–421, Feb. 2017.

[293] K. Kitagawa, T. Komine, T. Yamamoto, and S. Konishi, "A handover optimization algorithm with mobility robustness for LTE systems," in *IEEE 22nd International Symposium on Personal, Indoor and Mobile Radio Communications*, pp. 1647–1651, 2011.

[294] S. Nie, D. Wu, M. Zhao, X. Gu, L. Zhang, and L. Lu, "An Enhanced Mobility State Estimation Based Handover Optimization Algorithm in LTE-A Self-organizing Network," *Procedia Computer Science*, vol. 52, pp. 270–277, June 2015.

[295] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, pp. 236–262, Sep. 2015.

[296] I. Chih-Lin, J. Huang, R. Duan, C. Cui, J. X. Jiang, and L. Li, "Recent progress on C-RAN centralization and cloudification," *IEEE Access*, vol. 2, pp. 1030–1039, Aug. 2014.

[297] A. S. Rajan, S. Gobriel, C. Maciocco, K. B. Ramia, S. Kapury, A. Singhy, J. Ermanz, V. Gopalakrishnanz, and R. Janaz, "Understanding the bottlenecks in virtualizing cellular core network functions," in *21st International Workshop on Local and Metropolitan Area Networks*, pp. 1–6, IEEE, Apr. 2015.

[298] X. An, F. Pianese, I. Widjaja, and U. G. Acer, "dMME: Virtualizing LTE mobility management," in *36th Conference on Local Computer Networks (LCN)*, pp. 528–536, IEEE, Oct. 2011.

[299] P. Andres-Maldonado, P. Ameigeiras, J. Prados-Garzon, J. J. Ramos-Munoz, and J. M. Lopez-Soler, "Virtualized MME design for IoT support in 5G systems," *Sensors*, vol. 16, no. 8, p. 1338, 2016.

[300] P. Bhat, S. Nagata, L. Campoy, I. Berberana, T. Derham, G. Liu, X. Shen, P. Zong, and J. Yang, "LTE-advanced: an operator perspective," *IEEE Communications Magazine*, vol. 50, pp. 104–114, Feb. 2012.

[301] R. Tanbourgi, S. Singh, J. G. Andrews, and F. K. Jondral, "Analysis of non-coherent joint-transmission cooperation in heterogeneous cellular networks," in *IEEE International Conference on Communications (ICC)*, pp. 5160–5165, June 2014.

[302] 3GPP Technical Specification 36.839, "Evolved universal terrestrial radio access (E-UTRA); Mobility enhancements in heterogeneous networks," *LTE ETSI TS*, Jan. 2013.

[303] S. Sesia, I. Toufik, and M. Baker, *LTE, The UMTS Long Term Evolution: From Theory to Practice.* Wiley Publishing, 2009.

[304] Next Generation Mobile Networks Alliance, "Optimised backhaul requirements," Aug. 2008.

[305] 3GPP Technical Specification 36.881, "Study on latency reduction techniques for LTE," *LTE ETSI TS*, Sep. 2015.

List of Acronyms

ADAM Adaptive Moment Estimation.

AI Artificial Intelligence.

ALS Autocovariance Least Squares.

AQM Active Queue Management.

AR Augmented Reality.

BALIA Balanced Link Adaptation.

BBR Bottleneck Bandwidth and Round-trip propagation time.

BDP Birth-Death Process.

BLE Bluetooth Low Energy.

BS Base Station.

BTT Backpropagation Through Time.

CDF Cumulative Distribution Function.

CDN Content Delivery Network.

CKF Congestion Kalman Filter.

CN Core Network.

CR Cognitive Radio.

CSI Channel State Information.

D2D Device to Device.

DASH Dynamic Adaptive Streaming over HTTP.

DKF Delivery Kalman Filter.

DQN Deep Q-network.

eNB evolved Node Base.

FEC Forward Error Correction.

FeNB Femto eNB.

FESTIVE Fair, Efficient, and Stable adapTIVE algorithm.

GB Graphical Bayesian.

GP Gaussian Process.

GPS Global Positioning System.

HetNet Heterogeneous Network.

HMM Hidden Markov Model.

HTTP HyperText Transfer Protocol.

I2V Infrastructure to Vehicle.

IoT Internet of Things.

JFI Jain Fairness Index.

k-NN k-Nearest Neighbors.

LEAP Latency-controlled End-to-End Aggregation Protocol.

LPWA Low-Power Wide-Area.

LSTM Long-Short Term Memory.

LTE Long Term Evolution.

M2M Machine to Machine.

MCS Modulation and Coding System.

MDP	Markov Decision Process.
MeNB	Macro eNB.
MIMO	Multi-Input Multi-Output.
MIP	Mixed Integer Programming.
MLP	Multilayer Perceptron.
MME	Mobility Management Entity.
MMPP	Markov-Modulated Poisson Process.
MPC	Model Predictive Control.
MPD	Media Presentation Description.
MPTCP	Multi-path TCP.
MSS	Microsoft Smooth Streaming.
NAT	Network Address Translation.
NB-IoT	Narrowband IoT.
NFV	Network Function Virtualization.
NN	Neural Network.
OFDM	Orthogonal Frequency Division Modulation.
PANDA	Probe and Adapt.
PDF	Probability Distribution Function.
PID	Proportional-Integral-Derivative.
QoE	Quality of Experience.
QoE-RAHAS	QoE-driven Rate Adaptation Heuristic for Adaptive video Streaming.
QoS	Quality of Service.
QUIC	Quick UDP Internet Connections.

RAN Radio Access Network.

RBF Radial Basis Function.

ReLU Rectified Linear Unit.

RF Random Forest.

RL Reinforcement Learning.

RMSE Root Mean Square Error.

RNN Recurrent Neural Network.

RSS Received Signal Strength.

RSSI Received Signal Strength Indicator.

RTP Real-time Transport Protocol.

RTT Round Trip Time.

SC-MPTCP MPTCP with Systematic Coding.

SCOOT Split Cycle Offset Optimization Technique.

SNR Signal to Noise Ratio.

SON Self-Organized Networking.

SSIM Structural Similarity Index.

SV Support Vector.

SVM Support Vector Machine.

SVR Support Vector Regression.

TCP Transmission Control Protocol.

TfL Transport for London.

TTT Time-to-Trigger.

UDP User Datagram Protocol.

UE User Equipment.

UTC Urban Traffic Control.

V2I Vehicle to Infrastructure.

V2V Vehicle to Vehicle.

VI Value Iteration.

VM Virtual Machine.

vMME Virtual MME.

VPN Virtual Private Network.

VR Virtual Reality.

Acknowledgments

After three years and more than 200 pages, it's time to give thanks where they are due. They say it takes a village, and at least in my case it was actually true, so here it is.

First, I'd like to thank Andrea Zanella for all his advice and help, which went way beyond the requirements of the job; I was lucky to have him as my advisor. The second person who deserves credit is Stepan Kucera, who helped me and believed in me more than I did while I was working with him. Michele Zorzi, Michele Rossi, Nicola Laurenti, Giorgio Di Nunzio, Angelo Cenedese, Holger Claussen, and Leonardo Badia also gave me precious advice throughout these three years and helped me get my work published.

Of course, this work would not have been possible without all my other co-authors: Chiara Pielli, Michele Polese, Matteo Gadaleta, Daniel Zucchetto, Massimo Dalla Cia, Federico Mason, Davide Peron, Davide Del Testa, Rita Coutinho, Laura Toni, Stefano D'Aronco, Pascal Frossard, Davide Talon, Luca Attanasio, Massimo Condoluci, Toktam Mahmoodi, Riccardo Zanol, Davide Cazzaro, Mattia Gentil, Alessandro Galeazzi, Enrico Lovisotto, and Enrico Vianello.

On the personal side, I'll start with my family, who had to shoulder a lot of the weight of this PhD. The people who took me the rest of the way would take several pages to name one by one, but I'll try to at least list them by group: all my colleagues-slash-drinking mates during these three years, Giulia, Babi and all of ADI Padova, who were like a second family, Ottavia, Marta, and Arianna, who listened to my Irish woes over pints every Saturday night in Donnybrook, Elena, Irene and Nicola, who are still my go-to friends after all this time, Kristi, Robert and Luca, who welcomed me on the other side of the pond, all the FELDs team, Giorgia, Nicola, and Leo (hope to see you guys soon!), the Pantiera drinking team, and Daina and the rest of my Liguria friends. Last but not least, I'd like to thank all my virtual friends from HD, who feel as real as any of the others even though I've only seen some of them in a Telegram video message.

I'm sure I forgot someone, but if you're in here, at least a small piece of this thesis is yours. Thank you.