STOCHASTIC OPTIMAL CONTROL AND PERFORMANCE ANALYSIS OF WIRELESS AD HOC NETWORKS

DANIEL BALAGUER YAGAN

NATIONAL UNIVERSITY OF SINGAPORE

2007

STOCHASTIC OPTIMAL CONTROL AND PERFORMANCE ANALYSIS OF WIRELESS AD HOC NETWORKS

DANIEL BALAGUER YAGAN

(*B.Eng. (Hons), NUS*)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2007

To my father and mother

Acknowledgments

I believe this quotation applies to every aspect of life: "It takes vision and courage to create. It takes faith and courage to prove". I think every scientist, engineer, or mathematician is, to some degree, a man of faith. It requires faith to believe that there exists elegant solutions to difficult problems. Taking it a step further, it takes faith to believe that both the world we live in and the ability to understand it are gifts from God.

Therefore, first and foremost, I thank God for giving me the grace to acquire knowledge, and for sustaining me throughout my postgraduate studies.

I would like to express my deepest sense of gratitude to my supervisor, Associate Professor Tham Chen Khong, for his guidance, patience, and generous support. He has been my advisor for more than 5 years, ever since my third year of undergraduate studies. I'm grateful to him, not only as a supervisor who has guided my studies, but as a good friend in life.

I would like to thank the Department of Electrical and Computer Engineering and the National University of Singapore for the generous offer of research scholarship and conference grants. I also would like to thank the members of Computer Networks and Distributed Systems Laboratory for their friendship and hospitality.

This dissertation is dedicated to my mom and dad, and my two younger sisters, who have been my inspiration in my studies and career.

Abstract

As the next generation of mobile networks emerges, the design for more advanced networking techniques becomes more difficult and non-trivial, due to the increasing quality-of-service (QoS) demands of multimedia applications, while considering system issues and constraints.

In this thesis, we consider a stochastic optimal control approach for resource allocation and provisioning in wireless ad hoc networks. Specifically, we study the problem of multi-class network scheduling under a time-varying channel and network topology. We formulate the problem using the decision theoretic framework known as Markov Decision Process (MDP) . We present four variants of MDP formulations to highlight important results and contributions.

The first model uses the theory of ψ-irreducibility for controlled Markov chains to formulate an average-cost MDP for each node acting as an agent, with the goal of finding the policy that minimizes the expected average congestion level. Using stability concepts of ψ-irreducible chains, we present the *first* novel method of achieving optimization and stability conditions *simultaneously* for a general Markov queueing network, and for deriving performance bounds from the control algorithm, as the algorithm converges to the optimal solution.

The second model considers a Semi-Markov Decision Process (SMDP) where each node adaptively performs network-level bandwidth allocation and buffer management. The main objective is to maximize average long term network reward and at the same time, minimize QoS violations with respect to bandwidth, queueing delay, and buffer

loss. Due to the dynamic nature of the network, estimating the state transition of the Markov chain is a non-trivial task. Hence, we use the *model-free* framework known as Neuro-Dynamic Programming (NDP), also known as Reinforcement Learning (RL), that uses stochastic approximation and simulation-based (i.e. online) techniques to approximate or find the near-optimal policies.

In order to have a faster and more robust convergence, we also consider the provisioning problem as a Hierarchical Semi-Markov Decision Process (HSMDP) that exploits a *task structure* in the original SMDP problem. Specifically, the problem is divided into a hierarchy of subtasks in a task graph. In solving the HSMDP, we use the corresponding Hierarchical Reinforcement Learning (HRL) technique that reuse subtask solutions in the task graph structure.

Finally, we formulate the queue scheduling problem as a Decentralized Partially Observable Markov Decision Process (DEC-POMDP) where the joint actions or policies of agents affect the performance of the system. DEC-POMDP is a multi-agent extension of MDP for decentralized control where each agent observes a different partial view of the current network condition. We also address the issue of locality of interaction among neighboring nodes and apply a model-free algorithm to approximate the optimal joint policies. Using stability concepts of ψ-irreducible chains, we also present the *first* method for analyzing the stability and optimization of a decentralized Markov network and derive performance bounds, as the algorithm converges.

We also verify our analysis from simulation results to show the effectiveness and convergence of our proposed algorithms for the four types of MDP formulations.

Contents

List of Figures

List of Tables

List of Symbols

S	general state space of a Markov chain
A	general action space of a Markov chain
$(S, \beta(S))$	general measurable space for state space S and $\beta(S)$ as σ-field
w	control policy for a Markov chain
Φ^w	a Markov chain under policy w
$P_w^t(s, B)$	t-step transition probability for a general Markov chain under policy w starting from state $s \in S$ to $B \in \beta(S)$
$K_w(s, B)$	resolvent kernel under policy w from state $s \in S$ to $B \in \beta(S)$
π	invariant probability distribution for a general Markov chain
$V(s)$	value function for state s of a general Markov chain
N	total number of nodes or agents
J	total number of network classes
$x_j(k)$	queue length in bits for class j at time slot k
$a_j(k)$	arrival process in bits for class j at time slot k
$\mu_j(k)$	rate allocated to class j at time slot k

$C_h(k)$	wireless channel state at time slot k
$\mu_{j,av}$	rate convergent value for service process $\mu_j(k)$
λ_j	rate convergent value for arrival process $a_j(k)$
$\Gamma_{C_h(k)}$	constraint set of available transmission rates during the channel state $C_h(k)$
X	state space of the queue length process evolving as a Markov chain
$\overline{x}(k)$	queue length vector in bits at time slot k for J classes
$J(w, \overline{x}(0))$	average long term congestion level starting from initial state $\overline{x}(0)$ under scheduling policy w
$\psi(B)$	a measurable function for $B \in \beta(X)$ under the state space X of queue length in the scheduling problem
$\delta_C(x)$	indicator function for set C for state x
$\overline{\eta}$	a problem-dependent positive constant used in the Foster-Lyapunov Drift Inequality
m_j	the second moment of the increment process $\{a_j(k) - \mu_j(k)\}$ for all $k \geq 0$
$d_{j,max}$	queueing delay constraint for class queue j
$l_{j,max}$	buffer loss constraint for class queue j
C_h	time-varying channel capacity
$w_{sched,j}$	scheduling weight for class queue j
$r_{j,min}$	target bandwidth for class queue j for handling rate and delay contraints
w_i	local policy for agent i in a multiagent domain

L	maximum possible number of links in a directed graph $G(V, E)$ where V is the set of nodes and E is the set of links
$r_{i,l}^{j}(t)$	topology matrix element for class j in row i and column l at time t
$R^{j}(t)$	$N \times L$ topology matrix with elements $r_{i,l}^{j}(t)$ for class j
$h(l)$	destination node of link l
$q(l)$	origin node of link l
$x_{i}^{j}(t)$	queue length in bits for class j in node i at time slot t
$\mu_{l}^{j}(t)$	alocated share of bandwidth in bits for link l for class j
$d_{i}^{j}(t)$	number of bits arriving at the j^{th} class queue, as generated by the source application at node i, if there's any
I_i	internal states for the MFSC for agent i
$\phi_i \in \mathbb{R}^{n_{\phi_i}}$	n_{ϕ_i}-dimensional vector to parameterise the internal state transition probabilities for agent i
$\theta_i \in \mathbb{R}^{n_{\theta_i}}$	n_{θ_i}-dimensional vector to parameterise the action probabilities for agent i
$C_{loc,i}(t)$	immediate localized cost for agent i at time slot t
$\mu_{i,out}^{j}(t)$	total allocated rate for all outgoing links from node i for class j

$\mu_{i,out}^j$	convergent rate of the process $\mu_{i,out}^j(t) := \sum_{\{\forall l: q(l)=i\}} \mu_l^j(t)$
$\mu_{i,in}^j$	convergent rate of the process $\mu_{i,in}^j(t) := \sum_{\{\forall l: h(l)=i\}} \mu_l^j(t)$
λ_i^j	convergent rate of the process $\{d_i^j(t)\}$
m_i^j	second moment of the process $y_i^j(t)$, where $y_i^j(t) := \sum_{\{\forall l: q(l)=i\}} \mu_l^j(t) - \sum_{\{\forall l: h(l)=i\}} \mu_l^j(t) - d_i^j(t)$

List of Abbreviations

QoS	Quality of Service
MDP	Markov Decision Process
SMDP	Semi-Markov Decision Process
NDP	Neuro-Dynamic Programming
RL	Reinforcement Learning
HSMDP	Hierarchical Semi-Markov Decision Process
HRL	Hierarchical Reinforcement Learning
POMDP	Partially Observable MDP
DEC-POMDP	Decentralized POMDP
MANET	Mobile Ad Hoc Networks
MAC	Medium Access Control
SLA	Service Level Agreement
DP	Dynamic Programming
MLP	Multi-Layer Perceptron
CMAC	Cerebellar Model Articulation Controller
TD	Temporal-Difference
SMART	Semi-Markov Average Reward Technique
NEXP	Non-Deterministic Exponential-Time
DCOP	Distributed Constraint Optimization
LID-RLPS	Locally Interacting Distributed RL Policy Search
DEC-MDP	Decentralized MDP

CHO-AR	Continuous-Time Hierarchically Optimal Average Reward
ACOE	Average Cost Optimality Equation
NS2	Network Simulator version 2
DCF	Distributed Coordination Function
AODV	Ad hoc On-Demand Distance Vector
CBR	Constant Bit Rate
WF^2Q	Worst-Case Fair Weighted Fair Queueing
FLP	Foster-Lyapunov Provisioning
PDR	Packet Delivery Ratio
BA	Bandwidth Allocation
BM	Bandwidth Management
JoBS	Joint Buffer Management and Scheduling
WFNN	Wire Fitted Neural Network
WFRLP	Wire-Fitted RL Provisioning
JoBS-NS2	JoBS under NS2
HAM	Hierarchy of Abstract Machines
FSC	Finite State Controller
MFSC	Multi-agent Finite State Controller
LID-JESP	Locally Interacting Distributed Joint Equilibrium Search for Policies
ND-POMDP	Networked Distributed-POMDP
DBA	Distributed Breakout Algorithm
IRLP	Independent RL Provisioning

List of Algorithms

Chapter 1

Introduction

Mobile ad hoc networks (MANETs) have been envisioned as a type of next generation wireless network that is self-organizing, rapidly deployable and requires no infrastructure. They form a wireless network of mobile routers inter-connected with multi-hop communication paths. MANETs offer distinctive advantages in areas such as search and rescue operation in disaster areas, in collaborative mobile computing, and recently, in a heterogeneous sensor network where a mobile node acts as a more powerful sink and cluster head or gateway for resource-constrained sensor devices.

In order to realize the practical benefits of MANETs, it is essential to develop efficient networking techniques that take full advantage of available resources with consideration of the system capabilities and constraints. A number of issues must be addressed, such as routing, scheduling, medium access control (MAC), mobility management, power control and quality of service (QoS).

This thesis considers the problem of *network-level* resource allocation and provisioning under a time-varying channel in a multi-class ad hoc network. Specifically, we use a *stochastic optimal control* approach for queue scheduling or bandwidth allocation and buffer management. Our model captures the situation of having different source-destination pairs, different channel data and error rates, general traffic arrival patterns and arbitrary number of nodes.

There has been a number of research works in resource allocation and scheduling

under a time-varying channel and in a multi-hop network. Due to the inherently lossy characteristics of the wireless medium, researchers have used adaptive techniques for resource management. In [1] and [2], the authors formulated a constrained convex optimization of resource allocation for MANETs subject to QoS and fairness constraints. Specifically, they have considered power control for optimizing throughput, service level agreement (SLA) feasibility under network constraints (i.e. with respect to bandwidth, delay, delivery probability and guaranteed data rate), unused capacity maximization and minimizing transmission delay under SLA with the network constraints.

In [3], the authors presented a price-based approach of bandwidth allocation and considered pricing as a means to stimulate cooperation where nodes charge other nodes for relaying data packets. Assuming that the users maximize their own benefit, they have proposed an iterative price and rate adaptation algorithm and showed that it converges to a socially optimal bandwidth allocation.

The main weakness of these approaches [1, 2, 3] is the assumption that the network is static. Once the network conditions vary, each agent may need to re-compute its optimal resource allocation policy. A better and more practical approach is thus to include the network dynamics and find the optimal course of actions for the agents. There is also no clear notion of time or decision periods in the above-mentioned schemes. It is evident that the amount of available resources for future connections and expected amount of future net benefit depend on how much reservation is made at present. Our solution differs from these approaches by treating the resource allocation problem as a stochastic control problem, where each agent actively performs queue management and scheduling to optimize long term performance.

The theory of Markov Decision Process (MDP) have been widely used as a mathematical framework for sequential decision-making in stochastic domains and for addressing performance optimization. This thesis presents four variants of MDP formu-

lations to highlight our contributions and results.

1.1 Optimal Stochastic Control using ψ-irreducible Markov chains

The first type considers a general queueing network under a time-varying channel, where each node acts as an agent. We derive a Markov chain for each node and formulate a MDP, also known as a controlled Markov chain, with the objective of minimizing the average congestion level. The optimal solution to the MDP can be found by using standard dynamic programming (DP) techniques such as value iteration and policy iteration [4]. However, due to the time-varying nature of the network, the *irreducibility* property of the Markov chain may not hold and thus, DP algorithms cannot be applied directly to compute the optimal policy. We thus use a novel framework known as ψ-irreducibility for controlled Markov chains. We note that ψ-irreducibility is easier to verify and is more applicable than the standard definition of *irreducibility* of having only a single communicating class where any state can be visited from any initial condition [4].

Obtaining stochastic control policies for wireless queueing networks are developed based on the theory of *Lyapunov drift* [5, 6, 7]. This theory has been used in the development of stabilizing control laws for data networks, but has not been used to address performance optimization. However, by using the framework of ψ-irreducible Markov chains, not only that we can address performance optimization in finding the optimal policies, we can also show stability for the queueing model.

In particular, for obtaining the optimal policies for ψ-irreducible Markov chains, we introduce stability concepts such as c-regularity, *regular* chains and *regular* policies. Consequently, our analysis also uses the theory of *Lyapunov's second method*, together with *Foster's stability criterion* for *uncontrolled* Markov chains. The merging

of these techniques is known as the *Foster-Lyapunov drift condition* [8] that provides a stability inequality condition for our wireless queueing model.

By using the stability conditions for ψ-irreducible Markov chains and a modified *value iteration* algorithm, we also derive performance bounds from the controlled model itself, as the control algorithm converges to the optimal policy. Specifically, due to the *Foster-Lyapunov drift condition* and regularity of intermediate policies in the iteration, we can guarantee a bounded average congestion level and queueing delay.

The idea of using a drift condition to derive performance bounds is similar to the work in [9]. However, the authors in [9] did not formulate the problem as a MDP and only used a *linear programming* approach at every time slot to search for the optimal parameter settings. In general, the value iteration algorithm for MDP is less computationally expensive than linear programming, since the former improves on its computed policy at every succeeding iteration, while the latter searches the solution space at every time step without consideration of previously obtained values.

To the best of the author's knowledge, by using concepts of ψ-irreducible Markov chains, we present the *first* method of achieving optimization and stability conditions *simultaneously* in a general wireless Markov queueing network, and for deriving performance bounds *directly* from the scheduling algorithm, as the algorithm converges to the optimal solution.

1.2 Near-Optimal and Model-Free QoS Provisioning

The second variant of MDP formulation considers a Semi-Markov Decision Process (SMDP) where each node adaptively performs network-level bandwidth allocation and buffer management in a multi-class network. The main objective is to maximize average network reward and at the same time, minimize per-class QoS violations with

respect to bandwidth, queueing delay, and buffer loss. Due to the fact that in a dynamic network, estimating the state transition probabilities of the underlying Markov chain is a non-trivial task, we use the novel *model-free* mathematical framework known as *Neuro-Dynamic Programming* (NDP) [10], also termed as *Reinforcement Learning* (RL) [11].

It is also well known that Dynamic Programming techniques, such as value iteration and policy iteration, suffer from *curse of dimensionality* [4], especially when the state space is large as in the case of QoS provisioning. NDP or RL solves these issues by finding an *approximate* solution to the optimal policy, while the agent interacts with the system. The distinguishing characteristics of this approach is that it can be used in practical and real-world scenarios, whereby each node determines its near-optimal policy through a sequence of direct interactions with the network. A model-free solution does not need prior knowledge of the state transition probabilities of the Markov chain. Thus, RL is less computationally expensive than DP techniques, as it does not require the exact model of the system.

In [12], the authors proposed a stochastic control approach for resource allocation under a time-varying channel. However, they used a model-based DP algorithm to find the optimal policy. In our research, we employ a model-free solution due to its practicality and applicability in actual network deployments. In addition to the time-varying channel, we also consider MANET-specific characteristics, such as dynamic topology and different MAC and routing mechanisms.

A provisioning method based on the SMDP framework was proposed in [13] for adaptive multimedia in cellular wireless networks. They discussed a bandwidth adaptation algorithm in conjunction with call admission control. They considered a multi-class network and formulated an average reward SMDP. They applied a model-free RL algorithm to maximize the network revenue while satisfying the following QoS constraints: probability of hand-off dropping, average allocated bandwidth and intra-

class fairness. Due to the large state space of the problem, they used a neural network structure known as *Multi-Layer Perceptron* (MLP) with a single hidden layer for approximating the *state-action* values in the RL algorithm. The state-action values were then used to construct and find the optimal policy.

Our SMDP formulation is similar to [13], however, we present a network-level bandwidth allocation and buffer management scheme in a multi-class MANET with consideration of per-class QoS constraints. In applying RL to find the near-optimal policy, we use a linear neural network structure known as *Cerebellar Model Articulation Controller* (CMAC), which is computationally cheap to use and is suitable for fast and real-time learning. The CMAC network performs linear function approximation for compactly estimating and storing state-action values in the RL algorithm [14]. Linear function approximators, such as CMAC neural networks, in contrast with *non-linear* MLP networks used in [13], are also known to be beneficial for the convergence of RL algorithms, such as *Temporal-Difference* (TD) learning, for problems involving large or *continuous* state space [15].

The main contribution of this approach is by formulating the QoS provisioning problem as a stochastic control problem (i.e. SMDP) and using a model-free solution (i.e. RL), we are able to provision bandwidth and manage buffer resources to satisfy QoS requirements in a cost-effective and practical manner, while attaining the near-optimal policy and without knowing the exact model of the system.

1.3 Hierarchical Optimal Control for Resource Allocation

The third variant of MDP formulation extends the SMDP model for QoS provisioning as described in the previous section, especially for continuous state and action vector spaces. Using the idea of *divide-and-conquer*, we propose to divide the original SMDP

provisioning problem into a collection of smaller problems. We then compose policies obtained from the smaller problems into the optimal policy for the original problem.

Intuitively, this mechanism accelerates the process of finding the optimal solution, since the smaller problems are relatively easier to solve. Formally, we use the framework known as *Hierarchical Semi-Markov Decision Process* (HSMDP) [16] that uses the idea of decomposing the large state space into regions of sub-spaces.

A complex SMDP problem can often be solved by decomposing it into a collection of smaller problems. The smaller problems can then be solved and recombined into a solution for the original problem. In dividing the state space, HSMDP essentially decomposes the given problem into different tasks. Each task can then be further decomposed into a collection of subtasks and so on, up to the desired level of the *task hierarchy*. HSMDP uses the idea of temporal abstraction where, at a given task level in the hierarchy, decisions are not required at each step, but invokes a sequence of temporally-extended activities or tasks, also known as *temporally-abstract actions*. At the lowest level of the task hierarchy are *primitive actions* that immediately terminate after execution.

In the original SMDP problem in Section 1.2, we observe that, even though the model-free solution was used in finding the near-optimal policy for bandwidth allocation and buffer management, each agent still suffers from slow convergence, as it needs considerable amount of experience in estimating the optimal policy. In resource constrained devices, this may not be practical.

As for the HSMDP formulation, finding the optimal policy requires obtaining the best policy at each level of the *task hierarchy*. We use the corresponding model-free *Hierarchical Reinforcement Learning* (HRL) technique [16]. A primitive action of the task hierarchy corresponds to the action used in the *flat or non-hierarchical* RL algorithms, such as Q-Learning [17] and the SMART algorithm used in [13, 18]. It should be noted that flat RL algorithms are only applicable for non-hierarchical MDP

problems.

In [18], even though the action vector with the highest state-action value can be retrieved compactly, we observed that the flat RL algorithm is still searching from a large continuous vector space. This effectively does not prevent the agent from choosing costly actions, especially during the initial exploration phase, which also contributes to slower convergence. HRL accelerates convergence by reusing the policies learned by the subtasks in the task hierarchy in the HSMDP formulation. It is also possible to define actions at each subtask to prevent the agent from choosing costly and unfavorable policies.

The main contribution of this approach is that by decomposing the network-level QoS provisioning problem into a task hierarchy under the stochastic control HSMDP framework, and using the corresponding model-free HRL algorithm, we can achieve better average long term performance.

1.4 Decentralized Optimal Control for Resource Allocation

The fourth type of MDP formulation considers the queue scheduling problem as a *decentralized control* problem. Specifically, we use the framework known as *Decentralized Partially Observable Markov Decision Process* (DEC-POMDP) [19] where the performance of the network is affected by the joint actions or policies of the agents.

DEC-POMDP is an extension of the theory of Markov Decision Process for decentralized control (DEC-MDP), but each agent observes a different partial view of the current network condition. The observation of an agent may only include the local queue information and policies of neighboring agents. In finding the optimal joint actions, the DEC-POMDP formulation is essentially a multi-agent system that allows the agents to collaborate, cooperate and control a single MDP *without* com-

plete observability of the global network state. This approach is thus more applicable and realistic than any other MDP-based framework, as it considers a decentralized multi-agent system in actual communication network scenarios.

It is known that exact and model-based solutions to a DEC-POMDP are complete for the complexity class non-deterministic exponential time (NEXP-complete) [19]. In other words, a general DEC-POMDP does not admit polynomial-time algorithms since P $\neq$ NEXP. In this thesis, we propose to solve the queue scheduling problem as a DEC-POMDP using *model-free* RL techniques. It should be noted that standard RL algorithms only solve a single MDP formulation for each agent independently.

Hence, in order to find and approximate the optimal joint policy in a decentralized manner, we propose to extend RL algorithms to solve a multi-agent collaborative DEC-POMDP. Specifically, our approach is based on the RL technique known as *policy gradient* that parameterizes and updates the policies of each agent during execution.

We also observe the fact that in actual communication networks, each agent has only limited interactions with a small number of neighboring agents. In other words, each agent only affects those agents geographically close to it. We address this issue by exploiting this idea of *locality of interaction* by combining the ideas of DEC-POMDP and a formalism known as *Distributed Constraint Optimization* (DCOP) [20, 21].

We propose a model-free decentralized algorithm called *Locally Interacting Distributed Reinforcement Learning Policy Search* (LID-RLPS) to solve a DEC-POMDP. LID-RLPS uses a multi-agent *finite state controller* for each node to approximate the joint policy while considering the locality of interaction, and without the state and observation transition model of the DEC-POMDP. In our analysis, we also use the ψ-irreducibility framework discussed earlier, but applied in a multi-agent environment.

Consequently, we show a stability condition known as *V-uniform ergodicity* [22] for

ψ-irreducible controlled Markov chains. This stability property provides an inequality condition that enables us to derive performance bounds from the control algorithm. We emphasize that *V-uniform ergodicity* differs from the *Foster-Lyapunov drift condition* described earlier in Section 1.1 as it is more applicable in a model-free approach for a decentralized multi-agent framework.

To the best of the authors' knowledge and by also using stability concepts of ψ-irreducible chains, we present the *first* method of achieving optimization cooperatively in a decentralized manner in a general wireless Markov queueing network, and for deriving stability conditions and performance bounds *simultaneously* and *directly* from the control algorithm, as the LID-RLPS algorithm converges to the optimal solution.

1.5 Thesis Contributions

Figure 1.1 shows the contributions of our thesis. The breakdown of Figure 1.1 is explained as follows:

1. Given a wireless MANET, we identify a controlled Markov chain or Markov Decision Process. Two classifications of MDP framework are used: single-agent and multi-agent.

2. Given a particular MDP, we address the issue of stability. We introduce the ψ-irreducibility framework, and its stability concepts such as *c-regular* chains, *petite sets*, drift conditions, and V-uniform ergodicity.

3. We also address the issue of finding the optimal policy to optimize certain performance metrics. This can be done either via model-based exact DP algorithms such as value iteration, or via model-free algorithms such as policy gradient mechanism and NDP or RL.

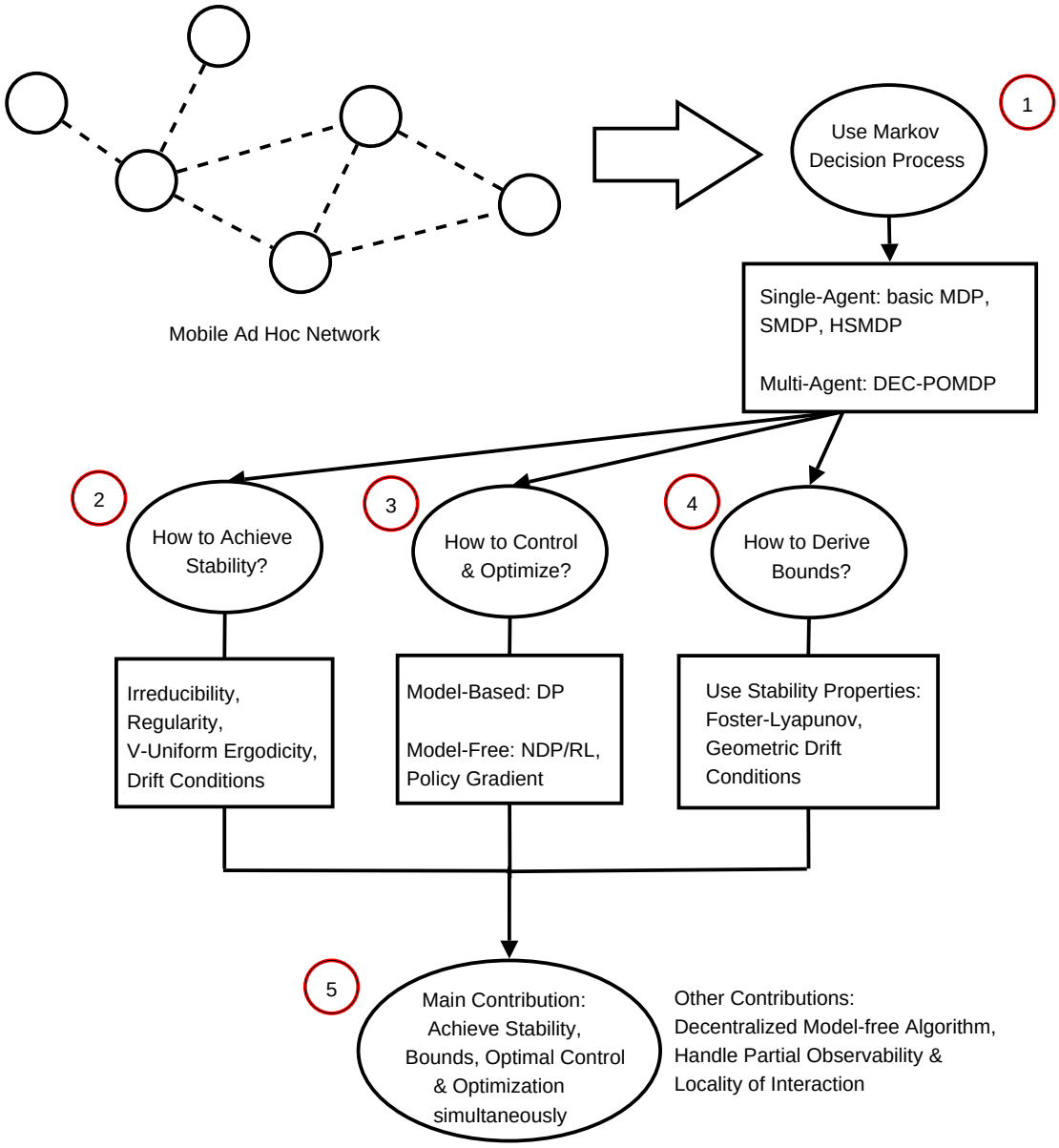


Figure 1.1: Thesis Contributions

4. In addition, we want to derive performance bounds for the network, under the controlled Markov chain. In this thesis, we use the Foster-Lyapunov and geometric drift inequality conditions under the ψ-irreducibility framework.

5. To the best of the author's knowledge and as our main contribution, we present the *first* technique based on a stochastic optimal control framework that achieves stability and optimization *simultaneously,* and for deriving performance bounds in a general Markov wireless queueing network *directly* from the control algorithm as the algorithm converges to the optimal solution.

In addition, we propose a number of techniques in the RL research field such as:

- A novel function approximation technique known as Wire-fitted CMAC that handles continuous state and action vector spaces for value-based algorithms
- A technique called Continuous-time Hierarchically Optimal Average Reward (CHO-AR) RL algorithm for solving the HSMDP problem with continuous state and action vector spaces
- Decentralized Model-free algorithm known as Locally Interacting Distributed Reinforcement Learning Policy Search (LID-RLPS) for solving the DEC-POMDP while handling locality of interaction, partial observability, and stability.

1.6 Summary of MDP models & algorithms

Figure 1.2 shows a summary of the MDP models, in terms of *control* and *optimization*. The first three models (i.e. ψ-irreducible controlled Markov chains, SMDP, HSMDP) fall under the single-agent framework. The last MDP variant (i.e. DEC-POMDP) is an extension of decentralized MDP (DEC-MDP) and falls under the multi-agent framework. In the first three variants, each node acts as an agent that sees its own

MDP locally and independently from the other nodes. Figure 1.3 shows our general system model for MANETs for the independent MDP agents.

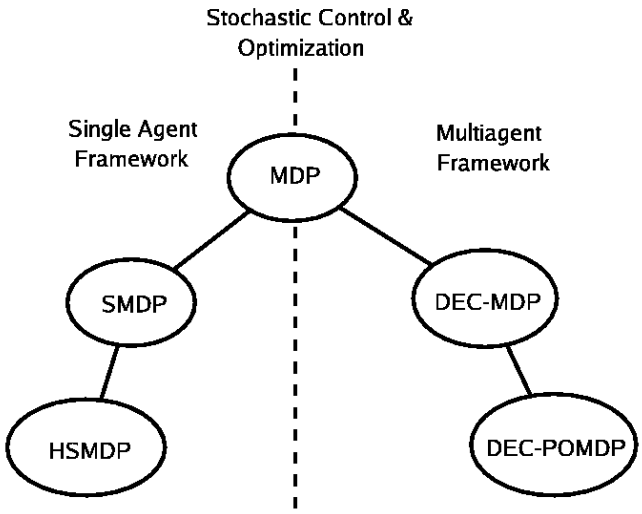


Figure 1.2: MDP models

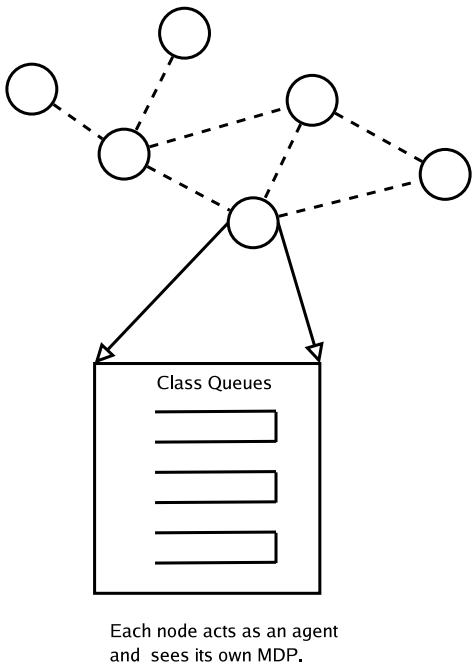


Figure 1.3: Independent MDP agents for resource allocation in MANETs

Under the ψ-irreducibility framework, Figure 1.4 shows the different properties

that express *stability* conditions needed for performance analysis in our wireless scheduling model.

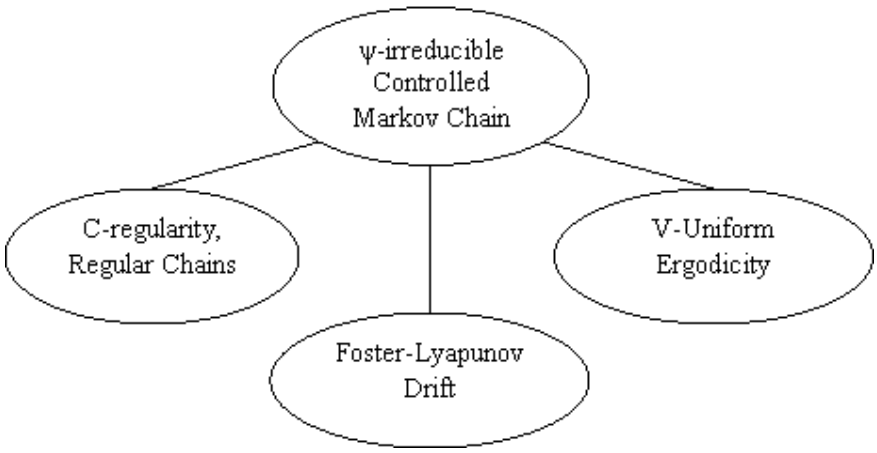


Figure 1.4: Stability and Performance Analysis of Controlled Markov Chains

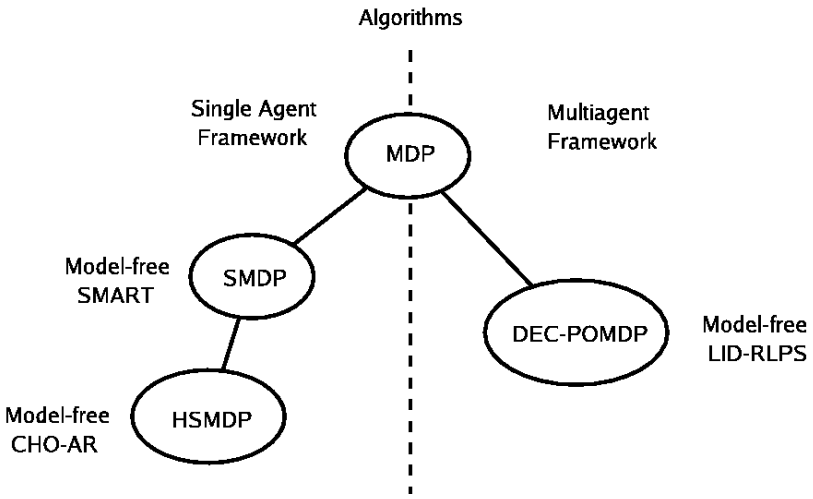


Figure 1.5: Model-Free MDP Algorithms

For the ψ-irreducibility framework in Chapter 3, we use a modified Dynamic Programming-based value iteration algorithm that considers both stability and optimization *simultaneously* (i.e. find optimal and stable policies). On the other hand, Figure 1.5 shows the corresponding algorithms used to solve the different MDP variants used in Chapters 4, 5, and 6. For the SMDP framework, we use a model-free RL algorithm known as Semi-Markov Average Reward Technique (SMART). We also

extend SMART for *continuous* state and action vector spaces to approximate the optimal policy.

For the HSMDP framework, we use an extended RL-based algorithm for a hierarchical task structure, known as Continuous-Time Hierarchically Optimal Average Reward (CHO-AR) algorithm. As mentioned in Section 1.3, this accelerates the convergence of finding the optimal policy as compared with the original SMDP model.

Finally, for the DEC-POMDP multi-agent model, we propose a model-free LID-RLPS algorithm that finds a near-optimal and stable policy structure for our resource allocation problem.

1.7 Organization of Thesis

Chapter 2 introduces the novel mathematical framework known as ψ-irreducibility for Markov chains. We shall use this framework for studying the stability and performance analysis for our wireless queueing model.

Chapter 3 first presents a general queueing model with a time-varying *channel state process*. We introduce the concepts of *rate convergence* and *channel convergence* and prove that the queueing model is a controlled Markov chain or MDP. We also consider the concept of a *topology state process* and explain how our approach can be used to include varying topology and MAC mechanisms. Since the *irreducibility* property of Markov chains may not apply due to the time-varying nature of the network, we apply the concepts of ψ-irreducibility for Markov chains from Chapter 2. Specifically, we discuss the optimization and stability conditions for ψ-irreducible controlled Markov chains with the objective of minimizing the average congestion level. We then derive the performance bounds for average queueing delay and congestion levels from the stability condition and control algorithm.

Chapter 4 introduces the SMDP framework and applies it for bandwidth allocation and buffer management. We then use the model-free RL algorithm to find the near-optimal policy that maximizes the average network reward, while minimizing QoS constraint violations. We also propose a novel function approximation technique known as *Wire-Fitted CMAC* that solves the issues of continuous state and action vector spaces in QoS provisioning.

Chapter 5 extends the SMDP framework of Chapter 4 by reformulating the QoS problem as HSMDP. The key principle behind HSMDP is to compose policies from the subtasks in the task structure. We then present the corresponding model-free HRL solution to approximate the hierarchically optimal policy. We compare the HRL-based provisioning scheme with the non-hierarchical solution in Chapter 4, in terms of average long term performance, such as network reward, class queueing delay, and buffer loss.

Chapter 6 presents and emphasizes the importance of decentralized control approach for collaborative scheduling and resource allocation, which differs from previous research. This chapter differs from the prior chapters as it addresses global network-wide optimization, and not on a per-node basis. We then present the DEC-POMDP framework and apply it for the multi-class queue scheduling problem in MANETs under the average cost criterion. The main objective is to find the joint optimal policy of the agents that minimizes the average long term congestion level. We then discuss some non-trivial issues of exact model-based solutions for DEC-POMDP. In order to find and *approximate* the joint optimal policy in a decentralized manner, we propose a model-free control algorithm known as LID-RLPS. We also exploit the idea of finding a policy structure to capture the *locality of interaction* among neighboring agents. Furthermore, we use the ψ-irreducibility property for Markov chains to study the performance and stability of our decentralized system model. We also derive performance bounds on average queueing delay and congestion level from the

stability condition known as *V-uniform ergodicity* for ψ-irreducible Markov chains.

Finally, we conclude our work and describe future research directions in Chapter 7.

Chapter 2

Optimization and Stability of Markov Decision Process

This chapter introduces the mathematical theory for a general Markov Decision Process, with emphasis on the ψ-irreducibility property, stability, and optimization of ψ-irreducible Markov chains.

As briefly introduced in Section 1.1, the framework of ψ-irreducible Markov chains is required since the usual concept of *irreducibility* for Markov chains may not hold, due to the time-varying network condition. The lemmas and theorems in this chapter are from previous research [4, 8, 23, 24, 25].

This important theory shall be used in Chapter 3 specific to our scheduling problem in a general wireless queueing network for MANETs.

2.1 Controlled Markov Chain or Markov Decision Process

We consider a general MDP with the state process $\Phi = \{\Phi(t) : t \geq 0\}$ evolving on a *countable* state space S and action space A. For each $s \in S$, there exists a non-empty set $A(s) \subseteq A$ that contains the admissible actions when the state Φ_t takes the value s at time t. Let $\beta(S)$ denote the set of all subsets of S. The pair $(S, \beta(S))$ is generally

known as a *measurable space* with S as the abstract set of points and $\beta(S)$ as the σ-field. The set of admissible state-action pairs $\{(s,a) : s \in S, a \in A(s)\}$ is assumed to be a measurable subset of the product space $S \times A$.

The process Φ itself can be thought of as lying in a sequence or path space formed by a countable product $\Omega = (S \times A)^{\infty} = \prod_{i=0}^{\infty} (S \times A)_i$. The set of all subsets of Ω is similarly defined with the corresponding σ-field $\mathcal{F}$. We also define an initial condition $s \in S$ for the sample path with a probability measure P_s so that $P(\Phi \in G)$ is well defined for any set $G \in \mathcal{F}$. The triple $\{\Omega, \mathcal{F}, P_s\}$ thus defines a stochastic process [23].

The transitions of the state process is governed by $P_a(s,B)$ which describes the probability that the next state is in B for any $B \in \beta(S)$ given the current state is $s \in S$ and the chosen action is $a \in A(s)$. A policy is defined by a mapping of the state process into the choice of action. In this chapter, we only consider a stationary Markov policy $w : S \to A$, such that $w(s) \in A(s)$ for all s and does not depend of the past choice of actions and time-period.

The process Φ^w under a fixed policy w is a Markov chain on $(S, \beta(S))$. We define the t-step transition probabilities for this chain as:

$$P_w^t(s,B) := P(\Phi_t^w \in B \mid \Phi_0^w = s)$$

for $s \in S$, $B \in \beta(S)$, and $t \in Z_+$. We also use the operator-theoretic notation: $P_w^t h(s) := E\{h(\Phi_t^w) \mid \Phi_0^w = s\}$. We also define the *resolvent kernel* as:

$$K_w(s,B) := \sum_{t=0}^{\infty} 2^{-(t+1)} P_w^t(s,B) \tag{2.1}$$

The stability of a Markov chain Φ^w is frequently defined in terms of the following return times:

$$\tau_B := \min\{n \geq 1 : \Phi_t^w \in B\}$$

$$\sigma_B := \min\{n \geq 0 : \Phi_t^w \in B\}$$

where τ_B is called the *first return time* while σ_B is the *first hitting time* on B.

We also define the *return time probability* as:

$$L(s, B) := P_s(\tau_B < \infty) \tag{2.2}$$

where $L(s, B)$ represents the probability that the chain enters B starting from state s.

We assume that a *cost function* $c : S \times A \to [1, \infty)$ is given. This assumption is used in [8, 23] for a general state space MDP and is crucial in providing convergence *without* the usual assumption of *irreducibility*. In this chapter, we only consider the *average cost criterion*. The average cost of a particular policy w, for a given initial condition $\Phi_0^w = s$, is defined as:

$$J(w, s) := \lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n} E_s^w \{c(\Phi_t^w, a_t)\} \tag{2.3}$$

where action $a_t = w(\Phi_t^w)$. A policy w^* is optimal if $J(w^*, s) \leq J(w, s)$ for all policies w, and *any* initial state s. The construction of the optimal policy is usually derived from the following equations:

$$\eta_* + h_*(s) = \min_{a\in A(s)} [c(x, a) + P_a h_*(s)] \tag{2.4}$$

$$w^*(s) = \arg\min_{a\in A(s)} [c(s, a) + P_a h_*(s)], \ s \in S \tag{2.5}$$

where $P_a h_*(s) := E\{h_*(\Phi_1^a) \mid \Phi_0^a = s\}$. The equality (2.4) is known as the *average*

cost optimality equation (ACOE). If a policy w^*, a *measurable* function h_* and a constant η_* exist that solve (2.4) and (2.5), then the stationary Markov policy w^* is optimal [4, 8]. Formally, this result is presented as follows:

Theorem 2.1: *[24, Theorem 2.1] If the triple (w^*, h_*, η_*) solves (2.4) and (2.5), and for any $x \in X$ and any policy w satisfying $J(w, s) < \infty$ and that:*

$$\frac{1}{n} P_w^n h_*(s) \to 0, \qquad n \to \infty.$$

Then w^* is an optimal control and η_* is the optimal cost such that:

$$\lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n} E_x^w \left\{ c(\Phi_t^{w^*}, a_t) \right\} = \eta_* \tag{2.6}$$

and $J(w, s) \geq \eta_*$ for all policies w, and all initial states s.

Since the goal is to find the policy that minimizes the steady state cost in (2.3), it is reasonable to first understand when the cost can be expected to be finite. In the next sections, we will introduce the concepts of ψ-irreducible Markov chains, *petite sets*, *f-regularity,* and c-regular chains [23]. These concepts are needed for characterizing the limit in (2.3) and for the stability of general Markov chains. We emphasize that the usual concept of *irreducibility* for Markov chains is relaxed in our model and thus another mathematical framework is required.

2.2 Concept of ψ-irreducibility for general state space chains

Definition 2.1: *A Markov chain Φ^w under policy w is ψ-irreducible if there exists a measure ψ on $B \in \beta(S)$ such that, whenever $\psi(B) > 0$, the resolvent kernel*

$K_w(s, B) > 0$, *for all* $s \in S$. *We call* ψ *a* **maximal irreducibility** *measure.*

Let $\beta^+(S) := \{A \in \beta(S) : \psi(A) > 0\}$. If the chain Φ^w is ψ-irreducible, the process has a positive probability of entering any set in $\beta^+(S)$. In other words, there exists some $n > 0$, such that, for any initial condition $s \in S$, $P_w^n(s, A) > 0$, where $A \in \beta^+(S)$. The chain Φ^w is also considered as ψ-irreducible if the return time probability satisfies: $L(s, A) > 0$ whenever $\psi(A) > 0$, for all $s \in S$. Equivalently, for a *countable* state space model, there exists a state $\theta \in S$ which is *accessible* where: $\sum_{t=0}^{\infty} P_w^t(s, \theta) > 0$. The following result is from [8].

Lemma 2.1: *A Markov chain is* ψ*-irreducible for a* **countable** *state space model if there exists a single communicating class of states which is reachable from any initial condition.*

Definition 2.2: *For a* ψ*-irreducible chain, a set* C *is defined as a* **petite set** *if for each* $A \in \beta^+(S)$*, and for any* $s \in C$*, there exists* $n \geq 1$ *and* $\delta > 0$*, such that:* $P_s(\tau_A \leq n) \geq \delta$ *where* $P_s(\tau_A \leq n)$ *denote the probability that the chain, starting from* x*, reaches* A *in at most* n *steps. This is equivalently expressed as:* $K_w(s, A) \geq \delta$ *under policy* w*. This also implies that in a* ψ*-irreducible chain, there always exists a countable covering of the state space by petite sets [23, Chapter 5].*

2.3 f-Regularity and Stability

A central concept of our model is the notion of f*-regularity*, where f is a measurable function of the state space and $f \geq 1$. In our case, $f = c_w$, where $c_w(s) = c(s, w(s))$ is the one-step cost incurred by policy w, for $s \in S$.

In a controlled ψ-irreducible chain and for any $A \in \beta^+(S)$, a set C is c_w*-regular*

if:

$$\sup_{s \in C} E_s^w \left\{ \sum_{t=0}^{\tau_A} c_w(\Phi_t^w) \right\} < \infty \tag{2.7}$$

where τ_A is the first return time to A.

A c_w-regular set is always a *petite set* due to the characterization of petite sets in Definition 2.2 and the result in [23, Theorem 14.2.4]. The Markov chain is itself a c_w-*regular* chain if the state space S admits a countable covering of c_w-regular sets. The Markov policy w is a *regular* policy if the controlled Markov chain is c_w-*regular*. The importance of regular policies is highlighted as follows:

Theorem 2.2: *[25, Theorem 2.1] For any regular policy w, there exists a unique invariant (i.e. steady state) probability distribution π_w and the controlled process Φ^w satisfies the following bound for the* steady state cost $\pi(c_w)$*:*

$$\pi(c_w) := \sum c_w(s)\pi_w(s) < \infty. \tag{2.8}$$

The average cost is equal to $\pi(c_w)$, independent of the initial condition s, and the following limit holds:

$$J(w) = J(w, s) = \pi(c_w) = \lim_{n \to \infty} \frac{1}{n} \sum_{t=0}^{n} E_s^w \left\{ c_w(\Phi_t^w) \right\} \tag{2.9}$$

The concepts of regular policies and c-regular Markov chains are thus crucial in finding the optimal policy w^* of controlled ψ-irreducible chains due to the Theorems 2.1 and 2.2.

The following result is a consequence of the f-norm Ergodic Theorem in [8, 23, Chapter 14] that uses a *drift* characterization of c-regular chains.

Theorem 2.3: *Let Φ^w be a Markov chain satisfying the* **Foster-Lyapunov drift inequality** *for $s \in S$:*

$$P_w V(s) := E\left\{V(\Phi^w_{t+1}) \mid \Phi^w_t = s\right\} \leq V(s) - c_w(s) + \bar{\eta} \tag{2.10}$$

where $V : S \to \mathbb{R}_+$ and $\bar{\eta} > 0$ is a finite constant. This inequality is usually written as: $P_w V \leq V - c_w + \bar{\eta}$. The cost function satisfies $c_w(s) \geq 1$. Suppose that the set $U = \{s : c_w(s) \leq 2\bar{\eta}\}$ is petite (see Definition 2.2). Then,

(i) Φ^w is a c_w-regular Markov chain satisfying the following bound: For each $A \in \beta^+(S)$, there exists a constant $d(A) < \infty$:

$$E^w_s \left\{\sum_{t=0}^{\tau_A - 1} c_w(\Phi^w_t)\right\} \leq V(s) + d(A) \tag{2.11}$$

(ii) There is a unique invariant probability π_w and $\pi(c_w) \leq \bar{\eta}$, where $\pi(c_w)$ is the steady state cost defined in (2.8).

Theorem 2.3 gives necessary conditions for a chain to be c-regular. Another relevant result in using the Foster-Lyapunov drift inequality is known as the *Comparison Theorem* from [23, Theorem 14.2.2]. This theorem will be used in Chapter 3 for showing c-regularity for our model.

Theorem 2.4: *If the Markov chain Φ_t satisfies the* drift inequality *for $s \in S$:*

$$PV \leq V - g + u$$

where the functions V, g, and u take values in $\mathbb{R}_+$. Then for any stopping time τ,

$$E_s\left\{V(\Phi_\tau)\right\} + E_s\left\{\sum_{t=0}^{\tau-1} g(\Phi_t)\right\} \leq V(s) + E_s\left\{\sum_{t=0}^{\tau-1} u(\Phi_t)\right\} \tag{2.12}$$

An important concept for *stability* of ψ-irreducible chains is defined as follows:

Definition 2.3: *A ψ-irreducible chain is called* **Harris** *if $L(s, A) = 1$ for any $A \in \beta^+(S)$ and any $s \in S$, where $L(s, A)$ is defined in (2.2). If the chain admits an invariant probability measure π, then the chain is called* ***positive Harris.***

From this definition, a c-regular chain is automatically *positive Harris*. In [5], for a wireless queueing system, the authors defined *stability* as satisfying: $L(s, A) := P_s(\tau_A < \infty) = 1$ where the state space S represents the queue length. However, they did not formulate the problem as a controlled Markov chain and did not use the idea of c-regular chains and regular policies and of finding the optimal policy from the chain itself.

For our case, we define *stability* as satisfying the c-regularity property for the Markov chain. In our model, the *Foster-Lyapunov drift inequality* in Theorem 2.3 plays a significant role for *queue stability* and for deriving performance bounds.

2.4 Existence of optimal policies of ψ-irreducible controlled chains

In Theorem 2.2, it is guaranteed that the limit of the average cost function exists for any regular policy. In this section, we discuss that this result is related to the existence criteria of the optimal policy for ψ-irreducible controlled chains. Specifically, we present the characterization of the triple (w^*, h, η_*) on the ACOE in (2.4) and (2.5).

We define the *cost over one cycle* η_w under a Markov policy w: For any $A \in \beta^+(S)$,

$$\eta_w := \min \left\{ \eta \geq 1 : E_s^w \left[\sum_{t=0}^{\tau_A - 1} (c_w(\Phi_t^w) - \eta) \right] \leq 0 \right\} \tag{2.13}$$

Let η_{min} be the minimal cyclic cost over all Markov policies. If the policy w is regular, then from the result in [8], the *cost over one cycle* under the stationary policy w satisfies: $\eta_w = \pi(c_w) = J(w, s)$ for $s \in S$.

The *minimal relative value function* h_{min} is defined point-wise as:

$$h_{min}(s) := \inf \left\{ E_s^w \left[\sum_{t=0}^{\tau_A - 1} (c_w(\Phi_t^w) - \eta_{min}) \right] \right\} \tag{2.14}$$

where the minimum is taken over all Markov policies.

We term a function $\underline{c}$ *norm-like* if the sub-level set $\{s : \underline{c}(s) \leq b\}$ is a finite subset of S for any finite constant b. The following assumptions are used for the existence criteria of the optimal solution to the ACOE:

$A(1)$: There exists a policy w_0, a function $V_0 : S \to \mathbb{R}_+$ and positive constant $\bar{\eta} < \infty$ satisfying the corresponding *Foster-Lyapunov* drift inequality: $P_{w_0} V_0 \leq V_0 - c_{w_0} + \bar{\eta}$. This assumption implies that there is at least one regular policy.

$A(2)$: The cost function $c(s, a)$ is *norm-like* on the product space $S \times A$, and there exists a *norm-like* function $\underline{c} : S \to \mathbb{R}_+$ such that $c(s, a) \geq \underline{c}(s)$ for any $s \in S$, $a \in A(s)$.

$A(3)$: For any Markov policy w, there exists $\theta \in \beta(S)$ and $\delta > 0$ such that $K_w(s, \theta) > \delta$ for all $s \in S_0$, where $S_0 = \{s : \underline{c}(s) \leq 2\bar{\eta}\}$. This implies that S_0 is a *petite set* for any policy w. This condition is a generalization of the Definition 2.2 in Section 2.2.

The following important theorem is from [8].

Theorem 2.5: *Suppose the Assumptions* $A(1) - A(3)$ *hold. Then:*

(i) The minimal relative value function h_{min} *is a solution to the ACOE.*

(ii) Suppose the policy w_{min} satisfies:

$$w_{min}(s) = \arg\min_{a \in A(s)} \left[c(s,a) + P_a h_{min}(s) \right], \; s \in S \tag{2.15}$$

where $P_a h_{min}(s) := E\left\{ h_{min}(\Phi_1^a) \mid \Phi_0^a = s \right\}$. *Then,* w_{min} *is optimal over all Markov policies.*

Following the proof of Theorem 2.5 in [8], it can be easily shown that the policy w_{min} in (2.15) is also a regular policy and that $\eta_{min} = \pi(c_{w_{min}}) = J(w_{min}, s)$ for $s \in S$. Hence, $\eta_* = \eta_{min}$, $h_* = h_{min}$, and $w_* = w_{min}$ as defined in (2.6) in Theorem 2.1. Now that we know that the solution exists, the next section explains an algorithm to obtain the solution.

2.5 Value Iteration Algorithm for ψ-irreducible controlled chains

The *value iteration algorithm* (VIA) is a *model-based* Dynamic Programming (DP) technique that approximates the *value function* in the optimality equation. For the average-cost criterion of ψ-irreducible controlled chains, VIA is inductively defined as follows [25]. If the *value function* V_n is given at the n^{th} iteration, the action $w^n(s)$ is defined as:

$$w^n(s) = \arg\min_{a \in A(s)} \left[c(s,a) + P_a V_n(s) \right], \; s \in S$$

The value function is then updated as follows:

$$V_{n+1}(s) = c_{w^n}(s,a) + P_{w^n}V_n(s) \tag{2.16}$$
$$= \min_{a\in A(s)}\left[c(s,a) + P_a V_n(s)\right]$$

This then makes it possible to obtain the next action $w^{n+1}(s)$. For notational convenience, we use the following:

$$c_n = c_{w^n}; \qquad P_n = P_{w^n} \tag{2.17}$$

Let E^n be the expectation operator induced by the stationary policy w^n:

$$w^n = \{w^n(\Phi_0), w^n(\Phi_1), ...\}$$

In Assumption $A(3)$ of Theorem 2.5, it is assumed that for any policy w, there exists a *distinguished state* $\theta \in \beta(S)$. We define the following functions for $s \in S$ and $n \in \mathbb{Z}_+$:

$$h_n(s) := V_n(s) - V_n(\theta); \qquad g_n(s) := V_{n+1}(s) - V_n(s)$$

The performance of the algorithm strongly depends on the initial value function V_0. The following theorem gives the necessary conditions for convergence.

Theorem 2.6: *[25, Theorem 2.2] Suppose that the value iteration algorithm is initialized with the function V_0 found in Assumption $A(1)$, and the Assumptions $A(1)-A(3)$ are satisfied. In addition, suppose that the optimal policy satisfies: $\lim_{n\to\infty}\frac{1}{n}P^n_{w^*}V_0(s) = 0$ for $s \in S$. Then:*

(i) For each s, the sequences $\{g_n(s)\}$ and $\{h_n(s)\}$ are bounded, and

$$\lim_{n\to\infty} \frac{1}{n} V_n(\theta) = \lim_{n\to\infty} g_n(\theta) = \eta_*$$
$$\limsup_{n\to\infty} \frac{1}{n} V_n(s) \le \limsup_{n\to\infty} g_n(s) \le \eta_*, \qquad s \in S$$

(ii) Each intermediate policy w^n is regular with unique invariant probability π_n, and each V_n serves as a **Lyapunov** *function for the n^{th} policy:*

$$P_n V_n \le V_n - c_n + \bar{\eta}, \quad n \ge 0$$

(iii) The average cost satisfies: $J(w^n) = \pi_n(c_n) \le \bar{\eta}_n = \sup_{x\in X} g_n(x)$, where $\pi_n(c_n)$ is the steady state cost defined in (2.8) and that $\lim_{n\to\infty} J(w^n) = \eta_*$.

(iv) Any point-wise limit point of the policies $\{w^n\}$ is regular and optimal. If h_∞ is any point-wise limit of the sequence $\{h_n(s)\}$, then the pair $\{h_\infty, \eta_*\}$ is a solution to the ACOE.

Theorem 2.6 states that if the value function is initialized with a *Lyapunov* function together with a few assumptions above, then every succeeding policy in the iteration is regular. This key result is used in the derivation of performance bounds in Chapter 3.

Chapter 3

ψ-irreducible MDP for Wireless Queueing Model

This chapter presents an application of ψ-irreducible Markov chains in Chapter 2 for a general wireless queueing model. Using the condition known as the *Foster-Lyapunov drift inequality* from Theorem 2.3, we derive results for finding the optimal policy and stability conditions for average queueing delay and congestion level.

By using the concepts of ψ-irreducible Markov chains and to the best of the author's knowledge, we present the *first* method of achieving optimization and stability conditions *simultaneously* in a general wireless Markov queueing network, and for deriving performance bounds *directly* from the control DP algorithm, as the algorithm converges to the optimal solution.

Section 3.1 discusses a general queueing model with a time-varying *channel state process*. We introduce the concepts of *rate convergence* and *channel convergence* and show that the queueing model is a controlled Markov chain using the proof of Theorem 3.1. We also consider the concept of a *topology state process* and explain how our approach can be used to include varying topology and MAC mechanisms in Section 3.2.

Section 3.3 applies the concepts of Chapter 2 for finding the optimal policy and stability conditions for our system model in Section 3.1. Section 3.4 then derives the performance bounds for average queueing delay and congestion levels from the

stability condition and control algorithm. In Section 3.5, we present and discuss simulation results obtained using the NS2 network simulator. Figure 3.1 summarizes the key ideas of this chapter.

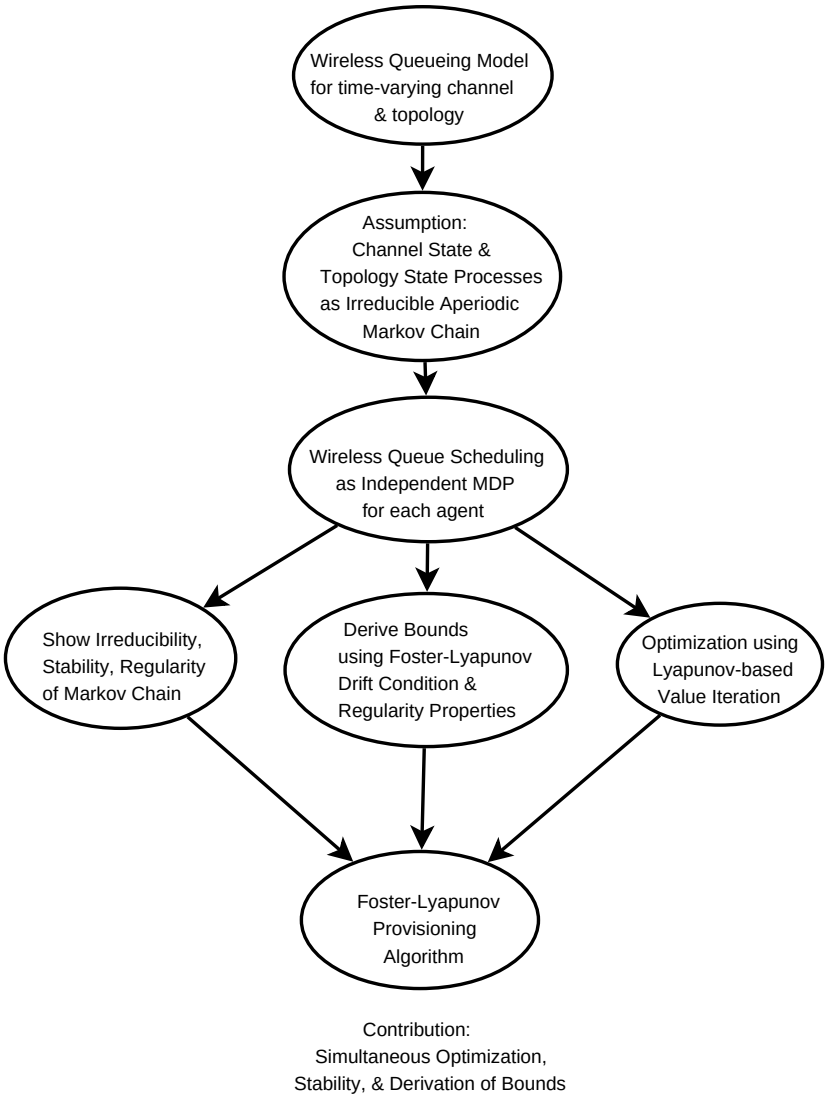


Figure 3.1: Summary of techniques for ψ-irreducible Markov chains for wireless queue scheduling

3.1 System Model: Time-Varying Channel State Process

Consider a wireless network with N nodes and J network classes. Each node is assumed to act as an agent that actively performs scheduling on its own J local queues, where packets are enqueued in their respective class queues. In each node, assuming that time is slotted, the queue dynamics at the j^{th} queue can be generally expressed for $\forall j$ as:

$$x_j(k+1) = (x_j(k) + a_j(k) - \mu_j(k))^+ \tag{3.1}$$

where:

$x_j(k)$ is the number of bits at time slot k

$a_j(k)$ is the number of bits arriving to the j^{th} queue

$\mu_j(k)$ is the number of bits that were transmitted out of the node

$(x)^+ := \max(x, 0)$

We assume that an agent observes the wireless *channel state*, denoted as $C_h(k)$, and is fixed for the entire time slot k and only changes at slot boundaries. The channel state may include the characteristics of the network that affect the transmission. It can be obtained either through direct measurement or through a combination of measurement and channel prediction. In particular, we assume that the channel process $\{C_h(k)\}_{k=1}^{\infty}$ evolves as an irreducible aperiodic finite state-space Markov chain. Let $\{C_{h,l}(k) : l = 1, .., L\}$ be the set of channel states with a total of L states. This assumption is commonly used in literature for modeling a time-varying channel process and has been shown to be valid in actual network conditions [6, 12, 26, 27, 28].

For each channel state $C_h(k)$ at time slot k, $\{\mu_j(k)\}_{j=1}^{J}$ is contained in a constraint set $\Gamma_{C_h(k)}$ that represents the set of available transmission rates for scheduling. At every time slot k, the agent chooses the service rate vector $\overline{\mu}(k) = (\mu_1(k), ..., \mu_J(k))' \in \Gamma_{C_h(k)}$. The mapping from current channel state to service rate vector is a stationary scheduling policy. Let H be the set of all stationary scheduling policies.

It is shown in the next theorem that the service rate $\mu_j(k)$ under any policy $h_0 \in H$ forms a stochastic process $\{\mu_j(k)\}_{k=0}^{\infty}$ that is *rate convergent* as defined as follows [29]:

Definition 3.1: *A stochastic process $\{A(k)\}_{k=0}^{\infty}$ is rate convergent with rate λ when:*

(i) $\lim_{k\to\infty} \frac{1}{k}\sum_{k=0}^{\infty} A(k) = \lambda < \infty$

(ii) For any $\delta > 0$, there exists an interval K such that, for any initial time k_0 and regardless of past history:

$$\left| E\left\{ \frac{1}{K}\sum_{k=0}^{K-1} A(k_0 + k) \right\} - \lambda \right| \leq \delta$$

The following theorem shows that the service process $\{\mu_j(k)\}_{k=0}^{\infty}$ is rate convergent and its rate, defined as $\mu_{j,av} := \lim_{k\to\infty} \frac{1}{k}\sum_{k=0}^{\infty} \mu_j(k)$, can be expressed from the statistics of the channel process itself $\{C_h(k)\}_{k=1}^{\infty}$.

Theorem 3.1: *Given an irreducible aperiodic finite state-space Markov chain $\{C_h(k)\}_{k=0}^{\infty}$ for the channel process, the service process $\{\mu_j(k)\}_{k=0}^{\infty}$ is* rate convergent *with rate $\mu_{j,av} = \sum_{l=1}^{L} \pi_l R_l$, where $\{\pi_l > 0; l = 1, ..., L\}$ is the steady state probability distribution of the Markov chain with L channel states; and $R_l := E\{\mu_j(k) \mid C_{h,l}(k)\}$ is defined as the expected service rate when the channel state is $C_{h,l}(k)$ under the stationary policy h_0, which is independent of past history.*

PROOF: It is commonly known that an irreducible aperiodic finite state-space Markov chain has a unique steady state probability distribution [4]: $\pi_l > 0$, for $l = 1, ..., L$. In the interval $[k_0, k_0 + K - 1]$, let $T_l(k_0, K)$ be the set of time slots where the channel state is in state l. Let $\|T_l(k_0, K)\|$ denote the number of time slots where the channel state is l in this interval. The channel process is known to be *channel convergent* [29] as the steady state probabilities of the Markov chain satisfy the following conditions:

(i) $\frac{\|T_l(k_0,K)\|}{K} \rightarrow \pi_l$ as $K \rightarrow \infty$.

(ii) For $\delta > 0$, there exists a fixed interval K such that:

$$\sum_{l=1}^{L} \left| \pi_l - \frac{E\left\{\|T_l(k_0, K)\|\right\}}{K} \right| \leq \delta$$

Following [29], the empirical rate over K slots for the service process is given as:

$$\begin{aligned} \frac{1}{K}\sum_{k=0}^{K-1} \mu_j(k) &= \frac{1}{K}\sum_{l=1}^{L} \left\{ \sum_{k \in T_l(0,K)} \mu_j(k) \right\} \\ &= \sum_{l=1}^{L} \frac{\|T_l(k_0, K)\|}{K} \left\{ \frac{1}{\|T_l(k_0, K)\|} \sum_{k \in T_l(0,K)} \mu_j(k) \right\} \end{aligned} \tag{3.2}$$

By the Law of Large Numbers, as $K \rightarrow \infty$,

$$\frac{1}{\|T_l(k_0, K)\|} \sum_{k \in T_l(k_0,K)} \mu_j(k) \rightarrow R_l$$

Hence, $\mu_{j,av} := \lim_{k \rightarrow \infty} \frac{1}{k} \sum_{k=0}^{\infty} \mu_j(k) = \sum_{l=1}^{L} \pi_l R_l$, thus proving the first condition for rate convergence in Definition 3.1. The second condition for rate convergence can be shown by taking expectations of (3.2) in the interval $[k_0, k_0 + K - 1]$:

$$\mu_{j,av} - E\left\{\frac{1}{K}\sum_{k=k_0}^{k_0+K-1}\mu_j(k)\right\} =$$

$$\sum_{l=1}^{L}\pi_l R_l - E\left\{\frac{1}{K}\sum_{l=1}^{L}\left(\sum_{k\in T_l(k_0,K)} E\left\{\mu_j(k) \mid C_{h,l}(k)\right\}\right)\right\}$$

$$= \sum_{l=1}^{L}\pi_l R_l - \sum_{l=1}^{L}\frac{E\left\{\|T_l(k_0,K)\|\right\}}{K}R_l$$

$$= \sum_{l=1}^{L} R_l\left(\pi_l - \frac{E\left\{\|T_l(k_0,K)\|\right\}}{K}\right)$$

$$\left|\mu_{j,av} - E\left\{\frac{1}{K}\sum_{k=k_0}^{k_0+K-1}\mu_j(k)\right\}\right| \leq \delta R_{max}$$

where R_{max} is the maximum value of R_l for all channel states. The last inequality is due to the channel convergent property condition. Hence, the service process is indeed rate convergent. ■

We assume that $\{a_j(k)\}_{k=0}^{\infty}$ are independent and identically distributed sequence of random variables and form a stationary and ergodic arrival process such that: $E\left[a_j(k)\right] = \lambda_j < \infty$ and its second moment are also bounded on every slot k: $E\left[a_j^2(k)\right] \leq A_{j,max}^2 < \infty$, for $j = 1, ..., J$. This arrival process can also be easily shown to be rate convergent with rate λ_j. It is also assumed to be independent of the service process above.

For any policy in H, the queue length process $\{x_j(k)\}_{k=0}^{\infty}$ is *influenced* by the Markov channel process and the independent and *rate convergent* arrival and service processes for each class j. From these concepts, we conclude that the queue length process is indeed a Markov chain. The queueing model in (3.1) for each class j is therefore a controlled Markov chain or MDP.

We propose in Section 3.3 that each agent *independently* solves its MDP based on its local observed state condition. The case of a multi-agent system where agents collaborate among themselves is discussed in Chapter 6.

In [7], the queue length process in (3.1) was shown to be an aperiodic irreducible Markov chain. This was done by assuming that the arrival process is modeled as a Markov-modulated Poisson process where the corresponding state space evolves in a countable and irreducible Markov chain. The number of arrivals is also a Poisson random variable. In our case, we relax this assumption for the arrival process. This implies that the queue length process $\{x_j(k)\}_{k=0}^{\infty}$ (i.e. queue length in bits) may not be an irreducible Markov chain in the usual sense (i.e. having only a single communicating class). As also mentioned in [5], under the constraint set $\Gamma_{C_h(k)}$ for the available transmission rates, we cannot guarantee *irreducibility*. The queue length process (in bits) cannot be irreducible since not every possible length (in bits) can be visited, obviously.

The controlled Markov chain in our model is also *not recurrent* under a stationary policy h_0, since by definition, *recurrent* chains are only concerned with irreducible chains [4].

The work in [7] also discusses stable policies under a time-varying channel. Our model is generally more applicable by relaxing the Markov-modulated Poisson arrival assumption and not having an irreducible Markov chain for the queue length process. Furthermore, we analyze *stability* and *performance optimization* simultaneously under a *stochastic control* framework by finding stable and optimal policies.

3.2 Concept of Topology State Process

In the previous section, we have discussed that each node is influenced by the *channel state process* that evolves as a Markov chain. We claim that this concept can be

extended to capture other important MANET related characteristics, such as varying topology and medium access control (MAC) mechanisms. For instance, the success of transmission depends on other nodes' attempts as well as the *topology state* of the network. The topology state includes all the characteristics of the network that affect transmission and may vary with time. It may include the changing connectivity among nodes as they move, and transmission rates in each link with changing quality. Other characteristics that may not be directly related to transmission can be also included in the topology state. This concept of topology state is applied in [27].

By considering slotted time, we can assume that the topology state, which also captures the channel state, forms a stochastic process that evolves as an irreducible aperiodic finite state-space Markov chain. Hence, we can apply the same reasoning and conclude that the queue length process in each node is influenced by the topology state of the network. Theorem 3.1 is also applicable and thus, the queue length process is a controlled Markov chain.

3.3 MDP for Wireless Queueing Model

As explained in Section 3.1, given the channel state $C_h(k)$ at time slot k, the agent chooses the service rate vector $\overline{\mu}(k) = (\mu_1(k), ..., \mu_J(k))^{'} \in \Gamma_{C_h(k)}$.

However, under the MDP formulation, we redefine the *state vector* as the queue length process itself: $\overline{x}(k) = (x_1(k), ..., x_J(k))^{'} \in X$, where X is the state space of the queue length process.

Given the current state or local class queue lengths, each agent chooses the *action vector* representing the service rate vector $\overline{\mu}(k)$. This is possible since the queue length process *locally* for each class j is already a Markov chain that obeys the following queueing law from Section 3.1:

$$x_j(k+1) = (x_j(k) + a_j(k) - \mu_j(k))^+$$

We have also shown that the service process $\{\mu_j(k)\}_{k=0}^{\infty}$ and arrival process $\{a_j(k)\}_{k=0}^{\infty}$ are independent *rate convergent* processes with rates $\mu_{j,av}$ and λ_j, respectively. Thus, the queue length process $\overline{x}(k) = (x_1(k), ..., x_J(k))^{'}$ is itself controlled Markov chain. In other words, we do not anymore consider the channel or topology state process, since the queue length process is already a Markov chain.

The mapping from current state vector $\overline{x}(k)$ (i.e. *local* queue length vector in bits) to the service rate vector $\overline{\mu}(k)$ is defined as stationary scheduling policy for the controlled Markov chain. The goal is for each agent to find the stationary policy w that minimizes the expected average long term congestion level starting with some initial state $\overline{x}(0)$:

$$J(w, \overline{x}(0)) := \lim_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n} E^{w}_{\overline{x}(0)} \{c_{all}(\overline{x}(k), \overline{\mu}(k))\} \tag{3.3}$$

where $c_{all}(\overline{x}(k), \overline{\mu}(k)) := \sum_j x_j(k)$ is defined as the buffer or congestion level of the queues. The existence of the expectation operator $E^{w}_{\overline{x}(0)}$ is due to the fact that the actual congestion levels vary depending on the channel quality and actual transmission rate. If the channel is error-free, the maximum amount of bits can be sent per time slot at the transmission rate. The expectation operator also captures the uncertainty of packet loss due to the time-varying channel and change in topology and route paths.

Figure 3.2 shows the multi-class MANET where each node acts as an agent. We emphasize that each agent *independently* solves its own MDP based on its local observed state condition (i.e. local queue lengths). As discussed in Section 3.2, this gen-

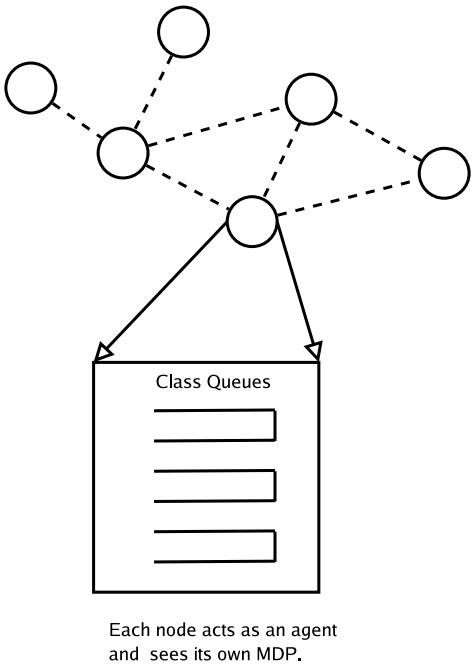


Figure 3.2: Independent MDP agents for queue scheduling in MANETs

eral system model captures varying topology, MAC mechanisms and wireless channel conditions.

The case of a multi-agent framework where agents collaborate among themselves is discussed in Chapter 6.

As shown in Section 3.1, the queue length process for each class j is already a controlled Markov chain in itself. For ease of presentation in the following sections, we only consider per-class processes, such as the arrival, service and queue length processes for class j in a single node.

It should also be noted that in the MDP formulation, each node does not need the channel state and traffic arrival statistics, since the local queue lengths for each class j already evolve as a controlled Markov chain.

3.3.1 ψ-irreducibility

After defining the MDP, the next step is to establish the concept of ψ-irreducibility for the controlled Markov chain.

Let $d_j(k) = a_j(k) - \mu_j(k)$. Since the arrival and service processes are independent, the process $D_j = \{d_j(k)\}_{k=0}^{\infty}$ itself is also rate convergent with rate $(\lambda_j - \mu_{j,av})$ and these *increment* variables $d_j(k)$ are i.i.d. random variables taking integer values in $\mathbb{Z}$.

Let $\Gamma_j(z) = P(d_j(k) = z)$, $z \in \mathbb{Z}$ be the probability distribution of the process D_j. The queue length for each class j evolves as a *random walk on a half line*:

$$x_j(k+1) = (x_j(k) + d_j(k))^+ \tag{3.4}$$

For any $B \in \mathbb{Z}_+$, the probability distribution for the class queue j process with initial condition of $x_j(0) = x_0$, $x_0 \in \mathbb{Z}_+$ can be specified as:

$$\begin{aligned} P(x_0, B) &= P(B = x_0 + d_j(1) \mid x_0) \\ &= P(d_j(1) = B - x_0) = \Gamma_j(d_j(1) = B - x_0) \end{aligned}$$

This implies that if $P(x_0, B) = \Gamma_j(d_j(1) = B - x_0) > 0$, then any succeeding queue length $B > 0$ can be reached. On the other case, we have for $x_0 > 0$:

$$\begin{aligned} P(x_0, \{0\}) &= P(x_0 + d_j(1) \leq 0 \mid x_0) \\ &= P(d_j(1) \leq -x_0) = \Gamma_j(d_j(1) \leq -x_0) \end{aligned}$$

It is clear that if $\Gamma_j(-\infty, 0) := P_j(d_j(k) \leq 0) > 0$ for any $k \geq 0$, then the chain reaches the empty queue $\{0\}$.

Formally, suppose for some $\delta, \epsilon > 0$, $\Gamma_j(-\infty, -\epsilon) > \delta$. Then for any n, if $x_0/\epsilon < n$

for $x_0 > 0$ then,

$$P^n\left(x_0, \{0\}\right) \geq \delta^n > 0 \tag{3.5}$$

We see that the queueing model in (3.4) is indeed ψ-irreducible if $\Gamma_j\left(-\infty, 0\right) := P\left(d_j(k) \leq 0\right) > 0$, so that the empty queue $\{0\}$ for class j is *accessible* from *any* initial condition $x_j(0) = x_0$, $x_0 \in \mathbb{Z}_+$. This also satisfies the conditions of Lemma 2.1 under the *countable* state space model.

In [23, Proposition 4.3.1], it is shown that the measurable function ψ satisfies: $\psi(B) = \sum_{t=0}^{\infty} 2^{-t} P^t(0, B)$ for $B \in \beta(X)$ under the state space X (i.e. queue length x in bits in our scheduling problem).

From (3.5), we can also conclude that for any $x \in C = [0, x_0]$, the probability of reaching $\{0\}$ starting from x is bounded:

$$P_x\left(\tau_0 \leq \frac{x_0}{\epsilon}\right) \geq \delta^{1+\frac{x_0}{\epsilon}} \tag{3.6}$$

In other words, the compact set $C = [0, x_0]$ is a *petite set* from Definition 2.2 with the accessible set $\{0\}$ for any $x_0 > 0$ provided that $\Gamma_j\left(-\infty, 0\right) := P\left(d_j(k) \leq 0\right) > 0$.

We make the observation that ψ-irreducibility is easier to verify and more applicable than the standard definition of *irreducibility* of only having a single communicating class under a countable state space model.

3.3.2 Queue Stability

In Section 2.3, we define that a Markov chain is *stable* if it satisfies the c-regularity property. In applying this concept for our queueing model, we separately define *queue stability* as follows:

Definition 3.2: *A queue is stable if the queue length process* $\{x_k\}_{k=0}^{\infty}$ *satisfies:*

$$\limsup_{M\to\infty} \frac{1}{M} \sum_{k=0}^{M-1} E_x \{x_k\} < \infty \tag{3.7}$$

We establish in the next subsection that the c-regularity property of the Markov chain leads to queue stability from this definition.

3.3.3 Establishing c-regularity and Queue Stability

In showing that the ψ-irreducible chain in (3.4) is c-regular, we use *Foster-Lyapunov drift inequality* condition in Theorem 2.3. We show that the rate convergent process $D_j = \{d_j(k)\}_{k=0}^{\infty}$ affects the c-regularity property. We have the following result:

Theorem 3.2: *Let* $\beta_j = (\lambda_j - \mu_{j,av})$ *denote the rate of the process* $D_j = \{d_j(k)\}_{k=0}^{\infty}$ *with probability distribution* $\Gamma_j(z) = P(d_j(k) = z)$, $z \in \mathbb{Z}$. *The queue length process for class* j *is c-regular if* $\beta_j < 0$.

PROOF: Since $\beta_j < 0$, there exists a finite $x_0 > 0$ such that:

$$\sum_{z=-x_0}^{\infty} z\Gamma_j(z) \leq \frac{\beta_j}{2} < 0 \tag{3.8}$$

For the *countable* state space Markov chain Φ_t^w which represents the queue length process under the Markov policy w, the *Foster-Lyapunov drift inequality* can be established as follows. For any $x, y \geq 0$,

$$P_w V(x) := E\left\{V(y = \Phi_{t+1}^w) \mid \Phi_t^w = x\right\} = \sum_{y=0}^{\infty} P(x,y)V(y)$$

Let $\triangle V(x) := P_w V(x) - V(x)$ and $z = (y - x)$ for $z \in \mathbb{Z}$. We choose the Lyapunov function $V(x) = x^2$. Then for any $x \geq 0$,

$$\begin{aligned}
\triangle V(x) &= \sum_{y=0}^{\infty} P(x,y)V(y) - V(x) \\
&= -V(x) + \sum_{z=-x}^{\infty} P(y = x+z \mid x)V(x+z) \\
&= -x^2 + \sum_{z=-x}^{\infty} \Gamma_j(z)(x+z)^2 \\
&= -x^2 + x^2 \sum_{z=-x}^{\infty} \Gamma_j(z) + 2x \sum_{z=-x}^{\infty} z\Gamma_j(z) + \sum_{z=-x}^{\infty} z^2\Gamma_j(z)
\end{aligned}$$

From Section 3.3.1, we have shown that the compact set $C = [0, x_0]$ is petite for $x_0 > 0$. Assuming that the distribution $\Gamma_j(z)$ has a finite second order moment (i.e. $\sum_{z=-\infty}^{\infty} z^2\Gamma_j(z) < \infty$), and by the finite negative rate β_j in (3.8) and the fact that $\sum_{z=-x}^{\infty} \Gamma_j(z) \leq 1$, then for any $x \geq 0$:

$$\triangle V(x) := P_w V(x) - V(x) \leq -f_0 c_w(x) + d_0 \delta_C(x) \tag{3.9}$$

for some constants $f_0 > 0$, $d_0 < \infty$, where $c_w(x) = (x+1)$ and the indicator function for the petite set $C = [0, x_0]$, for some $x_0 > 0$, is defined as: $\delta_C(x) = 1$ if $x \in C$ or 0 otherwise.

Let $\theta = \{0\}$ be the accessible state with τ_θ as the first return time of the chain Φ_t^w under policy w. By applying Theorem 2.4, we have for any $x \geq 0$:

$$V(\theta) + f_0 E_x^w \left\{ \sum_{t=0}^{\tau_\theta - 1} c_w(\Phi_t) \right\} \leq V(x) + d_0 E_x^w \left\{ \sum_{t=0}^{\tau_\theta - 1} \delta_C(\Phi_t) \right\}$$

Following the idea of Assumption $A(3)$ of Theorem 2.5, for the petite set C and for any Markov policy w, there exists a constant $m_0 > 0$ such that $K_w(x, \theta) \geq m_0$ for all $x \in C$. This is possible from (3.6) and due to the petiteness property in Definition

2.2.

It is shown in [25, Lemma A.3] that for the petite set C and an accessible set $\{\theta\}$ satisfying $K_w(x,\theta) \geq m_0$ for all $x \in C$, the following inequality holds:

$$E_x^w \left\{ \sum_{t=0}^{\tau_\theta - 1} \delta_C(\Phi_t) \right\} \leq \frac{1}{m_0}$$

Hence, the following inequality follows:

$$E_x^w \left\{ \sum_{t=0}^{\tau_\theta - 1} c_w(\Phi_t) \right\} \leq \frac{V(x) + d_0/m_0}{f_0}$$

Since $V(x) = x^2$ is bounded for any $x \in C$, the right hand side of this inequality is bounded and thus, by definition in (2.7), the set C is regular. Furthermore, each of the sub-level sets $C_n = \{x : V(x) \leq n\}$ for $n \in \mathbb{Z}_+$ is regular, since every petite set $C = [0, x_0]$ for any $x_0 > 0$ satisfies the inequalities above. Therefore, the process is itself c_w-regular and there exists a *regular* Markov policy w. ■

We now show that c-regularity leads to queue stability by the following result.

Lemma 3.1: *If the queue length process is c-regular, then the queue is* stable *from Definition 3.2.*

PROOF: We rewrite the *Foster-Lyapunov drift inequality* in (3.9) as follows:

$$\triangle V(x)/f_0 \leq -c_w(x) + d_0 \delta_C(x)/f_0$$

Since $f_0 > 0$ and we only require the *Lyapunov* function $V(x) \geq 0$, we can replace this function with a *normalized* version $V_0(x) := V(x)/f_0$ and the c-regularity property shown earlier still holds. In other words, the drift inequality for the chain

Φ_t^w can be expressed as:

$$P_w V_0(x) - V_0(x) \leq -c_w(x) + \overline{\eta} \tag{3.10}$$

for positive constant $\overline{\eta} < \infty$ and any $x \in X$. Furthermore, this can be rewritten for all $t \geq 0$, under a regular policy w as:

$$E_x^w \{V_0(x_{t+1}) - V_0(x_t) \mid x_t\} \leq -c_w(x_t) + \overline{\eta} \tag{3.11}$$

where $\Phi_t^w := x_t$. Taking expectations of this inequality over the distribution of x_t, and summing over t from $t = 0$ to $t = M - 1$ for some $M \in \mathbb{Z}_+$ yields:

$$E_x^w \{V_0(x_M) - V_0(x_0)\} \leq M\overline{\eta} - E_x^w \left\{ \sum_{t=0}^{M-1} c_w(x_t) \right\}$$

Dividing by M and using the fact that $V_0(x_M) \geq 0$, we have:

$$\tfrac{1}{M} E_x^w \left\{ \sum_{t=0}^{M-1} c_w(x_t) \right\} \leq \tfrac{1}{M} E_x^w \{V_0(x_0)\} + \overline{\eta}$$

By taking note that $c_w(x_t) = x_t + 1$, we have:

$$\limsup_{M \to \infty} \frac{1}{M} \sum_{t=0}^{M-1} E_x^w \{x_t\} \leq \overline{\eta} - 1 < \infty \tag{3.12}$$

The last inequality thus proves queue stability as defined in Definition 3.2. ■

3.3.4 Using the Value Iteration Algorithm

After we have shown c-regularity and queue stability, the next step is then to find the optimal policy using the value iteration algorithm. Specifically, we show how the different assumptions of the value iteration algorithm in Theorem 2.6 are satisfied as

follows:

Assumption $A(1)$ of Theorem 2.6 requires the existence of a regular policy w_0 satisfying the corresponding *Foster-Lyapunov* drift inequality: $P_{w_0}V_0 \leq V_0 - c_{w_0} + \bar{\eta}$. This condition is satisfied as shown in the proof of Theorem 3.2 and particularly in (3.11) of Lemma 3.1.

As defined in Section 2.4, a function $\underline{c}$ is *norm-like* if the sub-level set $\{x : \underline{c}(x) \leq b\}$ is a finite subset of X (i.e. *state space* of queue length) for any finite constant b. The cost function $c_w(x) = (x+1)$ is clearly *norm-like* for any x in the petite set $C = [0, x_0]$, for $x_0 > 0$ as shown earlier, since $c_w(x)$ is bounded and finite in the set C. For Assumption $A(2)$ to be satisfied, we require another norm-like function $\underline{c} : X \to \mathbb{R}_+$ such that $c(x,a) \geq \underline{c}(x)$ for any $x \in X$, $a \in A(x)$. This can be clearly met by the indicator function $\underline{c}(x) := \delta_C(x)$ in (3.9) for the petite set C.

For Assumption $A(3)$, the existence of a distinguished state $\theta \in \beta(X)$ for any Markov policy w is satisfied by the state $\{0\}$. Specifically, for the petite set C and any Markov policy w, there exists a constant $m_0 > 0$ such that $K_w(x, \theta) \geq m_0$ for all $x \in C$. This has been shown in the proof of Theorem 3.2. This is again possible from (3.6) and due to the petiteness property in Definition 2.2.

Finally, for any Markov policy w, $P_w^n V_0(x) := E\left\{V_0(\Phi_t^w) \mid \Phi_0^w = x\right\}$. As also shown in the proof of Theorem 3.2, each of the sub-level sets $C_n = \{x : V_0(x) \leq n\}$ for $n \in \mathbb{Z}_+$ is regular. Since in a c-regular Markov chain, there is a countable covering of the state space X by c-regular sets from Section 2.3, we can say that the *Lyapunov* function $V_0(x)$ is bounded for all $x \in X$. Hence, for any regular policy w, $\lim_{n\to\infty} \frac{1}{n} P_w^n V_0(x) = 0$ for $x \in X$. From Theorem 2.5, the optimal policy w^* is also a regular policy. Hence, the final assumption of Theorem 2.6 is satisfied.

3.4 Performance Bounds

In this section, we shall use the ideas of Chapter 2 and the results in the previous sections to derive performance bounds, as the algorithm converges to the optimal solution.

3.4.1 Application of Value Iteration Algorithm

In Lemma 3.1, we have shown that under any Markov policy w, the Markov chain for the wireless queueing model is c-regular and hence, the queue length process for class j is stable. In Section 3.3.4, we have shown how the value iteration algorithm can be used with respect to our model by discussing its conditions and assumptions.

Theorem 2.6 states that if the algorithm is initialized with a regular policy w_0 then every intermediate policy w_n in the iteration is also regular for $n \in \mathbb{Z}_+$. This then leads us to the following result:

Theorem 3.3: *The value iteration algorithm with an initial regular policy w_0 stabilizes the queueing model and yields the time average congestion bound:*

$$\limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} E\{x_\tau\} \leq \bar{\eta} - 1$$

PROOF: Following the notations of Section 2.5 and Theorem 2.6, we have for any $x \in X$:

$$P_n V_n(x) \leq V_n(x) - c_n(x) + \bar{\eta}, \quad n \geq 0$$

where $c_n(x) = c_{w^n}(x)$ and $P_n = P_{w^n}$ from (2.17). Let Φ_S denote the actual sample path, or the set of states visited by the chain as it evolves through slotted time $k \geq 0$. We assume that, without loss of generality, $n = k$, where n is the *index of iteration* from the algorithm itself. In other words, at every time slot k, we allow

the policy to change according to the value iteration algorithm itself. From this assumption, the sample path is characterized by the set of states that were visited from the intermediate policies w^n at every time slot k: $\Phi_S = \{x_k : \Phi_k^{w^n} = x_k\}$ for $k \geq 0$.

As $n \to \infty$, let $w_S := \{w^0(x_0), w^1(x_1), ..., w^n(x_k)\}$ be the *point-wise limit* policy which denotes the policy that is obtained from the execution of the intermediate policies w^n at each time slot k. From construction, the sample path itself Φ_S is also a c-regular Markov chain and the corresponding *Foster-Lyapunov drift inequality* holds for $k \geq 0$:

$$P_k V_k(x_k) := E_x^{w_S} \{V_k(x_{k+1}) \mid x_k\} \leq V_k(x_k) - c_k(x_k) + \bar{\eta} \tag{3.13}$$

Taking the expectation of the inequality in (3.13) over the distribution of x and using the fact that $c_{w_S}(x) = c_k(x)$,

$$E_x^{w_S} \{V_k(x_{k+1}) - V_k(x_k)\} \leq -E_x^{w_S} \{c_{w_S}(x_k)\} + \bar{\eta} \tag{3.14}$$

We note that at time slot k, the corresponding *Lyapunov* function V_k is different from the previous time slot function, V_{k-1}. We use the relation in (2.16) for two consecutive *Lyapunov* functions in the value iteration for $x \in X$ as:

$$\begin{aligned} V_{k+1}(x) &:= c_k(x) + P_k V_k(x) \\ &= c_k(x) + E_x^{w_S} \{V_k(x) \mid x\} \end{aligned} \tag{3.15}$$

For notational convenience, let $c(x) = c_{w_S}(x)$ and $E = E_x^{w_S}$. For the state x_{k+1},

and by using (3.15) and (3.14), we have the following:

$$E\left\{V_{k+1}(x_{k+2}) - V_{k+1}(x_{k+1})\right\} \leq -E\left\{c(x_{k+1})\right\} + \bar{\eta} \tag{3.16}$$

$$\begin{aligned} &E\left\{c(x_{k+2}) + V_k(x_{k+2})\right\} - E\left\{c(x_{k+1}) + V_k(x_{k+1})\right\} \\ &\leq -E\left\{c(x_{k+1})\right\} + \bar{\eta} \\ E\left\{V_k(x_{k+2}) - V_k(x_{k+1})\right\} &\leq -E\left\{c(x_{k+2})\right\} + \bar{\eta} \end{aligned} \tag{3.17}$$

The inequality in (3.17) expresses the difference of the expected values of the *Lyapunov* function V_k for two consecutive sample points: x_{k+1} and x_{k+2}. The motivation for this is to rewrite the corresponding *Foster-Lyapunov drift inequality* in (3.16) with V_{k+1} in terms of only the *Lyapunov* function V_k. In other words, we fix the *Foster-Lyapunov* function as V_0 and express the succeeding drift conditions in terms of V_0 alone. We can thus rewrite (3.14) and (3.17) as:

$$E\left\{V_0(x_{k+1}) - V_0(x_k)\right\} \leq -E\left\{c(x_k)\right\} + \bar{\eta} \tag{3.18}$$

$$E\left\{V_0(x_{k+2}) - V_0(x_{k+1})\right\} \leq -E\left\{c(x_{k+2})\right\} + \bar{\eta} \tag{3.19}$$

Similarly, for x_{k+2} , x_{k+3} and x_{k+4}, we can easily derive the following inequalities:

$$E\{V_0(x_{k+3})\} - E\{V_0(x_{k+2})\}$$
$$\leq -E\{2c(x_{k+3})\} + E\{c(x_{k+2})\} + \bar{\eta} \tag{3.20}$$
$$E\{V_0(x_{k+4})\} - E\{V_0(x_{k+3})\}$$
$$\leq -E\{3c(x_{k+4})\} + E\{2c(x_{k+3})\} + \bar{\eta} \tag{3.21}$$
$$E\{V_0(x_{k+5})\} - E\{V_0(x_{k+4})\}$$
$$\leq -E\{4c(x_{k+5})\} + E\{3c(x_{k+4})\} + \bar{\eta} \tag{3.22}$$

Continuing in this manner from x_k to x_{k+M} for some $M \in \mathbb{Z}_+$ and summing the corresponding inequalities in (3.18) to (3.22), we have for $k \geq 0$:

$$E\{V_0(x_{k+M})\} - E\{V_0(x_k)\}$$
$$\leq -E\{(M-1)c(x_{k+M})\} - E\{c(x_k)\} + M\bar{\eta} \tag{3.23}$$

To derive the required bounds from (3.23), we let $k = iM + t_0$ for $t_0 \in \{0, ..., M-1\}$. Summing over i from $i = 0$ to $i = K-1$ for some $K \in \mathbb{Z}_+$ creates a telescoping series yielding:

$$E\{V_0(x_{t_0+KM})\} - E\{V_0(x_{t_0})\} \leq KM\bar{\eta}$$
$$-E\{(M-1)c(x_{t_0+KM})\} - E\{c(x_{t_0})\}$$
$$-M\sum_{i=1}^{K-1} E\{c(x_{t_0+iM})\}$$

Dividing by K and using the fact that $V_0(x_{t_0+KM}) \geq 0$ for a *Lyapunov* function,

$$\frac{M}{K}\sum_{i=0}^{K-1} E\left\{c(x_{t_0+iM})\right\} + \frac{(M-1)}{K} E\left\{c(x_{t_0+KM}) - c(x_{t_0})\right\}$$
$$\leq M\bar{\eta} + \frac{1}{K} E\left\{V_0(x_{t_0})\right\}$$

Summing over $t_0 \in \{0, ..., M-1\}$ and dividing by M^2 yields:

$$\frac{1}{KM}\sum_{\tau=0}^{MK-1} E\left\{c(x_\tau)\right\}$$
$$+\frac{(M-1)}{MK}\frac{1}{M}\sum_{t_0=0}^{M-1} E\left\{c(x_{t_0+KM}) - c(x_{t_0})\right\}$$
$$\leq \bar{\eta} + \frac{1}{KM^2}\sum_{t_0=0}^{M-1} E\left\{V_0(x_{t_0})\right\}$$

By taking lim sup of the inequality above as $K \to \infty$ and noting that $c(x) = x+1$, we have:

$$\limsup_{t\to\infty} \frac{1}{t}\sum_{\tau=0}^{t-1} E\left\{x_\tau\right\} \leq \bar{\eta} - 1 \tag{3.24}$$

Hence, the time average congestion bound is satisfied as the value iteration algorithm *converges* in the iteration. In addition, by Little's Theorem, the average queue congestion level for class j is proportional to the average bit delay $D_{j,bit}$. In other words, $D_{j,bit} \leq \frac{\bar{\eta}-1}{\lambda_j}$ where λ_j is the rate of the *rate convergent* arrival process for class j. ■

3.4.2 Application of Queueing Law

The *Foster-Lyapunov drift inequality* in (3.10) has been established using the following arguments:

(i) ψ-irreducible controlled Markov chain with $\beta_j = (\lambda_j - \mu_{j,av}) < 0$

(ii) Existence of the petite set $C = [0, x_0]$ for any $x_0 > 0$ under any Markov policy w, such that $K_w(x, \theta) \geq m_0 > 0$ in the proof of Theorem 3.2.

(iii) *Lyapunov* function $V_0(x) = x^2/f_0$ for some constant $f_0 > 0$ and a positive constant $\overline{\eta} < \infty$.

(iv) Application of Theorem 2.4

We emphasize that this methodology is the general manner of validating the *Foster-Lyapunov drift inequality* from a ψ-irreducible controlled Markov chain.

In this subsection, we also claim that the bound parameter $\bar{\eta}$ in (3.24) can be expressed in terms of the parameters of the independent *rate convergent* arrival and service processes. Specifically, the *Foster-Lyapunov drift inequality* can also be verified using the queueing model itself.

Theorem 3.4: *From the* rate convergent *arrival process* $\{a_j(k)\}$ *and service process* $\{\mu_j(k)\}$ *with rates* λ_j *and* $\mu_{j,av}$*, respectively, such that* $\beta_j = (\lambda_j - \mu_{j,av}) < 0$*, the following drift condition holds for class* j *and for all* $k \geq 0$*.*

$$E\{V(x_{k+1}) - V(x_k) \mid x_k\} \leq -c(x_k) + \overline{\eta}$$

where $V(x) := \frac{x^2}{2(\mu_{j,av} - \lambda_j)}$*;* $c(x_k) = (x_j(k) + 1)$*;* $\overline{\eta} = \left(\frac{m_j}{2(\mu_{j,av} - \lambda_j)} + 1\right)$*; and* m_j *is the second moment of the* increment *process* $\{a_j(k) - \mu_j(k)\}$ *for all* $k \geq 0$*.*

PROOF: The queue length process is a Markov chain that obeys the following queueing law from (3.1):

$$x_j(k+1) = (x_j(k) + a_j(k) - \mu_j(k))^+$$

This can be rewritten as follows:

$$x_j(k+1) \leq \max\left(x_j(k) + a_j(k) - \mu_j(k), 0\right)$$

The expression above is an inequality because new arrivals may depart before the current slot interval is finished. By letting $d_j(k) = (a_j(k) - \mu_j(k))$ and squaring both sides, we have:

$$x_j^2(k+1) \leq x_j^2(k) + d_j^2(k) + 2x_j(k)d_j(k)$$

Since the sequence $\{d_j(k) : k \geq 0\}$ is i.i.d with common mean $\beta_j = (\lambda_j - \mu_{j,av}) < 0$ and a common second moment $m_j := E_d\left\{d_j^2(k)\right\} > 0$, and by taking expectations with respect to $d_j(k)$, we have:

$$x_j^2(k+1) \leq x_j^2(k) + m_j + 2\beta_j x_j(k)$$

By taking $V(x_k) = \frac{x_j^2(k)}{2(\mu_{j,av} - \lambda_j)}$ as the *Lyapunov* function and letting $c(x_k) = (x_j(k) + 1)$, and taking expectations with respect to $x_j(k)$, we can write:

$$E\left\{V(x_{k+1}) - V(x_k) \mid x_j(k)\right\} \leq \overline{\eta} - c(x_k) \tag{3.25}$$

where $\overline{\eta} := \frac{m_j}{2(\mu_{j,av} - \lambda_j)} + 1$. The inequality in (3.25) thus satisfies the *Foster-Lyapunov drift inequality*. Applying the result of Lemma 3.1 and (3.12), we can easily deduce that: $f_0 = 2\left(\mu_{j,av} - \lambda_j\right)$. ■

3.4.3 Relationship among Class Queues

In our analysis above, we derive theorems and performance bounds for a single class queue j, where each queue length process for class j is already a controlled

ψ-irreducible Markov chain.

In this subsection, we emphasize that the actual *Foster-Lyapunov drift inequality* for the MDP formulation in Section 3.3 can be expressed by summing the conditions of Theorem 3.4 for all j as follows:

$$E\left\{V(\bar{x}_{k+1}) - V(\bar{x}_k) \mid \bar{x}_k\right\} \leq -c(\overline{x}_k) + \overline{\eta}_{all} \tag{3.26}$$

where:

$$\begin{aligned}
\overline{x}_k &= (x_1(k), ..., x_J(k))' \\
\overline{\eta}_{all} &= \sum_{j=1}^{J} \left(\frac{m_j}{2\left(\mu_{j,av} - \lambda_j\right)} + 1 \right) \\
m_j &= E\left\{ \left(a_j(k) - \mu_j(k)\right)^2 \right\} \\
V(\bar{x}_k) &= \tfrac{1}{2} \sum_{j=1}^{J} \tfrac{x_j^2(k)}{(\mu_{j,av} - \lambda_j)} \\
c(\overline{x}_k) &= \sum_{j=1}^{J} \left(x_j(k) + 1\right)
\end{aligned}$$

The actual cost or congestion level $c_{all}(\overline{x}(k), \overline{\mu}(k))$ in Section 3.3 can be easily deduced from the cost function $c(\overline{x}_k)$ in (3.26).

In summary, in Section 3.3, we showed ψ-irreducibility and c-regularity of the wireless queueing model, which then leads to queue stability as defined in Definition 3.2. We then discussed how the assumptions of the value iteration algorithm in Theorem 2.6 can be satisfied.

In Section 3.4, we applied the modified value iteration algorithm initialized with a Lyapunov function so that queue stability is satisfied, as the algorithm converges to the optimal solution. Using this technique and the *Foster-Lyapunov drift condition*, we are able to derive bounds *directly* from the algorithm.

3.5 Simulation Results

A wireless multi-hop network of 20 mobile nodes in a 1,000m by 1,000m area is simulated in the NS2 simulator [30]. We use the IEEE 802.11 Distributed Coordination Function (DCF) for the MAC. For the routing protocol, the Ad hoc On-Demand Distance Vector (AODV) protocol is used [31]. A two-ray ground reflection model is used for the radio propagation model. The nodes are simulated with a speed of 0 to 10m/s with a random way-point mobility model and varying pause times. The maximum channel capacity is 2 Mbps, while the size of the network interface and routing protocol queues have a depth of 50 and 100 packets, respectively. The simulation is done for 3,000 seconds.

We define three traffic classes and simulate eight long-lived Constant Bit Rate (CBR) connections with the characteristics shown in Table 3.1. We choose CBR flows since this type of flows captures the worst case and average long term performance. However, we emphasize that our theoretical results still apply for other types of traffic such as Pareto and Exponential ON/OFF sources. The control packets from the routing protocol are marked as Class I and the data packet size is 64 bytes.

Figure 3.3 shows the simulation scenario with eight CBR flow connections. For example, S1 represents the source, D1 is the destination, and the arrows from node S1 to D1 represent the flow path. The dotted line represents the wireless link, while the unlabeled nodes represent intermediate nodes. The source traffic is classified according to its type in Table 3.1.

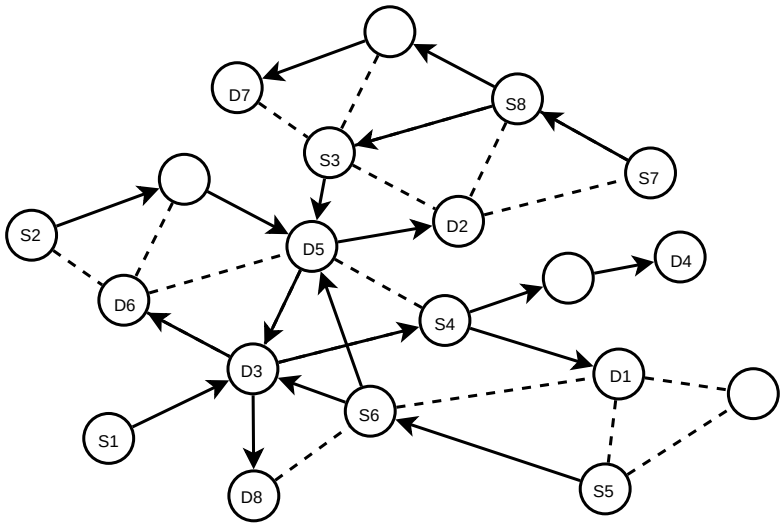


Figure 3.3: MANET Simulation Scenario with eight flows

Table 3.1: Traffic Source Characteristics

Traffic Source	Traffic Type	Rate (kbps)
Nodes 1 & 2	I	128
Nodes 3 & 4	III	32
Nodes 5 & 6	II	100
Nodes 7 & 8	III	128

We have discussed in Section 3.2 that the MAC mechanisms and varying mobility and topology issues are captured in our model through the concept of a *topology state process* that evolves as an irreducible aperiodic Markov chain. From the same reasoning, the queue length process is also influenced by the topology state and thus, our results and theorems apply in this case. It should also be noted that there have been a number of works that analyze the performance of DCF in MANETs using the Markov chain theory [32, 33]. Our approach is different because we formulate the problem using the ψ-irreducibility framework for a *controlled* Markov chain for each node acting as an agent.

We note that the value iteration algorithm in Section 2.5 requires the state tran-

sition probabilities of the Markov chain. Following the notations of Section 2.5, we can define the *state-action* values as:

$$Q_n(x,a) = c(x,a) + P_a V_n(x)$$

The stochastic value function V_n in (2.16) can then be expressed as:

$$V_n(x) = \min_{a \in A} Q(x_n, a)$$

The state-action values can also be computed using the equivalent formulation of value iteration as:

$$Q_{n+1}(x,a) = c(x,a) + P_a \left(\min_{b \in A} Q(x,b) \right) \tag{3.27}$$

Theorem 2.6 states that if the value function V_n is initialized with a *Lyapunov* function, then every succeeding policy in the iteration is regular. In the state-action formulation, only the *minimum* state-action value is initialized with the *Lyapunov* function as presented in Section 3.4.3.

For simulation purposes and due to the fact that estimating the state transition probabilities of the Markov chain is a non-trivial task, we make use of the sample-based or *model-free* framework known as *Neuro-Dynamic Programming* (NDP) [10], also known as *Reinforcement Learning* (RL) [11]. RL is a simulation-based method where the optimal value function is approximated while the agent directly interacts with the environment *without* the need of the state transition probabilities. Following [22], we use RL for ψ-irreducible controlled Markov chains. We summarize the RL-based value iteration algorithm for the average cost criterion as follows [34]: If action a_n is chosen at the n^{th} decision period with state x_n, the corresponding state-action value $Q_n(x_n, a_n)$ is updated as:

$$Q_{n+1}(x_n, a_n) = Q_n(x_n, a_n) + \alpha_n \left[c(x_n, a_n)\right.$$
$$\left. -\eta_n + \min_{b \in A} Q_n(x_{n+1}, b) - Q_n(x_n, a_n)\right] \tag{3.28}$$

where α_n is the learning rate parameter in the n^{th} decision period. The estimate of the average cost η_n is updated as:

$$\eta_n = (1 - \beta_{n-1})\eta_{n-1} + \beta_{n-1}\frac{T(n-1)\eta_{t-1} + c(x_n, a_n)}{T(n)} \tag{3.29}$$

where β_n denote the learning rate parameter and $T(n)$ is the total time spent until the n^{th} decision period. If each action is executed in each state an *infinite* number of times and all the states are visited while the learning rates β_n and α_n are decayed appropriately, the algorithm converges to the optimal solution [34]. The optimal policy in this case can be obtained from the action with the minimum state-action value: $w^n(x) = \arg\min_{a \in A} Q(x, a)$.

As mentioned earlier, the optimal solution can be obtained after an *infinite* visits to each state. In dealing with real problems, RL facilitates this idea by using exploration in the selection of actions. Specifically, with a small probability p_n, actions other than that of the minimum state-action value are chosen. In decaying the parameters β_n, α_n, and p_n, the Darken-Chang-Moody search-then-converge algorithm [35] is used.

We emphasize that RL is only used in this section for simulation purposes only and for updating the state-action value in (3.27), together with the Lyapunov function as described in Section 3.4.3. In Chapter 4, we shall discuss RL in greater details.

In deciding the action for the service rate vector, we perform *bandwidth allocation* and *provisioning* among the class queues. We use the work-conserving scheduler known as *worst-case fair weighted fair queueing* (WF^2Q) [36]. Specifically, the RL

algorithm *learns* the WF^2Q weights for each class queue.

WF^2Q is applicable, since in Theorem 3.1, we have proved that the amount of bits being transmitted out of the queue is *rate convergent* with rate $\mu_{j,av}$ for class j. We note that this result of *rate convergence* is only valid if the *topology state* process evolves as an irreducible aperiodic Markov chain. We emphasize that, since the service rate process $\{\mu_j(k)\}_{k=0}^{\infty}$ is rate convergent, then WF^2Q is applicable, and not the other way around.

Essentially, each node acts as an intelligent RL agent that finds the best WF^2Q weights depending on the current state of its ψ-irreducible controlled Markov chain (i.e. *local* class queue lengths). WF^2Q is used by each node to adaptively provision $\sum_{j=1}^{J}\mu_{j,av}$ among its J local class queues. We shall refer the proposed *adaptive* solution as the *Foster-Lyapunov Provisioning* (FLP) algorithm.

We compare its performance with that of *static* provisioning where the WF^2Q weights are equal and fixed among the classes. We shall also verify the stability conditions and performance bounds that were derived in the previous sections.

To show the *Foster-Lyapunov* condition, c-regularity and *queue stability*, we require that $\beta_j = (\lambda_j - \mu_{j,av}) < 0$ for class j. We refer to λ_j and $\mu_{j,av}$ as the *effective* arrival and service rates which are measured in a node using a time window mechanism under NS2.

Table 3.2 shows the effective arrival and service rate measurements, averaged over the simulation period and over all nodes. This clearly shows that $\beta_j < 0$ for all classes. In addition, we compare the average congestion level measurements and the theoretical bound computed as: $\left(\frac{m_j}{2(\mu_{j,av}-\lambda_j)}\right)$, where m_j is the second moment of the *increment* process $\{a_j(k) - \mu_j(k)\}$, which is obtained from the data samples and discussed in Theorem 3.4. We also show the normalized congestion bound for each class calculated as: $\left(\frac{AveBits_j}{MaxBits_j}\right)$, where $AveBits_j$ is shown in the 5^{th} column and $MaxBits_j$ is the maximum possible amount of bits in queue j. The simulation

results show that the measured average congestion level is well within the theoretical congestion bound under FLP for each class.

Table 3.2: Average Measurements under FLP for 5 secs pause time

Traffic Type	Average Arrival Rate (bps)	Average Service Rate (bps)	Average Congestion (bits)	Theoretical Bound for Average Congestion (bits)	Normalized Theoretical Bound
I	5866.04	5879.90	2304.60	3806.72	0.1487
II	3107.09	3117.13	1088.14	2677.76	0.1046
III	8255.53	8274.54	3162.71	3486.72	0.1362
Average Congestion Level among Classes				3409.56	0.1298

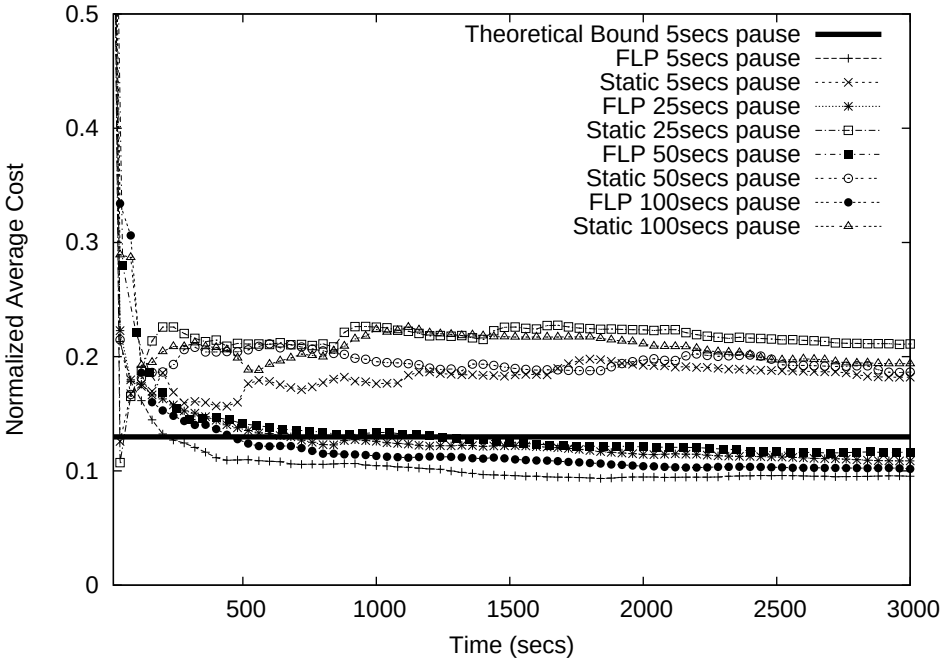


Figure 3.4: Normalized average cost for varying pause times

Figure 3.4 shows the average long-term cost under different scenarios and pause times. The average *long-term cost* effectively represents the average *congestion level* among the classes as discussed in Section 3.4.3. Static provisioning incurs higher congestion level and thus violates the congestion bound. Table 3.2 gives the average normalized theoretical bound value of 0.1298 for FLP for the 5 seconds pause time scenario, which is a tight bound as shown in Figure 3.4. The figure also shows that

the bound is not initially met for FLP. This is due to the fact that the RL algorithm is learning or updating the state-action values and has not yet converged to the optimal solution in (3.28). It is also observed that FLP satisfies the theoretical bound value of 0.1298 for the other scenarios and pause times. This result supports our claim that the MAC and topology issues are captured in our model as discussed in Section 3.2.

From the value iteration algorithm and Theorem 2.6, we can say the convergence limit of the average cost is the *minimum* average cost, which also corresponds to the *minimum* average congestion level. We emphasize that the theoretical bound value *can only be met by the optimal policy* due to the result in part (iv) of Theorem 2.6.

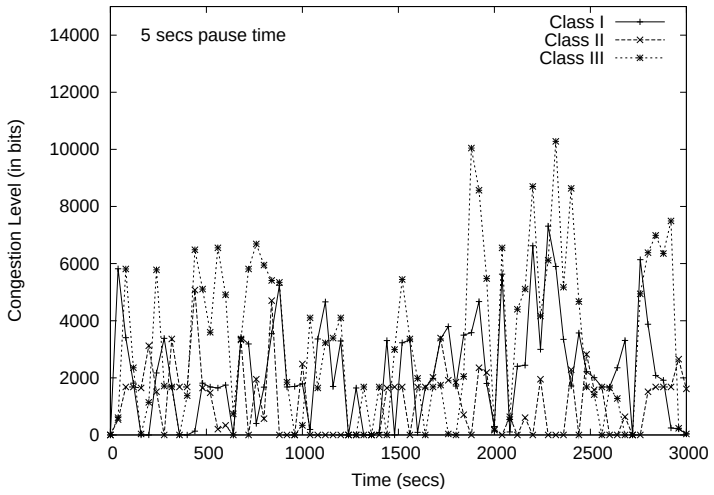


Figure 3.5: Congestion level using FLP for 5 secs pause time

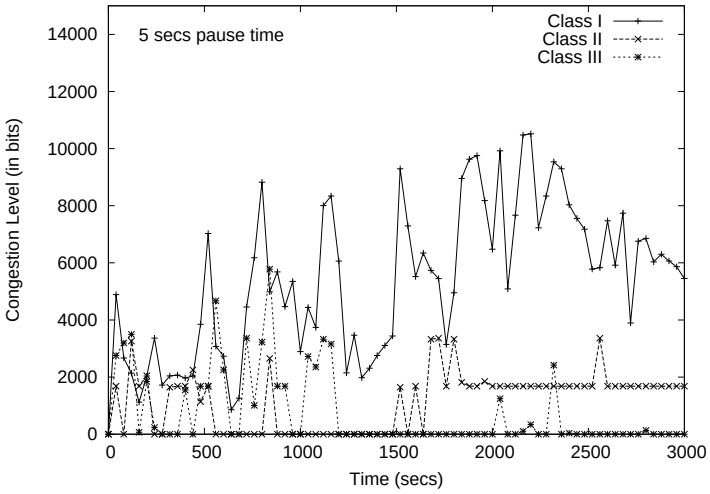


Figure 3.6: Congestion level using static provisioning for 5 secs pause time

Figures 3.5 and 3.6 show the congestion level measurements under FLP and static provisioning, respectively. The latter incurs higher congestion level for Class I, as it was not able to allocate and provision sufficient bandwidth for it. Even though Table 3.1 shows that there are more Class III than Class I packets generated from the source nodes, the Class I routing control packets are greater. The Class III packets can only be found in the flow path from the source to the destination and not from local queues of every node as in the case of Class I control packets.

FLP also performs significantly better in maintaining bounded buffer overflow (i.e. buffer loss) by adaptively allocating bandwidth among classes to minimize the congestion level as compared to static provisioning as shown in Figures 3.7 and 3.8.

The results in Figures 3.5, 3.6, 3.7 and 3.8 essentially supports our claim that the FLP algorithm has achieved a near-optimal policy for minimizing the average long term congestion level as shown in Figure 3.4.

Table 3.3 summarizes the queueing delay and buffer loss measurements, averaged over the simulation period and among all nodes. As expected, static provisioning incurs significantly larger loss compared to FLP for all classes. It also suffers large queueing delay especially for Classes I and II. Static provisioning was not able to

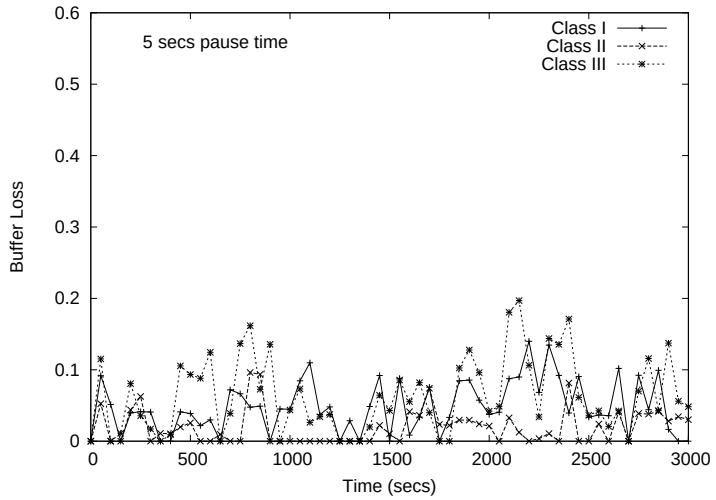


Figure 3.7: Buffer Loss using FLP for 5 secs pause time

allocate bandwidth efficiently and hence congestion and buffer loss increase.

Table 3.3: Average Queueing Delay (secs) and Buffer Loss (%) Measurements (Node-level statistics) for FLP and Static Provisioning

Scheme - Pause Time (secs)	I		II		III	
	Delay	Loss	Delay	Loss	Delay	Loss
FLP - 5	0.20	4.59	0.06	2.20	0.17	5.90
Static - 5	23.81	26.41	15.28	8.30	0.31	6.44
FLP - 25	0.21	5.22	0.05	3.19	0.26	4.72
Static - 25	30.98	23.81	10.07	4.15	0.54	5.13
FLP - 50	0.20	4.19	0.35	4.07	0.25	6.06
Static - 50	47.24	21.38	11.71	4.90	0.56	10.41
FLP - 100	0.28	4.08	0.33	3.13	0.16	5.08
Static - 100	45.41	27.96	10.77	3.52	0.95	9.48

Table 3.4 summarizes the average end-to-end delay and throughput or packet delivery ratio (PDR) as measured and averaged from the eight classified CBR flows. As the queueing delay for static provisioning is higher under FLP especially for Classes

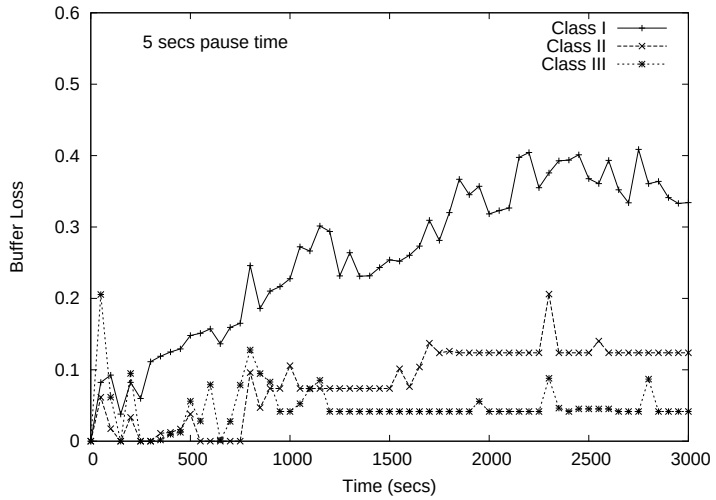


Figure 3.8: Buffer Loss using static provisioning for 5 secs pause time

I and II, it is expected that static provisioning incurs higher end-to-end delay. On the other hand, FLP achieves higher PDR in most scenarios due to its adaptive bandwidth provisioning mechanism. As FLP minimizes the average congestion level over time, it gives smaller buffer loss and higher amount of packets can be transmitted over an interval. This then gives higher throughput or PDR under FLP.

Table 3.4: Average End-to-End Delay (secs) and Packet Delivery Ratio (PDR %) Comparison for FLP and Static Provisioning

Scheme - Pause Time (secs)	I		II		III	
	Delay	PDR	Delay	PDR	Delay	PDR
FLP - 5	0.93	14.75	0.68	10.49	0.77	20.83
Static - 5	20.58	13.92	10.67	9.62	0.87	13.40
FLP - 25	0.43	23.43	0.90	12.73	0.66	14.87
Static - 25	23.32	20.88	9.38	9.61	0.96	13.38
FLP - 50	1.14	7.79	0.92	14.18	0.80	16.76
Static - 50	20.42	6.53	5.34	12.05	1.02	6.93
FLP - 100	0.96	9.30	0.65	18.35	0.82	16.22
Static - 100	30.68	8.32	10.68	2.29	0.97	10.07

3.6 Possible Weaknesses of FLP algorithm and ψ-irreducibility theory

The Foster-Lyapunov Provisioning algorithm uses a model-free RL technique for each agent to find the best WF^2Q weights that depends on the current local state of the ψ-irreducible Markov chain (i.e. local queue length for all classes in bits). In this section, we enumerate possible weaknesses of this technique.

Firstly, each agent effectively learns the WF^2Q weights from a *continuous* action space (i.e. vector of real numbers in the space $\Re^J$, where J is the number classes). For a given state or local queue length condition, (i.e. say Class 1 has empty queue length), the agent can have many possible WF^2Q weight combinations or solutions. In other words, there may be many optimal sets of actions that are learned by FLP. This may seem to be a problem, as *there is no unique action set for a given state.*

However, as explained in Section 2.4 and (2.14), the minimum relative value function is *unique* that solves the Average Cost Optimal Equation (ACOE) in (2.5). There maybe *many possible actions*, but there is only *one* value function that solves the ACOE for getting the optimal policy. This unique minimum value function is used to estimate the optimal or minimum long term cost. (i.e. minimum cost as $\eta_* = \eta_{min}$, minimum value function as $h_* = h_{min}$, and optimal policy $w_* = w_{min}$ in Section 2.4). In other words, we can obtain the optimal or minimum average cost, even though there is no unique action set for a given state.

Another possible weakness is the storage of WF^2Q weights in the *continuous* vector space $\Re^J$. A possible solution is the use of function approximation techniques, such as *artificial neural networks*. We shall deal this problem in Section 4.4 in greater details.

The FLP algorithm mainly relies on the use of WF^2Q for adaptive provisioning for each agent. WF^2Q is only possible, as we have said in Section 3.2 and Section

3.5, when the topology state (i.e. which also includes the channel state) evolves as an irreducible aperiodic (i.e. ergodic) Markov chain. From Theorem 3.1, when the topology state evolves as such, the service rate process $\{\mu_j(k)\}_{k=0}^{\infty}$ is *rate convergent*, and thus WF^2Q is applicable for each class j.

The assumption of an irreducible aperiodic Markov chain for a time-varying channel process has been commonly used in literature [6, 12, 26, 27]. In [37], the authors described a channel model for multi-node communication that captures the Signal-to-Interference (SIR) ratio or channel gain between each node. Suppose that the SIR values are partitioned into L intervals: $0 < \Gamma_1 < ... < \Gamma_L$. The channel gain is said to be in state l if it is between interval Γ_{l-1} and Γ_l. This mapping can then be reduced into an ergodic Markov chain, and the state transition probability completely specifies the dynamics of the channel.

Under the NS2 simulation [30], setting the transition probabilities can be easily done as NS2 already provides a finite-state Markov channel in its software distribution. Even though this assumption of an ergodic Markov chain for the channel process is commonly used in theory and is verified in simulations, such assumption still remains to be seen in actual network implementation.

Another possible issue is that each agent *independently* solves its own locally-observed MDP, without knowing the policies of other agents. Although this framework is easier to implement as each node does not need information about other nodes, accurate estimate of the global optimal cost may not always be obtained. We address this issue in Chapter 6.

As for the ψ-irreducibility theory on general Markov chains, showing this property of ψ-irreducibility may not be straightforward, especially when the state space is continuous and multi-dimensional. In our formulation in Section 3.3.1, ψ-irreducibility

was easily verified due to the fact that the MDP state descriptor is:

$$\overline{x}(k) = [x_1(k), ..., x_J(k)]$$

We can easily separate the analysis for each class, since the queue length process for each class j evolves as a *random walk on the half line* and is already a Markov chain.

However, for a general state descriptor such as in non-linear state space models, showing ψ-irreducibility may be difficult [23]. In this case, one may need to look at other types of stability formulations, such as how to guarantee recurrence and ergodicity of multi-dimensional Markov chains, and existence of Lyapunov functions [38]. In other words, although using ψ-irreducibility can give stability conditions and optimization *simultaneously*, it requires careful analysis and conditions specific to the problem.

3.7 Comparison of ψ-irreducibility with Lyapunov-based work

In this section, we summarize the recent state-of-the-art work by Neely in [6, 9, 26, 39, 40, 41] where he proposed a technique that uses a Lyapunov-based stability condition for communication networks.

In [39], Neely emphasized that "there was no Lyapunov method for optimization (such as stabilizing a system with minimum energy)". He then developed "a novel Lyapunov drift technique that enables system stability and performance to be achieved *simultaneously*". He claimed that his technique "bridges the gap between convex optimization and stochastic optimal control problems, and establishes a new framework for dynamic network optimization."

In [6], he claimed that his new technique that uses a Lyapunov-based stability con-

dition "unites network optimization and network control." His stochastic scheduling techniques are quite new [40, 41] which "build on the Lyapunov method to achieve optimal delay trade-offs, and make a significant contribution to the field of developing new scheduling algorithms that go beyond the classical gradient methods of optimization theory."

In this thesis, we claim that there exists a Lyapunov method for optimization, contrary to Neely's claim, and this is based on the theory of ψ-irreducibility. To the best of the author's knowledge, we present the theory of ψ-irreducibility that brings the gap between stochastic optimal control and stability in dynamic network optimization. We emphasize that, not only will ψ-irreducibility achieve optimal control, it is the ***only known framework*** *to handle network optimization, network control, and network stability* ***simultaneously*** *in a MDP formulation [22, 23, 24, 25].*

This thesis is the *first* research work that presents and applies this novel theory for a wireless network, in order to derive bounds, achieve stability and optimal control *simultaneously.*

To highlight another weakness of Neely's recent state-of-the-art work, we discuss the ideas of [9] where he proposed the following technique:

1. Firstly, a global Lyapunov drift condition is assumed to be satisfied for all time-slots t [9, Lemma 1]. This implies that for all time t in the future, the system is already stable, where the drift condition in (2.10) of Theorem 2.3 is satisfied for all time slots.

2. Using the Lyapunov drift condition, one can easily derive performance bounds. (see [9, Section IV])

3. By manipulating the equations in the Lyapunov drift inequality, Neely was able to find an optimization problem to be done at every time slot.

4. Neely then uses a Linear Programming-based algorithm to solve the optimization problem, to be done at every time slot (i.e. search for optimal parameters at current time slot, independent of previous time slots).

This technique is done in his recent state-of-the-art papers [6, 9, 26, 39, 40, 41].

In this thesis, we are essentially addressing the same problem: achieve optimization and stability *simultaneously*, for all time slots in a communication network, but in a different and better manner:

1. We do not assume that a global Lyapunov condition is satisfied for all time slots t. This is more realistic.

2. We only need a Lyapunov function at the first time slot. This Lyapunov function satisfies the Foster-Lyapunov drift inequality at time $k = 0$ only.

3. We then use the standard value iteration algorithm as our control algorithm. But, we initialize it with the Lyapunov function at $k = 0$.

4. By running the value iteration algorithm, this initializing technique then stabilizes the next policy at the next time slot, and every time slot after that.

In other words, we only need to stabilize at $k = 0$ and satisfy the Lyapunov drift inequality at $k = 0$, and the value iteration algorithm with the Lyapunov function stabilizes it for all time slots, automatically. Again, once we can stabilize at every time slot, then we can easily derive bounds using the Foster-Lyapunov drift condition in Theorem 2.3, just like Neely's results.

Neely's assumption of satisfying a global Lyapunov condition for all time slots is superfluous. This assumption implies that in a network, for all time slots in the future, the system is already stable. Then, Neely finds an algorithm to make it stable, to be done *independently*, at every time slot.

This thesis emphasizes a different and better technique. We emphasize that we do not need to assume for all time slots in the future, the system is already stable. We only need the initial time slot $k = 0$ to be stable, and by following a modified value iteration algorithm, we can stabilize all time slots in the future, automatically.

This is a new unique result, compared to Neely's results. His work and techniques do not deal with Markov chains too. In fact, the Linear Programming approach at every time slot by Neely in most of his recent state-of-the-art papers is not really so novel.

The value iteration algorithm is better than searching the parameter space in Linear Programming-based algorithms. It incrementally or iteratively finds the optimal value functions for every possible state at every time slot as it converges to the optimal solution. Linear Programming-based algorithms suffer a drawback, because it is just concerned with one time slot *independent* of other time slots, and it does not use the previously obtained values at previous time slots.

In summary, if one can find a general Markov chain for a network, one can find the optimal policy using standard value iteration techniques. In addition, by using the ψ-irreducibility theory, not only we can find the optimal policies, we can also stabilize the network, and other than that, we can derive bounds automatically. In addition, we can do these things in a more efficient manner, thus providing a framework for new scheduling techniques that combines network optimization and network control.

3.8 Chapter Summary

We have considered a stochastic optimal control approach to solve the problem of multi-class scheduling or bandwidth allocation and provisioning under a time-varying channel and topology in MANETs. Our proposed scheme is based on average-cost MDP, with the goal of finding the policy that minimizes the expected average conges-

tion level. We use a novel framework based on the theory of ψ-irreducible controlled Markov chains, c-regularity, *regular* chains, *regular* policies and *Foster-Lyapunov* drift inequality conditions that can be used to find stable and optimal policies.

Using this theory, we derive performance bounds on the average congestion level and queueing delay as the algorithm converges to the optimal solution. Specifically, at each iteration of the algorithm, a stability inequality condition is satisfied automatically when the value function is initialized with an appropriate *Lyapunov* function. Our simulation results show that the proposed scheme known as FLP is able to attain its objective of minimizing the average congestion level.

In summary, we have presented the *first* technique that uses the concepts of ψ-irreducible Markov chains for achieving the following *simultaneously* for a general Markov wireless queueing network:

1. Finding the optimal scheduling policy for each node that only depends on its queue length vector, by using a modified value iteration algorithm initialized with a Lyapunov function, and *without* considering the statistics of the channel and topology state processes

2. Obtain the policy that stabilizes the queue congestion level *directly* from the optimal scheduling policy, while the Markov chain satisfies the *Foster-Lyapunov drift* stability condition

3. Derive average performance bounds *directly* from the value iteration algorithm, as the algorithm converges to the optimal solution

Chapter 4

Resource Allocation: A Semi-MDP Approach

This chapter introduces the second variant of MDP formulations in this thesis. We present an adaptive approach for QoS provisioning, where each node acting as an agent performs bandwidth allocation (BA) and buffer management (BM). We consider an *event-based* scheme where each agent only executes its own policy at decision instants that depend on system events. These system events include changes in routing paths and MAC layer callback or notifications such as transmission failures.

We observe that the time interval between two successive decision instants is clearly non-deterministic, which depends on a number of factors such as the time-varying channel, random MAC schemes, and mobility. Furthermore, the local observed queue condition may vary during decision intervals. Formally, we use the Semi-Markov Decision Process (SMDP) framework to effectively capture this scenario. This framework differs from Chapter 3, due to the inclusion of the time interval between decision instants. We note that in a decision interval, the state of the Markov chain can vary as well.

The main objective is to maximize average network reward and at the same time, minimize per-class QoS violations with respect to bandwidth, queueing delay, and buffer loss. Due to the fact that in a dynamic network, estimating the state transition probabilities of the underlying Markov chain is a non-trivial task, we formally

introduce a *model-free* mathematical framework known as *Neuro-Dynamic Programming* (NDP) , also termed as *Reinforcement Learning* (RL) in this chapter.

In finding the optimal policy for SMDP, it is well known that *model-based* Dynamic Programming techniques, such as value iteration and policy iteration, suffer from *curse of dimensionality* [4], especially when the state space is large as in the case of QoS provisioning. NDP or RL solves these issues by finding an *approximate* solution to the optimal policy, while the agent interacts with the system. The distinguishing characteristics of this approach is that it can be used in practical and real-world scenarios, whereby each node determines its near-optimal policy through a sequence of direct interactions with the network. A model-free solution does not need prior knowledge of the state transition probabilities of the Markov chain. Thus, RL is less computationally expensive than DP techniques, as it does not require the exact model of the system.

We first introduce the SMDP framework in Section 4.1, followed by the RL algorithm in Section 4.2. Section 4.3 presents the SMDP formulation for each agent performing bandwidth allocation and buffer management. Section 4.4 discusses the complexity and implementation issues of the proposed scheme. In Section 4.5, we present and discuss simulation results based on the NS2 simulator.

4.1 Semi-Markov Decision Process

A Semi-Markov Decision Process (SMDP) is defined by a tuple (S, A, P, R) where S is a set of states, A is a set of actions, R is a reward function, and P is a probability distribution function defined as follows [4]:

$$P(t, s' \mid s, a) = P(s' \mid s, a)F(t \mid s, a) \tag{4.1}$$

where:

$P(t, s' \mid s, a)$ is the probability that the process will be in state s' for the next decision epoch, at or before t time units after choosing action a in state s.

$P(s' \mid s, a)$ denote the probability that action a taken will cause the system to transition from state s to s'.

$F(t \mid s, a)$ gives the probability that the next decision epoch occurs within t time units after action a in state s is chosen.

In state s, when action a is chosen, a lump sum reward $k(s, a)$ is received. The expected total reward between two decision epochs can be expressed as:

$$r(s, a) = k(s, a) + E_s^a \left\{ \int_0^\tau c(W_u, s, a) du \right\} \tag{4.2}$$

where:

$c(W_u, s, a)$ is the rate at which reward is accrued when the natural process is in state W_u.

τ is the transition time to the next decision epoch.

E_s^a is the expectation with respect to the transition distribution $F(t \mid s, a)$.

The natural process describes the evolution of the system at all times, while the SMDP model represents the snapshots of the system at decision points. The expected total reward up to time t, starting from initial state s is defined as [42]:

$$V_t^w(s) = E_s^w \left\{ \sum_{i=0}^{v_t - 1} k(s_i, a_i) + \int_0^t c(W_u, s_{v_u}, a_{v_u}) du \right\} \tag{4.3}$$

where:

v_t is the number of decisions made up to time t.

E_s^w denote the expectation with respect to policy w and initial state s.

The average reward starting from state s and using policy w (also known as the

gain of the policy w) is defined as:

$$\rho_w(s) = \lim_{M\to\infty} \frac{E_s^w \left\{ \sum_{i=0}^{M} \left(k(s_i, a_i) + \int_0^{\tau_i} c(W_u, s_i, a_i) du \right) \right\}}{E_s^w \left\{ \sum_{i=0}^{M} \tau_i \right\}} \tag{4.4}$$

where:

τ_i is the transition time between (i^{th}) and $(i+1)^{th}$ decision epochs.

We assume a *uni-chain* SMDP, where the gain of the policy is state independent: $\rho_w(s) = \rho_w$. For continuous-time uni-chain average reward SMDP, the expected average *adjusted* sum of rewards H^π up to time t for policy w is defined as:

$$H^w(s) = V_t^w(s) - \rho_w t \tag{4.5}$$

The main objective is to find the policy w^* that will maximize the average long term reward. The Bellman optimality equation for average reward SMDP can be stated as follows: For any *uni-chain* SMDP, there exists a scalar ρ^* and a *value function* H^* satisfying the system of equations [4] :

$$H^*(s) = \max_{a\in A} \left(r(s,a) - \rho^* \tau(s,a) + \sum_{s'\in S} P_{ss'}(a) H^*(s') \right) \tag{4.6}$$

where:

$\tau(s,a)$ is the average sojourn time in state s when action a is taken in it, until the next decision period.

$P_{ss'}(a)$ is the probability of transition from state s to state s' under action a.

ρ^* is the optimal gain.

The *state-action* representation of (4.6) can be written as follows: Let $R^w(s,a)$ represents the average *adjusted* value of choosing action a in state s once, and then following policy w subsequently [43]. Let $R^*(s,a)$ be the average adjusted value by

choosing actions optimally.

$$R^*(s,a) = r(s,a) - \rho^*\tau(s,a) + \sum_{s' \in S} P_{ss'}(a) \max_{a' \in A} R^*(s',a') \tag{4.7}$$

The optimal policy is $w^*(s) = \arg\max_{a \in A} R^*(s,a)$.

4.2 RL Solution for Average Reward SMDP

A model-free average reward Reinforcement Learning algorithm known as *Semi-Markov Average Reward Technique* (SMART) [13, 44] can be used to solve the SMDP. We summarize the SMART algorithm as follows: If action a_t is chosen at the t^{th} decision period with state s_t, the corresponding state-action value $R(s_t, a_t)$ is updated as:

$$R_{new}(s_t,a_t) = R_{old}(s_t,a_t) + \alpha_t \left[r(s_{t+1},s_t,a_t) - \rho_t\tau_t + \max_{a_{t+1} \in A} R_{old}(s_{t+1},a_{t+1}) - R_{old}(s_t,a_t) \right] \tag{4.8}$$

where:

α_t is the learning rate parameter in the t^{th} decision period.

$r(s_{t+1}, s_t, a_t)$ is the actual cumulative reward earned from s to s' under a.

τ_t is the sojourn time period from state s to s'.

The reward rate ρ_t is updated as follows:

$$\rho_t = (1 - \beta_{t-1})\rho_{t-1} + \beta_{t-1} \frac{T(t-1)\rho_{t-1} + r(s_{t+1},s_t,a_t)}{T(t)} \tag{4.9}$$

where:

β_t denote the learning rate parameter.

$T(t)$ is the total time spent until the t^{th} decision period.

If each action is executed in each state *infinite* number of times and all the states

are visited while the learning rates β_t and α_t are decayed appropriately, the SMART algorithm converges to optimality [44].

To facilitate this in a practical manner, exploration is performed where, with a small probability p_t, actions other than the highest state-action value (i.e. $\arg\max_{a \in A} R(s_t, a)$ or *greedy action*) should be executed. In decaying the parameters β_t, α_t, and p_t, the *Darken-Chang-Moody search-then-converge* algorithm [35] is used where, in the following expressions, ϑ can be substituted by β, α, and p. The following decaying equation is used: $\vartheta_t = \vartheta_0/(1 + \xi_t)$, where $\xi_t = t^2/(\vartheta_r + t)$, and ϑ_0 and ϑ_r are constants. This exploration scheme is a standard method of approximating the optimal value functions for RL [13, 35].

4.3 SMDP for Resource Allocation

In this section, we apply the concepts of the previous sections for resource allocation for MANETs. As mentioned in earlier, we use the SMDP framework, instead of the usual MDP due to the fact that, in a dynamic wireless network, the time interval between two successive decision events for an agent is non-deterministic. The length of the time interval depends on a number of factors such as the time-varying channel, random MAC schemes, and varying topology.

At each decision instant, each agent performs *network-level* resource allocation by doing bandwidth allocation (BA) and buffer management (BM) among its local class queues. We include predefined per-class QoS constraints with respect to queueing delay, bandwidth and buffer loss. The agent earns reward which depends whether it has achieved these constraints. The main objective is to find the optimal BA and BM policy so that the agent maximizes its average long term reward and at the same time, minimize average long term QoS violations for all the traffic classes.

This approach is similar to [13] in that we use an average reward SMDP and a

model-free RL solution. The authors in [13] formulated a *constrained* SMDP and use a *Lagrange* multiplier method for their QoS constraints. However, as we are dealing with a more dynamic MANET, instead of a centralized cellular network, absolute QoS bounds might be difficult to achieve. Hence, we choose to minimize the average long term QoS violations.

Our multi-class resource allocation scheme is also similar to the non-linear *Joint Buffer Management and Scheduling* (JoBS) optimization algorithm in [45]. However, JoBS requires knowing the system state, arrival, input and output curves for the whole history. For our case, the formulation strongly depends on the Markov property that the response at the next decision period depends only on the current state and action chosen, and not of the whole history [4].

4.3.1 System Model

We consider the wireless nodes as agents in a multi-class network and formulate a SMDP for each agent. Similar to the ψ-irreducible MDP framework in Section 3.3, we propose that each agent *independently* solves its SMDP based on its local observed state condition as shown in Figure 4.1. The case of a multi-agent system where agents collaborate among themselves is discussed in Chapter 6.

There are J classes of network services in the system. Each class j defines a minimum rate $b_{j,min}$, and absolute queueing delay and loss (i.e. packet buffer loss) constraints: $d_j \leq d_{j,max}$ and $l_j \leq l_{j,max}$, for $j = 1, 2, ..., J$. For each node, we define the state descriptor for BA and BM as:

$$\mathbf{S} = [d_1, l_1, d_2, l_2, ..., d_J, l_J] \tag{4.10}$$

where:

d_j is the normalized measured queueing delay for class j

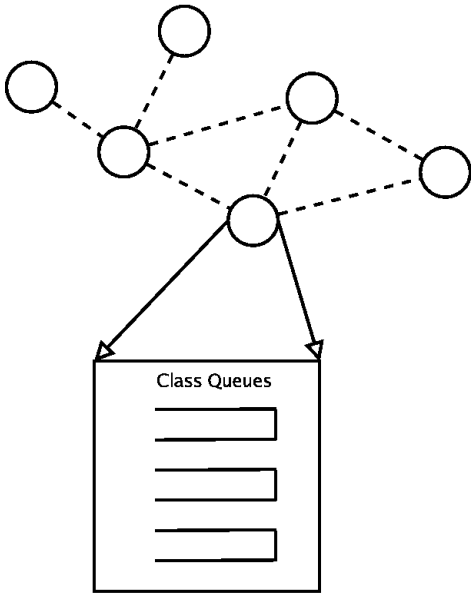


Figure 4.1: Independent SMDP agents for resource allocation in MANETs

l_j is the measured buffer loss for class j

We assume that the scheduling mechanism is work-conserving. We shall justify this assumption by the choice of the actual scheduling mechanism in Section 4.4.2.

We identify the following system events for the state transitions for the SMDP: a) Changes in the routing path, where the agent may need to reallocate bandwidth and perform buffer management for those service classes with flows using active routes, so as to earn more reward and effectively minimize deadline violations; b) MAC layer call-back or notifications such as transmission failures; c) Packet arrival. We also assume that only one event can occur in any time instant.

When an event occurs, the agent performs BA and BM through action **a** defined as:

$$\mathbf{a} = [r_1, r_2, ..., r_J, a_1, a_2, ..., a_J] \tag{4.11}$$

where:

a_j is the drop rate for class j

r_j is the rate allocated for class j

We define the cumulative reward function from state $\mathbf{S}$ to $\mathbf{S}'$ under action $\mathbf{a}$ as:

$$\begin{aligned} r(\mathbf{S}', \mathbf{S}, \mathbf{a}) &= \tau(\mathbf{S}', \mathbf{S}, \mathbf{a}) \sum_{j=1}^{J} \left\{ r_{j,price} \left(\frac{r'_j - b_{j,min}}{C_h} \right) \right. \\ &\quad \left. + d_{j,price}(d_{j,max} - d_j) + l_{j,price}(l_{j,max} - l_j) \right\} \end{aligned} \tag{4.12}$$

where:

$\tau(\mathbf{S}', \mathbf{S}, \mathbf{a})$ is the actual sojourn time from state $\mathbf{S}$ to $\mathbf{S}'$

$r_{j,price}$ is the rate or bandwidth price for class j

C_h is the *current capacity* of the channel that is assumed to be observed by the agent (see Section 4.4.2 for detailed explanation)

r'_j is the new allocated bandwidth for class j

$d_{j,price}$ is the price for queueing delay for class j

$l_{j,price}$ is the price for loss or buffer drops for class j

The reward definition in (4.12) gives more credit to those actions that satisfy the QoS constraints. If the chosen action results in QoS violations, the reward function penalizes the agent by a weighted scalar that is proportional to the deviations from the constraints.

Strictly speaking, the problem can be formulated as a *constrained* SMDP. The motivation behind the *unconstrained* SMDP formulation and reward definition above is as follows: For a general constrained SMDP with L constraints, the optimal policy for at most L of the states is *randomized* [46]. Since the state space is *continuous* in (4.10), a *non-randomized* policy obtained from RL is often a good approximation to the optimal policy [47].

Thus, we handle the constraints through our reward definition as above. Absolute QoS bounds are also difficult to achieve especially in MANETs. Thus, we try to

achieve our objective of maximizing average long term reward by effectively minimizing the average QoS violations. Our method can be used as a pricing scheme, where the service provider earns more when the system experiences good QoS and it tries to maximize its average long term profit.

The SMDP objective is then to maximize the average reward, starting from state $\mathbf{s}_1 = \mathbf{S}$ and using policy w, defined as:

$$\rho_w(\mathbf{S}) = \lim_{M \to \infty} \frac{E_{\mathbf{S}}^w \left\{ \sum_{t=1}^{M} r(\mathbf{s}_{t+1}, \mathbf{s}_t, w(\mathbf{s}_t)) \right\}}{E_{\mathbf{S}}^w \left\{ \sum_{t=1}^{M} \tau(\mathbf{s}_{t+1}, \mathbf{s}_t, w(\mathbf{s}_t)) \right\}} \tag{4.13}$$

where:

$w(\mathbf{s}_t)$ is the action taken using policy w when at state $\mathbf{s}_t$

$E_{\mathbf{S}}^w$ is the expectation with respect to policy w and initial state $\mathbf{s}_1 = \mathbf{S}$

Note that this multi-class formulation can be applied for a wired or wireless network. In Section 4.4.2, we explain the reason specific for MANETs, especially with time-varying channel medium and topology.

We do not pursue the discussion of the state transition probabilities that capture the underlying uncertainties of the network. The exact system model is often infeasible due to the following reasons. Firstly, the SMDP state-space formulation requires the transition probability that encompasses the joint distribution of uncertainties, such as node mobility, channel conditions and MAC schemes, that affect the state transition and cumulative reward. Estimating the probability distribution is also a non-trivial task during runtime. In addition, fixing a model before computing the optimal policy also means that it would not be robust if the actual system conditions depart from the assumed model. Our main motivation for this research is to use model-free algorithms such as RL and allow the nodes to adaptively learn the optimal policy.

4.4 Implementation Issues

4.4.1 RL algorithm-related issues

We assume that the SMDP in Section 4.3 is *uni-chain*, so that the optimal state-action values in (4.8) can be found.

In *value function-based* RL algorithms such as Section 4.2, when the total number of states and actions is small, using the algorithms is straightforward by having a look-up table to store and update the corresponding state-action values $R(s, a)$. However, when the state and action spaces are large or continuous and multi-dimensional, as in our case in (4.10) and (4.11), the state-action values are usually *approximated* due to storage limitations.

In dealing with continuous state space, researchers have used *function approximation* techniques such as neural networks. When using function approximation methods, it is crucial that the choice of such approximators facilitates the convergence of the RL algorithms. The RL provisioning algorithm in [13] uses a *Multi-Layer Perceptron (MLP)* neural network with a single hidden layer. *Non-linear* approximators, such as MLP networks, are more difficult to analyze mathematically and in general, may become divergent [15].

We thus use the *linear* function approximator known as *Cerebellar Model Articulation Controller (CMAC)* where the output (i.e. state-action values) is approximated by a weighted linear sum of the features of the state input vector. The CMAC is a *tile-coding* function approximator that displays *local* generalization. Tile coding is an established and well-understood method for Reinforcement Learning [14]. The linearity property is also instrumental in proving the algorithm's convergence [15] and in generalizing between similar states when the actions are *discrete* [14, 48, 49].

However, for both continuous state and action spaces, the CMAC neural network may not suffice alone. A number of recent research works have already tackled the

issues of continuous multi-dimensional state and action spaces for RL algorithms. In [50, 51], the authors proposed a continuous state, continuous action *value function-based* algorithm. A neural network acting as an *approximator* is combined with a component known as *wire fitter interpolator*. Their method, known as *Wire Fitted Neural Network* (WFNN), has the distinguishing property of finding the action with the highest expected value in real-time.

In the value-based RL algorithm in (4.8), there is a need to search for that action vector with the highest expected value. If the action is continuous and multi-dimensional, a straightforward and crude manner is to discretize the action vector in each dimension and sweep through all these representative vectors to obtain the maximum value. WFNN achieves a better and faster approach due to the property of the wire fitting interpolator that the highest interpolated value (i.e. maximum state-action value) coincides with the highest interpolation vector (i.e. *wire vector* with maximum state-action value) of the wire fitter [51].

In [50], the author favored a feed-forward neural network (i.e. a MLP network), which is a global approximator where changes to the neural network weights have some effect over the entire input space and can represent higher level relationships between the input and target output variables. However, as mentioned earlier, a non-linear global approximator may not be suitable in general due to its non-convergence for RL algorithms. Hence, we use the local and linear tile-coding approximator. We term our architecture as the *Wire-Fitted CMAC* for continuous and multi-dimensional state and action spaces.

Figure 4.2 shows the Wire-Fitted CMAC with input state vector $\overrightarrow{x}$, n wires with the *wire action vectors* $\overrightarrow{u_i}$ (i.e. representative action vectors), and state-action values (Q-values) q_i, for $i = 1, 2, ..., n$. A CMAC network is used to approximate $\overrightarrow{u_i}$ and q_i for each i.

The computation and updating of $Q(\overrightarrow{x}, \overrightarrow{u})$ from these components are explained

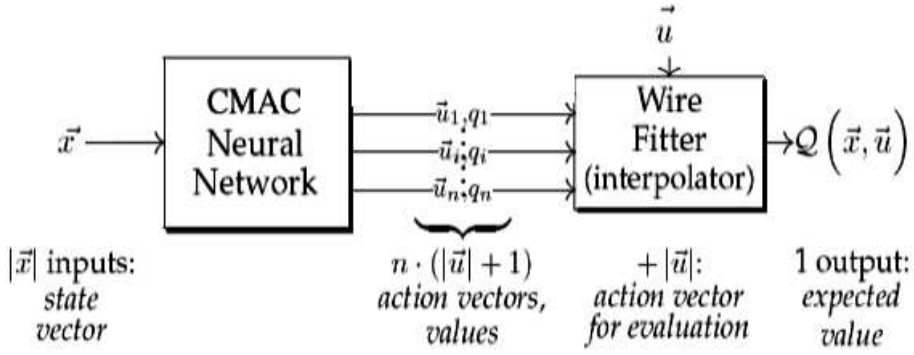


Figure 4.2: Wire-Fitted CMAC

in greater details in [50]. We summarize the wire-fitted interpolation of $Q(\overrightarrow{x}, \overrightarrow{u})$ with input vectors $\overrightarrow{x}$ (i.e. state vector) and $\overrightarrow{u}$ (i.e. action vector) as follows:

$$Q(\overrightarrow{x}, \overrightarrow{u}) = \lim_{\epsilon \to 0^+} \frac{\sum_{i=1}^{n} \frac{q_i(\overrightarrow{x})}{p_i(\overrightarrow{x})}}{\sum_{i=1}^{n} \frac{1}{p_i(\overrightarrow{x})}} \tag{4.14}$$

where:

$p_i(\overrightarrow{x}) = \| \overrightarrow{u} - \overrightarrow{u_i}(\overrightarrow{x}) \|^2 + \delta \left(\max_i q_i(\overrightarrow{x}) - q_i(\overrightarrow{x}) \right) + \epsilon$

$\overrightarrow{u_i}(\overrightarrow{x})$ is the i^{th} wire vector when the state is $\overrightarrow{x}$

$q_i(\overrightarrow{x})$ is the q-value of the i^{th} wire vector

ϵ, δ are small constant factors

The main function of the wire vectors $\overrightarrow{u_i}$ is: $\arg\max_{\overrightarrow{u}} Q(\overrightarrow{x}, \overrightarrow{u}) = \overrightarrow{u}_{\arg\max_{q_i}}$. The Q-values represent the corresponding state-action values $R(s_t, a_t)$ in the SMART algorithm in (4.8).

Thus, the Wire-Fitted CMAC combines the CMAC neural network with *wire-fitted interpolation* to approximate the state-action values in continuous vector spaces. The CMAC is used for its fast computational capability, its linearity and convergence properties in RL, while the wire-fitted interpolation helps in finding the action with the highest state-action value in real-time by searching only a few number of n representative wire action vectors. Wire-fitted interpolation also represents discontinuities

in the policy and value function where the action represented by each wire vector changes smoothly in response to changes in the state [51].

The choice of value for n affects the performance of the function approximation mechanism. Increasing n generally decreases the error of approximation, but it also increases the storage requirements (i.e. total number of action vectors). Other issues relating to Wire-Fitted function approximation are discussed in [50].

Since CMAC neural networks only display *local* generalization or approximation, the lack of global generalization is addressed by adding *noise terms* in the chosen action whenever the action is *non-greedy*. This is also in conjunction with the exploration scheme described in Section 4.2.

Another issue is the *state-action deviation* problem [50]: If the state-action values are stored approximately, it is likely that the approximation resources will be used to represent values of the states rather than actions in the states. Following [50], a modified SMART updating scheme based on *advantage learning* update is thus necessary:

$$R_{new}(s_t, a_t) = R_{old}(s_t, a_t) + \alpha_t \left\{ M_t - (1 - \frac{1}{q}) \left(M_t - \max_{a_t \in A} R_{old}(s_t, a_t) \right) \right\} \tag{4.15}$$

where:

$M_t = r(s_{t+1}, s_t, a_t) - \rho_t \tau_t + \max_{a_{t+1} \in A} R_{old}(s_{t+1}, a_{t+1})$

q is a small constant, taken as 0.1 in simulations

Similar to Section 3.5, for bandwidth allocation, we use the work-conserving scheduler known as *worst-case fair weighted fair queueing* (WF^2Q) [36]. The RL algorithm learns the WF^2Q weights for bandwidth allocation together with the packet drop rate for buffer management (see (4.11)).

Wire-fitted interpolation requires the cumulative reward in the update equation

to be normalized to the range of the action vectors [50]. We also require the WF^2Q weights and packet drop rate for BA and BM respectively to be in the range $[0, 1]$. The cumulative reward defined in (4.8) is initially passed into a *hyperbolic tangent function* before passing into the Wire-Fitted CMAC. This clamps the wire action vectors to be in the range $[-1, 1]$, which are then normalized to $[0, 1]$.

With the Wire-Fitted CMAC, together with a modified SMART algorithm, we term the proposed resource allocation and provisioning scheme as *Wire-Fitted Reinforcement Learning Provisioning (WFRLP)* algorithm.

We observe that the buffer management component can be used in a wired or wireless scenario. However, for bandwidth allocation, fair queueing techniques such as WF^2Q are generally known to be only applicable in a wired network [36]. In the next subsection, we justify the applicability of the bandwidth allocation component of WFRLP under a time-varying *channel medium* and *topology*.

4.4.2 Bandwidth allocation for MANETs

We use a similar principle from Section 3.1 to characterize the bandwidth allocation component of WFRLP.

Following the notations in Section 3.1, each node is assumed to act as an agent that actively performs scheduling or bandwidth allocation on its own J local queues, where packets are enqueued in their respective class queues. In each node, the queue dynamics at the j^{th} class queue for $j = 1, ..., J$, can be generally expressed as:

$$x_j(k+1) = \max\left(x_j(k) + a_j(k) - \mu_j(k), 0\right) \tag{4.16}$$

The index k represents the decision instant. By assuming that the channel process $\{C_h(k)\}_{k=1}^{\infty}$ is an *embedded* irreducible aperiodic finite state-space Markov chain, we have the same case as in Section 3.1. The difference is we consider the decision instants

or periods as the time slot boundaries of (4.16). In other words, the service process $\{\mu_j(k)\}_{k=0}^{\infty}$ is also *rate convergent* from Theorem 3.1, when the channel process is an *embedded* irreducible aperiodic Markov chain. We can then apply the same result as in Section 3.2 and claim that the service process is also rate convergent for a time-varying *topology state* process.

From Theorem 3.1, fair queueing techniques such as WF^2Q can thus be applied due to the *rate convergence* property of the service process for each class j. Specifically, the WF^2Q weights are used to provision and allocate the total effective rate $\sum_{j=1}^{J}\mu_{j,av}$ among the class queues. This total effective rate effectively represents the observed capacity C_h in the SMDP formulation in (4.12).

4.4.3 Action Search For Handling QoS Constraints

As the state-action values are being updated in (4.15) when the RL agent interacts with the environment, it is certain that the agent may choose actions that are apparently costly and ineffective, even though the action vector with the maximum state-action value can be easily retrieved through the Wire-Fitted CMAC in Figure 4.2. In BA, for learning the WF^2Q class weights $w_{sched,j}$ for $j = 1, 2, ..., J$, the RL algorithm does not prevent the agent to allocate zero bandwidth for a certain class j (i.e. $w_{sched,j} = 0$) as it initially searches the continuous action space. Similarly in BM, the action vector may result in dropping large proportion of packets in the class queue, even though there's no delay or rate constraint violations. These unwanted actions thus drastically penalize the agent and slow down the convergence of the RL algorithm.

In this subsection, we incorporate prior knowledge to address these issues. We use a similar approach in [52] to search for the desired actions (i.e. how much bandwidth to allocate and what is the packet drop rate for each class). We define the target bandwidth $r_{j,min}$ for network class i for handling the rate and delay constraints men-

tioned in Section 4.3 as follows: If $d_{j,max} > d_i$, $r_{j,min} = \max\left(\frac{B_j}{d_{j,max}-d_j}, b_{j,min}\right)$, where B_j is the total buffer size of class j (in bits).

The target bandwidth $r_{j,min}$ effectively captures the required bandwidth to clear the buffer, without any buffer loss, within the maximum allowed delay. If $d_{j,max} \leq d_j$, the target bandwidth is set to the observed current link capacity C_h, so as to quickly clear the buffer [52].

We use the same state and action descriptors in (4.10) and (4.11) for the SMDP formulation. However, we redefine the reward structure $r(\mathbf{S}', \mathbf{S}, \mathbf{a})$ in (4.12) by replacing $b_{j,min}$ with $r_{j,min}$. As can be deduced from the definition of $r_{j,min}$, the following cases arise:

Case 1: $\sum_j r_{j,min} < C_h$ and $d_{j,max} > d_j$, $\forall j$: No rate and delay violations. The spare bandwidth (i.e. $C_h - \sum_j r_{j,min}$) is allocated among the backlogged classes. The proportion for each class is obtained from the normalized rate components of the action vector. No buffer drops are made.

Case 2: $\sum_j r_{j,min} = C_h$: No rate violations. a) If there are no delay violations, the WF^2Q weight is set as: $w_{sched,j} = \frac{r_{j,min}}{C_h}$. No buffer drops are also made. b) If there are delay violations and due to our condition that $r_{j,min} = C_h$ if $d_{j,max} \leq d_j$, for some j, then only class j is backlogged and having delay violations. Packets are dropped until the delay constraints are satisfied or up to the proportion value obtained from the action vector's drop components.

Case 3: $\sum_j r_{j,min} > C_h$: Three sub-cases arise: a) If there are no rate or delay violations for any class, packets are dropped up to the *learned* proportion of the excess bandwidth (i.e. $\sum_j r_{j,min} - C_h$) obtained from the action vector's rate components. b) If there are delay violations, packets are dropped similar in 2b. c) If there are rate violations (i.e. $r_{j,min} > C_h$) for some j, packets are dropped similar to 3a.

For Cases 2 and 3, once the packets are dropped and delay and rate constraints are

satisfied, $\sum_j r_{j,min} \leq C_h, \forall j$. The WF^2Q weight is set as: $w_{sched,j} = \frac{r_{j,min}}{\sum_k r_{k,min}}$. This assures that the allocated rate is at least the target minimum rate: $r_j = w_{sched,j} \cdot C_h = \frac{r_{j,min}}{\sum_k r_{k,min}} \cdot C_h \geq r_{j,min}$.

4.5 Simulation Results

We simulate a similar scenario described in Section 3.5 and Figure 3.3 with 20 mobile nodes in a 1,000m by 1,000m area under the NS2 simulator. The channel capacity is 2 Mbps, while the interface queue and routing protocol's buffer have a depth of 50 and 100 packets, respectively. We define three different traffic classes with the parameters shown in Table 4.1 (see (4.12) for the price definition).

Table 4.1: QoS Constraints and Price Parameters

Traffic Type	BW Price	Min BW	Delay Price	Max Delay (msecs)	Loss Price	Max Buffer Loss %
I	50	128kbps	50	20	50	1
II	40	100kbps	40	60	40	2
III	20	50kbps	20	500	20	5

We have used eight long-lived CBR connections with the characteristics similar to Table 3.1. We choose CBR flows since this type of flows captures the worst case and average long term performance. However, our technique still applies for other types of traffic such as Pareto and Exponential ON/OFF sources.

We have discussed in Section 4.4.2 that the MAC mechanisms and varying topology in the network is captured in our model through the concept of the *topology state process* that evolves as an irreducible aperiodic finite-state Markov chain. It should also be noted that there have been a number of works in MANETs under the Markov chain theory [32, 33]. Our approach is different because we use the *controlled* Markov chain or SMDP framework.

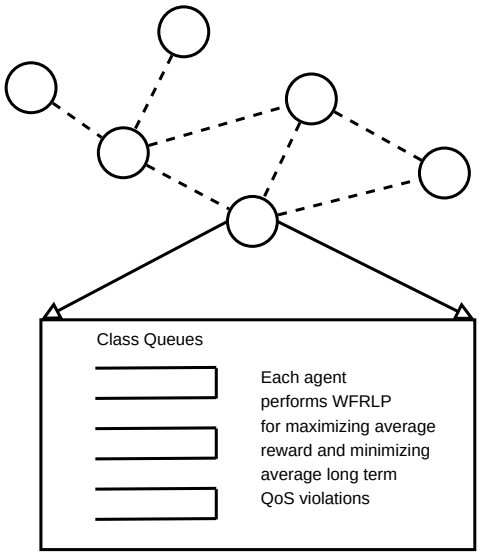


Figure 4.3: Each RL agent performs WFRLP independently

For WFRLP, the constants of the Darken-Chang-Moody scheme for the learning and exploration rates are chosen as $\beta_0 = \alpha_0 = 0.5$, $p_0 = 0.1$, and $\beta_r = \alpha_r = p_r = 10^{11}$ (see Section 4.2). Each CMAC has 4 tiles with 3 resolution elements in each dimension of the 6-dimensional state vector, giving a storage of 2916. We also use 10 wires or interpolation vectors. Hence, in one mobile node, the neural network approximator for WFRLP (see Figure 4.2) has 70 CMAC networks.

Figure 4.3 shows each node acting as an RL agent that performs the WFRLP algorithm with the corresponding state and action descriptor in (4.10) and (4.11), respectively. The WFRLP update equation is described in (4.15). Note that the agent forms its state descriptor from local available information in (4.10) and does not require traffic arrival, topology, or channel statistics.

WFRLP is compared with the JoBS algorithm found in the latest version of NS2. The NS2 implementation of JoBS (JoBS-NS2) uses a feedback-control based heuristic as described in [52]. The simulations were performed with varying pause times. It should be noted that JoBS-NS2 performs *joint buffer management and scheduling* (JoBS) optimization and was initially designed for wired networks.

However, as we have discussed earlier in Section 4.4.2, the service process is *rate convergent* under a time-varying channel and topology process. We believe JoBS-NS2 can be used for wireless networks, due to the rate convergence property and an irreducible aperiodic Markov chain assumption for the topology. For its average reward measurement, we modify JoBS-NS2 to use (4.12).

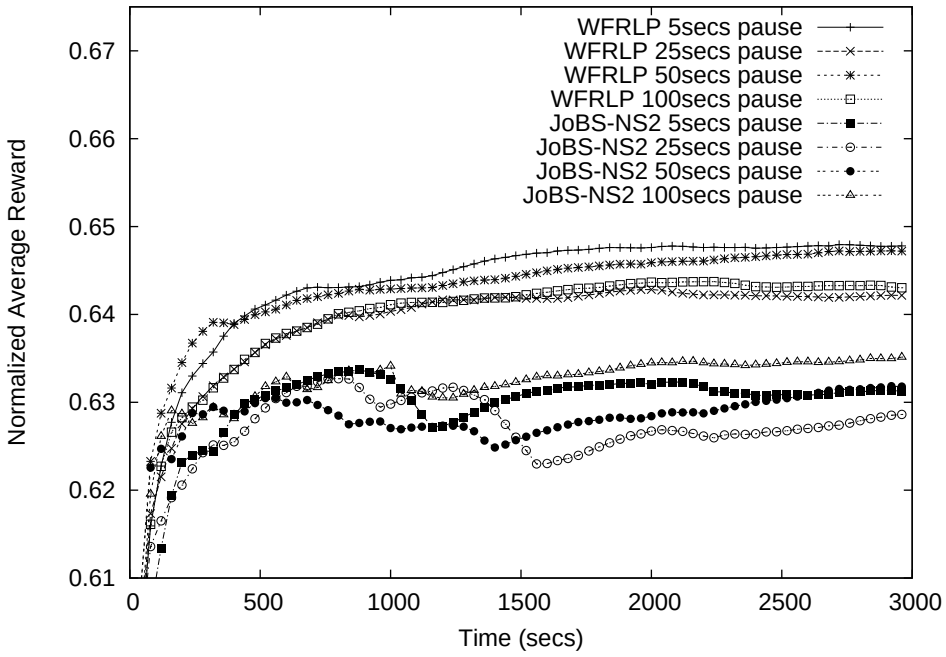
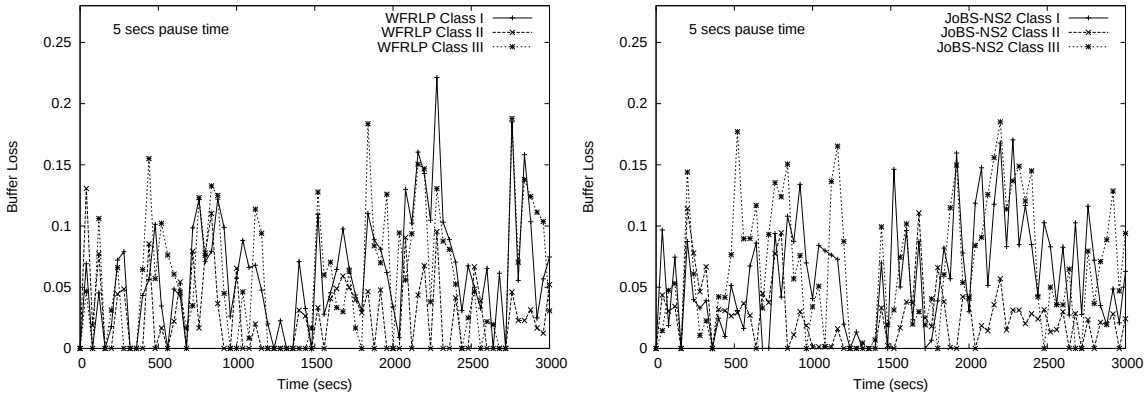


Figure 4.4: Normalized Average Reward for WFRLP and JoBS-NS2 for varying pause times

Figure 4.4 shows the normalized average network reward for both algorithms. It shows that WFRLP maximizes the average long term reward. This result also supports our claim that the different MAC, time-varying channel and topology issues are captured in our model as discussed in Section 4.4.2, as the normalized average reward converges under different scenarios and pause times.

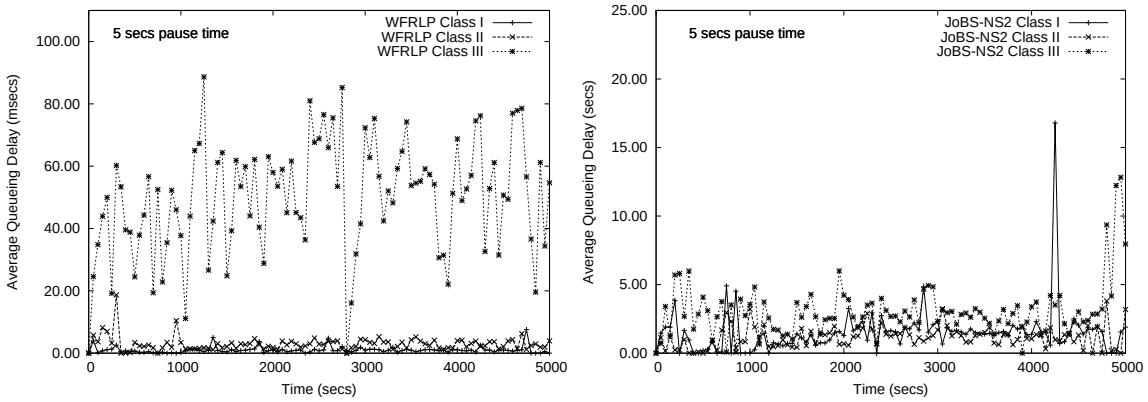
Figure 4.5 compares the buffer loss percentage for the traffic classes using WFRLP and JoBS-NS2 under the 5 seconds pause time scenario when the nodes are highly mobile. Both algorithms appear to incur similar buffer loss performance for all classes. The two algorithms did not satisfy the absolute loss constraints due to the dynamic

scenario.



(a) Under WFRLP (b) Under JoBS-NS2

Figure 4.5: Buffer Loss for WFRLP and JoBS-NS2 under 5 secs pause time

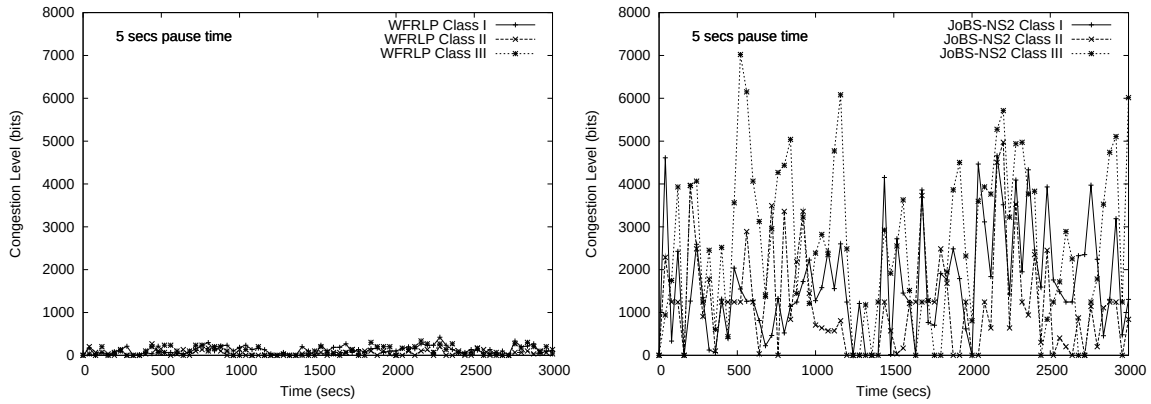


(a) Under WFRLP (msecs) (b) Under JoBS-NS2 (secs)

Figure 4.6: Queueing Delay for WFRLP and JoBS-NS2 under 5 secs pause time

Figure 4.6 shows the queueing delay measurements. The delay constraints are well satisfied under WFRLP. For JoBS-NS2, the delay constraints are not satisfied at all.

In JoBS-NS2, traffic is dropped from classes up to the maximum allowable packet loss when there's buffer overflow or delay and rate violations. Once the maximum level is reached, the delay and rate constraints are relaxed [30, 52]. Hence, it is expected that JoBS-NS2 incurs larger queueing delay. This is clearly shown in Figure 4.6b.



(a) Under WFRLP (b) Under JoBS-NS2

Figure 4.7: Congestion Level for WFRLP and JoBS-NS2 under 5 secs pause time

WFRLP incurs significantly lower delay by not allowing too many packets in the queue, due to its adaptive buffer management mechanism. JoBS-NS2 incurs larger queueing delay and thus higher congestion level. This result is also shown in Figure 4.7, even though JoBS-NS2 achieves similar buffer loss performance with WFRLP, as shown earlier in Figure 4.5.

In Section 4.5.1, we discuss the advantages of WFRLP over JoBS-NS2, compare the JoBS-NS2 mechanism with the proposed algorithm, and main reason for the results in Figures 4.5, 4.6, and 4.7.

Table 4.2 summarizes the delay and buffer loss measurements, averaged over the entire simulation period and from all mobile nodes. Table 4.3 summarizes the average end-to-end delay and packet delivery ratio as measured from the eight marked CBR flows. Since the WFRLP achieves significantly smaller queueing delay, it is expected that the end-to-end delay is smaller than in JoBS-NS2.

Table 4.2: Average Queueing Delay and Buffer Loss (%) for WFRLP and JoBS-NS2

Scheme - Pause Time (secs)	I		II		III	
	Delay	Loss	Delay	Loss	Delay	Loss
WFRLP - 0	1.69ms	15.39	4.50ms	10.59	48.74ms	18.44
JoBS-NS2 - 0	1.87s	17.06	0.56s	5.67	1.96s	9.55
WFRLP - 25	2.45ms	14.41	3.02ms	12.11	47.61ms	20.02
JoBS-NS2 - 25	1.77s	13.63	1.53s	6.92	2.23s	10.14
WFRLP - 50	5.19ms	14.80	6.85ms	11.70	45.89ms	20.65
JoBS-NS2 - 50	2.03s	15.59	1.38s	5.74	2.81s	16.43
WFRLP - 100	2.16ms	16.21	3.50ms	11.18	41.25ms	21.85
JoBS-NS2 - 100	2.93s	14.65	1.41s	5.93	4.67s	15.20

Table 4.3: End-to-End Delay (msec) and Packet Delivery Ratio (PDR %) for WFRLP and JoBS-NS2

Scheme - Pause Time (secs)	I		II		III	
	Delay	PDR	Delay	PDR	Delay	PDR
WFRLP - 0	49.81	21.15	70.63	42.78	168.14	51.10
JoBS-NS2 - 0	2766.01	17.81	2577.34	37.97	1642.35	49.62
WFRLP - 25	49.38	23.54	63.45	31.82	168.39	52.03
JoBS-NS2 - 25	2345.22	28.86	2465.38	29.04	2010.47	45.78
WFRLP - 50	48.08	30.13	61.59	38.61	178.44	47.07
JoBS-NS2 - 50	2896.42	28.68	2529.94	35.71	2558.74	38.63
WFRLP - 100	42.55	28.93	56.84	41.97	164.54	41.40
JoBS-NS2 - 100	2160.31	25.94	2211.74	37.77	2352.21	33.75

4.5.1 Advantages of WFRLP over JoBS-NS2

In this subsection, we first summarize the ideas behind JoBS-NS2. It was proposed in [52] as a *quantitative Assured Forwarding* service under the Differentiated Service architecture for providing absolute and proportional differentiation of loss, service rates, and packet delays.

JoBS-NS2 performs *simultaneous* scheduling and buffer management by enforcing per-class QoS guarantees on loss, delay, and throughput by adjusting the service rate allocation and selectively dropping traffic. The authors of JoBS-NS2 in [30, 52] proposed *delay* and *loss* feedback loops.

For the delay feedback loop, they proposed a *linear* control model for expressing the relationship between the rate adjustment and desired queueing delay. The service rates are translated into packet scheduling decisions resembling Deficit Round Robin [30, Section 7.5.1]. That is, the scheduler tries to achieve desired service rates by keeping track of the difference between the actual transmission rate for each class and desired rates for each class.

The control model consists of the delay feedback loop and is designed to be linear and time-invariant. They then derive a stability condition on the linearized approximate model by bounding the gain of the controller to satisfy the rate constraint for each class. The authors admitted that these assumptions may not hold in general. They also stated certain inequality conditions on the controller where the system cannot satisfy absolute delay and rate guarantees and proportional delay differentiation at the same time [52]. In this case, the JoBS-NS2 relaxes the constraints, according to the given class precedence order on the service guarantees.

For the loss feedback loop of JoBS-NS2, traffic is dropped from a class queue to satisfy the proportional loss differentiation within the limits imposed by the absolute loss guarantees. If there is class buffer overflow, or an absolute delay and rate violation, traffic is dropped until the conditions are satisfied, or until the maximum

amount of traffic has been dropped. Once the maximum possible amount of loss is met, and there is still absolute delay and rate violations, these constraints are relaxed [30, 52].

Hence, as expected and seen in Figures 4.6 and 4.7, JoBS-NS2 incurs significantly higher delay and higher congestion level (in bits) due to its linearized control model. This approximate model may not hold in general. WFRLP provides an advantage as it does not require control model parameters.

Another issue of JoBS-NS2 is that it only enforces service guarantees over the duration of a busy period. The authors of JoBS-NS2 admitted that if the busy periods are short, enforcing QoS guarantees with the information on the current busy period is unreliable. For WFRLP, each agent uses its experience on previous busy periods due to the sequential nature of the SMDP, and does not suffer from this issue.

4.6 Possible Weaknesses of WFRLP

WFRLP faces a similar issue in Section 3.6 since it also uses WF^2Q for bandwidth provisioning for each agent. As mentioned in Section 4.4.2, this is only possible if the topology state evolves as an irreducible aperiodic (i.e. ergodic) Markov chain. We have already discussed the implications of this assumption and its applicability in actual networks in Section 3.6.

In this chapter, each agent *independently* solves its own locally-observed SMDP, without knowing the policies of other agents. Although WFRLP is easier to implement as each node does not need information about other nodes, accurate estimate of the global optimal reward may not always be obtained. We address this issue in Chapter 6 for the decentralized case.

The proposed Wire-Fitted CMAC for continuous state and action vector spaces in Figure 4.2 uses a function approximation technique to find the near-optimal state-

action values. The state-action values are then used to construct the optimal policy. As explained in Section 4.4, this technique can achieve a fast search of the greedy action (i.e. $\arg\max_{a \in A} R^*(s,a)$ in (4.15)) by only searching a constant number of representative vectors, rather than searching the entire multi-dimensional space.

However, in-depth convergence analysis is needed to characterize the error bound as the algorithm converges to the optimal solution. Further investigation is needed to find the best structure for the Wire-Fitted CMAC (i.e. number of wire vectors, CMAC configuration, and learning and decaying rates) and this maybe specific to the problem.

4.7 Chapter Summary

We have proposed a *bandwidth allocation* and *buffer management* scheme in a time-varying channel and topology, with consideration of bandwidth, queueing delay and buffer loss constraints. Each node acts as an agent that observes its own average reward SMDP independently. Each agent uses a model-free RL algorithm that allows to learn the optimal policy through sequential decision without knowing the transition probabilities that encompasses the dynamic nature of the network.

Simulation results show that the proposed algorithm known as *Wire-Fitted Reinforcement Learning Provisioning (WFRLP)* is able to attain its objective of provisioning bandwidth and buffer to meet QoS requirements in a cost effective manner. Due to the continuous and multi-dimensional nature of the states and actions of the SMDP, we also propose a novel *Wire-Fitted CMAC* structure that is suitable for fast and real-time learning and is able to attain good convergence as shown in our simulation results.

Figure 4.8 summarizes the main ideas of this chapter. We have formulated the resource allocation problem (i.e. bandwidth allocation and buffer management) as a

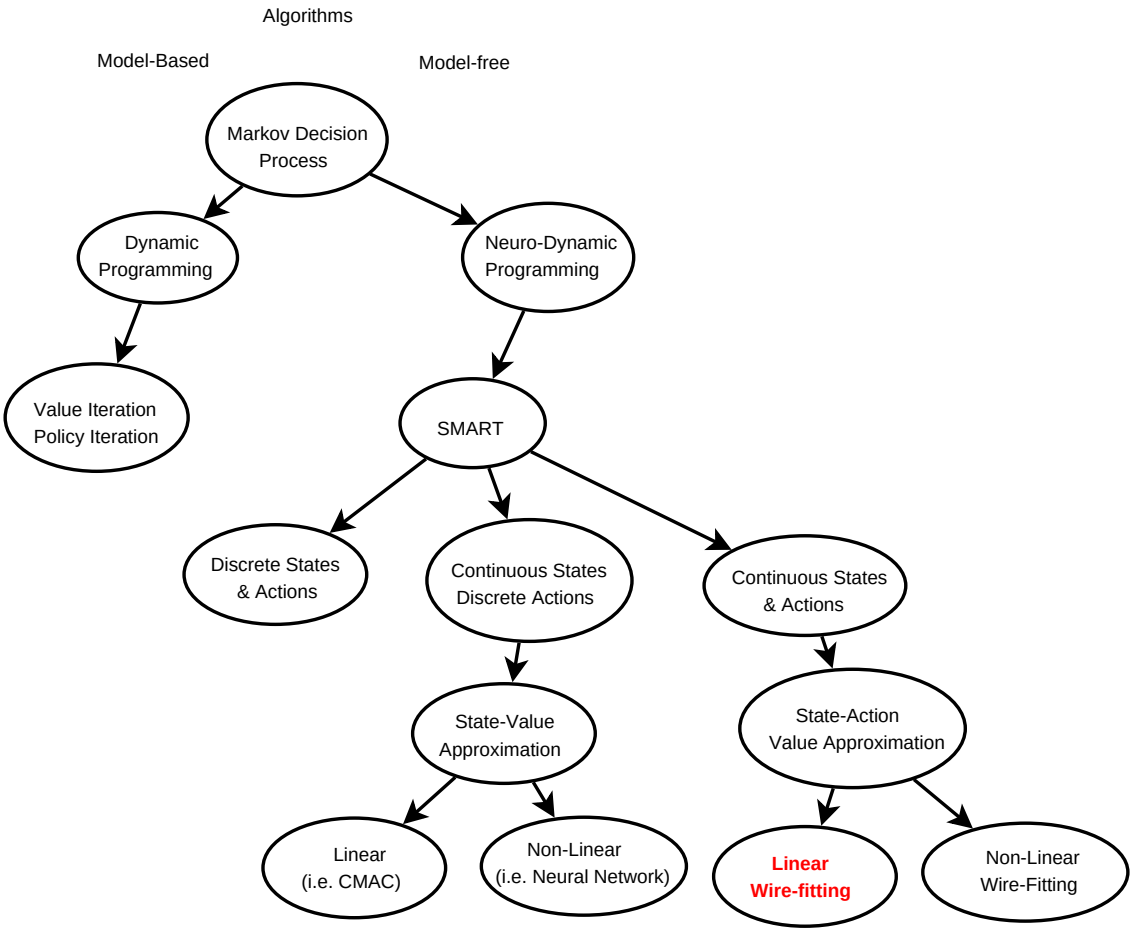


Figure 4.8: Model-Free Approach & Implementation Issues

SMDP for each independent agent. We then use a model-free RL technique known as SMART with state-action values. Due to the nature of having continuous state and action vector spaces, implementing the RL algorithm requires infinite storage. Thus, we propose a state-action value approximation structure known as Wire-Fitted CMAC, which is a combination of the CMAC linear approximator and wire-fitting for fast action interpolation. This idea is highlighted as *Linear Wire-fitting* in Figure 4.8.

Chapter 5

Hierarchical Semi-MDP Approach for QoS Provisioning

This chapter presents the third variant of MDP formulation in this thesis. This type extends the independent agent SMDP formulation in Chapter 4, especially in the QoS provisioning problem with large and continuous state and action vector spaces.

We observe that in Section 4.4, even though the desired action vector with the highest state-action value can be retrieved compactly, the *flat* RL algorithm is still searching from a large continuous vector space. This effectively does not prevent each agent from choosing costly actions, especially during the initial exploration space, which then contributes to slower convergence.

In this chapter, we present a novel solution to address this issue. Using the idea of *divide-and-conquer*, the original SMDP formulation for an agent is decomposed into smaller sub-problems. Intuitively, this mechanism accelerates the process of finding the optimal solution, since the smaller problems are relatively easier to solve. Formally, we present the Hierarchical Semi-Markov Decision Process (HSMDP) and Hierarchical Reinforcement Learning (HRL).

It should be noted that HSMDP and HRL are still based on the single-agent framework as described in Section 1.3, where each node acts as an agent that tries to solve its own locally-observed HSMDP, independent of other agents. Figure 5.1 shows a general system model for independent HSMDP agents.

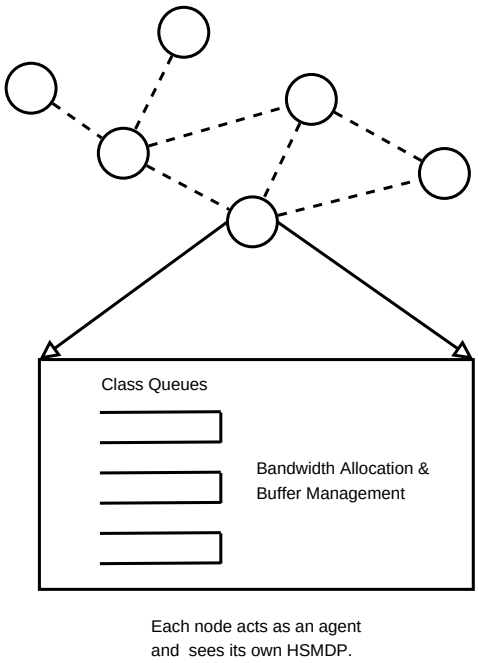


Figure 5.1: Independent HSMDP agents for resource allocation in MANETs

The case for the multi-agent framework is discussed in Chapter 6.

Section 5.1 formally introduces the HSMDP theoretical framework and a model-free HRL algorithm. Section 5.2 then describes our proposed HRL-based resource allocation scheme, where each node acts as an agent solving its own HSMDP independently. In Section 5.3, we present and discuss simulation results obtained using the NS2 network simulator. We effectively compare the performance of the non-hierarchical RL algorithm from Chapter 4 and the new proposed HRL-based method in this chapter.

5.1 Hierarchical SMDP and Hierarchical RL

Hierarchical Semi-Markov Decision Process (HSMDP) is a framework for large domains by using a task or action structure to restrict the search of policies. A large MDP problem is decomposed into a set of tasks, and each task can be further decomposed into a collection of subtasks and so on, up to a desired level of hierarchy.

The key principle behind HSMDP is to reuse learned policies by the subtasks in the task hierarchy. The corresponding model-free technique for HSMDP is Hierarchical Reinforcement Learning (HRL).

We emphasize that well-known *flat or non-hierarchical* RL algorithms, such as Q-Learning [17] and the SMART algorithm used in [13, 18] and Section 4.2, do not apply here anymore due to the *hierarchical task structure* of HSMDP.

At the lowest level of the task hierarchy are *primitive actions* that immediately terminate after execution. In a non-hierarchical setting, primitive actions correspond to the actual action obtained from the flat RL algorithm as in Section 4.2. In the hierarchical setting, the higher level tasks above the primitive actions are subtasks known as *non-primitive or temporally-abstract actions.* This type of actions can take a variable amount of time to finish. HRL determines how lower-level policies over subtasks or primitive actions can themselves be composed into higher level policies. Policies over primitive actions or tasks are considered *Semi-Markov* when composed at the next level up due to the variable execution times.

The SMDP model described in Section 4.1 has been the mathematical framework for analyzing this concept of *temporal abstraction* [16] used in HSMDP, where at a given task level in the task hierarchy, decisions are not required at each step, but invoke a sequence of temporally-extended activities which follow their own subtask policies until termination.

Several approaches to HSMDP and HRL have been proposed, including the options formalism [53], hierarchies of abstract machines (HAMs) [54] and the MAXQ framework [55]. The difference among these three well-known approaches lies in how to specify the subtasks in the hierarchy. In this chapter, we leverage on the MAXQ framework and its *value function decomposition*, due to its ability to represent the state and action values in a more compact and reusable manner. However, as the original MAXQ framework is formulated for discounted reward criterion, we use its

extension for average reward criterion found in [42].

5.1.1 Hierarchical Task Decomposition in HSMDP

HSMDP decomposes the overall task MDP M into a finite set of subtasks

$$\{M_0, M_1, ..., M_n\}$$

where M_0 is the root task and solving it solves the original MDP M.

Each non-primitive subtask i consists of a tuple (T_i, A_i, R_i) [42] where:

$T_i(s_i)$ is a termination predicate that partitions the MDP state space S into a set of active states S_i, and a set of terminal states T_i. The policy of subtask M_i can be executed only if $s \in S_i$. Subtask i terminates when it reaches a state in T_i.

A_i is a set of actions that can be executed to perform subtask M_i. These actions can be the primitive actions from the MDP action space A or can be other subtasks.

R_i is the reward structure inside subtask i. Besides the reward of the overall task (MDP M), each subtask can use additional rewards to guide its learning of policies.

A hierarchical policy w is defined as the set of policy for each of the subtasks in the hierarchy: $w = \{w_0, w_1, ..., w_n\}$. Under a hierarchical policy w, a multi-step transition probability function is defined for each subtask i: $P_i^w : S_i \times \aleph \times S_i \rightarrow [0, 1]$, where $P_i^w(s', K \mid s)$ is the probability that action $\mu_i(s)$ will cause the system transition from state s to s' in K primitive steps. Each subtask i can then be modeled by a SMDP with the components (S_i, A_i, P_i^w, R_i) .

An example of a task graph structure is shown in Figure 5.2. The *Root* task is found at the highest level, while the *ReduceDelay*, *ReduceLoss,* and *IncreaseThroughput* are the non-primitive actions or subtasks for *Root*. The *PerformBA* and *PerformBM* tasks are the possible actions or subtasks for the next lower level. The *ActionBA* for *PerformBA* (i.e. perform bandwidth allocation) is a primitive action

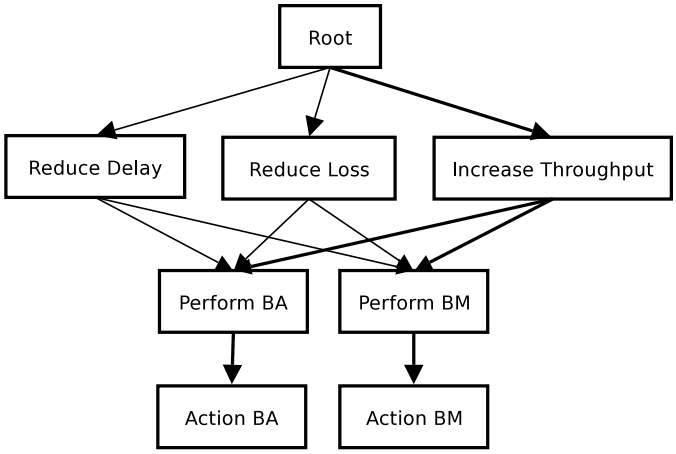


Figure 5.2: Example Task Graph

or task which corresponds to the actual action vector taken by the agent from the continuous action space. Similarly, *ActionBM* is a primitive action for *PerformBM*, the subtask for buffer management.

The concept of *temporal abstraction* can be explained as follows: the *Root* initially chooses among its non-primitive subtasks according to its policy, and these subtasks in turn follow their own policies until they themselves terminate. The primitive actions terminate immediately after execution.

5.1.2 Optimality

The hierarchical task graph structure is generally specified by the designer using prior knowledge about the problem. This essentially reduces the size of the space for searching a good policy. However, the hierarchy itself constrains the possible policies that it may not be able to represent the optimal policy or its value function. Two types of optimality have been defined to tackle this issue:

1. *Hierarchical optimal policy* is a hierarchical policy which has the best performance among all policies consistent with the hierarchy. The policy for each subtask may not be optimal, but the policy for the task hierarchy is optimal.

2. *Recursive optimal policy* is a hierarchical policy such that for each subtask M_i, the corresponding policy μ_i is optimal for the SMDP defined by (S_i, A_i, P_i^w, R_i).

In this work, we consider a hierarchical optimal policy for the QoS provisioning problem. Following Section 4.3, our objective is to maximize average long term network reward which effectively translates into minimizing average long term QoS violations. Hence, the goal is to find a hierarchical optimal policy that maximizes the average long term reward of the overall task.

5.1.3 Hierarchically Optimal Value Function Decomposition

In composing policies for tasks from lower level subtasks, the need for compactly storing the value of the state-action pairs at different task levels is crucial in order for policy reuse. This idea is exploited in the MAXQ framework [55] that allows the storing and decomposing of the value function in a distributed manner in the nodes of the task graph. However, MAXQ value decomposition is only proven to be *recursively optimal* and for discounted reward criterion. The separation of the value function for a hierarchical optimal policy is discussed in greater details in [42]. We summarize the hierarchical optimal value decomposition for average reward criterion as follows:

We assume the root task in the task hierarchy is a *continuing task* (i.e. the overall root task goes on without termination). In addition, we assume that for every possible stationary policy consistent with the overall hierarchy, the embedded Markov chain has uni-chain transition probability distribution and as a result, the whole task is *uni-chain* SMDP [56]. This assumption also implies that the average reward for the root (i.e. overall problem) is well defined for every hierarchical policy and does not vary with initial state. We define the overall gain to the hierarchical policy w as ρ^w , also defined as the gain of the root task.

The *hierarchical value function* of a subtask contains the reward received during subtask execution and the reward after it terminates. This is crucial in order to find

the hierarchical optimal policy. This also implies that the expected reward depends on the subtask and all its calling ancestors up to the root of the hierarchy. Similar to (4.5) in the flat uni-chain SMDP model, we define the hierarchical average adjusted value function $H^w(i,s)$ for hierarchical policy w and subtask M_i as the average adjusted sum of rewards earned until M_i terminates plus the average adjusted reward outside M_i:

$$H^w(i,s) = \lim_{K\to\infty} E_s^w \left\{ \sum_{t=0}^{K-1} \left(r(s_t,a_t) - \rho^w \tau_t \right) \right\} \tag{5.1}$$

where τ_t is the length of decision period.

Decomposing (5.1) can be stated as follows [42]: Suppose the first action chosen by policy w is executed and terminates after N_1 primitive steps in s_1 according to $P_i^w(s_1, N_1 \mid s, w_i(s))$, where $w_i(s)$= action taken by policy w_i for subtask M_i. Afterwards, subtask M_i itself executes for N_2 steps at the level of subtask M_i (i.e. N_2 is the number of actions taken by subtask M_i, not the number of primitive actions) and terminates in s_2 according to $F_i^w(s_2, N_2 \mid s_1)$, also known as the abstract transition probability at the level of subtask M_i. A Bellman equation can then be written as:

$$\begin{aligned} H^w(i,s) = r\left(s, w_i(s)\right) - \rho^w y_i\left(s, w_i(s)\right) \quad &+ \sum_{N_1, s_1 \in S_i} P_i^w\left(s_1, N_1 \mid s, w_i(s)\right) \hat{H}^w(i, s_1) \\ &+ \sum_{s_1 \in S_i} F_i^w\left(s_1 \mid s, w_i(s)\right) M_i^w(s_1, i) \end{aligned} \tag{5.2}$$

where:

$$M_i^w(s_1,i) = \sum_{N_2, s_2 \in S_i} F_i^w\left(s_2, N_2 \mid s_1\right) H^w(pt(i), s_2)$$

$$F_i^w\left(s_1 \mid s, w_i(s)\right) = \sum_{N=1}^{\infty} P_i^w\left(s_1, N \mid s, w_i(s)\right)$$

$pt(i)$ is the calling parent task of subtask M_i

$y_i\left(s, w_i(s)\right)$ is the expected length of time until the next decision period after invoking

action $w_i(s)$

$F_i^w(s_1 \mid s, w_i(s))$ is the single-step transition probability function for subtask M_i, defined by marginalizing the multi-step probability function P_i^w.

$\hat{H}^w(i, s_i)$ denote the *projected* average adjusted value function of hierarchical policy w and subtask M_i, defined as the average adjusted sum of rewards of executing the policy w_i and the policies of all the descendants of M_i starting in s_i, until M_i terminates

Since $r(s, w_i(s))$ is the expected total reward between two decision epochs of subtask M_i given that the system was at state s and executed action $w_i(s)$, we have: $r(s, w_i(s)) = \hat{H}^w(w_i(s), s) + \rho^w y_i(s, w_i(s))$. The Bellman equation can then be written as:

$$\begin{aligned} H^w(i,s) &= \hat{H}^w(w_i(s), s) \\ &+ \sum_{N_1, s_1 \in S_i} P_i^w(s_1, N_1 \mid s, w_i(s))\, \hat{H}^w(i, s_1) \\ &+ \sum_{s_1 \in S_i} F_i^w(s_1 \mid s, w_i(s))\, M_i^w(s_1, i) \end{aligned} \tag{5.3}$$

The corresponding hierarchical average adjusted action-value function for (5.3), similar to the action-value representation in (4.7) for flat uni-chain SMDP, can be expressed as:

$$\begin{aligned} L^w(i,s,a) &= \hat{H}^w(a, s) \\ &+ \sum_{N_1, s_1 \in S_i} P_i^w(s_1, N_1 \mid s, a)\, \hat{H}^w(i, s_1) \\ &+ \sum_{s_1 \in S_i} F_i^w(s_1 \mid s, a)\, K_i^w(s_1, i) \end{aligned} \tag{5.4}$$

where:

$$K_i^w(s_1, i) = \sum_{N_2, s_2 \in S_i} F_i^w\left(s_2, N_2 \mid s_1\right) \hat{L}^w(pt(i), s_2, w_{pt(i)}(s_2))$$

$\hat{L}^w(i, s, a)$ denote the *projected* average adjusted action-value function of hierarchical policy w and subtask M_i, defined as the average adjusted sum of rewards of doing a in state s once, and executing the policy w_i and the policies of all the descendants of M_i thereafter, until M_i terminates

The projected average adjusted value function $\hat{H}^w(i, s)$ in (5.2) can be redefined as:

$$\hat{H}^w(i, s) = \begin{cases} \hat{L}^w(i, s, w_i(s)), & \text{if } i \text{ is a composite task} \\ \sum_{s'} P\left(s' \mid s, i\right)\left(r(s' \mid s, i) - \rho^w\right), & \text{otherwise} \end{cases} \tag{5.5}$$

A primitive task i is synonymous to a primitive action which terminates immediately after execution and is found at the lowest level of the task hierarchy. In the task graph structure, the primitive task i stores its own $\hat{H}^w(i, s)$. A non-primitive task i, also known as a composite task or action, is a higher level task that takes a variable amount of time to finish. It stores the value functions $\hat{L}^w(i, s, a)$ and $L^w(i, s, a)$ for each possible action a under non-primitive task i.

5.1.4 Hierarchically Optimal Average Reward RL Algorithm

As RL methods do not require the probability distribution functions, as defined in the previous subsection, the optimal policy is learned or approximated using the Bellman equations in (5.3), (5.4), and (5.5) for estimating the optimal values of the different value functions: $\hat{H}^w(i, s)$, $L^w(i, s, a)$, and $\hat{L}^w(i, s, a)$ [42, 56].

Once the optimal value functions are obtained, the optimal policy for subtask M_i is

given as: $w_i^*(s) = \arg\max_{a \in A_i} L^{w^*}(i, s, a)$. We term the algorithm as the Continuous-time Hierarchically Optimal Average Reward (CHO-AR) RL algorithm and its pseudo-code is shown in Algorithm 1. The exploration policy mentioned in the pseudo-code is similar to the exploration scheme described in Section 4.2 in the flat RL algorithm. This algorithm is similar to the ideas used in [42, 56], but applied in a continuous-time, hierarchically optimal context.

Algorithm 1 Continuous-time Hierarchically Optimal Average Reward RL algorithm

Function CHO-AR(Task i, State s)
Let Seq be the sequence of states visited while executing i
If i is a primitive action then
  Execute action i in state s, observe s' and
  reward $(k(s,i) + r(s',s,i)\tau)$, Update $\hat{H}_t^w(i,s)$:
  $\hat{H}_{t+1}^w(i,s) = (1-\alpha_t)\hat{H}_t^w(i,s) + \alpha_t\left(k(s,i) + r(s',s,i)\tau - \rho_t\tau\right)$,
  where τ= time interval between the decision epochs
  If i and all its calling ancestors are non-random actions then
    update the global average reward:
    $\rho_{t+1} = \frac{r_{t+1}}{\tau_{t+1}} = \frac{r_t + k(s,i) + r(s',s,i)\tau}{\tau_t + \tau}$
  End if
  Push state s into the beginning of Seq
Else
  While i has not terminated do
    Choose action a according to current exploration policy $w_i(s)$
    Let $ChildSeq$ = CHO-AR(a,s), where $ChildSeq$ is the
    sequence of visited states while executing action a
    Observe the resulting state s'
    Let $a^* = \arg\max_{a' \in A_i(s')} L_t^w(i,s',a')$
    For each s in $ChildSeq$ from the beginning do
      Let $target = \hat{H}_t^w(a,s) + \hat{L}_t^w(i,s,a^*)$
      $\hat{L}_{t+1}^w(i,s,a) = (1-\alpha_t)\hat{L}_t^w(i,s,a) + \alpha_t target$
      $L_{t+1}^w(i,s,a) = (1-\alpha_t)L_t^w(i,s,a) + \alpha_t target$
    End for
    Append $ChildSeq$ onto the front of Seq
    $s = s'$
  End while
End if
Return Seq
End Function CHO-AR

5.2 HRL for QoS Provisioning

Using the HSMDP framework requires the designer to decompose the problem into a task structure using available prior knowledge. As mentioned in Section 5.1, we use this framework to efficiently search for actions in the continuous action space in order to handle the QoS constraints and to prevent the agent from invoking actions that are costly and unfavorable.

In incorporating prior knowledge, we use a similar approach in [52] to search for the desired actions (i.e. how much bandwidth to allocate and what is the packet drop rate for each class). We define the target bandwidth $r_{j,min}$ for network class j for handling the rate and delay constraints mentioned in Section 4.3 as follows: If $d_{j,max} > d_j$,

$$r_{j,min} = \max\left(\frac{B_j}{d_{j,max} - d_j}, b_{j,min}\right) \tag{5.6}$$

where B_j is the total buffer size of class j (in bits)

The target bandwidth $r_{j,min}$ effectively captures the required bandwidth to clear the buffer, without any buffer loss, within the maximum allowed delay. If $d_{j,max} \leq d_j$, the target bandwidth is set to the observed link capacity C_h, so as to quickly clear the buffer [52].

We use the same state and action descriptors in (4.10) and (4.11) for the HSMDP formulation. However, we redefine the reward structure by replacing $b_{j,min}$ with $r_{j,min}$ as follows:

$$r(\mathbf{S}', \mathbf{S}, \mathbf{a}) = \tau(\mathbf{S}', \mathbf{S}, \mathbf{a}) \left\{ \sum_{j=1}^{J} r_{j,price} \left\{ \frac{r'_j - r_{j,min}}{C_h} \right\} + \sum_{j=1}^{J} d_{j,price}(d_{j,max} - d_i) + \sum_{j=1}^{J} l_{i,price}(l_{j,max} - l_j) \right\} \tag{5.7}$$

As defined in (4.10), the state descriptor contains the actual queueing delay and buffer loss measurements in a node. The state effectively captures the achieved QoS information. Given a state s, the target bandwidth for the network classes can be easily computed from (5.6). As can be deduced, the state s can be found in one of the following regions in the QoS provisioning state space:

Region A_1: $\sum_j r_{j,min} < C_h$ and $d_{j,max} > d_j$, $\forall j$: No rate and delay violations.

Region A_2: $\sum_j r_{j,min} = C_h$: No rate violations.

Region A_3: $\sum_j r_{j,min} > C_h$.

The three regions above help us to define the proposed task graph hierarchy for QoS provisioning. Region A_1 is considered as the best region since the required bandwidth is satisfied to clear the class queue without any buffer loss and delay violations.

In the network, as the actual delay and loss measurements vary, the position of state s also varies among the three regions. The QoS provisioning problem can thus be stated as follows: Given the current varying state s, the agent needs to perform BA and BM to *bring* or *maneuver* the state into the best region, where the average long term reward is maximized. By intuition, the ideal solution is when the agent can bring the state into Region A_1 and stay in this region. However, due to network dynamics and interaction or communication among the nodes, this case may not be always possible as the achieved QoS vary. The goal is still for the agent to perform BA and BM to *navigate* the state into a region where it can maximize its long term reward, which effectively translates into minimizing long term QoS violations. The problem can thus be easily translated into a *navigation problem* for the agent.

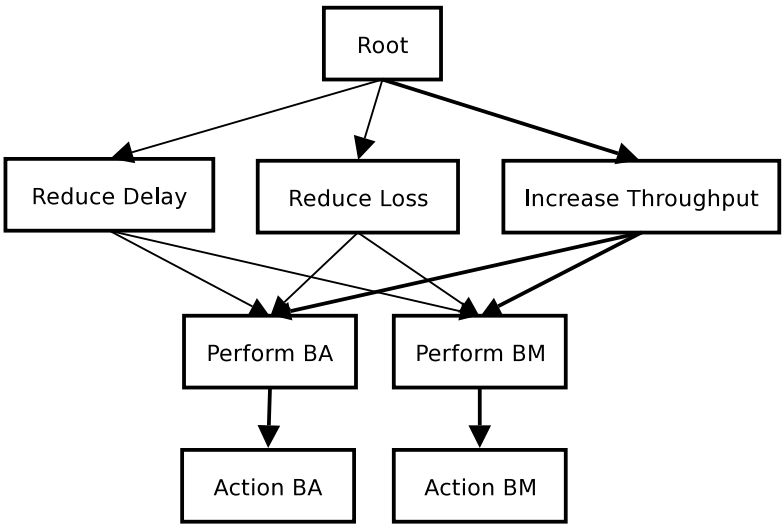


Figure 5.3: Task Graph for QoS provisioning

Using the idea of navigation for QoS provisioning, we use a simple task graph structure for the HRL formulation in Figure 5.3. We define the following tasks:

- *Root*. This is the overall task. We assume that the root is a continuing task that does not terminate as mentioned in Section 5.1.3, with the objective of maximizing average long term reward.
- *ReduceDelay*. This composite task is used to reduce the queueing delay.
- *ReduceLoss*. This composite task is used to reduce the buffer loss.
- *IncreaseThroughput*. This composite task is used to increase throughput.
- *PerformBA*. This composite task is used for performing bandwidth allocation. *ActionBA* is a primitive continuous action.
- *PerformBM*. This composite task is used for buffer management. *ActionBM* is a primitive continuous action.

The *PerformBA* and *PerformBM* tasks are the composite actions for *ReduceDelay*, *ReduceLoss* and *IncreaseThroughput*. The *PerformBA* obtains the actual action vector from the primitive action *ActionBA* for bandwidth allocation, while primitive action *ActionBM* gives the action vector for buffer management in the *PerformBM* task. This level of abstraction is needed since the composite tasks *PerformBA* and *PerformBM* are searching the continuous action space.

The sequence of action selection for the different subtasks is obtained from the task policy itself that is being learned. *PerformBA* and *PerformBM* terminates immediately after the corresponding action vector (i.e. *ActionBA* or *ActionBM*) has been taken. For the *ReduceDelay* subtask, it terminates once the queueing delay of at least one class has been reduced. The termination predicates of *ReduceLoss* and *IncreaseThroughput* are defined in a similar manner. The agent is using the different tasks in the task hierarchy for navigation in the different regions.

Figure 5.4 shows the *MAXQ graph* for the task graph defined in Figure 5.3. The MAXQ graph is a graphical representation of the value function decomposition and is used for the design and implementation of the RL algorithm itself [55]. The graph contains two types of nodes: *Max nodes* and *Q nodes*. The Max nodes correspond to the subtasks in the task decomposition as shown in Figure 5.3, where each primitive action is represented by a Max node, while each subtask, including the Root, is also represented by a Max node. The Q nodes correspond to the actions that are available for each subtask.

For the CHO-AR pseudo-code in Algorithm 1, each primitive Max node i stores the value of $\hat{H}^w(i,s)$. Each Q node for parent task i, state s, and subtask a stores the values of $L^w(i,s,a)$ and $\hat{L}^w(i,s,a)$.

As mentioned earlier, the subtasks *PerformBA* and *PerformBM* are searching the *continuous* action space and hence, there are infinitely many possible primitive action vectors. To represent this in the MAXQ graph, we define a *continuous* Q node to

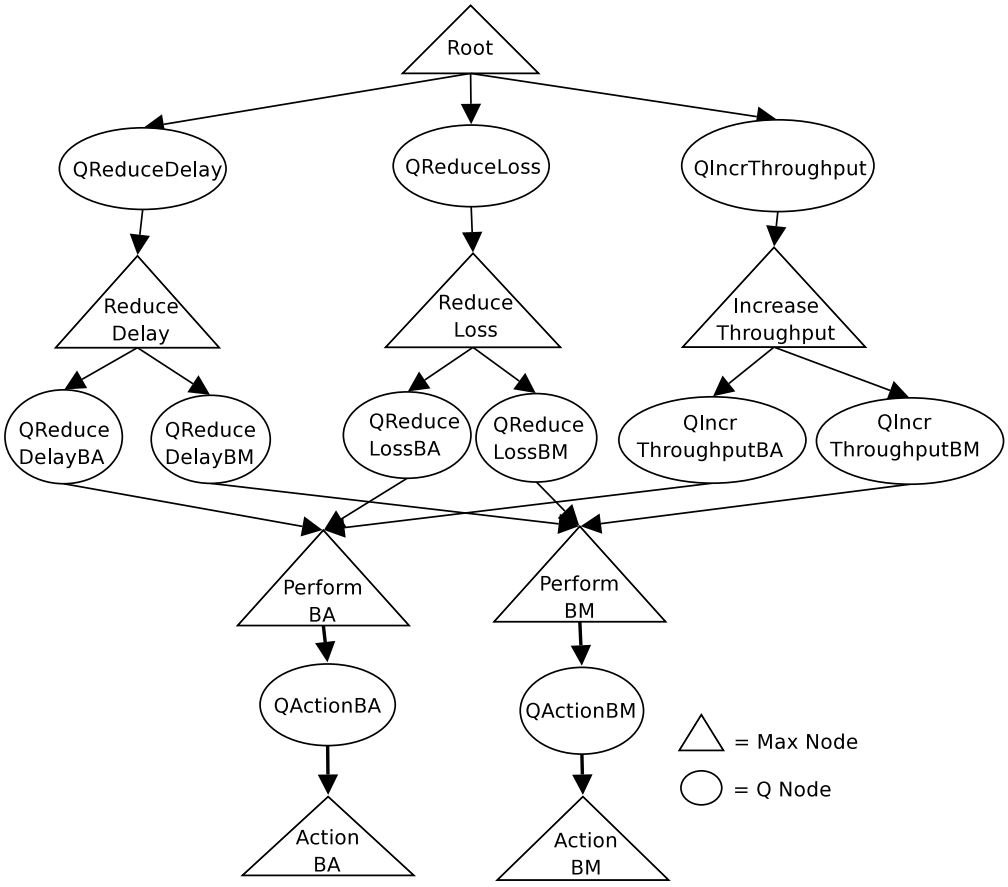


Figure 5.4: MAXQ graph for QoS Provisioning

handle a continuous action space. The node *QActionBA* is a continuous Q node which stores the value functions $L^w(i,s,a)$ and $\hat{L}^w(i,s,a)$ for the possible action vectors a for BA. The node *ActionBA* is also defined as a *continuous* primitive Max node and stores $\hat{H}^w(a,s)$ for the possible action vectors a for BA. Similarly, nodes *QActionBM* and *ActionBM* are continuous Q node and continuous primitive Max node for BM, respectively. These value functions in the respective continuous Q nodes and Max nodes are stored compactly using the Wire-Fitted CMAC structure in Figure 4.2.

Note that this state-action representation for continuous vector space with continuous Q nodes and Max nodes is our proposed extension to the usual MAXQ graph for discrete vector spaces. In addition, the proposed CHO-AR technique in Algorithm 1 uses this compact representation which differs from [42, 56].

We term the proposed HRL-based solution for QoS provisioning as the *Hierarchical Optimal Reinforcement Learning Provisioning* (HORLP) algorithm.

Similar to the flat RL-based algorithm in Section 4.4, we use the work-conserving scheduler known as worst-case fair weighted fair queueing (WF^2Q). The HORLP algorithm learns the WF^2Q weights for bandwidth allocation and the packet drop rate for buffer management.

We observe that the buffer management component of HORLP can be used in a wired or wireless scenario. However, for bandwidth allocation, fair queueing techniques such as WF^2Q are generally known to be only applicable in a wired network [36]. To justify the use of WF^2Q under a time-varying *channel medium* and *topology*, we use the same idea as in Section 4.4.2 and Theorem 3.1.

Essentially, by assuming that each agent observes a time-varying channel process that evolves as an embedded irreducible aperiodic finite-state Markov chain, we have the same case as in Section 3.1. Again, the difference is we consider the decision instants or periods as the single time slot boundaries of the queueing law in (4.16). For a time-varying topology process, the same result applies, where the service process or the amount of bits coming out of a local class queue is *rate convergent*. From Theorem 3.1, fair queueing techniques such as WF^2Q can thus be applied due to the *rate convergence* property of the service process for each class j. Therefore, the WF^2Q weights are used to provision and allocate the total effective rate $\sum_{j=1}^{J} \mu_{j,av}$ among the class queues. This total effective rate effectively represents the observed capacity C_h in the HSMDP formulation in (5.7).

5.3 Simulation Results for HRL-based provisioning

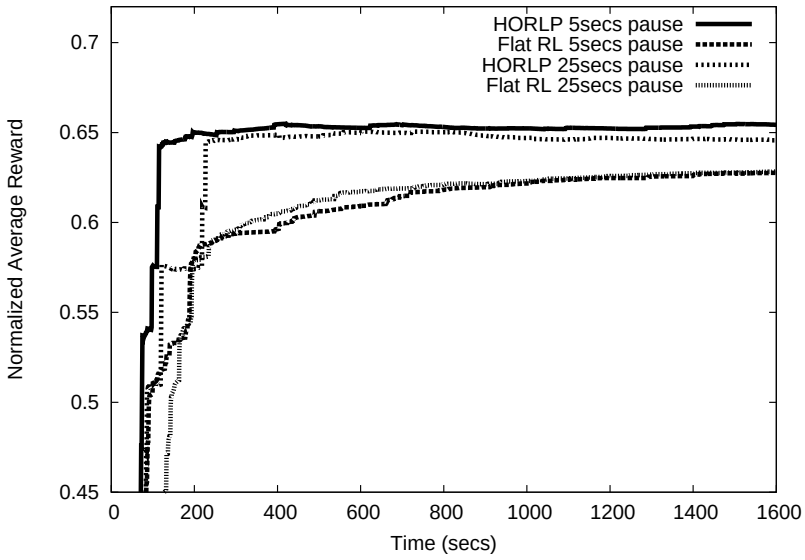
In this section, we simulate the same scenario described in Section 4.5, with a network of 20 mobile nodes in a 1,000m by 1,000m as shown in Figure 3.3. The maximum

channel capacity is 2 Mbps, while both the queue size of the interface queue and routing protocol have a depth of 50 packets.

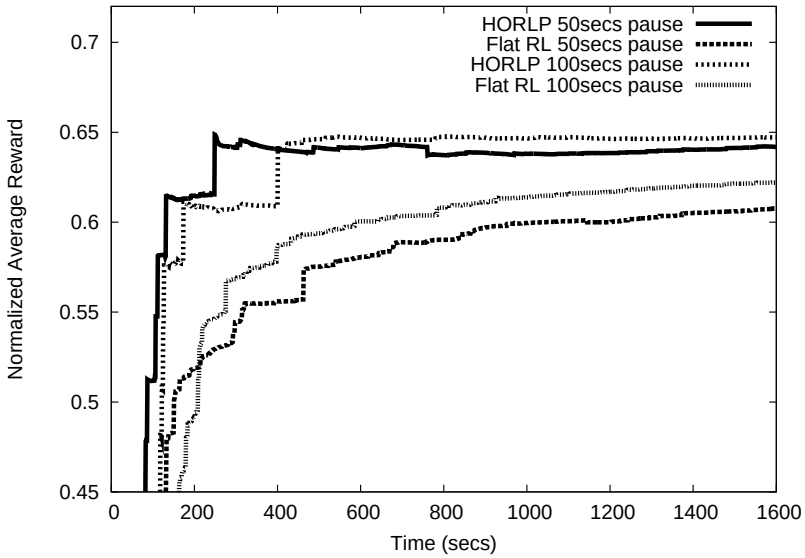
We use the same traffic class definitions and flow characteristics from Tables 4.1 and 3.1, respectively. Specifically, we simulate eight long-lived CBR flows. We have used CBR connections since this type of flows captures the worst case and average long term performance.

We have discussed in Section 4.4.2 that the MAC mechanisms and varying topology in the network is captured in our model through the concept of a *topology state process* that evolves as an irreducible aperiodic Markov chain. It should also be noted that there have been a number of works that uses DCF in MANETs with varying scenarios under the Markov chain theory [32, 33]. Our approach is different because we use the *controlled* Markov chain under the HSMDP framework.

We compare the performance of the HRL-based HORLP algorithm with the Flat RL algorithm discussed in Section 4.4 with respect to the normalized average reward. It should be noted that the average reward is a measure of how well the algorithm satisfies the QoS constraints in (5.7).



(a) With 0 and 25 secs pause times

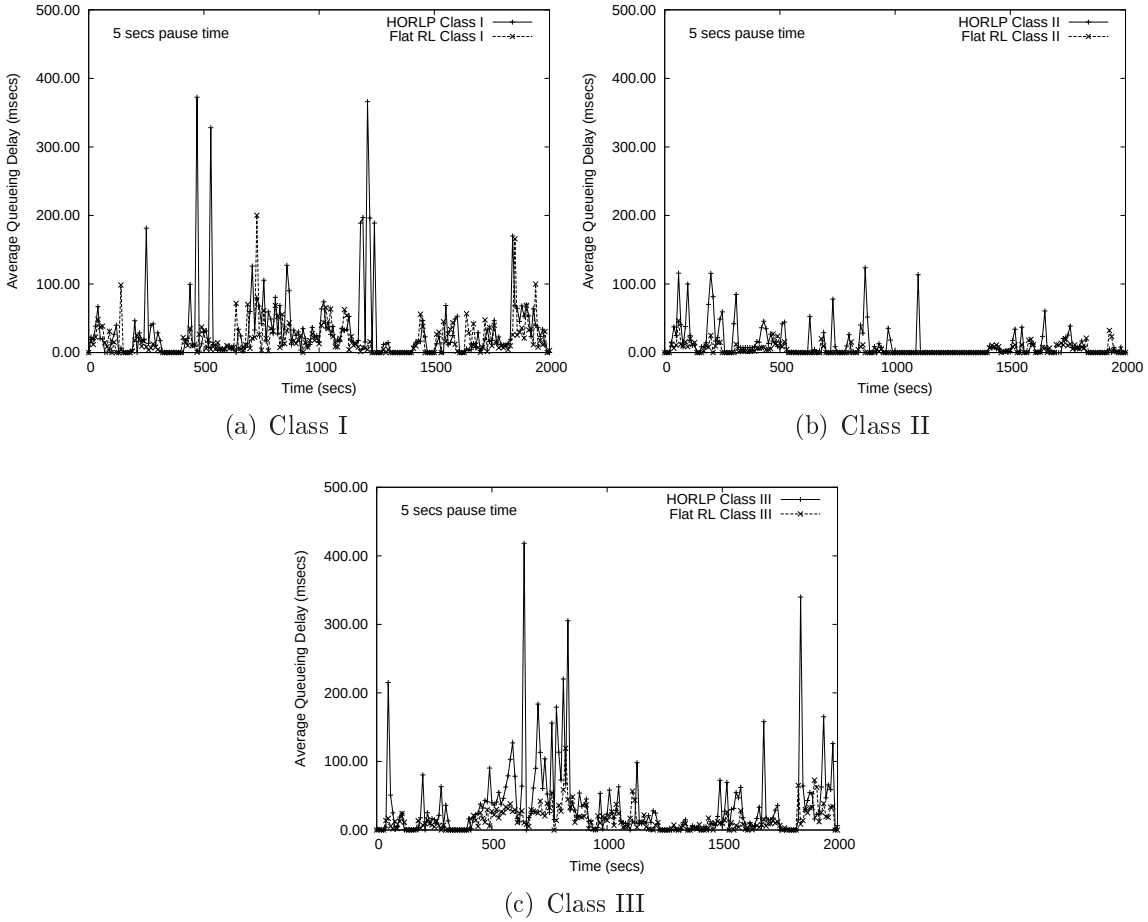


(b) With 50 and 100 secs pause times

Figure 5.5: Normalized Average Reward for HORLP and Flat RL with different pause times

Figure 5.5 shows the average reward for HORLP and Flat RL algorithms under varying pause times. It shows that HORLP achieves faster and more robust convergence as compared with the Flat RL algorithm. This can be attributed to the task

structure in HRL that allows the agent to reuse subtask policies whenever the agent is in any region in the QoS provisioning state space as described in Section 5.2. The figure also shows that HORLP attains a convergence limit close to 0.6513 for different scenarios and pause times. This result supports our claim in Section 4.4.2 that different MAC mechanisms and varying topology issues are captured in our model due to the *topology state process* concept.



(a) Class I

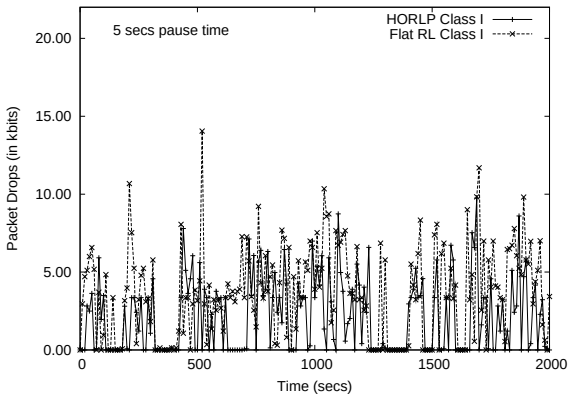
(b) Class II

(c) Class III

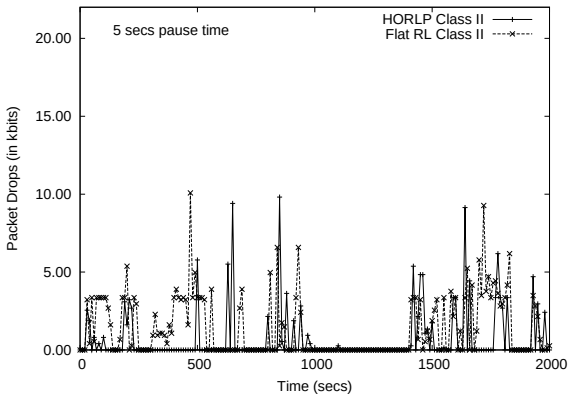
Figure 5.6: Queueing delay for HORLP and Flat RL under 5 secs pause time

Figure 5.6 shows the queueing delay measurements in milliseconds. Due to the dynamic network scenario, the absolute delay bound is not satisfied for both algorithms. However, our objective is to minimize the average long term QoS violations, which is effectively reflected by the average long term reward. This objective is satisfied

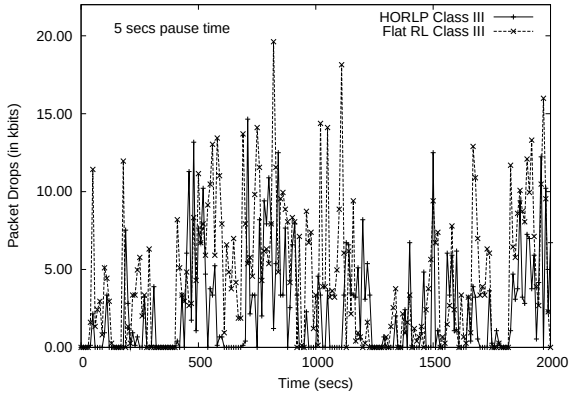
where HORLP achieves more robust and faster convergence in Figure 5.5. It should also be noted that HORLP appears to incur higher queueing delay than Flat RL. The main reason for this is explained later in conjunction with buffer drop measurements in Figure 5.7.



(a) Class I (in kbits)



(b) Class II (in kbits)



(c) Class III (in kbits)

Figure 5.7: Buffer drop measurements for HORLP and Flat RL under 5 secs pause time

Figure 5.7 shows the packet buffer drop measurements in bits using HORLP and Flat RL for the three different classes. The figure shows that Flat RL incurs relatively higher packet drops than HORLP for all classes. As we have discussed in Section 5.1, Flat RL does not prevent the agent from choosing costly and ineffective actions as it searches in the continuous action space, and thus suffer slower convergence as shown in Figure 5.5.

As expected, since Flat RL incurs higher packet drops, the amount of packets in the class queues is less than that of HORLP. This explains the reason why HORLP appears to incur higher queueing delay as mentioned earlier and shown in Figure 5.6, since HORLP has more packets in the queue. There is a clear trade-off between packet drops and queueing delay.

Table 5.1: Average Queueing Delay (secs), Buffer Drops (bits) and Congestion Level (in bits) Measurements for HORLP and Flat RL

Scheme - Pause Time (secs)	I			II			III		
	Delay	Drops	Congestion	Delay	Drops	Congestion	Delay	Drops	Congestion
HORLP - 5	0.042	2418.83	2097.98	0.013	995.31	111.437	0.041	2602.79	323.56
Flat RL - 5	0.023	4100.53	1962.53	0.005	1726.42	96.45	0.017	4567.77	257.64
HORLP - 25	0.048	2573.98	2389.15	0.023	1662.62	1526.89	0.036	2211.61	2071.05
Flat RL-25	0.024	4302.86	2003.05	0.010	2844.56	1300.64	0.015	4173.46	1849.02
HORLP - 50	0.067	2215.70	1884.02	0.029	1462.85	1841.70	0.039	2740.90	2778.60
Flat RL - 50	0.022	3618.06	1701.36	0.012	3301.91	1601.86	0.018	4712.36	2301.83
HORLP - 100	0.032	2242.44	1682.60	0.023	1429.61	1647.22	0.031	2394.85	2108.9
Flat RL - 100	0.019	3607.47	1618.55	0.010	2784.06	1325.96	0.015	3869.59	1779.53

Table 5.1 shows the measured average queueing delay, average packet drops in bits, and average congestion level (i.e. average packet buffer content in bits). The measurements are obtained from the QoS statistics from all nodes and averaged over the simulation period. The table supports our earlier results in Figures 5.6 and 5.7 as it also shows that Flat RL has fewer packets in the queue (i.e. less congestion) and slightly smaller queueing delay than HORLP. However, Flat RL incurs higher packet

drops. HORLP is advantageous since a task structure permits efficient searching of actions to avoid ineffective drop rates, thus improving the overall performance in terms of average long term reward.

5.4 Possible Weaknesses of HORLP algorithm and HRL

HORLP faces a similar issue with WFRLP and the FLP algorithm in Section 3.6 since it also uses WF^2Q for bandwidth provisioning for each agent. As mentioned in Section 4.4.2, this is only possible if the topology state evolves as an irreducible aperiodic (i.e. ergodic) Markov chain. We have already discussed the implications of this assumption and its applicability in actual networks in Section 3.6.

HORLP also faces the same issue as the WFRLP in Section 4.6 since each agent *independently* solves its own locally-observed HSMDP, without knowing the policies of other agents. Although this is easier to implement as each node does not need information about other nodes, accurate estimate of the optimal network reward may not always be obtained. We address this issue in Chapter 6 for the decentralized case.

HORLP also uses the Wire-Fitted CMAC in its task graph representation in Section 5.2, and thus suffers similar issue with WFRLP in Section 4.6 with regards to storage implementation of continuous state-action multidimensional space.

In addition, HORLP uses a task graph structure in decomposing the original SMDP problem, as discussed in Section 5.2. In our provisioning problem, we have only used a simple task graph in Figure 5.3. Finding the *best* task graph is the next issue. Aside from this, one requires problem-specific conditions to specify the *termination* conditions in the task graph. Further investigation is needed to address this problem. There are also some recent work on automating the search for building the task graph [57].

5.5 Chapter Summary

Simulation results show that HORLP is able to attain its objective of network provisioning to meet QoS requirements in a cost effective manner. By decomposing the network-level QoS provisioning problem into a simple task hierarchy under a stochastic control HSMDP framework, and using the corresponding model-free HRL algorithm, we are able to achieve better average long term performance, in terms of network reward and reduction in QoS constraint violations.

Chapter 6

Decentralized Optimal Control for Resource Allocation

This chapter presents the fourth variant of MDP formulation in this thesis. In the previous chapters, we have used a single-agent framework, where each agent sees its own controlled Markov chain independently, and finds its optimal policy. In this chapter, we formulate the queue scheduling problem as a *decentralized* control problem. Specifically, we use the framework known as *Decentralized Partially Observable Markov Decision Process* (DEC-POMDP) [19] where the performance of the network is affected by the joint actions or policies of the agents. DEC-POMDP is an extension of the theory of MDP for decentralized control where each agent observes a different partial view of the current network condition. The observation of an agent may only include the local queue information and policies of neighboring agents.

In finding the optimal joint actions, the DEC-POMDP formulation is essentially a multi-agent system that allows the agents to collaborate, cooperate and control a single MDP *without* complete observability of the global network state. This framework is thus more applicable and realistic in actual network deployment.

In Chapters 4 and 5, due to the known complexity of model-based DP techniques, such as value iteration and policy iteration, we have also introduced the model-free approach known as Neuro-Dynamic Programming or Reinforcement Learning. In this chapter, we also employ a model-free solution as it does not require the transi-

tion probability distribution of the underlying Markov chain in finding the optimal solution. RL effectively solves a single MDP formulation with independent agents, however, we extend standard RL algorithms to solve a multi-agent collaborative DEC-POMDP.

It is also known that exact solutions to a DEC-POMDP are complete for the complexity class non-deterministic exponential time (NEXP-complete) [19]. In other words, a general DEC-POMDP do not admit polynomial-time algorithms since P $\neq$ NEXP. Hence, using online RL-based algorithms to approximate the optimal solution is not only more applicable in a dynamic network, but also practical and less in complexity.

We also note that in prior research work, the performance analysis of communication networks is usually done using a separate mathematical framework. For instance, the work in [58] uses a general $G/G/1$ queue for MANETs to establish the probability distributions and performance bounds on congestion level and queueing delay *without* an explicit use of a control algorithm. In this thesis, we also derive performance bounds *directly* from the *controlled* Markov chain itself, as the model-free algorithm *converges* in finding the best scheduling policy. This approach is also more realistic since the nodes acting as agents can control the actual performance achieved.

We also use the ψ-irreducibility framework discussed in Chapters 2 and 3 for studying the stability and performance analysis of our DEC-POMDP scheduling problem. Note that ψ-irreducibility is more applicable than the standard definition of *irreducibility* where the latter refers to the case when there is only a single communicating class and any state of the Markov chain can be visited from any initial condition [4].

Consequently, our analysis uses a stability condition known as *V-uniform ergodicity* [22] for ψ-irreducible controlled Markov chains. This stability property provides an inequality condition that enables us to derive performance bounds from the control

algorithm. We emphasize that *V-uniform ergodicity* differs from the *Foster-Lyapunov drift condition* in Theorem 2.3 as it is more applicable in a model-free approach for a decentralized multi-agent framework.

We believe that the DEC-POMDP formulation captures a multi-agent system for communication networks more appropriately than any other decision-theoretic, game-theoretic or MDP-based framework. To the best of the authors' knowledge, this novel approach is the *first* method of achieving optimization cooperatively in a decentralized manner, and for deriving stability conditions and performance bounds *simultaneously* in a general wireless Markov queueing network *directly* from the control algorithm, as the algorithm converges to the optimal solution. Furthermore, our proposed solution does not require the complete knowledge of the topology, traffic and channel statistics.

This chapter is organized as follows. Section 6.1 emphasizes the importance of a decentralized control framework for collaborative scheduling and resource allocation, which differs from earlier research. We then introduce the DEC-POMDP framework and apply it for the multi-class queue scheduling problem in MANETs under the average cost criterion. The main objective is to find the optimal joint policy that minimizes the average congestion level of the network. We also explain how a time-varying channel medium and topology can be easily captured in our system model.

Section 6.2 then discusses non-trivial complexity issues of exact and optimal algorithms for DEC-POMDP. In order to find and *approximate* the optimal policies efficiently and in a decentralized manner, we propose a model-free control algorithm in Section 6.3. We introduce the concept of a *multi-agent finite state controller* (MFSC). Our approach is based on the RL technique known as *policy gradient* that parameterizes and updates the policies of agents during execution. We also exploit the idea of finding a policy structure to capture the *locality of interaction* among neighboring agents.

In Section 6.4, we use the ψ-irreducibility property for Markov chains from Chap-

ter 2 to study the performance and stability of our system model. Specifically, we derive performance bounds on average queueing delay and congestion level from the stability condition known as *V-uniform ergodicity* for ψ-irreducible controlled Markov chains.

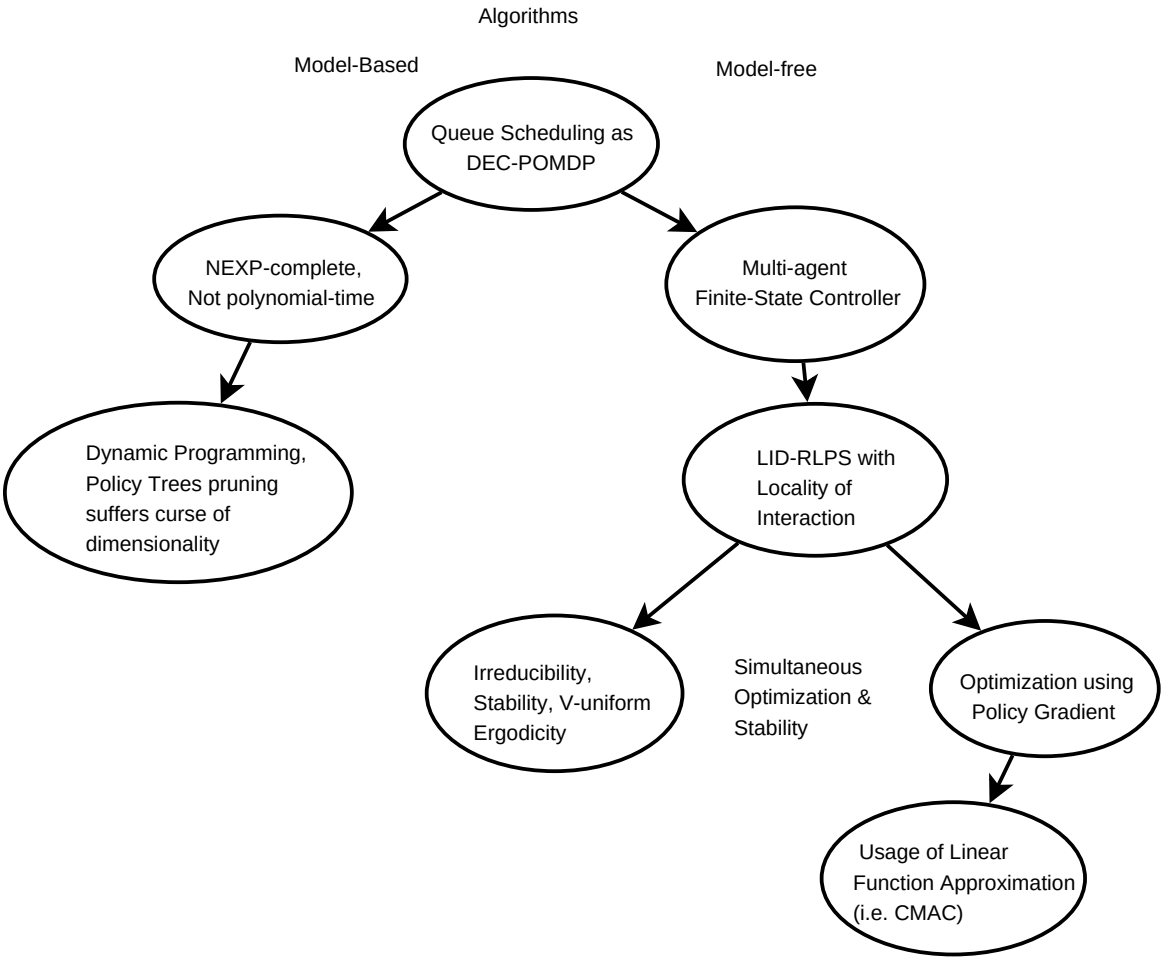


Figure 6.1: Summary of DEC-POMDP methodology

Figure 6.1 summarizes the key ideas of this chapter. A general queue scheduling problem under a time-varying channel and topology is first formulated as a DEC-POMDP. Due to the complexity issues of model-based algorithms [19], we use the *multi-agent finite state controller* and propose the *Locally Interacting Distributed Reinforcement Learning Policy Search* (LID-RLPS) in Algorithm 4 with neighborhood locality of interaction. We use the ψ-irreducibility property and *V-uniform ergodic-*

ity in studying the performance and stability of our model. By using a distributed policy gradient mechanism in LID-RLPS, we can achieve optimization and stability *simultaneously*. LID-RLPS also uses function approximation techniques (i.e. CMAC linear neural network) in storing neighborhood policy parameters.

6.1 Decentralized Control for Resource Allocation

6.1.1 Importance of Decentralized Control

Markov Decision Process have been widely used as a mathematical framework for sequential decision-making in stochastic domains. In particular, a single decision maker controls the system to optimize a global objective. The agent *completely* observes the state of the controlled Markov chain and acts based on its policy [4].

For resource allocation in communication networks, the authors in [18, 59] have used the MDP framework where each node, acting as an agent, treats relevant local information as the state of the MDP. For instance, in [59] and Chapter 3, the *local* queue length represented the state of the controlled Markov chain, while the immediate cost depended only on the locally-observed state. The goal was to find the best scheduling policy that minimizes the average cost or congestion level of the network. The solution was obtained using a model-free RL technique that approximates the optimal policy without the transition probability distribution of the MDP. While this approach is novel, the model in [59] and Chapter 3 is only a *single agent* framework, where each agent solves its own locally-observed MDP independently, *without* directly considering the policies of other agents. Since the agents do not exchange information about the local states and policies of other neighboring agents in the resource allocation problem, attaining the global optimum may not be possible.

The idea of using a collaborative framework among the agents is not new and has already been studied in game theory. For instance, the theory of *stochastic games*

provides the foundation for recent research work on multi-agent planning and learning [60]. A stochastic game can be considered as an extension of single-agent MDP where there are multiple agents with possible conflicting goals, and the joint actions of agents determine the global state transition and rewards. Much of the literature on stochastic games assume that each agent has complete information (i.e. complete observability) about the global state of the system. However, this assumption is obviously not satisfied especially in a large and dynamic network. For MANETs, such assumption is impractical since each agent requires the instantaneous local information from all other agents to act optimally.

In this chapter, we are interested in a single MDP that is collaboratively controlled by multiple agents. However, each agent does not have access to the global state of the network which evolves as a Markov chain and is affected by the joint actions of agents. Specifically, an agent only observes its local information, such as the queue lengths in its class queues, and possibly the local information and policies of neighboring agents. This type of model is known as a Decentralized Partially Observable Markov Decision Process (DEC-POMDP).

We highlight that the spatial extent of the neighboring locality is discussed in Section 6.3.2.

6.1.2 DEC-POMDP

A N-agent DEC-POMDP can be expressed as the tuple [19]:

$$\left\langle S, \overrightarrow{A}, P, C, \overrightarrow{\Omega}, O, p_o \right\rangle \tag{6.1}$$

where:

S is a finite set of states.

$\overrightarrow{A} = \{A_i\}$ is a finite set of joint actions, where A_i is the set of actions available to

agent i.

$P(S'|\overrightarrow{a}, S)$ denote the probability that the next state is S' given that the agents execute the joint action $\overrightarrow{a} = \{a_1, ..., a_N\}$ when the current state is S.

$\overrightarrow{\Omega} = \{\Omega_i\}$ is a finite set of joint observations, where Ω_i is the set of observations by agent i.

$O(\overrightarrow{o}|S, \overrightarrow{a}, S')$ is the probability of observing $\overrightarrow{o} = \{o_1, ..., o_N\}$ when the agents take actions $\overrightarrow{a}$ in state S, resulting to state S'.

$C(S, \overrightarrow{a}, S')$ denote the immediate cost function,

p_0 is the initial state distribution of the system.

In a DEC-POMDP, the joint action of the agents and the current state determine the next state. However, each agent only observes its own local observation o_i and none of the agents know the complete state of the system. Note that this formulation is similar to the centralized single-agent Partially Observable Markov Decision Process (POMDP) framework [61, 62] where the agent only has observations.

We define a local policy w_i for agent i to be a mapping from local history of observations $\overrightarrow{o_i^h}(t) = \{o_i(1), ..., o_i(t)\}$ to local actions $a_i(t)$. A joint policy $\overrightarrow{w} = \{w_1, ..., w_N\}$ is defined to be a tuple of local policies. Solving a DEC-POMDP can be seen as finding a set of N policies, one for each agent. Here, we only consider the average cost criterion. The average cost of a particular joint policy $\overrightarrow{w}$ for a given initial state $S(0) = s_0$ is defined as:

$$J(\overrightarrow{w}, s_0) := \lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n} E_{s_0}^{\overrightarrow{w}} \{C(S(t), \overrightarrow{a}(t), S(t+1))\} \tag{6.2}$$

A policy $\overrightarrow{w^*}$ is optimal if $J(\overrightarrow{w^*}, s_0) \leq J(\overrightarrow{w}, s_0)$ for all policies $\overrightarrow{w}$ and any initial state s_0.

6.1.3 Queue Scheduling as DEC-POMDP

Consider a wireless network with N nodes acting as agents and J network classes. Each node is assumed to act as an agent that actively performs scheduling on its own J local queues, where packets are enqueued in their respective class queues.

In formulating the queue scheduling problem as a DEC-POMDP, we study the queue length dynamics of the agents in a time-varying channel and topology. Specifically, we shall show that for each class, the queue length dynamics can be concatenated from all agents to form the state vector which evolves as a Markov chain. This idea is similar to the centralized model in [5], but we extend it to the decentralized case.

Following [5], we represent the network as a directed graph $G(V, E)$, where V is the set of nodes and E is the set of links. We assume slotted time and there are L maximum possible links, where $L := N(N-1)$ and N is the number of nodes. To represent the dynamic network topology, we define the $N \times L$ *topology* matrix R^j for class j as follows. We define the element of R^j in its i^{th} row and l^{th} column for $i = 1, ..., N$ and $l = 1, ..., L$:

$$r^j_{i,l} = \begin{cases} 1 & \text{if } h(l) = i \\ -1 & \text{if } q(l) = i \\ 0 & \text{otherwise} \end{cases} \tag{6.3}$$

where:

$h(l)$ represents the destination node of link l.

$q(l)$ represents the origin node of link l.

We let $\{N_i\}$ be the set of *neighboring* nodes of node i, which is defined as the set of nodes that share an active link with node i (i.e. $r^j_{i,l} \neq 0$). The set of neighbors determines the *locality of interaction* among nodes or agents. We shall elaborate on this concept in Section 6.3.2.

Let $M_l^j(t)$ be a binary variable such that $M_l^j(t) = 1$ if a packet of class j from $q(l)$ is successfully transmitted to $h(l)$ during slot t; otherwise $M_l^j(t) = 0$ as the packet remains at node $q(l)$. Let $x_i^j(t)$ denote the number of bits in node i at its j^{th} class queue during time slot t. The queue dynamics can be generally expressed for $\forall j$:

$$\overrightarrow{x^j}(t+1) = \overrightarrow{x^j}(t) + R^j(t)M^j(t)\overrightarrow{\mu^j}(t) + \overrightarrow{D^j}(t) \tag{6.4}$$

where:

$\overrightarrow{x^j}(t) = [x_1^j(t), ..., x_N^j(t)]^T$ is a vector of the queue lengths in bits by the end of time slot t.

$R^j(t)$ is the topology matrix with elements $r_{i,l}^j(t)$ defined in (6.3).

$M^j(t)$ is a $L \times L$ diagonal matrix where the l^{th} diagonal element is $M_l^j(t)$.

$\overrightarrow{\mu^j}(t) = [\mu_1^j(t), ..., \mu_L^j(t)]^T$, where $\mu_l^j(t)$ denote the share of bandwidth (in bits) that was allocated for link l by the agent or node $q(l)$ for class j.

$\overrightarrow{D^j}(t) = [d_1^j(t), ..., d_N^j(t)]^T$, where $d_i^j(t)$ denote the number of bits arriving at the j^{th} class queue, as generated by the source application at node i, if there's any.

We assume that $\{d_i^j(t)\}_{t=1}^{\infty}$, $\{r_{i,l}^j(t)\}_{t=1}^{\infty}$, and $\{M_l^j(t)\}_{t=1}^{\infty}$ are i.i.d. sequences of random variables for all $i = 1, ...N$, $l = 1, ..., L$, and $j = 1, ..., J$. In addition, we assume that these processes are independent among themselves and the second moments of the arrival processes are finite.

Due to the broadcast nature of the wireless medium and by assuming that each node has only one network interface device, performing rate allocation on each outgoing link l from node $q(l)$ is effectively the same as each agent i managing its own *single* outgoing link (i.e. single network interface). For notational convenience, we write$\mu_i^j(t)$ as the rate allocated by agent i for each class j for its single network interface.

In contrast with the model in [5] where there exists a centralized entity that has complete knowledge of the vector queue length $\overrightarrow{x^j}(t) = [x_1^j(t), ..., x_N^j(t)]^T$, in our case,

no agent has complete knowledge of $\overrightarrow{x^j}(t)$. Agent i must decide based on its own locally observed history of information.

Under the statistical assumptions above and for any joint policy by the agents, the queue length process $\{\overrightarrow{x^j}(t)\}_{t=1}^{\infty}$ is a controlled Markov chain for each class j and is independent among the classes. Generally speaking, each agent is acting similar to a single POMDP agent, where the latter uses its history of local observations and actions to find its optimal policy [61, 63].

For the DEC-POMDP formulation, we define the state vector as:

$$S(t) := \overrightarrow{x}(t) = [\overrightarrow{x^1}(t), .., \overrightarrow{x^J}(t)] \tag{6.5}$$

where:

$\overrightarrow{x}(t)$ is a *factored* representation or concatenation of the queue length vectors $\overrightarrow{x^j}(t) = [x_1^j(t), ..., x_N^j(t)]^T$ from all network classes.

Thus, $\overrightarrow{x}(t)$ also evolves as a controlled Markov chain. The set of joint actions of N agents is represented by the rate allocation vector $\overrightarrow{\mu}(t) = [\overrightarrow{\mu^1}(t), ..., \overrightarrow{\mu^J}(t)]^T$, where $\overrightarrow{\mu^j}(t) = [\mu_1^j(t), ..., \mu_N^j(t)]^T$. The rate allocation action vector can also be rewritten as $\overrightarrow{\mu}(t) = [\overrightarrow{\mu_1}(t), ..., \overrightarrow{\mu_N}(t)]^T$, where $\overrightarrow{\mu_i}(t) := \overrightarrow{a_i}(t) = [\mu_i^1(t), ..., \mu_i^J(t)]^T$ is the action vector for agent i. Each agent i only observes the queue length vector $\overrightarrow{o_i}(t) := \overrightarrow{x_i}(t) = [x_i^1(t), ..., x_i^J(t)]$ from its local class queues. The cost function for each class j is defined as the congestion level from all agents:

$$C(x^j(t), \mu^j(t)) = \sum_{i=1}^{N} x_i^j(t) \tag{6.6}$$

The actual immediate cost is $C_{all}(\overrightarrow{x}(t), \overrightarrow{\mu}(t)) = \sum_{j=1}^{J}\sum_{i=1}^{N} x_i^j(t)$. The average net-

work congestion level starting from an initial state $\overrightarrow{x}(0)$ is defined as:

$$J(\overrightarrow{w}, \overrightarrow{x}(0)) := \lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n} E^{\overrightarrow{w}}_{\overrightarrow{x}(0)} \{C_{all}(\overrightarrow{x}(t), \overrightarrow{\mu}(t))\} \tag{6.7}$$

The main objective is to find the optimal joint policy $\overrightarrow{w} = \{w_1, ..., w_N\}$ so that $J(\overrightarrow{w}, \overrightarrow{x}(0))$ is minimized starting from any initial state $\overrightarrow{x}(0)$. Figure 6.2 summarizes the ideas of queue scheduling as a DEC-POMDP problem.

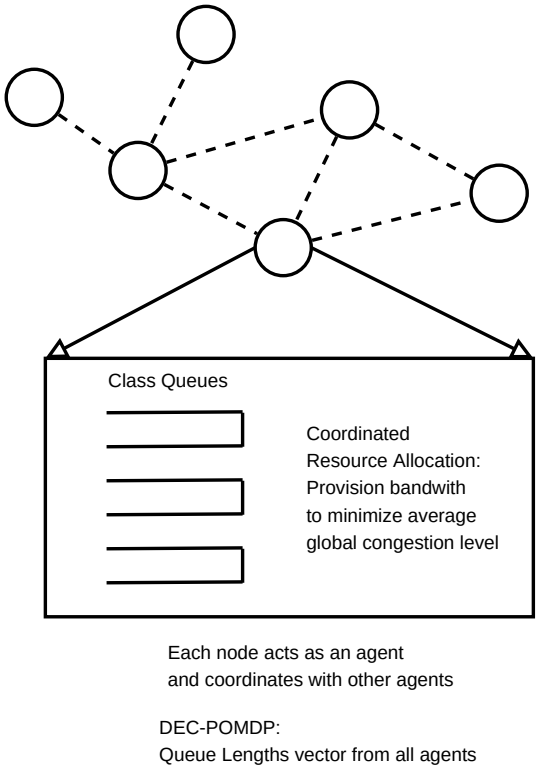


Figure 6.2: Queue Scheduling as multi-agent DEC-POMDP

6.1.4 Time-Varying Channel and Topology

In modeling the time-varying wireless medium, researchers have used the concept of the *channel state* process. The channel state includes characteristics of the network that affect transmission. It can be obtained either through direct measurement or through a combination of measurement and channel prediction. The authors in [12,

6, 26, 27] have assumed that the channel state, denoted as $\{C_h(t)\}_{t=1}^{\infty}$, evolves as an irreducible aperiodic finite state-space Markov chain.

In this subsection, we claim that this important concept can be easily captured in the general queue dynamics in (6.4). Specifically, the channel state process is captured in the diagonal element $M_l^j(t)$ of the diagonal matrix $M^j(t)$, since $M_l^j(t)$ is from $\{0, 1\}$ that effectively represents a success or failed transmission from node $q(l)$ to $h(l)$.

In addition, we also claim that other important MANET related characteristics, such as varying topology, routing protocols and MAC mechanisms, can be easily included in the queueing model in (6.4). We know that the success of transmission depends on other nodes' attempts as well as the *topology state* of the network. The topology state includes all the characteristics of the network that affect transmission and may vary with time. It may include the changing connectivity among nodes as they move, and transmission rates in each link with changing quality. Other characteristics that may not be directly related to transmission can be also included in the topology state. Effectively, the topology state also captures the channel state process above. By considering slotted time, the authors in [27, 59] have assumed that the topology state forms a stochastic process that evolves as an irreducible aperiodic finite state-space Markov chain.

In the queueing model above, the topology state process can be expressed in the topology matrix $R^j(t)$ and in conjunction with the earlier assumption that the elements of $R^j(t) = \{-1, 0, 1\}$ are i.i.d random variables.

Hence, from these concepts, we conclude that the time-varying channel and topology state processes are captured in our general queueing model. We observe that, once a Markov chain is identified for the entire network, the DEC-POMDP formulation seems straightforward. Although the DEC-POMDP formulation is novel and handles a decentralized multi-agent system for communication networks more appro-

priately than any other MDP-based framework, we highlight some non-trivial issues for exact optimal DEC-POMDP algorithms in the next section.

6.2 Complexity Issues of DEC-POMDP

As mentioned in the Section 6.1.3, DEC-POMDP appears to be a POMDP, where each agent only has local observation and the joint actions of the agents determine the next state transition, without observing the actual state of the system. In this sense, one can intuitively convert it to a POMDP and use established techniques [64, 65].

However, the authors in [19] have shown that, on the contrary, a DEC-POMDP requires a fundamentally different algorithmic structure. By reducing the control problem to a *tiling* problem, they have shown that if the underlying transition probability function is known, the DEC-POMDP in a finite-horizon with a constant number of agents (i.e. $N \geq 2$) is *complete* for the complexity class non-deterministic exponential time (NEXP). This implies that problems modeled as DEC-POMDP *provably* do not admit polynomial-time algorithms. This trait is not shared by finite-horizon MDP or POMDP problems and thus, has direct implications when solving problems involving distributed agents. It should be noted that, even if one can convert a DEC-POMDP into a single-agent POMDP, one for each agent, exact POMDP methods are PSPACE-hard [63, 64] and so approximate tractable solutions are preferable.

Even for infinite-horizon POMDPs, it has been shown that exact algorithms based on Dynamic Programming suffer from an infinite number of belief states. This condition implies that the problem of determining convergence is undecidable [63, 66]. Since a POMDP is a special case of a DEC-POMDP (i.e. $N = 1$), the corresponding DEC-POMDP problems are also undecidable [19].

Recently, an exact Dynamic Programming algorithm was proposed for a general DEC-POMDP [67]. Though the algorithm was used in a finite-horizon context, the

authors mentioned ways to extend it to the infinite-horizon case. Their model-based algorithm uses *policy trees* that enumerate the possible policies at each state and every possible next state transitions up to a given depth in the tree, and performs pruning of tree branches. The algorithm obviously suffers from large memory requirements with each iteration as the tree grows and in practice, has only been used to solve very small problems. It is likely that any exact optimal algorithm would suffer this curse of dimensionality and exponential-time complexity, due to the *NEXP-complete* result in [19].

6.3 Model-Free Algorithm for Queue Scheduling as a DEC-POMDP

6.3.1 Finite-State Controller Model

Due to the complexity issues of exact model-based algorithms for DEC-POMDP as discussed in the previous section, we use model-free techniques to approximate the optimal joint policy of agents.

When an agent does not completely observe the state of the system and that the underlying transition probability distribution is unknown, the agent needs *memory* of the past observations and actions to act optimally, as in the case of single-agent POMDP [63].

Following [63] for POMDP, we use the concept of a *finite-state controller* (FSC) to capture relevant past information to act optimally. Each agent i contains a finite-state controller which is represented as the tuple:

$$\langle I_i, \phi_i, f_i(\cdot), \theta_i, \mu_i(\cdot) \rangle \tag{6.8}$$

where:

I_i is the set of *internal states* (I-states) of the FSC.

$f_i(\cdot)$ and $\mu_i(\cdot)$ are I-state transition and action selection distribution functions, respectively.

$\phi_i \in \mathbb{R}^{n_{\phi_i}}$ and $\theta_i \in \mathbb{R}^{n_{\theta_i}}$ are vector parameters.

FSC uses $\phi_i \in \mathbb{R}^{n_{\phi_i}}$ as a n_{ϕ_i}-dimensional vector to parameterise the I-state transition probabilities based on the current I-state and local observation. In other words, the next I-state is chosen stochastically from the distribution $f_i(\cdot \mid \phi_i, g, y)$, where $g \in I_i$ and y is the current local observation. Similarly, $\theta_i \in \mathbb{R}^{n_{\theta_i}}$ is a n_{θ_i}-dimensional vector to parameterise the action probabilities $\mu_i(\cdot \mid \theta_i, h, y) > 0$ for each I-state h and local observation y.

Each agent learns to use the I-states to remember only what is needed to act optimally. Specifically, the I-state transitions and policies are *learned* by searching the space of parameters ϕ_i and θ_i. The action selection or mapping based on $\mu_i(\cdot \mid \theta_i, h, y)$ is also known as a *randomized policy.*

Algorithm 2 summarizes the FSC for a single-agent framework. In Step 5, the gradient mechanism used mostly depends on the I-state and policy distributions $f_i(\cdot \mid \phi_i, g, y)$ and $\mu_i(\cdot \mid \theta_i, h, y)$, respectively. We shall elaborate this policy gradient mechanism specific to our proposed solution in Section 6.3.3.

In single-agent POMDP literature, the FSC is used to capture the uncertainty of the system using the concept of *belief states*. In a single-agent case, a belief state is defined as: $B_{single}(t) = Pr(S(t) \mid \overrightarrow{o^h}(t))$ where $S(t)$ is the unobserved state and $\overrightarrow{o^h}(t)$ represents a vector of the history of local observations up to time t. It is known that $B_{single}(t)$ is a sufficient statistic because the agent can compute an optimal policy based on $B_{single}(t)$ without having to consider the actual observation history sequence $\overrightarrow{o^h}(t)$ [68].

Algorithm 2 Finite-State Controller (FSC) for Partially-Observable Environment (i.e. Single-Agent POMDP)

Let S_t= unobserved state of the environment at time t
g_t= I-state of the agent i.
$\langle \phi_i, \theta_i \rangle$ = FSC parameters

1. Agent i observes y_t which depends on S_t.
2. Agent i then chooses its next I-state g_{t+1} from the distribution $f_i(\cdot \mid \phi_i, g_t, y_t)$
3. Agent i then chooses its action a_t from $\mu_i(\cdot \mid \theta_i, g_{t+1}, y_t)$
4. The environment transits to the next state S_{t+1} and the immediate cost C_{t+1} is obtained by the agent i
5. The agent updates its FSC parameters $\langle \phi_i, \theta_i \rangle$ using a gradient estimate mechanism.
6. $t = t + 1$. Go to step 1.

As the agent uses the I-states in the FSC above, this process can be viewed as an automatic *quantization* of the belief state space to provide the optimal policy representable by $\|I_i\|$ internal states. As $\|I_i\| \to \infty$, we can represent the optimal policy accurately, *without* knowing the exact model of the system [69].

In a multi-agent DEC-POMDP case, an agent faces a complex but normal single-agent POMDP if the policies of all other agents are *fixed* at a given decision instant. However, $B_{single}(t)$ is not sufficient since the agent must also reason about the action selection and observation histories of other agents.

Following [70], at each time t, agent i decides based on the tuple:

$$\overrightarrow{e_i}(t) = \left\langle S(t), \overrightarrow{o^h_{\neq i}}(t) \right\rangle$$

where $\overrightarrow{o^h_{\neq i}}(t) = \left\langle \overrightarrow{o^h_1}(t), ..., \overrightarrow{o^h_{i-1}}(t), \overrightarrow{o^h_{i+1}}(t), ..., \overrightarrow{o^h_N}(t) \right\rangle$ is the joint observation histories of all agents except i. By treating the vector $\overrightarrow{e_i}(t)$ as the state of the agent i at time t, we can define the transition function and observation for the single-agent POMDP for agent i as follows:

$$P^{'}(\overrightarrow{e_i}(t+1)|a_i(t),\overrightarrow{e_i}(t)) = P(S(t+1)|S(t),\overrightarrow{a}(t)) \cdot O_{\neq i}(\overrightarrow{o_{\neq i}}(t+1)|\overrightarrow{a}(t),S(t+1)) \tag{6.9}$$

$$O^{'}(o_i(t+1)|a_i(t),\overrightarrow{e_i}(t+1)) = O_i(o_i(t+1)|\overrightarrow{a}(t),S(t+1)) \tag{6.10}$$

where:

$P(S(t+1)|S(t),\overrightarrow{a}(t))$ is the state transition distribution function of the original DEC-POMDP.

$O_{\neq i}(\overrightarrow{o_{\neq i}}(t+1)|\overrightarrow{a}(t),S(t+1))$ is the probability that all other agents except i observes vector $\overrightarrow{o_{\neq i}}(t+1)$ (i.e. not part of history $\overrightarrow{o^h_{\neq i}}(t)$) given previous joint action $\overrightarrow{a}(t)$, resulting to $S(t+1)$.

$O_i(o_i(t+1)|\overrightarrow{a}(t),S(t+1))$ is the probability that agent i observes $o_i(t+1)$ given previous action $\overrightarrow{a}(t)$, resulting $S(t+1)$.

The multi-agent belief state for an agent i given the distribution over the initial state $p_0(s) = P(S(0) = s)$ is defined as [70]:

$$B_{i,mul}(t) = Pr(\overrightarrow{e_i}(t)|\overrightarrow{o^h_i}(t),\overrightarrow{a^h_i}(t-1),p_0(s)) \tag{6.11}$$

where:

$\overrightarrow{a^h_i}(t-1)$ is the history vector of actions up to time $(t-1)$, while $\overrightarrow{o^h_i}(t)$ is the observation history for agent i up to time t.

In other words, when reasoning about the agent's policy in the *context* of other agents (i.e. other agents' policies are fixed at the *current context*), we maintain a distribution over $\overrightarrow{e_i}(t)$, rather than simply the current state $S(t)$ as in $B_{single}(t)$.

We thus extend of the FSC in (6.8) to capture the multi-agent belief state $B_i(t)$.

Specifically, we use the same tuple $\langle I_i, \phi_i, f_i(\cdot), \theta_i, \mu_i(\cdot)\rangle$, with the parameters ϕ_i and θ_i, and set of internal states I_i. We redefine the internal state transition distribution as $f_i(\cdot \mid \phi_i, g, y, a_i(t-1))$, where $g \in I_i$ and y is the current local observation, and $a_i(t-1)$ is the previous local action. Similar to (6.8), the action selection policy is based on the action probabilities $\mu_i(\cdot \mid \theta_i, h, y) > 0$ for each I-state h and local observation y. The I-state transitions and policies are learned by searching the space of parameters ϕ_i and θ_i.

This search process effectively performs an automatic quantization of the multi-agent belief state $B_{i,mul}(t)$ in (6.11), similar to the single-agent POMDP case earlier. We refer to this concept as the *multi-agent finite state controller* (MFSC).

In summary, the agent searches the space of parameter vectors ϕ_i and θ_i to act optimally, with consideration of other agents' observation history statistics, *without* knowing the state transition and observation probabilities of the DEC-POMDP in (6.1).

6.3.2 Locality of Interaction among Neighboring Agents

While decentralized or distributed POMDP captures real-world uncertainty in multi-agent domains, such as time-varying topology and channel in MANETs, it fails to exploit the fact that each agent has limited interactions with a small number of neighboring agents. In other words, each agent only affects the local observation and policies of those agents close to it. A general DEC-POMDP does not exploit the *locality of interaction* structure of a communication network.

In [71], the authors introduced this important concept for distributed POMDPs for finite horizon problems. In this section, we apply this idea under the infinite-horizon criterion for our DEC-POMDP scheduling problem.

We have earlier shown in Section 6.1.3 that the queue length process $\{\overrightarrow{x^j}(t)\}_{t=1}^{\infty}$ is already a controlled Markov chain for each class j. For notational convenience, we

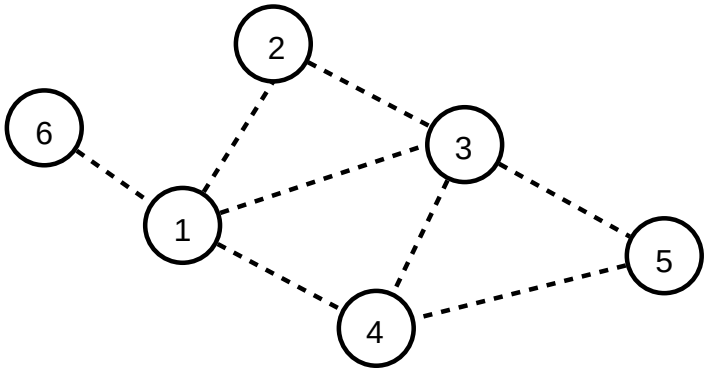


Figure 6.3: Locality of interaction among neighboring nodes

drop the class index j and we perform our analysis for a single class queue only.

Due to the interaction among neighbors, we observe that the immediate cost function $C(x^j, \mu^j)$ in (6.6) for class j can be expressed as a summation of the corresponding cost functions of a sub-group of agents. For example, consider the network scenario with six nodes in Figure 6.3. We know that in a single time slot t, the queue length of a node is only affected by neighboring nodes. The immediate cost or congestion function for N nodes can be simply expressed as:

$$C(\overrightarrow{x}, \overrightarrow{a}) = \sum_{i=1}^{N} C(x_i, x_{N_i}, a_i, a_{N_i}) \tag{6.12}$$

where:

x_{N_i} is a vector representing the local observations or queue lengths of the neighboring agents (i.e. $x_{N_1} = [x_2, x_3, x_4, x_6]^T$ in Figure 6.3).

a_i is the chosen action of agent i.

a_{N_i} is a vector for the actions of the neighboring agents (i.e. $a_{N_1} = [a_2, a_3, a_4, a_6]^T$ in Figure 6.3).

$C(x_i, x_{N_i}, a_i, a_{N_i})$ is the immediate cost incurred by agent i which depends on its current local observation x_i and action a_i, and the relevant information x_{N_i} and a_{N_i} from neighbors.

Let $C_{loc,i}(t) := \sum_k C(x_k, x_{N_k}, a_k, a_{N_k})$, $\forall k \in \{i \cup N_i\}$ is the immediate localized cost. This effectively represents the sum of the individual cost functions in (6.12) where x_i and a_i components are included at the current time slot t. We use this quantity to separate the immediate cost function into two independent components: one is affected by agent i, while the other is independent of i. For instance, in Figure 6.3, the immediate cost function in (6.12) can be expressed in the following ways, depending on the index i in $C_{loc,i}$:

$$
\begin{aligned}
&C(\overrightarrow{x}, \overrightarrow{a}) = C_{loc,1} + C(x_3, x_4, x_5, a_3, a_4, a_5) \\
&= C_{loc,2} + C(x_1, x_3, x_4, x_5, a_1, a_3, a_4, a_5) + \\
&C(x_3, x_4, x_5, a_3, a_4, a_5) + C(x_1, x_6, a_1, a_6) \\
&= C_{loc,3} + C(x_1, x_6, a_1, a_6) \\
&= C_{loc,4} + C(x_1, x_2, x_3, a_1, a_2, a_3) + C(x_1, x_6, a_1, a_6)
\end{aligned}
$$

We define the *local neighborhood utility* of agent i as $B_{\overrightarrow{w}}(N_i, s)$ to represent the expected average long term cost for executing joint policy $\overrightarrow{w} = \{w_1, ..., w_N\}$ due to the links containing agent i, starting with state s_0:

$$
B_{\overrightarrow{w}}(N_i, s_0) := \lim_{n \to \infty} \frac{1}{n} \sum_{t=0}^{n} E_{s_0}^{\overrightarrow{w}} \{C_{loc,i}(t)\} \tag{6.13}
$$

Lemma 6.1: *To find the best policy for agent i given its neighbors' policies in optimizing its local neighborhood utility, agent i does not need to consider the non-neighbors' policies.*

PROOF: We observe from (6.13) that the local neighborhood utilities of agent i for two joint policies $\overrightarrow{w_a} = [w_{a,1}, ..., w_{a,N}]$ and $\overrightarrow{w_b} = [w_{b,1}, ..., w_{b,N}]$ are equal, if the corresponding policy vector components are equal: $B_{\overrightarrow{w_a}}(N_i, s_0) = B_{\overrightarrow{w_b}}(N_i, s_0)$ if $w_{a,k} = w_{b,k}$ for all $k \in \{i \cup N_i\}$. Thus, any policy vector $\overrightarrow{w_b}$ that has different policies for only

non-neighborhood agents as compared to policy $\overrightarrow{w_a}$ has equal value as $B_{\overrightarrow{w_a}}(N_i, s)$. Furthermore, given the neighbors' policies, optimizing the local neighborhood utility of agent i does not affect the local neighborhood utility of agent k if $k \notin \{N_i\}$. ■

Lemma 6.1 is known as the property of *locality of interaction* which we shall use in the algorithm in the next subsection. We also emphasize that the local neighborhood utility for agent i also captures varying network topology especially in MANETs, since $B_{\overrightarrow{w_a}}(N_i, s)$ is independent of time and is the *time average* of the immediate localized cost $C_{loc,i}(t)$ incurred from the topology at every time slot t.

6.3.3 Model-Free Policy Generation algorithm

The idea of exploiting locality of interaction in distributed agents to optimize a global objective function has already been addressed in the formalism known as *Distributed Constraint Optimization* (DCOP) [20, 21].

A DCOP problem includes a set of variables, each variable is assigned to an agent who can control its value, and agents must coordinate their choice of values. DCOPs have successfully exploited limited agent interactions in multi-agent systems, with over a decade of algorithm development. However, DCOPs do not capture planning under uncertainty as compared to DEC-POMDP.

In an attempt to synthesize DCOPs and DEC-POMDP in order to handle locality of interaction and decentralized stochastic planning, the authors in [71] proposed a novel model known as *Networked Distributed-POMDP* (ND-POMDP). They have proposed a novel algorithm called *Locally Interacting Distributed Joint Equilibrium Search for Policies* (LID-JESP), which combines the ideas of Dynamic Programming in policy search, and the *Distributed Breakout Algorithm* (DBA) for DCOPs [20].

ND-POMDP can be thought of as an N-ary DCOP where N is the number of agents and the DCOP variable at each node is the individual agent's policy. The

model-based LID-JESP algorithm in [71] can be summarized as follows: Each agent i starts with a random local policy and exchanges its policies with its neighbors. It then computes its local neighborhood utility (see (6.13)) with respect to its *current* policy and its neighbors' policies (i.e. current context). Agent i then uses a *value-based* Dynamic Programming technique to get the local neighborhood utility of agent i's *best* policy given the policies of its neighbors. The difference between the two local neighborhood utilities is represented as the *gain* message. Each agent broadcasts its gain message among its neighbors for the current context. Agent i is allowed to act if its gain message is larger than all the gain messages it receives from all its neighbors. Essentially, agent i changes its policy to the computed best local policy if it is the winner at the current context or *cycle* of the algorithm. This process is then repeated.

The idea of exchanging policies and gain messages in the LID-JESP algorithm to improve agent i's policy with respect to its neighbors' policies in a distributed manner is based on the DBA for DCOPs [20]. However, LID-JESP includes planning under uncertainty, where the value of the local neighborhood utility depends on the *expected* long term value, whereas DBA does not handle uncertainty in the variables of the DCOP.

In this sub-section, we extend the idea of ND-POMDP to our decentralized queueing problem. Specifically, we extend the *model-based* LID-JESP algorithm to use the MFSC learning framework in Section 6.3.1, and apply *model-free* techniques to search for the policy parameters in representing and *estimating* the local neighborhood utility in (6.13).

As mentioned earlier in Section 6.2, a model-based Dynamic Programming algorithm suffers from the curse of dimensionality especially for DEC-POMDP, even with small number of states, observations, and actions. Hence, we propose that the policy generation of LID-JESP should be represented using the MSFC and without using the state and observation transition probabilities.

For our DEC-POMDP formulation on queue scheduling in Section 6.1.3, the state vector $S(t) := \overrightarrow{x}(t) = [\overrightarrow{x^1}(t), .., \overrightarrow{x^J}(t)]$ in (6.5) is a *factored* representation or concatenation of the local observations $\overrightarrow{x^j}(t) = [x_1^j(t), ..., x_N^j(t)]^T$ of agents for each class j. Following Section 6.3.2 and for notational convenience, our analysis here is based on a single class only, since the queue length processes among the classes are independent.

As shown in Lemma 6.1, given the neighbors' policies, agent i does not need to consider non-neighboring agents to find its best policy. The local neighborhood utility $B_{\overrightarrow{w}}(N_i, s_0)$ has effectively localized the effect of agent i's policy to the global cost function in (6.12). In other words, agent i has to only optimize its $B_{\overrightarrow{w}}(N_i, s_0)$, given the policies of the neighboring agents. The exchange of gain messages determines who among the agents can update its local policy at every cycle of the algorithm.

From this concept, we reformulate the MFSC for agent i to capture the locality of interaction. Following [71], instead of considering all other agents except i in the tuple $\overrightarrow{e_i}(t)$ in (6.9) and (6.10), we only consider the neighboring agents. In order to exploit the locality of interaction, we redefine the tuple $\overrightarrow{e_i}(t)$ as:

$$\overrightarrow{e_i}(t) = \left\langle S_{i,N_i}(t), \overrightarrow{o^h_{N_i}}(t) \right\rangle \tag{6.14}$$

where:

$S_{i,N_i}(t)$ is a vector containing the local observations of neighboring agents and agent i at the current time slot t.

$\overrightarrow{o^h_{N_i}}(t)$ represents the joint observation histories of neighbors up to time t.

Given the policy of neighboring agents, treating $\overrightarrow{e_i}(t)$ as state of agent i results in a single-agent POMDP with transition functions:

$$P^{'}(\overrightarrow{e_i}(t+1)|a_i(t),\overrightarrow{e_i}(t)) =$$
$$P(S_{i,N_i}(t+1)|S_{i,N_i}(t),\overrightarrow{a_{i,N_i}}(t))$$
$$\cdot O_{N_i}(\overrightarrow{o_{N_i}}(t+1)|\overrightarrow{a_{i,N_i}}(t),S_{i,N_i}(t+1)) \tag{6.15}$$

$$O^{'}(o_i(t+1)|a_i(t),\overrightarrow{e_i}(t)) =$$
$$O_i(o_i(t+1)|\overrightarrow{a_{i,N_i}}(t),S_{i,N_i}(t+1)) \tag{6.16}$$

where:

$O_{N_i}(\overrightarrow{o_{N_i}}(t+1)|\overrightarrow{a_{i,N_i}}(t),S_{i,N_i}(t+1))$ is the probability that neighboring agents observe $\overrightarrow{o_{N_i}}(t+1)$ (i.e. not part of history $\overrightarrow{o^h_{N_i}}(t)$) given previous action vector $\overrightarrow{a_{i,N_i}}(t)$, resulting to $S_{i,N_i}(t+1)$.

$O_i(o_i(t+1)|\overrightarrow{a_{i,N_i}}(t),S_{i,N_i}(t+1))$ is the probability that agent i observes $o_i(t+1)$ given previous action vector $\overrightarrow{a_{i,N_i}}(t)$, resulting to $S_{i,N_i}(t+1)$.

The multi-agent belief state in (6.11) is defined similarly with respect only to the neighboring agents: $B_{i,mul}(t) = Pr(\overrightarrow{e_i}(t)|\overrightarrow{o^h_i}(t),\overrightarrow{a^h_i}(t-1),p_0(s))$. We emphasize that $\overrightarrow{e_i}(t)$ is a single-agent POMDP only for a given set of policies of *neighbors* at the *current context* where their policies are fixed. In other words, to capture the multi-agent belief state under the MFSC with unknown transition probabilities, the MFSC must also depend on the policy parameters of neighboring agents, and not just ϕ_i and θ_i for agent i.

We thus redefine the internal state transition distribution in Section 6.3.1 as $f_i(\cdot \mid \phi_i, g, y, a_i(t-1), \delta_{N_i})$, where $g \in I_i$ and y is the current local observation, $a_i(t-1)$ is the previous local action, and $\delta_{N_i} \in \mathbb{R}^{n_{\delta_i}}$ is a n_{δ_i}-dimensional vector that captures the

Algorithm 3 Multi-agent Finite-State Controller (MFSC) with Locality of Interaction for DEC-POMDP

Let S_t= *unobserved* state of the environment at time t
g_t= I-state of the agent i.
$\langle \phi_i, \theta_i \rangle$ = FSC parameters

1. Agent i observes y_t which depends on S_t.
2. Agent i then chooses its next I-state g_{t+1} from the distribution $f_i(\cdot \mid \phi_i, g_t, y_t, a_i(t-1), \delta_{N_i})$, where $a_i(t-1)$ is its previous action, δ_{N_i} represents a current *feature* vector from the FSC parameters of its neighbors N_i
3. Agent i then chooses its action a_t from $\mu_i(\cdot \mid \theta_i, g_{t+1}, y_t, \delta_{N_i})$
4. The immediate localized cost $C_{loc,i}(t)$ is obtained by the agent i
5. The agent updates its FSC parameters $\langle \phi_i, \theta_i \rangle$ using a gradient estimate mechanism. (see Algorithm 4 for the complete description)
6. $t = t + 1$. Go to step 1.

effect of the neighbors' policy parameters $\langle \theta_k, \phi_k \rangle$ for all $k \in N_i$. Similarly, the action selection policy is based on the action probabilities $\mu_i(\cdot \mid \theta_i, h, y, \delta_{N_i}) > 0$ for each I-state h, local observation y, and parameters θ_i and δ_{N_i}. It should be noted that δ_{N_i} is fixed in a given cycle of the algorithm since the neighbors' policies are fixed during a cycle. Algorithm 3 gives a short summary of the MFSC with locality of interaction. The MFSC extends the ideas of Algorithm 2 with the addition of the *feature* vector δ_{N_i} for agent i from its neighbors.

It is known from [62] that if $\overrightarrow{e_i}(t)$ is the state of the single-agent POMDP and $g(t) \in I_i$ is an internal state of the MSFC, then the tuple $\langle \overrightarrow{e_i}(t), g(t) \rangle$ forms a Markov chain. Let $P(\phi_i, \theta_i)$ be the probability matrix of the Markov chain. Interaction begins at an initial state $\overrightarrow{e_i}(0)$ and agent i completely observes its own initial I-state $g(0) \in I_i$.

Given the neighbors' policies $\overrightarrow{w_{N_i}}$, the goal of agent i is to find ϕ_i and θ_i that

minimizes the local neighborhood utility, independent of the initial state:

$$B(N_i, \phi_i, \theta_i | \overrightarrow{w_{N_i}}) := \lim_{n \to \infty} \frac{1}{n} \sum_{t=0}^{n} E_{\phi_i, \theta_i} \{ C_{loc,i}(t) | \overrightarrow{w_{N_i}} \} \tag{6.17}$$

where the expectation E_{ϕ_i, θ_i} denote the expectation over all trajectories:

$$\langle \overrightarrow{e_i}(0), g(0) \rangle, \langle \overrightarrow{e_i}(1), g(1) \rangle, ...$$

with transitions generated using $P(\phi_i, \theta_i)$. We emphasize that the model-free searching of policy parameters $\langle \theta_i, \phi_i \rangle$ in the MFSC performs an automatic quantization of the belief state $B_i(t)$ for a given set of internal states, as explained in Section 6.3.1.

We call our proposed model-free algorithm as: *Locally Interacting Distributed Reinforcement Learning Policy Search* (LID-RLPS) that uses the MFSC to generate the policies while considering locality of interaction, and without the state and observation transition model of DEC-POMDP. The pseudo-code of LID-RLPS is shown in Algorithm 4. Note that the policy w_i of agent i is represented by the policy vector $\langle \theta_i, \phi_i \rangle$. Hence, during exchange of policies in LID-RLPS, the actual policy vector is communicated among the neighbors.

It is assumed that there is no error in the communication of the policy vector $\langle \theta_i, \phi_i \rangle$ among neighbors. The case when there is error can be translated into the problem of minimizing the error of function approximation in representing the policy parameters. This issue is discussed in the next subsections.

6.3.4 Algorithm Implementation Issues

The function $LocalBestPolicy(w_i, \delta_{N_i}, y_t, C_{loc,i}(t))$ in Algorithm 4 returns the best response policy: $w_i^* = \arg\min_{w_i} B(N_i, \phi_i, \theta_i | \overrightarrow{w_{N_i}})$. This function is implemented as a *policy gradient* algorithm similar to the model-free *IState-GPOMDP* algorithm pro-

Algorithm 4 Locally Interacting Distributed Reinforcement Learning Policy Search (LID-RLPS)

Let $t = 0$ and $T =$ required number of iterations.
Each agent i starts with a random policy w_i represented as $\langle \theta_i, \phi_i \rangle$.
Let w_{N_i} be the policies of agent i's neighbors N_i, and are represented as the set of policy parameters $\{\theta_k, \phi_k\}$ $\forall k \in N_i$.
Let g_t be the I-state of the agent at time t.
While $t < T$
    Obtain y_t as local observation
    Agent i exchanges policies w_i with neighbors N_i
    Form δ_{N_i} as a *feature* vector from $\langle \theta_k, \phi_k \rangle$ $\forall k \in N_i$
    Choose g_{t+1} from $f_i(\cdot \mid \phi_i, g_t, y_t, a_i(t-1), \delta_{N_i})$
    Choose and execute action $a_i(t)$ from $\mu_i(\cdot \mid \theta_i, g_{t+1}, y_t, \delta_{N_i})$
    Get the immediate localized cost $C_{loc,i}(t)$
    $w_i^* = \langle \theta_i^*, \phi_i^* \rangle = \mathit{LocalBestPolicy}(w_i, \delta_{N_i}, y_t, C_{loc,i}(t))$
    $cValue = \mathit{GetEstimate}(w_i, \delta_{N_i})$
    $mValue = \mathit{GetEstimate}(w_i^*, \delta_{N_i})$
    $gain_i = \|cValue - mValue\|$
    Broadcast $gain_i$ to N_i
    $maxGain = \max_{k \in \{i \cup N_i\}} gain_k$
    $winner = \arg \max_{k \in \{i \cup N_i\}} gain_k$
    If $maxGain > 0$
        If $i = winner$ then
            Update policy: $w_i = w_i^*$ or $\langle \theta_i, \phi_i \rangle = \langle \theta_i^*, \phi_i^* \rangle$
            Broadcast w_i^* to N_i
        Else
            Receive policy w_{winner} from winner
            Update w_{N_i}
        End If
    Else
        Break While;
    End If
    $t = t + 1$
End While

posed in [63]. The pseudo-code for $\mathit{LocalBestPolicy}(w_i, \delta_{N_i})$ is shown in Algorithm 5.

The function $\mathit{GetEstimate}(w_i, \delta_{N_i})$ in Algorithm 4 computes the estimated local neighborhood utility $B(N_i, \phi_i, \theta_i | \overrightarrow{w_{N_i}})$ given the neighbors' policies. If we simply follow the *IState-GPOMDP* algorithm in [63], this estimate is obtained from the immediate localized cost $C_{loc,i}(t)$ as follows: Let $\eta_{B_i}(t)$ be the current estimate of

Algorithm 5 Finding the Best Local Policy Response

Given $\beta \in [0,1)$. $\alpha_t = \frac{1}{t}$ = learning rate
Set $z_0^{\theta_i} = z_{new}^{\theta_i} = [0]$, $z_0^{\phi_i} = z_{new}^{\phi_i} = [0]$;
$\triangle_0^{\theta_i} = \triangle_{new}^{\theta_i} = [0]$; $\triangle_0^{\phi_i} = \triangle_{new}^{\phi_i} = [0]$;
$\phi_{i,new} = [0]$, $\theta_{i,new} = [0]$;
where $z_0^{\theta_i}, z_{new}^{\theta_i}, \triangle_0^{\theta_i}, \triangle_{new}^{\theta_i}, \theta_{i,new} \in \mathbb{R}^{n_{\theta_i}}$,
$z_0^{\phi_i}, z_{new}^{\phi_i}, \triangle_0^{\phi_i}, \triangle_{new}^{\phi_i}, \phi_{i,new} \in \mathbb{R}^{n_{\phi_i}}$.
Function *LocalBestPolicy*(w_i, δ_{N_i},y_t,$C_{loc,i}(t)$)

$$z_{new}^{\phi_i} = \beta z_t^{\phi_i} + \frac{\nabla f_i(g_{t+1}|\phi_i, g_t, y_t, a_i(t-1), \delta_{N_i})}{f_i(g_{t+1}|\phi_i, g_t, y_t, a_i(t-1), \delta_{N_i})}$$

$$z_{new}^{\theta_i} = \beta z_t^{\theta_i} + \frac{\nabla \mu_i(a_i(t)|\theta_i, g_{t+1}, y_t, \delta_{N_i})}{\mu_i(a_i(t)|\theta_i, g_{t+1}, y_t, \delta_{N_i})}$$

$$\triangle_{new}^{\phi_i} = \triangle_t^{\phi_i} + \frac{1}{t+1}\left[C_{loc,i}(t) z_{new}^{\phi_i} - \triangle_t^{\phi_i}\right]$$

$$\triangle_{new}^{\theta_i} = \triangle_t^{\theta_i} + \frac{1}{t+1}\left[C_{loc,i}(t) z_{new}^{\theta_i} - \triangle_t^{\theta_i}\right]$$

$$\phi_{i,new} = \phi_i - \alpha_{t+1} \triangle_{new}^{\phi_i}$$

$$\theta_{i,new} = \theta_i - \alpha_{t+1} \triangle_{new}^{\theta_i}$$

Return $w_i^* = \langle \phi_{i,new}, \theta_{i,new} \rangle$
End Function

$B(N_i, \phi_i, \theta_i | \overrightarrow{w_{N_i}})$ given the *current* neighbor policies $\overrightarrow{w_{N_i}}$. It is then updated as follows:

$$\eta_{B_i}(t+1) = \eta_{B_i}(t) + \frac{1}{t+1}\left[C_{loc,i}(t+1) - \eta_{B_i}(t)\right] \tag{6.18}$$

However, this update structure may not be suitable in storing the estimates of *every* possible policies of neighbors $\overrightarrow{w_{N_i}}$. This is due to the fact that δ_{N_i} is obtained from the set of neighbor parameters $\langle \theta_k, \phi_k \rangle$ $\forall k \in N_i$, which are from *continuous* vector spaces. We then propose that the estimated utility $\eta_{B_i}(t)$ is approximated using a form of *neural network*, known as *Cerebellar Model Articulation Controller* (CMAC).

A CMAC is a tile-coding structure that performs linear function approximation, where the output (i.e. $\eta_{B_i}(t)$) is a weighted linear sum of the features of the input vector parameters. The input vector for the CMAC is represented as $[g_{t+1}, y_t]$. The CMAC internal neural network weights are represented by the vector parameters $\langle \delta_{N_i}, \theta_i, \phi_i \rangle$.

Figure 6.4 shows the CMAC neural network for representing the estimate utility

$\eta_{B_i}(t)$. This representation is required since the vector δ_{N_i} is continuous and we require to retrieve $\eta_{B_i}(t)$ for every possible set of neighbor policy vectors.

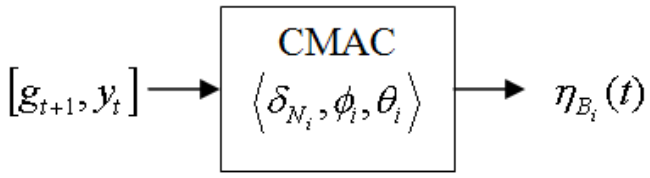


Figure 6.4: Linear Function Approximation for estimate utility $\eta_{B_i}(t)$

The CMAC displays *local generalization* for approximating the estimated utility. With this structure, different policies of neighbors are represented compactly and the estimate $\eta_{B_i}(t)$ is retrieved easily. The CMAC neural network weights are then updated with the immediate localized cost $C_{loc,i}(t)$ as the target value. More details on CMAC networks can be found in [14].

In Algorithm 4, the vector parameter δ_{N_i} is obtained from the policy vectors of the neighbors N_i. In our experiments, δ_{N_i} is computed where its vector elements are the *average* of the corresponding elements in the policy vectors: $\langle \theta_k, \phi_k \rangle \ \forall k \in N_i$. Other possible *feature* representation for δ_{N_i} can also be investigated.

In representing the MFSC distribution functions $f_i(g_{t+1} \mid \phi_i, g_t, y_t, a_i(t-1), \delta_{N_i})$ and $\mu_i(a_i(t) \mid \theta_i, g_{t+1}, y_t, \delta_{N_i})$, we face the same issue for storing the function values for every possible neighbor policies, since the vector parameter δ_{N_i} is a continuous real-valued vector. Hence, we use neural networks to represent the these distribution functions. Following [63], we use the *soft-max* function to generate the distributions.

Specifically, for $f_i(\cdot \mid \phi_i, g_t, y_t, a_i(t-1), \delta_{N_i})$, a neural network is used where the input vector is represented as $[g_t, y_t, a_i(t-1)]$, which is a concatenation of the current I-state g_t, current observation y_t, and previous action $a_i(t-1)$. The weights of the neural network are represented by the parameter vectors $\langle \delta_{N_i}, \phi_i \rangle$. The output is a $\|I_i\|$-dimensional vector $[m_1, ..., m_{\|I_i\|}]$, where $\|I_i\|$ is a constant total number of I-

states. The *soft-max* distribution for each possible next I-state $h \in I_i$ is obtained as follows:

$$f_i(h \mid \phi_i, g_t, y_t, a_i(t-1), \delta_{N_i}) = \frac{\exp(m_h)}{\sum_{h' \in I_i} \exp(m_{h'})} \tag{6.19}$$

Figure 6.5 shows the neural network representation for the soft-max distribution $f_i(h \mid \phi_i, g_t, y_t, a_i(t-1), \delta_{N_i})$. This is required to capture the continuous vector parameter δ_{N_i}. The next I-state g_{t+1} is chosen from the soft-max distribution in (6.19).

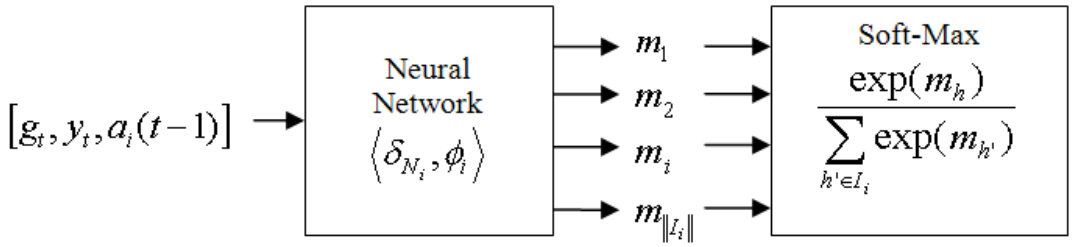


Figure 6.5: Neural Network for the Soft-Max Distribution: $f_i(h \mid \phi_i, g_t, y_t, a_i(t-1), \delta_{N_i})$

For notational convenience, let $f_i(h|\phi_i, l_i) = f_i(h \mid \phi_i, g_t, y_t, a_i(t-1), \delta_{N_i})$. As required in Algorithm 5, the log gradient of the distribution with respect to the components of the policy vector $\phi_i = [\phi_{i,k}]$, for $k = 1, ..., n_{\phi_i}$, is represented as $\frac{\nabla f_i(h|\phi_i, l_i)}{f_i(h|\phi_i, l_i)}$ and computed as follows:

$$\begin{aligned} \frac{\frac{\partial f_i(h|\phi_i, l_i)}{\partial \phi_{i,k}}}{f_i(h|\phi_i, l_i)} &= \frac{1}{f_i(h|\phi_i, l_i)} \sum_{h' \in I_i} \frac{\partial f_i(h \mid \phi_i, l_i)}{\partial m_{h'}} \frac{\partial m_{h'}}{\partial \phi_{i,k}} \\ &= \sum_{h' \in I_i} \left(\chi_{h'}(h) - f_i(h' \mid \phi_i, l_i) \right) \frac{\partial m_{h'}}{\partial \phi_{i,k}} \end{aligned} \tag{6.20}$$

where $\chi_{h'}(h) = 1$ if $h' = h$ else 0.

The first factor in the summation in (6.20) is derived from (6.19), while the sec-

ond factor $\frac{\partial m_{h'}}{\partial \phi_{i,k}}$ is the gradient of the neural network output with respect to each weight parameter $\phi_{i,k}$. The whole expression is implemented similarly to error back propagation, which is a standard procedure for training neural networks [14]. However, instead of propagating the gradient of an error measure, we back propagate the soft-max gradient for the agent's choice of h.

We derive $\frac{\nabla \mu_i(a_i(t)|\theta_i, g_{t+1}, y_t, \delta_{N_i})}{\mu_i(a_i(t)|\theta_i, g_{t+1}, y_t, \delta_{N_i})}$ in Algorithm 5 in the same way by having another neural network with input as $[g_{t+1}, y_t]$ and evaluating the soft-max distribution for each possible action $a_t(t)$ by using the real-valued outputs of the neural network.

6.3.5 Effect on Overall Network Congestion Level

Theorem 6.1: *When applying the LID-RLPS algorithm, the estimate of the global average network congestion level* $J(\overrightarrow{w}, \overrightarrow{x}(t))$ *in* (6.7) *is strictly decreasing.*

PROOF: We know from Sections 6.3.2 and 6.3.3 that the local neighborhood utility $B(N_i, \phi_i, \theta_i|\overrightarrow{w_{N_i}})$ has effectively localized the effect of agent i's policy to the global immediate cost function in (6.12). $B(N_i, \phi_i, \theta_i|\overrightarrow{w_{N_i}})$ is estimated in the current context or cycle where the neighbors' policies are fixed. From the LID-RLPS algorithm, only non-neighboring agents can modify their policies in the same cycle. If agent i has the largest gain or improvement among its neighbors, it sets its policy to the best local policy response. From (6.12) and (6.13), decreasing $B(N_i, \phi_i, \theta_i|\overrightarrow{w_{N_i}})$ results in decreasing the global average cost function. By locality of interaction, if an agent $k \notin \{i \cup N_i\}$ changes its policy to improve its local neighborhood utility, it will not affect $B(N_i, \phi_i, \theta_i|\overrightarrow{w_{N_i}})$, but will decrease the global average cost. Thus, at each cycle, the estimate of the global average network cost or congestion is strictly decreasing.■

The proof of Theorem 6.1 is similar to the case of the model-based LID-JESP algorithm [71], since both are using the concept of locality of interaction, local neigh-

borhood utility, and exchange of gain messages. The main difference lies in the computation of the local neighborhood utility, which is approximated by the model-free LID-RLPS algorithm using a policy-gradient technique in Algorithm 5. In Section 6.4, we shall show the convergence proof of LID-RLPS, especially in optimizing the local neighborhood utility.

6.3.6 LID-RLPS under Time-Varying Channel & Topology

In this subsection, we explain some issues concerning the proposed LID-RLPS algorithm for resource allocation under a time-varying channel and topology. From (6.4), at each node i and each local class queue j, the queue length (in bits) generally evolves as:

$$x_i^j(t+1) = \max\left(x_i^j(t) - \sum_{\{\forall l: q(l)=i\}} \mu_l^j(t), 0\right) + \sum_{\{\forall l: h(l)=i\}} \mu_l^j(t) + d_i^j(t) \tag{6.21}$$

where $\{q(l) = i\}$ is the set of neighboring nodes with links receiving from i, while $\{h(l) = i\}$ is the set of neighboring nodes with links transmitting to i.

We have discussed earlier in Section 6.1.4, that both time-varying channel and topology processes can be incorporated in our general queueing model in (6.4). We now have the following result:

Theorem 6.2: *Given a time-varying* topology process *evolving as an irreducible aperiodic Markov chain, let* $\mu_{i,out}^j(t) = \sum_{\{\forall l: q(l)=i\}} \mu_l^j(t)$ *be the total allocated rate for all outgoing links from node* i *for class* j*. Then,* $\mu_{i,out}^j(t)$ *is* **rate convergent** *to some constant rate* $\mu_{i,av}^j$ *(see Definition 3.1).*

Furthermore, each agent i *can use the* WF^2Q *scheduling mechanism to adaptively provision* $\sum_{j=1}^{J} \mu_{i,av}^j$ *among its* J *local class queues as follows: let* $w_{sched,j}$ *be the local class weight of the* WF^2Q *scheduling mechanism for class queue* j*,* $\forall j = 1, ..., J$*. Agent* i *obtains* $w_{sched,j}$ *from the* ***learned*** *action selection probability* $\mu_i(a_j \mid$

$\theta_i, g_{t+1}, y_t, \delta_{N_i})$ *of the LID-RLPS algorithm, where discrete action* a_j *represents the class queue* j *itself.*

PROOF: The first part of the Lemma follows immediately from Theorem 3.1. By having a *topology state process* evolving as an irreducible aperiodic Markov chain and following the proof of Theorem 3.1, we can say that the *service* process $\mu^j_{i,out}(t)$ is *rate convergent* to some constant rate $\mu^j_{i,av}$ (see Definition 3.1) for each class j.

Following the reasoning in Section 3.5 and Section 4.4.2, we can thus use weighted fair queueing techniques for scheduling. Specifically, WF^2Q is used by each *scheduling* agent i to adaptively provision $\sum_{j=1}^{J} \mu^j_{i,av}$ among its J local class queues, since the service processes of the class queues are *rate convergent*.

This is similar to the single-agent framework in Chapters 3, 4 and 5. However, in this chapter, the scheduling mechanism is performed in conjunction with the decentralized LID-RLPS algorithm. We elaborate this concept as follows:

By definition, the WF^2Q weight $w_{sched,j}$ for class j represents the allocated share Bw_j (in bits per seconds) of bandwidth from the total share $\sum_{j=1}^{J} \mu^j_{i,av}$. In other words, $w_{sched,j} = \frac{Bw_j}{\sum_{j=1}^{J} \mu^j_{i,av}}$. Since the process $\mu^j_{i,out}(t)$ is rate convergent, w_j also represents the ***probability of allocating*** Bw_j ***bits over time*** to class j.

In other words, if agent i allocates the ratio $w_{sched,j}$ to class j, then ratio $w_{sched,j}$ of the time, agent i selects queue j to empty out its contents. This is the *same* as the agent choosing a class queue among its J class queues *over time* from its action selection distribution $\mu_i(a_j \mid \theta_i, g_{t+1}, y_t, \delta_{N_i})$, where action a_j is the class queue j itself. ■

The action selection distribution $\mu_i(a_j \mid \theta_i, g_{t+1}, y_t, \delta_{N_i})$ is learned using the policy gradient mechanism in Algorithm 5 of LID-RLPS with locality of interaction, and

not simply by *blindly searching of parameters,* nor independently among neighbors.

We also emphasize that in our multi-class queue network, the rate convergent result for the service process for each class queue j is always true, no matter what the local class queue condition is (i.e. even backlogged or not). This holds true provided the topology and channel state processes as discussed in Section 6.1.4 evolve as an irreducible aperiodic Markov chain.

6.3.7 Advantage over a Cluster-Based Approach

We observe that the locality of interaction under the DEC-POMDP framework in Section 6.3.2 can be compared with an hierarchical cluster-based architecture for MANETs [72, 73]. Some issues in a cluster-based architecture include mobility and formation of clusters (i.e. selection of cluster-heads).

A cluster-based approach is similar to the locality of interaction in the LID-RLPS algorithm and reduces the number of messages to get the required agent policy parameters. However, each cluster (i.e. cluster-head) has to coordinate among other neighboring clusters (i.e. cluster-heads) in order to optimize a given performance metric (i.e. average congestion level).

We observe that in this manner, the cluster-heads themselves form a higher-level DEC-POMDP with locality of interaction. Furthermore, depending on the number of levels in the hierarchical cluster-based architecture, we can find a corresponding DEC-POMDP among nodes on the same level.

In other words, our proposed DEC-POMDP (i.e. non-hierarchical) framework with locality of interaction is less complex than a cluster-based architecture, since it has already considered the coordination among nodes, *without* the need for cluster formation. The size of locality to share information is handled during the *automatic* search of winning agent in the locality with the highest gain as explained in Algorithm 4.

6.4 Performance analysis

In this section, we analyze the proposed model-free algorithm in terms of optimality and stability for the queue scheduling problem. As explained in Section 6.1.3, the queueing law in (6.4) evolves as a Markov chain, jointly controlled by multiple agents in a decentralized manner.

The stability and optimality of Markov chains are usually studied by analyzing how the state of the Markov chain evolves, what properties the chain must satisfy, and how to control the Markov chain to preserve such properties. For instance, the irreducibility property has been commonly used in model-free algorithms [10]. However, as explained in [5, 59] and Chapter 3, this irreducibility property may not hold, since in a dynamic network, the Markov chain may not have a single communicating class of states.

Thus, we use the ψ-irreducibility framework discussed in Chapter 3 to study the stability and performance analysis in our DEC-POMDP formulation for the queue scheduling problem. It should be noted that this ψ-irreducibility concept differs from [5] where the authors only considered a Markov chain, *without* any control algorithm for the MDP. The main reason for us to apply ψ-irreducibility is to obtain stability conditions under the model-free control algorithm, which can be used to derive performance bounds in Section 6.4.4 as the algorithm converges.

Figure 6.6 summarizes the key ideas in this section:

1. For the DEC-POMDP model for queue scheduling, we shall first show the ψ-irreducibility framework in Section 6.4.1. Specifically, we prove ψ-irreducibility in Lemma 6.2 under some stability assumptions and characteristics of the joint policy.

2. We then introduce the stability property known as *geometric drift* in Definition 6.2 and prove the convergence of the LID-RLPS algorithm in Theorem 6.3.

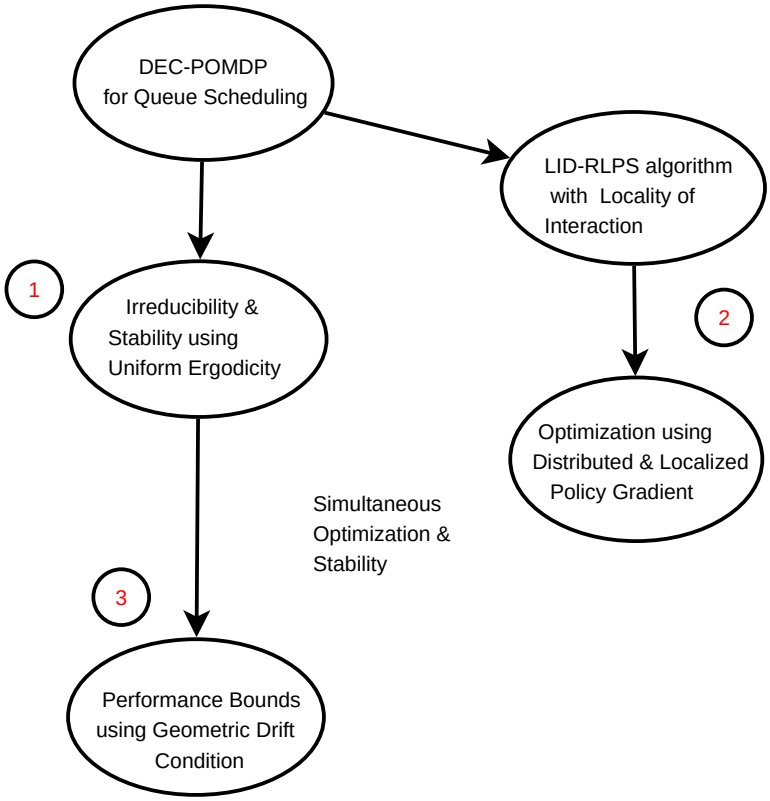


Figure 6.6: Summary of techniques used in performance analysis for DEC-POMDP

3. Finally, we use the geometric drift condition or V-*uniform ergodicity* to derive bounds in Section 6.4.4.

6.4.1 ψ-irreducibility for DEC-POMDP queueing problem

In [74], the authors proposed a multi-agent cross-product MDP from the DEC-POMDP itself together with the finite state controllers of all nodes. In this chapter, we apply this idea but with the locality of interaction among neighbors, as discussed in the previous section.

Formally, we only consider the DEC-POMDP components in the neighborhood of agent i and define the tuple:

$$\langle S_{i,N_i}, A_{i,N_i}, P, C_{loc,i}, \Omega_{i,N_i}, O, p_o \rangle \tag{6.22}$$

where:

S_{i,N_i} is the set of *factored state* vector containing the local queue lengths of agent i and its neighboring agents N_i. Note that each agent does not completely observe this state (i.e. each agent only observes its own local queue lengths).

A_{i,N_i} is the set of actions $\{A_k\}$, $\forall k \in \{i \cup N_i\}$ for the neighboring agents including agent i.

$P(S'_{i,N_i}|S_{i,N_i}, \overrightarrow{a_{i,N_i}})$ is the state transition probability.

$C_{loc,i}$ is the immediate localized cost in (6.13) which depends only on $\langle S_{i,N_i}, \overrightarrow{a_{i,N_i}} \rangle$.

Ω_{i,N_i} is the corresponding set of observation vectors

$O(\overrightarrow{o_{i,N_i}}|S_{i,N_i}, \overrightarrow{a_{i,N_i}}, S'_{i,N_i})$ is the observation function probability.

p_0 is the initial state probability distribution for S_{i,N_i}.

We also use the MFSC with neighbor locality for each agent k: $\langle I_k, \phi_k, f_k, \theta_k, \mu_k \rangle$ $\forall k \in \{i \cup N_i\}$. Specifically, let $\overrightarrow{g_i}(t) = [g_k \in I_k]$ be a vector of I-states at time t from agents i and its neighbors N_i. We define $\eta(\overrightarrow{o_{i,N_i}}|\overrightarrow{g_i}(t))$ be a mapping from the set of observations $\overrightarrow{o_{i,N_i}}$ to the set of next I-states vector $\overrightarrow{g_i}(t+1)$, given the current I-states vector $\overrightarrow{g_i}(t)$.

Following [74], the cross product MDP from the DEC-POMDP in (6.22) and MFSC for agent k, $\forall k \in \{i \cup N_i\}$ is defined as the tuple:

$$\left\langle \hat{S}, \hat{A}, \hat{P}, \hat{C} \right\rangle \tag{6.23}$$

where:

$\hat{S} = \left(\prod_k I_k \right) \times S_{i,N_i}$ is the cross product space of the S_{i,N_i} and I_k.

$\hat{A} = \prod_k \left(A_k \times I_k^{\Omega_k} \right)$ is the cross product space of the action space A_k and the set of I-states I_k with respect to the set of observations Ω_k.

$\hat{C}(\hat{S}(t), \hat{A}(t))$ is the immediate cost function, which is also equal to $C_{loc,i}(t)$.

$\hat{P}(\hat{S}(t+1)|\hat{A}(t), \hat{S}(t))$ is the state transition probability that can be written as:

$$\hat{P}(\hat{S}(t+1)|\hat{A}(t),\hat{S}(t)) = P(S_{i,N_i}(t+1)|\overrightarrow{a_{i,N_i}}(t), S_{i,N_i}(t))$$
$$\cdot \sum_{\overrightarrow{o_{i,N_i}}(t)\in\Omega_{i,N_i}} O(\overrightarrow{o_{i,N_i}}(t)|\overrightarrow{a_{i,N_i}}(t), S_{i,N_i}(t), S_{i,N_i}(t+1))$$

where:

$O(\overrightarrow{o_{i,N_i}}(t)|\overrightarrow{a_{i,N_i}}(t), S_{i,N_i}(t), S_{i,N_i}(t+1))$ is the probability of observing $\overrightarrow{o_{i,N_i}}(t)$ when agents in $S_{i,N_i}(t)$ take $\overrightarrow{a_{i,N_i}}(t)$ resulting to $S_{i,N_i}(t+1)$.

$\eta(\overrightarrow{o_{i,N_i}}(t)|\overrightarrow{g_i}(t)) = \overrightarrow{g_i}(t+1)$ is the mapping from $\overrightarrow{o_{i,N_i}}(t)$ to $\overrightarrow{g_i}(t+1)$, as obtained from the agents' policies.

$\hat{S}(t) = \langle \overrightarrow{g_i}(t), S_{i,N_i}(t)\rangle$

$\hat{A}(t) = \langle \overrightarrow{a_{i,N_i}}(t)\rangle$, which depends on $\overrightarrow{a_{i,N_i}}(t)$ itself and $\eta(\overrightarrow{o_{i,N_i}}(t)|\overrightarrow{g_i}(t))$.

Note that the DEC-POMDP components in (6.22) is similar to the single-agent POMDP for each agent i with state $\overrightarrow{e_i}(t) = \left\langle S_{i,N_i}(t), \overrightarrow{o^h_{N_i}}(t)\right\rangle$ in (6.14). Although $\overrightarrow{e_i}(t)$ is not used explicitly in LID-RLPS, in this subsection, we use $\hat{S}(t) = \langle \overrightarrow{g_i}(t), S_{i,N_i}(t)\rangle$ as the state of the cross product MDP. $\hat{S}(t)$ does not contain $\overrightarrow{o^h_{N_i}}(t)$ that represents the complete history of observations of other agents up to time t. This is not only for ease of representation, but also for analysis in verifying ψ-irreducibility.

We emphasize that in the cross product MDP, no agent can effectively know the global state $\hat{S}(t)$ especially when each agent i executes LID-RLPS. In spite of this, the main idea here is to show that the controlled Markov chain $\left\langle \hat{S}, \hat{A}, \hat{P}, \hat{C}\right\rangle$ is ψ-irreducible under the joint policy $\overrightarrow{w_{i,N_i}} = [w_i, \overrightarrow{w_{N_i}}]$, where the joint policies are *fixed* at the current context of LID-RLPS.

The state $\hat{S}(t) = \langle \overrightarrow{g_i}(t), S_{i,N_i}(t)\rangle$ consists of the I-states vector and the factored queue lengths of agents. Since the queue length processes among class queues are independent as discussed in Sections 6.1.3 and 6.3.3, our analysis is based on a single class queue. From the queueing law in (6.4) and by considering locality of interaction, it can be seen that the queue dynamics (in bits), given a fixed joint policy $\overrightarrow{w_{i,N_i}}$, can be written as:

$$\overrightarrow{x^j_{i,N_i}}(t+1) = \overrightarrow{x^j_{i,N_i}}(t) + R^j_{i,N_i}(t)M^j_{i,N_i}(t)\overrightarrow{\mu^j_{i,N_i}}(t) + \overrightarrow{D^j_{i,N_i}}(t)$$

This queueing law is similar to (6.4), but is only concerned among agent i and its neighbors N_i, where $S_{i,N_i}(t) = [\overrightarrow{x^1_{i,N_i}}(t), .., \overrightarrow{x^J_{i,N_i}}(t)]$ for J network classes. On the other hand, $\overrightarrow{g_i}(t)$ evolves from observation mapping: $\eta(\overrightarrow{o_{i,N_i}}|\overrightarrow{g_i}(t-1)) = \overrightarrow{g_i}(t)$. The action vector $\hat{A}(t)$ from the joint policy $\overrightarrow{w_{i,N_i}} = [w_i, \overrightarrow{w_{N_i}}]$ depends on the allocation rate $\overrightarrow{a_{i,N_i}}(t) := \overrightarrow{\mu^j_{i,N_i}}(t)$, observation vector $\overrightarrow{o_{i,N_i}}$, and internal state $\overrightarrow{g_i}(t)$. Thus, we can write:

$$\hat{S}(t+1) = \hat{S}(t) + H(\hat{S}(t)) + M(\hat{S}(t), \hat{W}(t)) \tag{6.24}$$

where $\{\hat{W}(t)\}$ is i.i.d. and independent of $\hat{S}(0)$. We assume that $\hat{W}(t)$ is taken from the space $\hat{S}$, but independent of $\hat{S}(t)$ itself. We assume that the functions H and M are smooth, while function H is Lipschitz.

The pair $\left\langle \hat{S}, \beta(\hat{S}) \right\rangle$ is a measurable space with $\beta(\hat{S})$ as the σ-field. The state space $\hat{S}$ consists of two components: subsets of multidimensional Euclidean space for $S_{i,N_i}(t)$; and subsets of multidimensional countable and discrete space for the I-states $\overrightarrow{g_i}(t)$. Thus, the space $\hat{S}$ is *compact* and a separable metric space.

We also assume that $\hat{S}$ evolves as an *aperiodic* Markov chain. We refer the reader to [23] for further terminology and notation. In addition, we have the following assumptions:

Assumption $A(1)$: Let $\hat{S}(t) = \hat{S}^*$ for $t \to \infty$. There exist $\hat{W}(t) = \hat{W}^*$ such that $M(\hat{S}^*, \hat{W}^*) = 0$, and for a continuous function $\xi : \mathbb{R}^d \to [0,1]$ with $\xi(\hat{W}^*) > 0$ and for $B \in \beta(\mathbb{R}^d)$: $Pr(\hat{W}(0) \in B) \geq \int_B \xi(z)dz$. This assumption is mainly for characterizing the density of $\hat{W}(t)$.

Assumption $A(2)$: The pair of matrices (F, G) is *controllable* with:

$F = \frac{\partial}{\partial \hat{S}} H(\hat{S}^*) + \frac{d}{d\hat{S}} M(\hat{S}^*, \hat{W}^*)$ and $G = \frac{d}{d\hat{W}} M(\hat{S}^*, \hat{W}^*)$. The concept of *controllable* pair (F, G) is used in linear control models if the control matrices (F, G) satisfy certain matrix structure properties so that each pair of states $\left\langle \hat{S}(0), \hat{S}(t) = \hat{S}^* \right\rangle$ can be reached. For more details on *controllable* matrices, see [23, Chapter 4].

Definition 6.1: *We term a joint policy* $\overrightarrow{w_{i,N_i}} = [w_i, \overrightarrow{w_{N_i}}]$ *that satisfies Assumptions* $A(1)$ *and* $A(2)$ *as a* dominating *policy.*

Lemma 6.2: *Under a dominating joint policy* $\overrightarrow{w_{i,N_i}} = [w_i, \overrightarrow{w_{N_i}}]$, *the cross product MDP* $\left\langle \hat{S}, \hat{A}, \hat{P}, \hat{C} \right\rangle$ *in* (6.23) *is a* ψ*-irreducible controlled Markov chain.*

PROOF: We follow some notations from Chapter 2. From Assumptions $A(1)$ and $A(2)$ and using the Implicit Function Theorem [22], the state space can be written as the union of open sets [23, Proposition 7.1.5]. Furthermore, if O is an open set containing $\hat{S}^*$ and $\hat{s} \in O$, then under policy $\overrightarrow{w}$, the t-step transition probability from state $\hat{s}$ satisfies:

$$P^t_{\overrightarrow{w}}(\hat{s}, R) = P(\hat{S}(t) \in R | \hat{S}(0) = \hat{s}) \geq \epsilon v(R) \tag{6.25}$$

where $\epsilon > 0$ is a constant, $R \in \beta(\hat{S})$, and $v(\cdot)$ is the uniform distribution on set O. The set O is also known as a *small* and *petite set* from the inequality condition in (6.25).

From (2.1), the resolvent kernel for set O, under fixed policy $\overrightarrow{w} = \overrightarrow{w_{i,N_i}}$, can be written as: $K_{\overrightarrow{w}}(s, O) := \sum_{t=0}^{\infty} 2^{-(t+1)} P^t_{\overrightarrow{w}}(s, O)$ for any $s \in \hat{S}$. From Assumption $A(1)$, $M(\hat{S}^*, \hat{W}^*) = 0$ and thus, by using (6.24), we have $\hat{S}(t) \in O$ for all sufficiently large t. In other words, $P^t_w(s, O) > 0$ for large t. Since $\hat{W}^*$ is also the support of the marginal distribution of $\{\hat{W}(t)\}$ from Assumption $A(1)$, it then follows that $K_{\overrightarrow{w}}(s, O) > 0$ from the resolvent kernel definition.

From the inequality in (6.25), we can write the following for some $m \in \mathbb{Z}^+$ and any $s \in \hat{S}$, $R \in \beta(\hat{S})$ (see [22]):

$$K_{\overrightarrow{w}}(s,R) \geq \int K_{\overrightarrow{w}}(s,dy)2^{-m}P^{m}_{\overrightarrow{w}}(y,R)$$
$$\geq 2^{-m}K_{\overrightarrow{w}}(s,O)\epsilon v(R)$$

Since the right-hand side expression of the last inequality above is always positive, the Markov chain in (6.24) is a T-chain [24]. Finally, from the result in [23, Proposition 6.2.1] with one reachable set O containing $\hat{S}^*$ and from Definition 2.1, the controlled Markov chain is thus ψ-irreducible. ■

6.4.2 Markov Stability and Convergence of LID-RLPS

In analyzing the convergence of the proposed model-free LID-RLPS algorithm, we first introduce the following stability condition [23, Chapter 16].

Definition 6.2: *Consider a ψ-irreducible Markov chain Φ with measurable space $\beta(\hat{S})$ and probability transition P. If there exists an extended-value function $V : \hat{S} \rightarrow [1,\infty]$ bounded in $\hat{S}$, a measurable set C, and constants $\gamma > 0$, $b < \infty$, such that for $s \in \hat{S}$:*

$$\triangle V(s) := \int P(s,dy)V(y) - V(s) \leq -\gamma V(s) + b\delta_C(s) \tag{6.26}$$

then Φ exhibits geometric drift *towards set C, where $\delta_C(s)$ is the indicator function: $\delta_C(s) = 1$ if $s \in C$ else 0.*

In this subsection, we shall use the geometric drift condition to show that the ψ-irreducible chain $\Phi = \left\langle \hat{S}, \hat{A}, \hat{P}, \hat{C} \right\rangle$ has a unique stationary distribution. Once this is established, we shall prove the convergence of the LID-RLPS algorithm, specifically Algorithm 5.

Formally, since the state space $\hat{S}$ is compact, and that the chain Φ is a ψ-irreducible and aperiodic T-chain from Lemma 6.2, then Φ is known as *uniformly ergodic* [23, Theorem 16.2.5].

Some properties of *uniformly ergodic* chains are:

1. Existence of a unique invariant probability measure π such that:

$$\sup_{s \in \hat{S}} \left\| P^t(s, B) - \pi(B) \right\| \to 0, \qquad t \to \infty.$$

   where $B \in \beta(\hat{S})$, $\pi(B)$ is the steady state probability distribution, and the norm $\|\nu(\cdot)\|$, for some signed measure $v(\cdot)$ on $\beta(\hat{S})$, is known as the *total variation norm*: $\|\nu(\cdot)\| = \sup_{B \in \beta(\hat{S})} v(B) - \inf_{B \in \beta(\hat{S})} v(B)$.

2. The geometric drift condition in (6.26) is satisfied for a *petite* set C, and a bounded function V.

Hence, the chain has a unique stationary distribution π for a given dominating policy $\overrightarrow{w} = \overrightarrow{w_{i,N_i}}$ in the MDP. This also implies under policy $\overrightarrow{w}$, the Markov chain is defined to be *stable* (i.e. uniformly ergodic).

Let $c(\overrightarrow{w}) \in \mathbb{R}^{\|\hat{S}\|}$ be a $\left\|\hat{S}\right\|$-dimensional vector, where $\left\|\hat{S}\right\|$ is the total number of states in the countable state space $\hat{S}$, such that the s-th component is:

$$c(\overrightarrow{w}, s) := E_{\overrightarrow{w}}\{\hat{C}(s, \overrightarrow{a})|s\} \tag{6.27}$$

for all $s \in \hat{S}$ and $\overrightarrow{a} \in \hat{A}$. By using the fact that $\hat{C}(\hat{S}(t), \hat{A}(t)) = C_{loc,i}(t)$, we can write the local neighborhood utility $B(N_i, \phi_i, \theta_i|\overrightarrow{w_{N_i}})$ in (6.17) to be equal to:

$$\eta_{\overrightarrow{w}} := B(N_i, \phi_i, \theta_i | \overrightarrow{w_{N_i}}) = \lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n} E_{\overrightarrow{w}} \{C_{loc,i}(t)\}$$
$$= \sum_{s\in\hat{S}} \pi(s) c(\overrightarrow{w}, s) \tag{6.28}$$

This expression is thus independent of the initial state. LID-RLPS tries to find $\langle \phi_i, \theta_i \rangle$ that minimizes $\eta_{\overrightarrow{w}}$ given the policies of neighbors. Hence, it is intuitive to look at the gradient of $\eta_{\overrightarrow{w}}$ with respect to the parameters $\langle \phi_i, \theta_i \rangle$. Specifically, the gradient can be expressed as: $\nabla\eta_{\overrightarrow{w}} = (\nabla\pi)^T c(\overrightarrow{w}) + (\nabla c(\overrightarrow{w}))^T \pi$, where π is written as a $\left\|\hat{S}\right\|$-dimensional vector.

Given $\beta \in [0, 1)$ from Algorithm 5, we define a vector $\zeta_{\overrightarrow{w},\beta} \in \mathbb{R}^{\|\hat{S}\|}$ where the s-th component is:

$$\zeta_{\overrightarrow{w},\beta}(s) = e_s^T \sum_{t=0}^{\infty} \beta^t P(\overrightarrow{w})^t c(\overrightarrow{w}) \tag{6.29}$$

where:

$e_s^T \in \mathbb{R}^{\|\hat{S}\|}$ is a unit vector where the s-th component is equal to 1.

$P(\overrightarrow{w})^t = (P(\overrightarrow{w}))^t$ is t-th power of the probability matrix $P(\overrightarrow{w})$ of the Markov chain Φ under policy $\overrightarrow{w}$ and $P(\overrightarrow{w})^0 = I$ is the identity matrix.

From [61], $\lim_{\beta\to 1} \pi^T \nabla P(\overrightarrow{w}) \zeta_{\overrightarrow{w},\beta} = (\nabla\pi)^T c(\overrightarrow{w})$. This implies that:

$$\nabla\eta_{\overrightarrow{w},\beta} = \pi^T \nabla P(\overrightarrow{w}) \zeta_{\overrightarrow{w},\beta} + (\nabla c(\overrightarrow{w}))^T \pi \tag{6.30}$$

where $\nabla\eta_{\overrightarrow{w},\beta}$ is a good estimate of $\nabla\eta_{\overrightarrow{w}}$ when β is close to 1.

Theorem 6.3: *During the update of the policy for agent i in Algorithm 5 when it is the* winner *and for a given $\beta \in [0, 1)$, let $\triangle_T := [\triangle_T^{\theta_i}, \triangle_T^{\phi_i}]$ be the gradient estimate after T cycles. Under the ψ-irreducible aperiodic T-chain Φ and assumptions in*

Lemma 6.2 with a dominating policy $\overrightarrow{w}$, $\lim_{T\to\infty} \triangle_T = \nabla\eta_{\overrightarrow{w},\beta} = [\frac{\partial\eta_{\overrightarrow{w},\beta}}{\partial\theta_i}, \frac{\partial\eta_{\overrightarrow{w},\beta}}{\partial\phi_i}]$.

PROOF: Since the chain Φ is uniformly ergodic and satisfies (6.26), then from [23, Lemma 15.2.2], the function V is known as *unbounded off petite sets* that satisfy the following definition: For any $n < \infty$, the sublevel set $C_V(n) = \{s : V(s) \leq n\}$ is petite. A set is petite if it satisfies the condition in (6.25).

The geometric drift condition in (6.26) can be easily written as:

$$\triangle V(s) \leq 0, \quad s \notin C \tag{6.31}$$

From this inequality condition and the result in [23, Theorem 9.1.8], and using the fact that C is a petite set from the uniform ergodicity property, then the chain Φ is known as *Harris recurrent chain* that satisfies the following: $L(s, C) = 1$, which is the *return time probability* starting with any state $s \in \hat{S}$ to the petite set C, as described in (2.2) and Section 2.1.

By definition in [23, Chapter 10], if Φ is ψ-irreducible, admits an invariant probability measure π, and is Harris recurrent, then Φ is called a positive Harris chain.

From Algorithm 5, let $\mu_i(a_i(t) \mid \theta_i, g_{t+1}, y_t, \delta_{N_i}) = \mu_i(a_i(t)|l_i(t))$ for ease of notation. Since the policy is fixed at $\overrightarrow{w}$, we can expand $\triangle_T^{\theta_i}$ for T steps:

$$\begin{aligned}\triangle_T^{\theta_i} := \tfrac{1}{T}\left[\sum_{k=0}^{T} C_{loc,i}(k)\frac{\nabla\mu_i(a_i(k)|l_i(k))}{\mu_i(a_i(k)|l_i(k))}\right] \\ +\tfrac{1}{T}\left[\sum_{k=0}^{T} C_{loc,i}(k)\sum_{t=0}^{k}\beta^{k-t}\frac{\nabla\mu_i(a_i(t)|l_i(t))}{\mu_i(a_i(t)|l_i(t))}\right]\end{aligned} \tag{6.32}$$

We shall show that the first and second terms of (6.32) converge to $(\nabla c(\overrightarrow{w}))^T \pi$ and $\pi^T \nabla P(\overrightarrow{w})\zeta_{\overrightarrow{w},\beta}$, respectively. The proof for the case of $\triangle_T^{\phi_i}$ is similar.

Using a similar idea in [75, Theorem 4], for positive Harris chain with the cost function $\hat{C}(s(t), \overrightarrow{a}(t)) = C_{loc,i}(t)$ as a Borel-measurable function, the first term in

(6.32) can be written as:

$$\lim_{T\to\infty}\frac{1}{T}\left[\sum_{k=0}^{T} C_{loc,i}(k)\frac{\nabla\mu_i(a_i(k)|l_i(k))}{\mu_i(a_i(k)|l_i(k))}\right] = E_\pi\left\{\hat{C}(s(0),\overrightarrow{a}(0))\frac{\nabla\mu_i(a_i(0)|l_i(0))}{\mu_i(a_i(0)|l_i(0))}\right\} \quad (a.s.) \tag{6.33}$$

To simplify this relation, we rewrite the cost vector component $c(\overrightarrow{w}, s)$ in (6.27) for $s \in \hat{S}$ as follows:

$$\begin{aligned} c(\overrightarrow{w}, s) &:= E_{\overrightarrow{w}}\{\hat{C}(s,\overrightarrow{a})|s,\overrightarrow{w_{N_i}}\} \\ &= \sum_{a_i\in A_i}\hat{C}(s,\overrightarrow{a})\mu_i(a_i \mid \theta_i, g, y, \delta_{N_i}) \\ &= \sum_{a_i\in A_i}\hat{C}(s,\overrightarrow{a})\mu_i(a_i \mid l_i) \end{aligned} \tag{6.34}$$

where:

$\overrightarrow{a} = [a_i, \overrightarrow{a_{N_i}}]$ represents all possible action vectors with $a_i \in A_i$ from agent i and $\overrightarrow{a_{N_i}}$ from its neighbors.

$\overrightarrow{w_{N_i}}$ is the policies of the neighbors N_i, represented as δ_{N_i}.

$\mu_i(a_i \mid \theta_i, g, y, \delta_{N_i}) := \mu_i(a_i \mid l_i)$ is the action selection distribution from the MFSC for an internal state g and observation y obtained from the state $s \in \hat{S}$, and $l_i = [\theta_i, g, y, \delta_{N_i}]$.

The representation in (6.34) is possible, since only the *winner* agent i at the current algorithm cycle can change its parameters, given the fixed policies $\overrightarrow{w_{N_i}}$ of its neighbors. By differentiating (6.34) with respect to $\langle\phi_i, \theta_i\rangle$, we have:

$$\nabla c(\overrightarrow{w}, s) = \sum_{a_i \in A_i} \hat{C}(s, \overrightarrow{a}) \nabla \mu_i(a_i \mid l_i)$$
$$= \sum_{a_i \in A_i} \hat{C}(s, \overrightarrow{a}) \frac{\nabla \mu_i(a_i \mid l_i)}{\mu_i(a_i \mid l_i)} \mu_i(a_i \mid l_i)$$
$$= c(\overrightarrow{w}, s) \frac{\nabla \mu_i(a_i|l_i)}{\mu_i(a_i|l_i)}$$

By using the stationary distribution π in (6.28), we can write the following vector expression:

$$(\nabla c(\overrightarrow{w}))^T \pi = \sum_{s \in \hat{S}} \pi(s) c(\overrightarrow{w}, s) \frac{\nabla \mu_i(a_i \mid l_i)}{\mu_i(a_i \mid l_i)} \tag{6.35}$$

Consequently, since the expression in (6.33) holds for every possible initial values $s(0)$, $\overrightarrow{a}(0)$, and $l_i(0)$, and by using (6.35), we can write (6.33) as:

$$\lim_{T \to \infty} \frac{1}{T} \left[\sum_{k=0}^{T} C_{loc,i}(k) \frac{\nabla \mu_i(a_i(k)|l_i(k))}{\mu_i(a_i(k)|l_i(k))} \right] = (\nabla c(\overrightarrow{w}))^T \pi$$

We then simplify the second term in (6.32) as follows. Using a similar argument in [75, Lemma 9] for positive Harris chains,

$$\lim_{T \to \infty} \frac{1}{T} \left[\sum_{k=0}^{T} C_{loc,i}(k) \sum_{t=0}^{k} \beta^{k-t} \frac{\nabla \mu_i(a_i(t)|l_i(t))}{\mu_i(a_i(t)|l_i(t))} \right] =$$
$$\sum_{k=0}^{\infty} \beta^k E_\pi \left\{ \frac{\nabla \mu_i(a_i(0)|l_i(0))}{\mu_i(a_i(0)|l_i(0))} \hat{C}(s(k+1), \overrightarrow{a}(k+1)) \right\} \tag{6.36}$$

From (6.29),

$$\pi^T \nabla P(\overrightarrow{w}) \zeta_{\overrightarrow{w},\beta} = \sum_{k=0}^{\infty} \beta^k \pi^T \frac{\partial P(\overrightarrow{w})}{\partial \theta_i} P(\overrightarrow{w})^k c(\overrightarrow{w}) \tag{6.37}$$

For any $k \geq 0$, by following [75, Theorem 4] and using the fact that only winner agent i changes its policy,

$$\pi^T \frac{\partial P(\overrightarrow{w})}{\partial \theta_i} P(\overrightarrow{w})^k c(\overrightarrow{w}) := \sum_{s,s',a_i} \pi(s)\mu_i(a_i|l_i)\frac{\nabla \mu_i(a_i|l_i)}{\mu_i(a_i|l_i)} P(s^{'}|s,a_i)\left[P(\overrightarrow{w})^k c(\overrightarrow{w})\right](s^{'}) = E_\pi\left\{\frac{\nabla \mu_i(a_i(0)|l_i(0))}{\mu_i(a_i(0)|l_i(0))}\hat{C}(s(k+1), \overrightarrow{a}(k+1))\right\} \tag{6.38}$$

where:

$P(s^{'}|s,a_i)$ is the state transition probability.

$\left[P(\overrightarrow{w})^k c(\overrightarrow{w})\right](s^{'})$ is the corresponding $s^{'}$-th vector element.

Using (6.36), (6.37), and (6.38), we have the desired limit for the second term of (6.32). The following equality then holds:

$$\lim_{T\to\infty} \triangle_T = \pi^T \nabla P(\overrightarrow{w})\zeta_{\overrightarrow{w},\beta} + \left(\nabla c(\overrightarrow{w})\right)^T \pi = \nabla \eta_{\overrightarrow{w},\beta}. \quad \blacksquare$$

Theorem 6.3 implies that, for each agent i, the gradient estimates $\triangle_T$ approaches the actual gradient of the local neighborhood utility $\nabla \eta_{\overrightarrow{w},\beta}$ for a given *fixed* dominating policy $\overrightarrow{w} = \langle \phi_k, \theta_k \rangle$ for $k \in \{i, N_i\}$.

LID-RLPS updates the parameters of the winning agent i simply by: $\theta_{i,new} := \theta_i - \alpha_{t+1}\triangle_{t+1}^{\theta_i}$ and $\phi_{i,new} := \phi_i - \alpha_{t+1}\triangle_{t+1}^{\phi_i}$. We note that once the parameters are changed, the joint policy also changes. Our next goal is to show that, despite this policy change, the new succeeding joint policy is still a dominating policy. As explain earlier, a dominating policy is a *stable* policy that makes the chain to be *uniformly ergodic*.

From Algorithm 4 and (6.18), the estimate $\eta_{B_i}(t)$ of the local neighborhood utility $\eta_{\overrightarrow{w}}$ in (6.28) is required to obtain the *gain* messages. The key idea is to initialize this estimate to a certain value that satisfy stability properties. Using similar arguments in [8, 59], we first initialize the estimate $\eta_{B_i}(t)$ to a *Lyapunov* function V that satisfies the geometric drift in (6.26). Specifically, since the geometric drift inequality can be reduced to the *Foster-Lyapunov inequality* in [8, 59] and Theorem 2.3, it can be shown that every succeeding policy in the iteration is also uniformly ergodic.

The parameters $\langle \theta_{i,t}, \phi_i \rangle$ of the winning agent are said to evolve on a slower time-scale than the gradient estimator $\triangle_T$. The convergence of two-time scale stochastic approximation theory is studied in [76]. Following [76] under suitable stability conditions, it can be shown that $\langle \theta_{i,t}, \phi_i \rangle$ converges to the parameter values that satisfy: $\nabla \eta_{\overrightarrow{w},\beta} = 0$.

In summary, by *starting* with a dominating policy $\overrightarrow{w}$ and initializing the estimate of the local neighborhood utility $\eta_{B_i}(t)$ to a Lyapunov function V in (6.26), we can guarantee convergence (i.e. $\nabla \eta_{\overrightarrow{w},\beta} = 0$) and stability (i.e. uniform ergodicity) of the succeeding joint policies in the LID-RLPS iteration.

We observe that this gradient-based mechanism may converge to a *locally* optimal solution. To handle this situation, we can use the idea of *actor-critic algorithms* that use a value function approximator (i.e. critic) in computing the value of the policy. This extends the ideas in [77] in expressing the gradient estimate $\triangle_T$.

6.4.3 Decentralized Queue Stability

The previous subsection discusses the stability and convergence of the LID-RLPS algorithm using the geometric drift condition in Definition 6.2. In applying this concept for our decentralized queueing model, we define decentralized queue stability as follows:

Definition 6.3: *A network of N queues is stable if the queue length process $\{x_i(t)\}$,$\forall i$ satisfies:*

$$\limsup_{M\to\infty} \frac{1}{M} \sum_{t=0}^{M-1} \left(\sum_{i=0}^{N} E\left\{x_i(t)\right\} \right) < \infty$$

Lemma 6.3: *A network of N queues is stable if and only if each queue is stable from Definition 3.2.*

PROOF: If all the queues are stable from Definition 3.2, then Definition 6.3 follows automatically, since:

$$\sum_{i=0}^{N} \left(\limsup_{M\to\infty} \frac{1}{M} \sum_{t=0}^{M-1} E\left\{x_i(t)\right\} \right) < \infty$$

If the network is stable, then by using the fact that *the lim sup of a sum is less than or equal to the sum of the lim sups* [29], then each queue is stable from Definition 3.2. ■

6.4.4 Performance Bounds using V-uniform ergodicity

From Section 6.4.2, we see that a bounded value function V exists that satisfies the geometric drift inequality. In this subsection, we show how to find V using the general queueing law in (6.4).

Lemma 6.4: *For a single class queue j, the drift value function $V^j(s) = \sum_{i=1}^{N} (x_i^j)^2$ satisfies the geometric drift inequality in Definition 6.2, where $s \in \hat{S}$ in (6.23) and x_i^j is the queue length in bits at node i.*

PROOF: From the queueing law in (6.4) and for each class queue j, suppose the arrival processes $\{d_i^j(t)\}$ have a rate $E\{d_i^j(t)\} = \lambda_i^j$ for $i = 1, ..., N$.

We rewrite the general queueing expression in (6.21) as follows:

$$x_i^j(t+1) = \max\left(x_i^j(t) - y_i^j(t), 0\right)$$

where $y_i^j(t) := \sum_{\{\forall l: q(l)=i\}} \mu_l^j(t) - \sum_{\{\forall l: h(l)=i\}} \mu_l^j(t) - d_i^j(t)$.

By squaring this expression and noting that $(\max(x,0))^2 \leq x^2$, we have:

$$\left(x_i^j(t+1)\right)^2 - \left(x_i^j(t)\right)^2 \leq \left(y_i^j(t)\right)^2 - 2x_i^j(t)y_i^j(t) \tag{6.39}$$

From Theorem 6.2, we know that the process $\mu_{i,out}^j(t) := \sum_{\{\forall l: q(l)=i\}} \mu_l^j(t)$ is *rate convergent* to some constant rate $\mu_{i,out}^j > 0$. By similar arguments to the proof of Theorem 6.2, the process defined as $\mu_{i,in}^j(t) := \sum_{\{\forall l: h(l)=i\}} \mu_l^j(t)$ is also *rate convergent* to some rate $\mu_{i,in}^j > 0$.

Assuming the *source* arrival process $\{d_i^j(t)\}$ is independent with $\{\mu_{i,out}^j(t)\}$ and $\{\mu_{i,in}^j(t)\}$, then the process $\{y_i^j(t)\}$ is also rate convergent with rate $\left(\mu_{i,out}^j - \mu_{i,in}^j - \lambda_i^j\right)$. Suppose that $\{y_i^j(t)\}$ has a second moment

$$m_i^j := E\left\{\left(y_i^j(t)\right)^2\right\} > 0 \tag{6.40}$$

By taking expectations of (6.39) with respect to $y_i^j(t)$, we have:

$$\left(x_i^j(t+1)\right)^2 - \left(x_i^j(t)\right)^2 \leq m_i^j - 2\left(\mu_{i,out}^j - \mu_{i,in}^j - \lambda_i^j\right)x_i^j(t)$$

By summing for all nodes i, letting $V^j(s_t) = \sum_{i=1}^{N}\left(x_i^j(t)\right)^2$, and taking conditional expectations on the vector $\overrightarrow{x_t^j} := [x_1^j(t), ..., x_N^j(t)]$, we obtain the geometric drift inequality as:

$$\triangle V^j(s_t) := E\left\{V^j(s_{t+1}) - V^j(s_t)|\overrightarrow{x_t^j}\right\} \leq -V^j(s_t)$$
$$+\sum_{i=1}^{N}\left(x_i^j(t)\right)^2 + \sum_{i=1}^{N} m_i^j - 2\sum_{i=1}^{N}\left(\mu_{i,out}^j - \mu_{i,in}^j - \lambda_i^j\right) E\left\{x_i^j(t)|\overrightarrow{x_t^j}\right\} \tag{6.41}$$

To identify a certain measurable set C (see (6.26)), we consider the indicator function: $\delta_C(s_t) := 1$ if $\sum_{i=1}^{N}\left(\mu_{i,out}^j - \mu_{i,in}^j - \lambda_i^j\right) > 0$, else 0 if $\sum_{i=1}^{N}\left(\mu_{i,out}^j - \mu_{i,in}^j - \lambda_i^j\right) \leq 0$.

We can then define the measurable set C as the *stability region* of the network, when $\sum_{i=1}^{N}\left(\mu_{i,out}^j - \mu_{i,in}^j - \lambda_i^j\right) > 0$. This definition is intuitive and not surprising, since it implies that for each class queue j in each node i: $\mu_{i,out}^j > \mu_{i,in}^j + \lambda_i^j$. This is also in conjunction with Lemma 6.3, implying that the measurable set C is the situation when every stable queue leads to network stability and vice versa. The result then follows.

■

We shall now use the geometric drift condition in (6.41) for deriving performance bounds for average queueing delay and congestion level.

Theorem 6.4: *The LID-RLPS algorithm stabilizes the queueing model according to Definition 6.3.*

PROOF: By letting $V(s) = \sum_{j=1}^{J}\sum_{i=1}^{N}(x_i^j)^2$, where $s \in \hat{S}$ in (6.23) and x_i^j is the class j queue length in bits at node i, and by summing (6.41) $\forall j$, we have the following geometric drift inequality for $\forall t$:

$$\triangle V(s_t) := E\left\{V(s_{t+1}) - V(s_t)|\overrightarrow{x_t}\right\} \leq -V(s_t)$$
$$+\sum_{j=1}^{J}\sum_{i=1}^{N}\left(x_i^j(t)\right)^2 + \sum_{j=1}^{J}\sum_{i=1}^{N} m_i^j - 2\sum_{j=1}^{J}\sum_{i=1}^{N}\left(\mu_{i,out}^j - \mu_{i,in}^j - \lambda_i^j\right) E\left\{x_i^j(t)|\overrightarrow{x_t^j}\right\} \tag{6.42}$$

where m_i^j is defined in (6.40), and

$$\overrightarrow{x_t} := [x_1^1(t), ..., x_N^1(t), x_1^2(t), ..., x_N^2(t), ..., x_1^J(t), ..., x_N^J(t)]$$

Since $V(s_t) \geq 0$, and by taking expectations over the distribution of $\overrightarrow{x_t}$ and summing inequality (6.42) over t from $t = 0$ to $t = M - 1$ for some $M \in \mathbb{Z}_+$ and simplifying, we have:

$$E\{V(s_M) - V(s_0)\} \leq M \sum_{j=1}^{J} \sum_{i=1}^{N} m_i^j - 2 \sum_{t=0}^{M-1} \sum_{j=1}^{J} \sum_{i=1}^{N} \left(\mu_{i,out}^j - \mu_{i,in}^j - \lambda_i^j\right) E\left\{x_i^j(t)\right\}$$

Dividing by M and using the fact that $V(s_M) > 0$, we have:

$$\frac{1}{M} \sum_{t=0}^{M-1} \sum_{j=1}^{J} \sum_{i=1}^{N} 2\left(\mu_{i,out}^j - \mu_{i,in}^j - \lambda_i^j\right) E\left\{x_i^j(t)\right\} \leq \frac{1}{M} E\{V(s_0)\} + \sum_{j=1}^{J} \sum_{i=1}^{N} m_i^j$$

By taking lim sup, we have the following bound:

$$\limsup_{M\to\infty} \frac{1}{M} \sum_{t=0}^{M-1} \sum_{j=1}^{J} \sum_{i=1}^{N} 2\left(\mu_{i,out}^j - \mu_{i,in}^j - \lambda_i^j\right) E\left\{x_i^j(t)\right\} \leq \sum_{j=1}^{J} \sum_{i=1}^{N} m_i^j$$

Noting the result in (6.4) that $\overrightarrow{x^j}(t) = [x_1^j(t), ..., x_N^j(t)]^T$ is already a Markov chain for each class j , we can write:

$$\limsup_{M\to\infty} \frac{1}{M} \sum_{t=0}^{M-1} \sum_{i=1}^{N} 2\left(\mu_{i,out}^j - \mu_{i,in}^j - \lambda_i^j\right) E\left\{x_i^j(t)\right\} \leq \sum_{i=1}^{N} m_i^j$$

Following the ideas of [29, Lemma 2] and rewriting the geometric drift for each node i and each class j,

$$\limsup_{M\to\infty} \frac{1}{M} \sum_{t=0}^{M-1} E\left\{x_i^j(t)\right\} \leq \frac{m_i^j}{2\left(\mu_{i,out}^j - \mu_{i,in}^j - \lambda_i^j\right)}$$

From Lemma 6.3, we can then write the following bound (see Lemma 6.4 for notations):

$$\limsup_{M\to\infty}\frac{1}{M}\sum_{t=0}^{M-1}\sum_{j=1}^{J}\sum_{i=1}^{N}E\left\{x_i^j(t)\right\}\leq\sum_{j=1}^{J}\sum_{i=1}^{N}\frac{m_i^j}{2\left(\mu_{i,out}^j-\mu_{i,in}^j-\lambda_i^j\right)} \tag{6.43}$$

■

Note that the bound in (6.43) only holds when the LID-RLPS algorithm is initialized with a dominating policy $\overrightarrow{w}$ and by initializing the estimate of the local neighborhood utility $\eta_{B_i}(t)$ with the *Lyapunov* function $V(s)=\sum_{j=1}^{J}\sum_{i=1}^{N}(x_i^j)^2$, so that the geometric drift condition in (6.42) is initially satisfied. This is in conjunction with Theorem 6.3 and the concepts in Section 6.4.2.

We also observe that the bound in (6.43) of Theorem 6.4 is an extension of the result from Section 3.4.3 for the single-agent case and in conjunction with Lemma 6.3. However, the result in this section is only met by following the LID-RLPS algorithm.

6.5 Simulation and Discussion

We simulate the same scenario as described in Section 3.5 with a network of 20 mobile nodes as shown in Figure 3.3. We set the maximum channel capacity at 2 Mbps, while both the network interface and routing protocol queues have a limit of 50 and 100 packets, respectively. The simulation is done with varying pause times for 3,000 seconds.

We have used eight long-lived CBR connections with the characteristics similar to Table 3.1. We choose CBR flows since this type of flows captures the worst case and average long term performance. However, our technique still applies for other types of traffic such as Pareto and Exponential ON/OFF sources. The control packets from

the routing protocol are marked as Class I and the data packet size is 64 bytes.

We have discussed in Section 6.1.4 that different MAC mechanisms and varying mobility and topology issues are captured in our general *decentralized* queueing model. It should also be noted that there have been a number of works that analyze the performance of DCF in MANETs using the Markov chain theory [32, 33]. Our approach is different because we formulate the problem using the DEC-POMDP framework for a *decentralized controlled* Markov chain.

We compare the performance of LID-RLPS with a single-agent based RL algorithm similar to [37, 59] and Section 4.2. We refer to the latter as Independent RL Provisioning (IRLP), where each node only considers its own locally-observed MDP independently and does not communicate its policies with any other agent.

As explained in (6.7) of Section 6.1.3, the main objective is to minimize the average congestion level of the network, which concerns all nodes and all network classes. Similar to Section 3.5 earlier in Chapter 3, we summarize the effective arrival rate and service class rate measurements, averaged over the simulation period and over all nodes in Table 6.1.

Table 6.1: Performance measurements under LID-RLPS for 5 secs pause time

Traffic Type	Effective Arrival Rate (bps)	Effective Service Rate (bps)	Average Congestion (bits)	Theoretical Bound for Average Congestion (bits)	Normalized Theoretical Bound
I	14671.40	14700.68	2892.53	3005.72	0.1174
II	5657.82	5741.80	991.30	1166.93	0.0455
III	15396.20	15407.39	2862.89	3200.15	0.1250
Total Congestion among Classes			6746.72	7372.80	0.2879

Furthermore, we compare the average congestion level measurements and the theoretical bound computed as: $\sum_{j=1}^{J}\sum_{i=1}^{N} \frac{m_i^j}{2\left(\mu_{i,out}^j - \mu_{i,in}^j - \lambda_i^j\right)}$ from (6.43), where m_i^j is the second moment of the the process $\{y_i^j(t)\}$ defined in (6.40), and is obtained from data samples. The normalized congestion bound for each class is calculated as:

$\left(\frac{AveBits_j}{MaxBits_j}\right)$, where $AveBits_j$ is shown in the 5^{th} column of Table 6.1 and $MaxBits_j$ is the maximum possible amount of bits in queue j (i.e. $50 * 64 * 8$ bits). The tabulated simulation results show that the measured average congestion level (i.e. 4^{th} column) is well within the theoretical congestion bound (i.e. 5^{th} column) under LID-RLPS for each class.

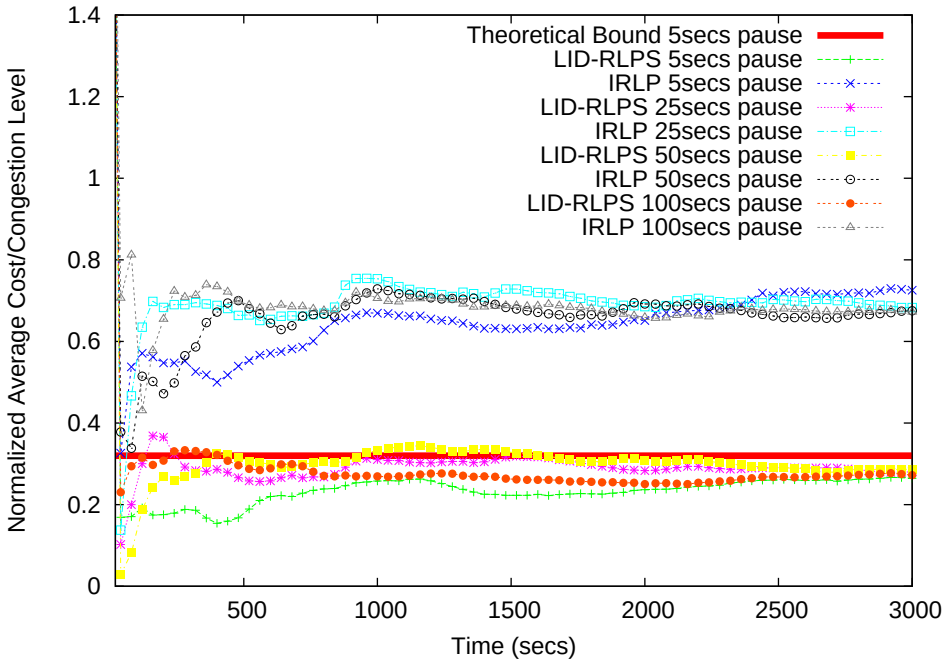


Figure 6.7: Normalized average congestion for LID-RLPS and IRLP under varying pause times

Figure 6.7 shows the average long term cost or congestion over time, under different scenarios and pause times. As expected, IRLP incurs higher congestion level even though each agent learns and adapts its policy. LID-RLPS achieves significantly lower since each agent coordinates among its neighbors before the winner agent changes its policy, as explained in Algorithm 4.

Table 6.1 shows a normalized bound value of 0.2879, summed for all J classes, for LID-RLPS under the 5 seconds pause time scenario. We observe that this is indeed a tight bound as shown in Figure 6.7. The figure also shows that the bound was not initially met due to the LID-RLPS *learning* algorithm that uses the multi-agent finite

state controller (MFSC) for each node. LID-RLPS is continually updating the MFSC parameters in each agent, and has not converged yet to the optimal solution.

We also observe that LID-RLPS satisfies the limiting bound value of 0.2879 for varying scenarios and pause times. This result supports our claim that the varying MAC mechanisms and topology are captured in our model as discussed in Section 6.1.4. We also emphasize that the theoretical bound value *can only be met by the optimal policy* from the result in Theorem 6.4.

6.6 Possible Weaknesses of LID-RLPS

As explained in Theorem 6.2 and similar to FLP, WFRLP and HORLP in the previous chapters, LID-RLPS uses WF^2Q provisioning if the topology state (i.e. also captures the channel state process in Section 6.1.4) evolves as an irreducible aperiodic Markov chain, as seen by each agent. This assumption has been commonly used in literature [6, 12, 26, 27].

Following the ideas in Section 3.6, we highlight again that this assumption can be met as follows: In a multi-node communication network, suppose the Signal-to-Interference (SIR) ratio or channel gain between each node can be measured. The SIR of a link i can be expressed as [37, 78]:

$$\Gamma^i(p_t^1, ..., p_t^N) = \frac{WA_t^i p_t^i}{R\left(\sum_{j \neq i} A_t^j p_t^j + \sigma^2\right)}$$

where W and R are the system bandwidth and transmission rate, p_t^i is the transmission power employed at node i, A_t^i is the path loss corresponding to link i, and σ^2 is the variance of the thermal noise. The path loss A_t^i depends on the distance between the transmitter i and receiver.

The SIR is essentially the main component affecting the transmission in the general decentralized queueing law in (6.4). Suppose that the SIR values are partitioned into L intervals: $0 < \Gamma_1 < ... < \Gamma_L$. The channel gain is said to be in state l if it is between interval Γ_{l-1} and Γ_l. This mapping can then be reduced into an ergodic Markov chain, and the state transition probability completely specifies the dynamics of the channel. Under the NS2 simulation [30], setting the transition probabilities can be easily done as NS2 already provides a finite-state Markov channel model. Even though this assumption of an ergodic Markov chain for the channel process is commonly used in theory and is verified in simulations, such assumption still remains to be seen in actual network implementation.

As mentioned in Section 6.3.4 and shown in Figure 6.4, a linear function approximation structure known as the CMAC is used to store the estimate $\eta_{B_i}(t)$ of the local neighborhood utility $B(N_i, \phi_i, \theta_i | \overrightarrow{w_{N_i}})$ given the neighbors' policies. This structure also faces an issue of how close is the actual estimate to the required value, especially during learning. Similar to the usage of the CMAC neural network in Section 4.6, further investigation is required to study the error bound between these two values. In relation to this storage issue, one can face similar problem when representing the softmax distribution $f_i(h \mid \phi_i, g_t, y_t, a_i(t-1), \delta_{N_i})$ in Figure 6.5, due to the approximation error of the artificial neural network structure.

We observe that even though LID-RLPS can solve the DEC-POMDP queue scheduling problem in a decentralized *model-free* manner, with locality of interaction, the price to pay is that it can only give a near-optimal policy and an increase of storage for storing and representing the different required quantities of LID-RLPS such as $\eta_{B_i}(t)$ and $f_i(h \mid \phi_i, g_t, y_t, a_i(t-1), \delta_{N_i})$.

We emphasize that the proposed technique is independent on the size of locality or neighboring nodes $\|N_i\|$. In the exchange of policy vectors in the locality, each agent performs feature extraction on the policy vectors: $\langle \theta_k, \phi_k \rangle \ \forall k \in N_i$, so that the

stored vector parameter δ_{N_i} for each agent has a constant dimension. As explained in Section 6.3.4, δ_{N_i} is computed as the average of the policy vectors. Other feature extraction mechanism can be investigated in the future.

The main idea is that, the policy learned with the current set of neighbors can be used when the set of neighbors changes. The vector space for δ_{N_i} is the same, while the values of the vector parameter δ_{N_i} vary, when the set of neighbors changes. This vector parameter δ_{N_i} is used during the LID-RLPS learning procedure to obtain the required policy, independent on the size or set of neighbors.

6.7 Chapter Summary

We have considered a stochastic optimal control approach to solve the problem of multi-class scheduling or bandwidth allocation and provisioning under a time-varying channel and topology in MANETs. Our proposed scheme is based on a novel framework known as Decentralized Partially Observable Markov Decision Process (DEC-POMDP), where the performance is affected by the joint actions of the agents. In addition each agent only observes a partial view of the current network state, which may only include its local queue lengths and policies of neighboring agents. In solving the DEC-POMDP, we propose a model-free algorithm known as LID-RLPS that performs cooperative decentralized optimization for a general Markov wireless queueing network, without requiring the topology, traffic, and wireless channel statistics. We also exploit the idea of finding a policy structure to capture the locality of interaction among neighbors. Simulation results have shown that the proposed scheme in resource allocation is able to attain its objective of minimizing the average congestion level, as compared to independent learning agents.

Chapter 7

Conclusion

In this thesis, we have considered a stochastic optimal control approach for *network-level* resource allocation and provisioning under a time-varying channel in a multi-class ad hoc network. Specifically, we study the problem of queue scheduling and buffer management, under the inherently lossy wireless medium and varying topology. Our model captures the situation of having different source-destination pairs, varying channel data and error rates, and general traffic arrival statistics and arbitrary number of nodes.

We have formulated the resource allocation problem using the decision theoretic framework known as Markov Decision Process (MDP). In this thesis, we have presented four variants of MDP formulations to highlight important results and contributions.

The first variant uses the theory of ψ-irreducibility to formulate an average cost MDP for each node acting as an agent, with the objective of minimizing the average congestion level. Using the stability conditions of ψ-irreducible Markov chains, we have presented the *first* technique of achieving optimal control, network optimization, and stability *simultaneously* for a general Markov queueing network, *without* the knowledge of traffic, topology, and channel statistics. From this theory, we have also derived performance bounds on the average congestion level and queueing delay *directly* from the value iteration algorithm initialized with a Lyapunov function, as

the algorithm converges to the optimal policy.

The second variant considers an event-based Semi-Markov Decision Process (SMDP) for each node that adaptively performs network-level bandwidth allocation and buffer management. The main objective is to maximize average long term network reward, and at the same time, minimize QoS violations with respect to bandwidth, queueing delay, and buffer loss. In solving the SMDP without knowing the underlying statistics of the channel, topology, and traffic processes, we have used a novel *model-free* approach known as Neuro-Dynamic Programming (NDP), also known as Reinforcement Learning, that uses simulation-based techniques to find near-optimal policies.

We have also considered the third variant of MDP formulation that extends the SMDP model earlier, especially for continuous state and action vector spaces in the QoS provisioning problem. By dividing the original SMDP formulation into different tasks or smaller problems, and composing policies from these tasks for the optimal policy of the original problem, we have shown that such mechanism accelerates the convergence of finding the optimal solution. Formally, we have used the framework known as Hierarchical Semi-Markov Decision Process (HSMDP) and use the corresponding model-free Hierarchical Reinforcement Learning (HRL) algorithm for provisioning. HRL provides the advantage of faster convergence and better performance in terms of long term average reward by reusing subtasks solutions in the task hierarchy of the HSMDP formulation. In the second and third variants, we have emphasized that the algorithm does not need to consider the traffic, topology, and channel statistics.

Finally, we discussed the importance of *decentralized control* for resource allocation and provisioning. We have used a novel *multi-agent* framework known as Decentralized Partially Observable Markov Decision Process (DEC-POMDP). In the earlier MDP formulations, each agent *independently* solves its own locally-observed Markov chain. On the contrary, under a DEC-POMDP, the joint policies of the nodes act-

ing as agents affect the overall performance. This framework is similar to stochastic game theory, where agents have to collaborate among themselves to find the optimal policy. However, in contrast to game theory where each agent completely observes the global state of the network, for DEC-POMDP, each agent only observes its local queue information and possibly policies of neighboring agents.

To the best of our knowledge, we believe that the DEC-POMDP formulation captures a multi-agent system for communication networks more appropriately than any other decision-theoretic, game-theoretic, or MDP-based framework. In this thesis, we present the *first* DEC-POMDP formulation for queue scheduling under a time-varying channel and topology. The solution to the DEC-POMDP gives the optimal joint policy for the agents. However, it is known that solving a DEC-POMDP is *NEXP-complete* and memory requirements grow exponentially even for finite-horizon problems. We address these issues by exploiting the *locality of interaction* among the agents and using *online* model-free techniques to approximate the optimal solution. We have proposed a model-free algorithm for achieving optimization and stability *cooperatively* and *simultaneously* in a decentralized manner, without knowing the statistics of the topology, traffic, and channel in a general Markov queueing network.

7.1 Possible Directions for Future Research

As we have presented in Chapter 3, under the ψ-irreducibility framework, one can achieve *simultaneous* optimization and stability for the queue scheduling problem. We observe that this theory for controlled Markov chains can be used to address optimization and stability for other types of networks. As mentioned earlier in Section 3.7, this technique outperforms recent state-of-the-art works by Neely. Generally speaking, using ψ-irreducibility allows us to derive similar bounds to Neely's work.

This novel theory of ψ-irreducibility has a rich mathematical framework [23], and

can be used to analyze system metrics in a communication network. Possible future areas include works similar to [6, 9, 26, 39, 40, 41] for power allocation, routing, congestion control, for analyzing energy expenditure, and study delay and rate trade-offs. We can also use this theory for cross-layer design and optimization, and derive performance bounds, similar to a recent Lyapunov-based work in [79], as long as one can find a controlled Markov chain. The main added advantage is that, we can analyze and achieve stability and optimization *simultaneously*.

As for the theory on Neuro-Dynamic Programming or Reinforcement Learning, possible future research includes adding the theory of ψ-irreducibility using the *model-free* algorithm to *directly* derive performance bounds. This is similar to the work in [22], but applied in an actual network. With this idea, one can theoretically derive performance bounds from the SMDP formulation described in Chapter 4.

For possible extension on the theory on HSMDP and HRL, one can also look at the inclusion of stability using ψ-irreducibility. We note that in Chapter 5, the HRL-based algorithm only tackles optimization, and no stability conditions were discussed. This can be further investigated in the future.

For the DEC-POMDP formulation, in this thesis, we have only proposed one model-free algorithm, with locality of interaction. As the DEC-POMDP theory is quite new, other types POMDP-based algorithms can also be investigated. Other areas of research can be done by extending the ideas in recent works by [80, 81] that use the concept of *graphical games* to study the locality of interaction among nodes, achieving *Nash equilibrium*, and in achieving performance optimization simultaneously.

Finally, for using the ψ-irreducibility stability framework on general Markov chains, showing this property of ψ-irreducibility may not be straightforward, especially when the state space is continuous and multi-dimensional. In our formulation in Section 3.3.1, ψ-irreducibility was easily verified due to the fact that the queue length process

for each class j evolves as a *random walk on the half line* and is already a Markov chain. However, for a general state descriptor such as in non-linear state space models, showing ψ-irreducibility may be difficult [23]. In this case, one may need to look at other types of stability formulations, such as how to guarantee recurrence and ergodicity of multi-dimensional Markov chains, and existence of Lyapunov functions and stability for non-linear cases [38].

Appendix

Publications:

1. Daniel Yagan and Chen-Khong Tham. "Adaptive QoS Provisioning in Wireless Ad Hoc Networks: A Semi-MDP Approach". In *IEEE Wireless Communications and Networking Conference (WCNC)*, New Orleans, Louisiana, USA, March 2005.

2. Daniel Yagan and Chen-Khong Tham. "Policy-based QoS Provisioning in Wireless Ad Hoc Networks". In *IEEE International Conference on Communications (ICC)*, Seoul, S. Korea, May 2005

3. Daniel Yagan and Chen-Khong Tham. "Self-Optimizing Architecture for QoS Provisioning in Differentiated Services". In *IEEE International Conference in Autonomic Computing (ICAC)*, Seattle, Washington, USA, June 2005.

4. Daniel Yagan and Chen-Khong Tham, "Distributed Model-Free Stochastic Optimization in Wireless Ad Hoc Networks". In *IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, San Francisco, California, USA, June 2006

5. Daniel Yagan and Chen-Khong Tham, "Coordinated Reinforcement Learning for Decentralized Optimal Control". In *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL)*, Hawaii, USA, April 2007

6. Daniel Yagan and Chen-Khong Tham, "Decentralized Optimal Control for Resource Allocation in Wireless Ad Hoc Networks", submitted to *IEEE International Conference on Computer Communication (INFOCOM)*, 2007

7. Daniel Yagan and Chen-Khong Tham, "Decentralized Optimal Control for Resource Allocation in Wireless Ad Hoc Networks". Journal version

8. Daniel Yagan and Chen-Khong Tham, "Stochastic Optimal Control and Performance Analysis of Wireless Ad Hoc Networks", submitted to *IEEE Transactions on Networking,* 2006 (under review)

9. Daniel Yagan and Chen-Khong Tham, "A Hierarchical Semi-MDP Approach for Adaptive QoS Provisioning in Ad Hoc Networks", submitted to *IEEE Transactions on Mobile Computing,* 2006 (under review)

10. Daniel Yagan and Chen-Khong Tham, "Adaptive Resource Allocation in Wireless Ad Hoc Networks: A Semi-MDP Approach", submitted to *IEEE Transactions on Mobile Computing*, 2006 (under review)

Bibliography

[1] D. Julian, M. Chiang, D. O'Neill, and S. Boyd, "QoS and Fairness Constrained Convex Optimization of Resource Allocation for Wireless Cellular and Ad Hoc Networks," in *Proc. IEEE INFOCOM'02*, New York, NY, USA, June 2002, pp. 477–486.

[2] M. Chiang, D. O'Neill, D. Julian, and S. Boyd, "Resource Allocation for QoS Provisioning in Ad Hoc Wireless Networks," in *Proc. IEEE GLOBECOM'01*, San Antonio, USA, Nov. 2001, pp. 2911–2915.

[3] Y. Qiu and P. Marbach, "Bandwidth Allocation in Ad Hoc Networks: a Price-Based Approach," in *Proc. IEEE INFOCOM'03*, vol. 2, San Francisco, CA, USA, Apr. 2003, pp. 797–807.

[4] M. L. Puterman, *Markov Decision Processes*. New York, USA: Wiley Interscience, 1994.

[5] L. Tassiulas and A. Ephremides, "Stability Properties of Constrained Queueing Systems and Scheduling policies for Maximum Throughput in Multihop Radio Networks," *IEEE Trans. Automat. Contr.*, vol. 37, no. 12, Dec. 1992.

[6] M. J. Neely, E. Modiano, and C. E. Rohrs, "Dynamic Power Allocation and Routing for Time Varying Wireless Networks," *IEEE J. Select. Areas Commun.*, vol. 23, no. 1, Jan. 2005.

[7] A. Eryilmaz, R. Srikant, and J. R. Perkins, "Stable Scheduling Policies for Fading Wireless Channels," *IEEE/ACM Trans. Networking*, vol. 13, no. 2, Apr. 2005.

[8] S. P. Meyn, "Algorithms for Optimization and Stabilization of Controlled Markov Chains," in *SADHANA (Proc. Indian Academy of Sciences, Engineering Sciences)*, vol. 24, India, Oct. 1999, pp. 339–368.

[9] M. Neely, E. Modiano, and C. P. Li, "Fairness and Optimal Stochastic Control for Heterogenous Networks," in *Proc. IEEE INFOCOM'05*, Miami, Florida, USA, Mar. 2005.

[10] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Belmont, MA, USA: Athena Scientific, 1996.

[11] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[12] A. Fu, E. Modiano, and J. Tsitsiklis, "Optimal Energy Allocation for Delay-Constrained Data Transmission over a Time-Varying Channel," in *Proc. IEEE INFOCOM'03*, San Francisco, CA, USA, Mar. 2003.

[13] F. Yu, V. Wong, and V. Leung, "A New QoS Provisioning Method for Adaptive Multimedia in Cellular Wireless Networks," in *Proc. IEEE INFOCOM'04*, Hong Kong, China, Mar. 2004, pp. 2130–2141.

[14] C. K. Tham, "Online Function Approximation for Scaling Up Reinforcement Learning," Ph.D. dissertation, Univ. of Cambridge, UK, Oct. 1994.

[15] J. N. Tsitsiklis and B. V. Roy, "An Analysis of Temporal-Difference Learning with Function Approximation," *IEEE Trans. Automat. Contr.*, vol. 42, no. 5, pp. 674–690, May 1997.

[16] A. G. Barto and S. Mahadevan, "Recent advances in Hierarchical Reinforcement Learning," *Discrete-Event Dynamical Systems: Theory and Applications*, vol. 13, pp. 341–379, 2003.

[17] C. J. C. H. Watkins and P. Dayan, "Q-Learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[18] D. Yagan and C. K. Tham, "Adaptive QoS Provisioning in Wireless Ad Hoc Networks: A Semi-MDP Approach," in *Proc. IEEE WCNC'05*, New Orleans, Louisiana, USA, Mar. 2005.

[19] D. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The Complexity of Decentralized Control of Markov Decision Process," *Mathematics of Operations Research*, vol. 27, no. 4, pp. 819–840, 2002.

[20] M. Yokoo and K. Hirayama, "Distributed breakout algorithm for sovling distributed constraint satisfaction problems," in *Proc. ICMAS*, 1996, pp. 401–408.

[21] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo, "An asynchronous complete method in distributed constraint optimization," in *Proc. 2nd International Conf. on Autonmous Agents and Multi-Agent Systems*, Sydney, Australia, 2003.

[22] V. S. Borkar and S. P. Meyn, "The O.D.E. Method for Convergence of Stochastic Approximation and Reinforcement Learning," *SIAM Journal on Control and Optimization*, vol. 38, no. 2, pp. 447–469, 2000.

[23] S. P. Meyn and R. L. Tweedie, *Markov Chains and Stochastic Stability.* New York, USA: Springer-Verlag, 1993.

[24] S. P. Meyn, "The Policy Iteration Algorithm for Average Reward Markov Decision Processes with General State Space," *IEEE Trans. Automat. Contr.*, vol. 42, pp. 1663–1680, 1997.

[25] R. R. Chen and S. P. Meyn, "Value Iteration and Optimization of Multiclass Queueing Networks," *Queueing Systems*, vol. 32, pp. 65–97, 1999.

[26] M. Neely, E. Modiano, and C. Rohrs, "Dynamic Power Allocation and Routing in Time Varying Wireless Networks," in *Proc. IEEE INFOCOM'03*, San Francisco, CA, USA, Mar. 2003.

[27] L. Tassiulas, "Scheduling and Performance Limits of Networks with Constantly Changing Topology," *IEEE Trans. Inform. Theory*, vol. 43, no. 3, May 1997.

[28] A. Konrad, B. Y. Zhao, A. D. Joseph, and R. Ludwig, "A Markov-based Channel Model Algorithm for Wireless Networks," in *Proc. of the 4th Int'l Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems (MSWIM 2001)*, Rome, Italy, July 2001.

[29] M. Neely, "Dynamic Power Allocation and Routing for Satellite and Wireless Networks with Time Varying Channel," Ph.D. dissertation, Massachusetts Institute of Technology, US, Nov. 2003.

[30] K. Fall and K. Varadhan, editors. The ns manual. The VINT Project, UC Berkeley, LBL, USC/ISI and XEROX PARC. [Online]. Available: http://www.isi.edu/nsnam/ns/ns-documentation.html

[31] C. Perkins. (2003, Nov.) Ad Hoc On-Demand Distance Vector Routing. Internet draft. [Online]. Available: http://tools.ietf.org/html/draft-ietf-manet-aodv-13.txt

[32] G. Bianchi, "Performance Analysis of the IEEE 802.11 Distributed Coordination Function," *IEEE J. Select. Areas Commun.*, vol. 18, no. 3, pp. 535–547, 2000.

[33] Y. Barkowski, S. Biaz, and P. Agrawal, "Towards the Performance Analysis of IEEE 802.11 in Multihop Ad Hoc Networks," in *Proc. of MobiCom*, Philadelphia, PA, USA, Sept. 2004.

[34] A. Gosavi, "Reinforcement Learning for Long-Run Average Cost," *European Journal of Operational Research*, vol. 155, pp. 654–674, 2004.

[35] C. Darken, J. Chang, and J. Moody, "Learning Rate Schedules for Faster Stochastic Gradient Search," in *Proc. IEEE Workshop on Neural Networks for Signal Processing*, Sept. 1992.

[36] J. Bennett and H. Zhang, "WF^2Q : Worst-Case Fair Weighted Fair Queueing," in *Proc. of IEEE INFOCOM'96*, vol. 1, San Francisco, CA, USA, Mar. 1996, pp. 120–128.

[37] C. Pandana and K. J. R. Liu, "Near-Optimal Reinforcement Learning Framework for Energy-Aware Sensor Communications," *IEEE J. Select. Areas Commun.*, vol. 23, no. 4, Apr. 2005.

[38] A. A. Borovkov, *Ergodicity and stability of stochastic processes.* New York, USA: Wiley, 1998.

[39] M. Neely, "Energy Optimal Control for Time Varying Wireless Networks," in *Proc. IEEE INFOCOM'05*, Miami, Florida, USA, Mar. 2005.

[40] ——, "Optimal Energy and Delay Tradeoffs for Multi-User Wireless Downlinks," in *Proc. IEEE INFOCOM'06*, Barcelona, Spain, Apr. 2006.

[41] ——, "Super-Fast Delay Tradeoffs for Utility Optimal Fair Scheduling in Wireless Networks," in *Proc. IEEE INFOCOM'06*, Barcelona, Spain, Apr. 2006.

[42] M. Ghavamzadeh and S. Mahadevan, "Hierarchical average reward reinforcement learning," Department of Computer Science, Univ. of Massachusetts, Amherst, MA, Tech. Rep. UM-CS-2003-19, 2003.

[43] S. Mahadevan, "Average Reward Reinforcement Learning: Foundations, Algorithms, and Empirical Results," *Machine Learning*, vol. 22, no. 1-3, pp. 159–195, 1996.

[44] A. Gosavi, *Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning.* Kluwer Academic Publishers, May 2003.

[45] J. Liebeherr and N. Christin, "JoBS: Joint Buffer Management and Scheduling for Differentiated Services," in *Proc. IEEE/IFIP Int'l Workshop on Quality of Service*, Karlsruhe, Germany, June 2001, pp. 404–418.

[46] E. Altman, *Constrained Markov Decision Processes.* Chapman and Hall/CRC, 1999.

[47] Z. Gabor, Z. Kalmar, and C. Szepesvari, "Multi-criteria Reinforcement Learning," in *Proc. International Conf. on Machine Learning*, Madison, WI, July 1998, pp. 197–205.

[48] J. Santamaria, R. S. Sutton, and A. Ram, "Experiments with Reinforcement Learning in problems with continuous state and action spaces," *Adaptive Behavior*, vol. 6, no. 2, pp. 163–217, 1998.

[49] R. Sutton, "Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding," in *Advances in Neural Information Processing Systems*, vol. 8. The MIT Press, 1996, pp. 1038–1044.

[50] C. Gaskett, "Q-Learning for Robot Control," Ph.D. dissertation, Robotic Systems Laboratory, Research School of Information Sciences and Engineering, Australian National University, 2002.

[51] L. Baird and A. Klopf, "Reinforcement Learning with High-dimensional, Continuous Actions," Wright-Patterson Air Force Base Ohio: Wright Laboratory, Tech. Rep. WL-TR-93-1147, 1993.

[52] N. Christin, J. Liebeherr, and T. Abdelzaher, "A Quantitative Assured Forwarding Service," in *Proc. of IEEE INFOCOM'02*, vol. 2, New York, NY, June 2002, pp. 864–873.

[53] R. Sutton, D. Precup, and S. Singh, "Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, pp. 181–211, 1999.

[54] R. Parr, "Hierarchical Control and Learning of Markov Decision Process," Ph.D. dissertation, University of California, Berkeley, 1998.

[55] T. Dietterich, "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition," *Journal of Artificial Intelligence Research*, vol. 13, pp. 227–303, 2000.

[56] M. Ghavamzadeh and S. Mahadevan, "Hierarchically Optimal Average Reward Reinforcement Learning," in *Proc. 19th International Conf. on Machine Learning*, Sydney, Australia, July 2002, pp. 195–202.

[57] B. Hengst, "Discovering Hierarchy in Reinfocement Learning with HEXQ," in *Proc. International Conf. on Machine Learning*, San Franscisco, CA, July 2002, pp. 243–250.

[58] O. Tickoo and B. Sikdar, "Queueing Analysis and Delay Mitigation in IEEE 802.11 Random Access MAC based Networks," in *Proc. IEEE INFOCOM'04*, Hongkong, China, Mar. 2004.

[59] D. Yagan and C. K. Tham, "Stochastic Optimal Control and Performance Analysis of Wireless Ad Hoc Networks," *IEEE/ACM Trans. Networking*, submitted for publication.

[60] M. Littman, "Markov Games as a framework for Multi-Agent Reinforcement Learning," in *Proc. International Conf. on Machine Learning*, New Brunswick, NJ, 1994, pp. 157–163.

[61] J. Baxter and P. Bartlett, "Infinite-Horizon Policy Gradient Estimation," *Journal of Artificial Intelligence Research*, vol. 15, pp. 319–350, 2001.

[62] D. Aberdeen and J. Baxter, "Scaling Internal-State Policy-Gradient Methods for POMDPs," in *Proc. 19th International Conf. on Machine Learning*, Sydney, Australia, July 2002, pp. 1–12.

[63] D. Aberdeen, "Policy-Gradient Algorithms for Partially Observable Markov Decision Process," Ph.D. dissertation, Australian National University, Australia, Apr. 2003.

[64] A. Cassandra, M. Littman, and N. Zhang, "Incremental Pruning: A simple, fast, exact method for Partially Observable Markov Decision Process," in *Proc. of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI–97)*, San Francisco, CA, 1997, pp. 54–61.

[65] T. Jaakkola, S. Singh, and M. Jordan, "Reinforcement Learning Algorithm for Partially Observable Markov Decision Process," in *Advances in Neural Information Processing Systems*, vol. 7, 1995, pp. 345–352.

[66] O. Madani, S. Hanks, and A. Condon, "On the undecidability of probabilistic planning and infinite-horizon Partially Observable Markov Decision Process problems," in *Proc. of the 16th National Conference on Artificial Intelligence*, 1999, pp. 541–548.

[67] E. Hansen, D. Bernstein, and S. Zilberstein, "Dynamic Programming for Partially Observable Stochastic Games," in *Proc. of the 19th National Conference on Artificial Intelligence*, 2004, pp. 709–715.

[68] E. Sondik, "The Optimal Control of Patirally Observable Markov Process," Ph.D. dissertation, Standford University, US, 1971.

[69] B. Bonet, "An ϵ-optimal grid-based algorithm for Partially Observable Markov Decision Processes," in *Proc. 19th International Conf. on Machine Learning*, C. Sammut and A. Hoffmann, Eds., Sydney, Australia, 2002, pp. 51–58.

[70] R. Nair, M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella, "Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings," in *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, 2003.

[71] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo, "Networked distributed POMDPs: A synthesis of Distributed Constraint Optimization and POMDPs," in *Proc. of the 20th National Conference on Artificial Intelligence*, 2005, pp. 133–139.

[72] S. Sivavakeesar and G. Pavlou, "Associativity-based Stable Cluster Formation in Mobile Ad hoc Networks," in *Proc. IEEE CCNC'05*, Las Vegas, USA, Jan. 2005.

[73] ——, "Cluster-based Location Services for Scalable Ad hoc Network Routing," in *Proc. IEEE MWCN'04*, Paris, France, Oct. 2004.

[74] D. Szer and F. Charpillet, "An Optimal Best-First Search Algorithm for Solving Infinite Horizon DEC-POMDPs," in *European Conf. on Machine Learning*, ser. Lecture Notes in Computer Science, vol. 3720. Springer, 2005.

[75] S. Singh, V. Tadic, and A. Doucet, "A Policy Gradient Method for Semi-Markov Decision Processes with Application to Call Admission Control," Cambridge University, Tech. Rep. CUED/F-INFENG/TR-523, 2005.

[76] V. Tadic and S. P. Meyn, "Asymptotic properties of two time-scale stochastic approximation algorithms with constant step sizes," in *Proc. American Control Conference*, June 2003.

[77] H. Yu, "A Function Approximation Approach to Estimation of Policy Gradient for POMDP with Structured Policies," in *Proc. of the 21th Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)*, Arlington, Virginia, 2005, pp. 642–64.

[78] D. J. Goodman and N. B. Mandayam, "Power Control for Wireless Data," *IEEE Personal Commun. Mag.*, vol. 7, no. 2, Apr. 2000.

[79] M. C. L. Chen, S. H. Low and J. C. Doyle, "Jointly Optimal Congestion Control, Routing, and Scheduling for Wireless Ad Hoc Networks," in *Proc. IEEE INFOCOM'06*, Barcelona, Spain, Apr. 2006.

[80] R. T. Maheswaran, J. P. Pearce, and M. Tambe, "Distributed Algorithms for DCOP: A Graphical-Game Based Approach," in *International Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS) Workshop on Challenges in the Coordination of Large-Scale Multi-Agent Systems*, New York, NY, July 2004.

[81] J. P. Pearce, R. T. Maheswaran, and M. Tambe, "Solution Sets for DCOPs and Graphical Games," in *International Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, Hakodate, Japan, May 2006.