SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ingegneria

Fault-tolerant routing in Next Generation Home Networks

Relatore

Chiar.mo Prof. Francesco Delli Priscoli

Dottorando:

Ing. Marco Castrucci

Coordinatore:

Chiar.mo Prof. Carlo Bruni

Dottorato di ricerca in Ingegneria dei Sistemi *XXII Ciclo*

Index

List of Figure

List of Tables

List of Acronyms

ARM	Architecture Reference Model
ATM	Asynchronous Transfer Mode
B-NT	Broadband Network Termination
CPE	Consumer Premise Equipment
DHCP	Dynamic Host Configuration Protocol
DP	Dynamic Programming
DSL	Digital Subsriber Line
ETSI	European Telecommunication Standardization Institute
EUT	End User Terminal
FPD	Functional Processing Device
FT	Fault-Tolerant
GPI	Generalized Policy Iteration
HDTV	High Definition Television
HGI	Home Gateway Initiative
HO	Hand-Over
HWO	Hybrid Wireless Optic
IGMP	Internet Group Management Protocol
I-MAC	Inter-MAC
IP	Internet Protocol
ITU	International Telecommunication Union
LAN	Local Area Network
MAC	Medium Access Control
MC	Markovian Chain
MDP	Markov Decision Process
NAT	Network Address Tranlsation
NGHN	Next Generation Home Network
NT	Network Termination
OMEGA	hOME Gigabit Access
PLC	Power Line Communication
PPP	Point-to-Point Protocol
P2P	Peer To Peer

QoE	Quality of Experience
QoS	Quality of Service
RL	Reinforcement Learning
TA	Terminal Adapter
TD	Temporal-Difference learning
TR	Techical Report
VoD	Video on Demand
VoIP	Voice over IP
UBB	Ultra-BroadBand
UT	User Terminal
UWB	Ultra Wide Band
WAN	Wide Area Network
WiFi	Wireless Fidelity
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network

Chapter I
Introduzione

Le reti di nuova generazione, attualmente in fase di studio e di standardizzazione presso i più importanti forum pubblici e privati mondiali come ETSI [46], ITU-T [47], WiMAX Forum [48], BroadBand Forum [49] e Home Gateway Initiative [50], introducono dei concetti di fondamentale interesse per l'ingegneria dei sistemi e del controllo, considerando che le reti impiegate diventano sempre più complesse, a causa di tecnologie eterogenee che necessariamente devono interoperare, ma che allo stesso tempo devono poter essere gestite in modo efficiente. Per questo motivo i modelli di rete moderni propongono un approccio in cui il mondo delle telecomunicazioni è suddiviso in tre piani principali: il *data plane*, che gestisce i flussi dati e le applicazioni degli utenti, il *control plane*, che include le funzionalità preposte ad eseguire un controllo in real-time della rete, e il *management plane*, che comprende le funzionalità dedicate alla configurazione a lungo termine e il fault management della rete.

Gli obiettivi della presente tesi sono stati molteplici:

- definire un modello di architettura per reti domestiche in grado di rendere possibile l'interoperabilità di diverse tecnologie trasmissive;
- definire il modello di riferimento per un sistema di controllo dell'instradamento dei flussi applicativi nelle reti domestiche multi-tecnologia, in grado di gestire diversi requisiti di Qualità di Servizio (QoS) e le frequenti indisponibilità dei percorsi;
- definire un algoritmo di controllo basato sul modello di riferimento proposto;
- validare il sistema di controllo proposto in un adeguato scenario simulativo.

Il primo obiettivo è stato raggiunto attraverso la definizione di un modello architetturale di riferimento per reti domestiche multi-tecnologia [12]. A tale scopo, il lavoro è consistito nella raccolta e l'analisi dei requisiti, la definizione delle specifiche, il design dettagliato dei moduli funzionali e delle interfacce. Tale architettura apre innumerevoli possibilità di applicazione delle teorie del controllo e dell'ottimizzazione alla gestione e il controllo delle reti domestiche (campo ancora poco esplorato per la mancanza di un adeguata architettura di rete come quella proposta) quali il controllo di ammissione e il controllo dell'instradamento dei flussi accettati.

Il lavoro si è quindi focalizzato sulla progettazione di un sistema di controllo per gestire l'instradamento dei flussi nella rete, tenedo conto dei requisiti di QoS dei flussi stessi. A tal fine è stato utilizzato un approccio basato sulla teoria del *Markov Decision Process* (MDP). Il sistema è stato quindi modellato come una catena di Markov, per la quale sono stati definiti gli stati e le transizioni tra i diversi stati. Tale approccio è stato scelto in quanto già applicato con successo per il controllo dell'instradamento di flussi informativi, anche se fino ad oggi era stato applicato in reti *core*, mentre non era mai stato considerato per essere applicato in reti domestiche. Inoltre, i particolari requisiti del contesto considerato hanno reso necessarie notevoli modifiche ai modelli proposti in letteratura, portando quindi alla definizione di un nuovo modello di rete: in pratica, è stato necessario definire nuovi stati e nuove transizioni. L'approccio usato si è rivelato una buona soluzione nell'ambiente domestico di applicazione, soggetto a numerosi e ripetuti *link fault*. Il controllore è stato infatti progettato con l'obiettivo di minimizzare il numero di re-instradameni dei flussi dovuti all'improvvisa indisponibilità di un percorso usato da flussi già attivi.

Una volta terminata la fase di modellazione del sistema e di progettazione del controllore, sono emersi problemi di scalabilità della soluzione proposta, che la rendono inapplicabile in contesti reali con requisiti di real-time. Nonostante ciò, tale lavoro costituisce un prezioso modello teorico di riferimento che può essere usato per sviluppare algoritmi implementabili.

A dimostrazione di ciò è stato quindi definito un algoritmo basato sulla teoria del *Reinforcement Learning*. In particolare è stato definito un algoritmo di *Q-learning* in grado di apprendere la scelta ottimale a seconda dello stato del sistema.

Per verificare le performance dell'algoritmo proposto sono state effettuate numerose simulazioni utilizzando il software di simulazioni MATLAB. I risultati delle simulazioni hanno permesso di verificare come l'algoritmo di Q-learning proposto consente di ottenere risultati prossimi a quelli ottenibili applicando l'algoritmo ottimo basato su MDP.

Attualmente, l'algoritmo proposto è in fase di implementazione al fine di essere integrato nel prototipo in fase di sviluppo presso i laboratori di ricerca di France Telecom, nell'ambito del progetto OMEGA, finanziato dalla Commissione Europea.

Nel dettaglio, il presente lavoro si articola in 7 capitoli.

Il presente *Capitolo 1* fornisce un'introduzione al fine di offrire una visione completa di tutto il lavoro svolto.

Il *Capitolo 2* presenta una panoramica sulle reti domestiche, al fine di descrivere il contesto del presente lavoro. In particolare, viene descritta l'evoluzione delle reti domestiche dalla loro nascita a oggi, attraverso la riproposizione dei modelli di rete che si sono succeduti nel corso degli anni e frutto del lavoro di ricerca a livello mondiale e presso gli enti di standardizzazione. Alla fine del capitolo viene inoltre presentato il modello di rete domestica del futuro, in cui diverse tecnologie trasmissive verranno utilizzate contestualmente per offrire maggiori capacità e quindi servizi a valore aggiunto agli utenti. In particolare il modello presentato è stato il frutto della fase iniziale del presente lavoro e definisce un innovativo modello di rete che dà una grande rilevanza al piano di controllo della rete stessa. Questo perché le potenzialità delle nuove reti domestiche potranno essere utilizzate in maniera efficiente solo se opportunamente controllate in maniera automatica e in real-time da appositi protocolli e algoritmi di controllo. Tra questi, viene definito il problema dell'instradamento di flussi in rete (routing), con i suoi obiettivi e le sue caratteristiche nell'innovativo scenario proposto.

Il *Capitolo 3* e il *Capitolo 4* forniscono le basi teoriche per l'algoritmo oggetto del presente lavoro. In particolare, l'obiettivo del Capitolo 3 è quello di presentare gli strumenti teorici, messi a disposizione nell'ambito del controllo stocastico, relativamente alle catene di Markov e ai processi decisionali di Markov. Tali strumenti sono stati utilizzati come approccio fondamentale per la definizione del problema di routing. Nel Capitolo 4 viene invece fornita un'introduzione al Reinfocement Learning (RL) e vengono presentati alcuni metodi di soluzione.

Il *Capitolo 5* è il capitolo principale della tesi, in quanto contiene la descrizione dell'algoritmo proposto. In particolare il capitolo è suddiviso in due parti. Nella prima parte viene descritto l'approccio utilizzato per modellare il problema di routing in reti domestiche di prossima generazione attraverso un processo decisionale di Markov. Tale lavoro costituisce un risultato di grande rilievo in quanto determina una base teorica per lo sviluppo di altri algoritmi che potranno essere progettati utilizzando diverse metodologie. In aggiunta, tale frame work potrà essere utilizzato come punto di riferimento per la valutazione delle prestazioni dei diversi algoritmi sviluppati. Nella seconda parte del capitolo viene invece proposto un particolare algoritmo di routing, progettato facendo uso delle metodologie di Reinforcement Learning e costruito sulla base del riferimento teorico descritto nella prima parte del capitlo.

Nel *Capitolo 6* sono presentati alcuni risultati numerici ottenuti attraverso la realizzazione di una serie di simulazioni eseguite con il software MATLAB. Inizialmente viene descritto lo scenario simulativo, rappresentante una rete domestica di nuova generazione e realizzata attraverso l'integrazione di 4 tecnologie trasmissive diverse. In seguito vengono presentati i risultati ottenuti attraverso la simulazione del processo decisionale di Markov, utilizzando tre politiche diverse. I risultati ottenuti mostrano le potenzialità dell'algoritmo proposto e la sua capacità di supportare flussi con diverse caratteristiche e requisiti e quindi appartenenti a diverse Classi di Servizio. Infine vengono presentati i risulati ottenuti attraverso la simulazione dell'algoritmo basato su Reinforcement Learning. I risulati mostrano come le prestazioni di questo algoritmo si avvicinano all'ottimo ottenibile co il processo decisionale di Markov.

Infine, nel *Capitolo 7*, vengono riportate le conclusioni del lavoro e viene fornita un'indicazione dei prossimi sviluppi di questo algoritmo, sia dal punto di vista teorico che implementativo.

Chapter II
Overview of home networking

II.1. Introduction

Home networking is defined by the CEA-HNIT's Board of Directors as follows: "*A home network interconnects electronic products and systems, enabling access and control of this products and systems, and any available content such as music, video or data*" [1]. Products need to be connected to each other; access to content (e.g., entertainment, information, services) must be provided; and the user must have control of the products and the distribution of content. Content may come from within the home, from a media centre hard disk, a personal video recorder, and so on; or remotely from somewhere outside the home (e.g., form a Wide Area Network that provide connection to Internet). One point that must be emphasized is the ease of use. The consumer should not even know a home network is being established in its home. Consumers buy applications, not home networks [2].

The significant interest in home networking today stems from the availability of low-cost communication technologies and from the need for network operators and service providers to overcome bandwidth limitations occourring today in home networks, that limit the diffusion and the provisioning of added-value services to users.

In this chapter is presented the evolution of home networks, starting from simple old implementation to the vision and the idea behind the next generation of home networks that nowadays are being object of worldwide research. The chapter ends with the presentation of the open research topics related to next generation home networks, highlitghing how the present work intends to provide a solution to one of the most important open issue.

II.2. Home network evolution

This section describes the evolution of the trends in architecture of home networking during the ten last years through standardization as well as practical implementations. Three different home networks model are presented: the ITU-T model, the DSL-Forum model and the triple play model. In addition to those presented here, there are also other standardization bodies and industrial forum (e.g. Home Gateway Initiative (HGI)) at European and international level that are working to define guidelines and standards for home networks.

II.2.1 ITU-T model

The first elaboration of models of home networking date back to the nineties, with the ITU-T efforts to standardize recommendations for digital subscriber lines. ITU-T 995.1 [3], for instance, introduces in 2001 the following entities:

- the NT1, terminating the access digital section of the broadband connection,
- the NT2, terminating the transport protocol for user traffic. It may implement switching/routing functions,
- the Terminal Adapter (TA), adapting the transport protocol to the specific requirements of a user terminal,
- the User Terminal, providing an interface for the user.

These entities are interconnected by interfaces (R, S, T, U) defined by the following representation:

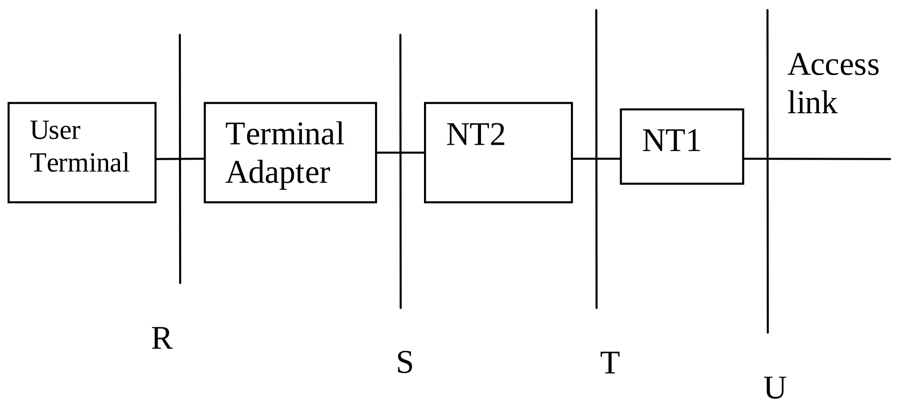


Figure 1 - The ITU-T 995.1 architecture for home networking

II.2.2 DSL-Forum model

In 2004, the DSL Forum defined in TR-094 [6] requirements and capabilities that a home network should provide to take advantage of the full capabilities of the multi services broadband access. TR-094 introduces in particular the following entities:

- the B-NT (Broadband Network Termination),
- the Routing Gateway,
- the Premises Distribution (client infrastructure),
- the FPD (Functional Processing Device), which is a component of the home network that processes voice, video or data for its intended application,
- the EUT (End User Terminal).

These entities are interconnected by interfaces (R, $T_{CN,}$ $T_{PDN,}$ U) defined by the following representation:

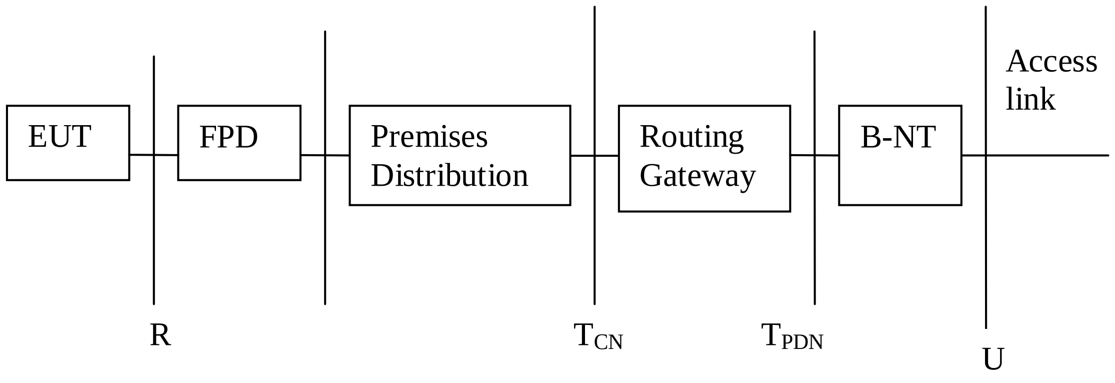


Figure 2 - The DSL Forum TR-094 architecture for home networking

The R interface is the type of interface that the FPD should support in order to provide connectivity to the EUT. The T_{CN} interface defines the interface between the Routing Gateway and the various premises distribution technologies. The T_{PDN} interface is physically discernable when the B-NT and Routing Gateway are implemented in separate devices: it is practically limited to a point to point layer 1+2 connection. The U interface is represented here in making abstraction of a possible splitter. Some entities of these representations can be merged into one single equipment, for instance the EUT and the FPD, or the Routing Gateway and the B-NT.

II.2.3 Triple play model

The triple play model [7] was adopted by the operators around 2004 in order to launch commercial offers based on three service components: the Internet, the conversational (VoIP, videophony) and the TV services. These offers were often based on a residential gateway with physical ports each dedicated to a specific service, which allows to simplify the implementation. The following figure gives an example (hybrid between the bridged model and the routed model) of such a gateway:

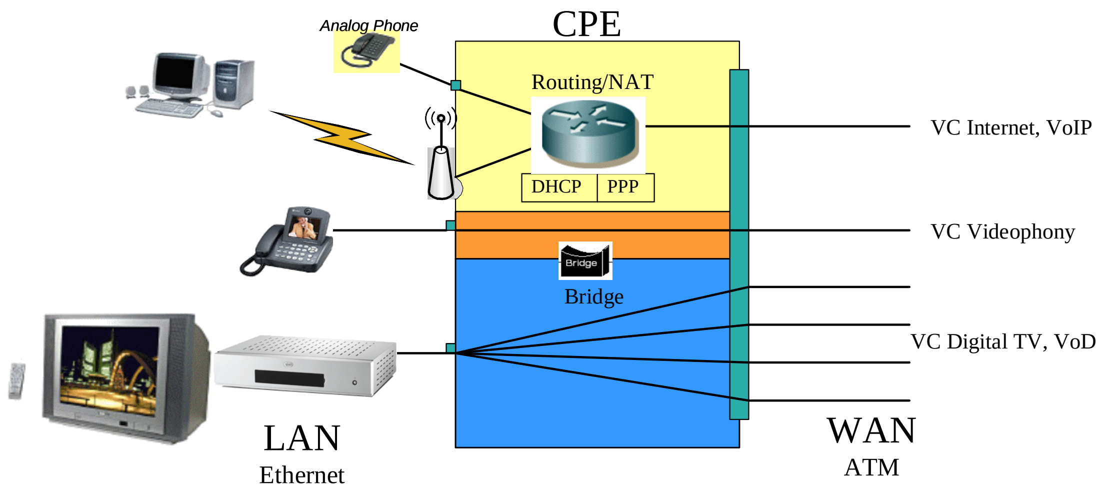


Figure 3 - The hybrid bridged/routed triple play architecture

Such an option leads naturally to an organization of the home network where a given technology is dedicated to a given service. In a longer term prospect, that architecture will likely evolve to a more flexible configuration, based on a full-routed solution, which would avoid the separation between the services and the constraint to connect each device to a given port, as showed on the following figure:

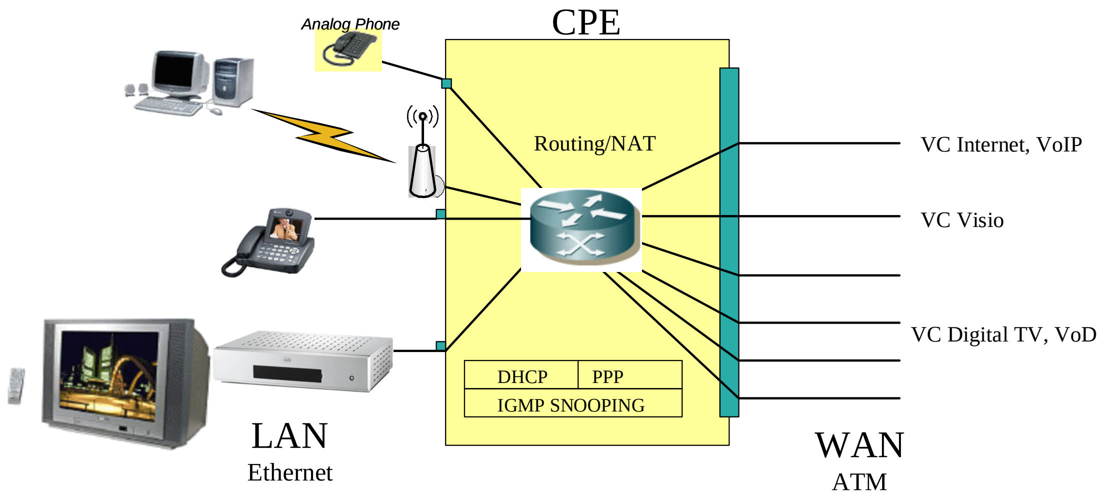


Figure 4 - The full-routed triple play architecture

II.3. Next Generation Home Networks

During last years, communication technology has evolved in terms of services diversification. Requests of different advanced services lead to a mass-market of a variety of devices and networks supporting heterogeneous and broadband technologies. Several solutions have been deployed to provide broadband and heterogeneous connectivity to users, especially in the access networks. But the diffusion of new bandwidth demanding services (like HDTV) will be possible only when technology limitations will be eliminated from the real network bottleneck: the users home area network.

Several technologies are nowadays adopted in home networks. Despite this diversity, it is possible to group all these technologies in two main categories, depending on the communication medium to be used: wired communications and wireless communications. Inside this two big clusters further distinctions take place. Among wired communications, Ethernet (IEEE 802.3) is for sure the most common technology used to interconnect different devices in a home environment. But in the last years the attention and the research is focusing on Power Line Communications (PLC), an emerging technology which uses power supply to convey the information through the network. As such as concerns wireless communication, technologies like Wi-Fi (802.11a/b/g) have already been exploited and new standards such as Wi-Max (802.16) or UltraWideBand (802.15.3) are actually contending the attention of the people. An emerging technology within wireless communications is the Free Space Optics technology, which introduce the concept of wireless infrared and visible light communications, alternative to the wireless radio frequency medium. As a consequence of this large diversification, many different networks have emerged inside the domestic ambient, causing de facto the impossibility to make interact devices connected to the network with different technologies.

A lot of research works has been done on convergence, and most of them propose to enhance terminals and network components with technology independent middleware frameworks. This is a good solution but not in a home gigabit access network. A middleware solution is not suitable for terminal capabilities, it requires in the most of cases to be installed, configured and maintained by the terminal user and can process only low data rate services. Services like HDTV, Broadband Internet access, on-line 3D gaming are extremely expensive in terms of resources. Therefore, these services requires a disruptive approach for the management of the resources in a so called Next Generation Home Network (NGHN), where convergence should be achieved maintaining simplicity, scalability and backward compatibility.

In Europe, the FP7 OMEGA project ([9]) is defining and prototyping a new architecture for home networks able to achive the above mentioned objectives. In particolar, OMEGA is proposing an innovative protocol stack for home networks where a new layer is introduced between the MAC layer and the IP layer: the so called Inter-MAC layer ([10]). It receives and processes the information from the upper layers (IP) in order to match the services requests with the availability offered by the various underlying technology dependent MACs. The Inter-MAC (see Figure 5) is technology independent and controls multiple technology networks by means of proper adapters. It also provides services as well as connectivity to all the devices in the house. Thanks to the introduction of the Inter-MAC layer, it is possible to obtain convergence inside the home among several heterogeneous Telecommunication technologies, thus paving the way to the possibility to achieve home network capacities od the order of Gigabit per second.

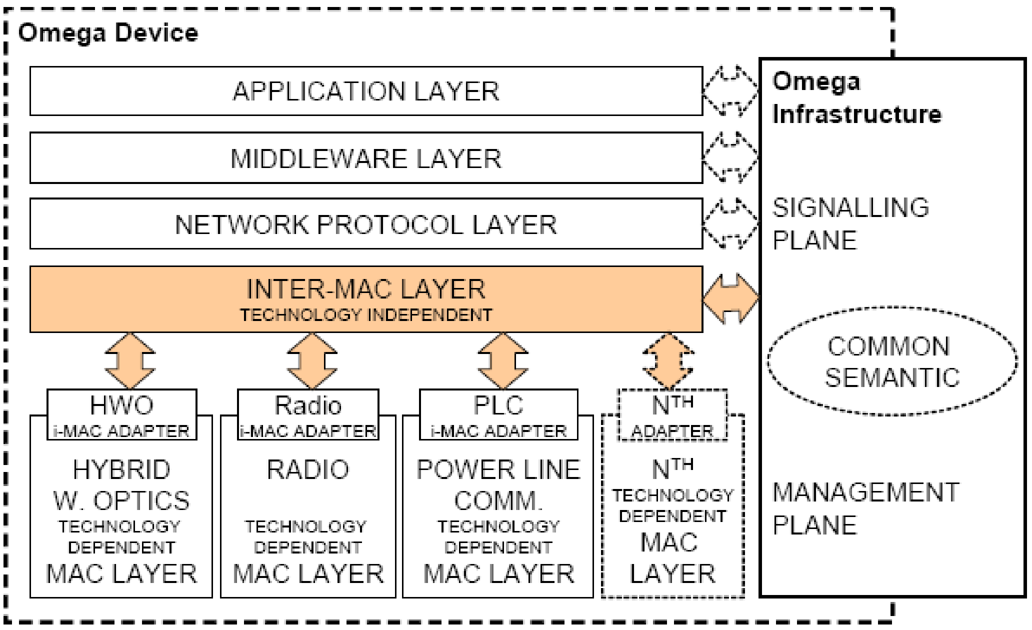


Figure 5 - Inter-MAC reference architecture

In Figure 6 a functional architecture is presented, where it is possible to distinguish the interfaces by which the Inter-MAC communicates with Network protocol layer, the technology-dependent MAC layers, and with the signalling and management plane; three main Inter-MAC functionalities are wrapped by a Monitoring & Event Manager. It handles the decision to enforce taking in input the information coming from the Signalling and Management Plane.

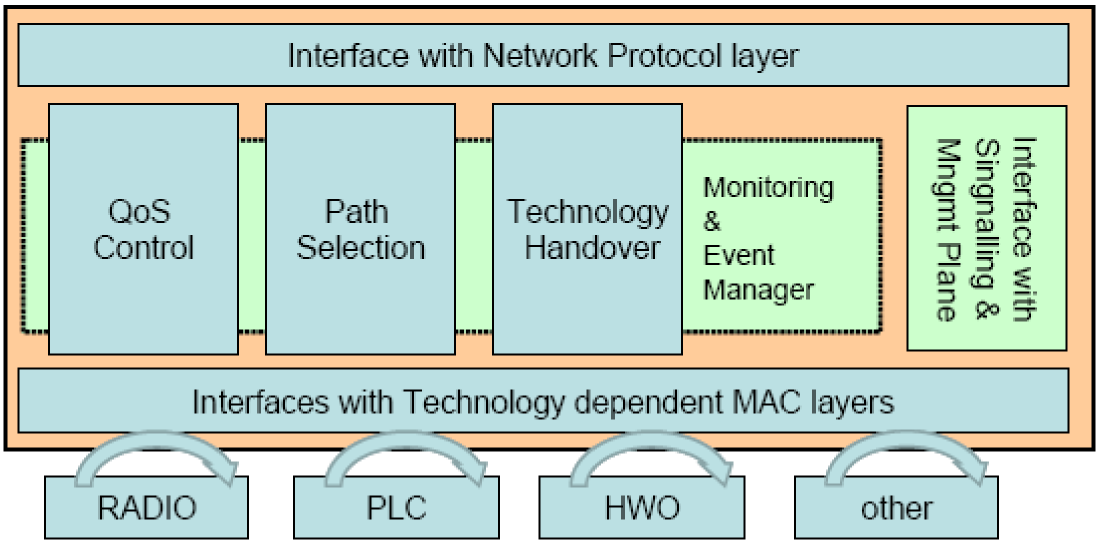

Interface with Network Protocol layer
QoS Control
Path Selection
Technology Handover
Monitoring & Event Manager
Interface with Singnalling & Mngmt Plane
Interfaces with Technology dependent MAC layers
RADIO
PLC
HWO
other

Figure 6 - Inter-MAC functional architecture

Each one of the functional components previously mentioned are described in detail:

- **QoS Control**: it manages the resource allocation of specific flows guaranteeing some QoS parameters: Bandwidth, Delay, Delay Variation, Loss Ratio and Error Ratio. Different classes of service can be handles by the Inter-MAC and the QoS Control then performs a complete scan over all MACs to estimate which of them can handle the specific flow belonging to that class of service.
- **Path Selection**: select all the possible paths to connect two or more nodes among various networks. It considers multi-hop solutions and take care of load balancing tecniques. Load balancing is needed whenever the QoS parameters could not be assured using only one available path. Path selection is a functionality strongly interconnected with QoS control. Existing solutions for multi-hop routing are tailored for homogenous networks and thus not suited for the heterogeneous home gigabit architecture. Implementing multi-hop connection in the home heterogeneous environment is novel, will be undertaken by this functional component, and is the objective of this work. In general, path selection can consider factors including class-of-service identification, policy-

based routing table derivation, dynamic bandwidth allocation, protection, reservation, priority routing, and priority queuing.

- **Technology Handover**: in order to provide access to different communications systems an efficient vertical handover mechanism is required. A technique that uses the common semantic to describe the available channels and chooses between them will be developed. The technology handover switches between two different technologies and is recalled whenever a network congestion, link failure or device mobility occur.
- **Monitoring & Event Manager**: it represents the link-up point for the functionalities described above. Its task is to trigger decisions, based on Signalling & Management Plane information. Since every Inter-MAC functionality is related to each other, if Monitoring & Event Manager detects that a particular link of the Home Network cannot support the service class imposed by QoS Control, then it will trigger Path Selection module in order to choose a better link. So, information produced from monitoring and event manager will be used by Inter-MAC to cast its main functionalities: Qos Control, Technology Handover and Path Selection.

The Next Generation Home Network Architecture Reference Model (ARM) has been then designed to fulfil the following conditions:

- it should be elaborated in the continuation of models already elaborated in standardization and currently used in the domain of home networking architecture;
- it should provide a good comprehension of the bounds of the network;
- it should clarify the internal and external interfaces of the network;
- it should highlight the structure of the NGHN into elementary network functionalities and capabilities.

In the prospect of the Gigabit data rate in Next Generation Home Networks, it appears interesting to distribute the functions of connectivity inside the home with the help of interconnection points spread in the home, and achieving the hybridization of several different wired and wireless access technologies through the introduction of the Inter-MAC layer described above. This scenario is illustrated in the following picture:

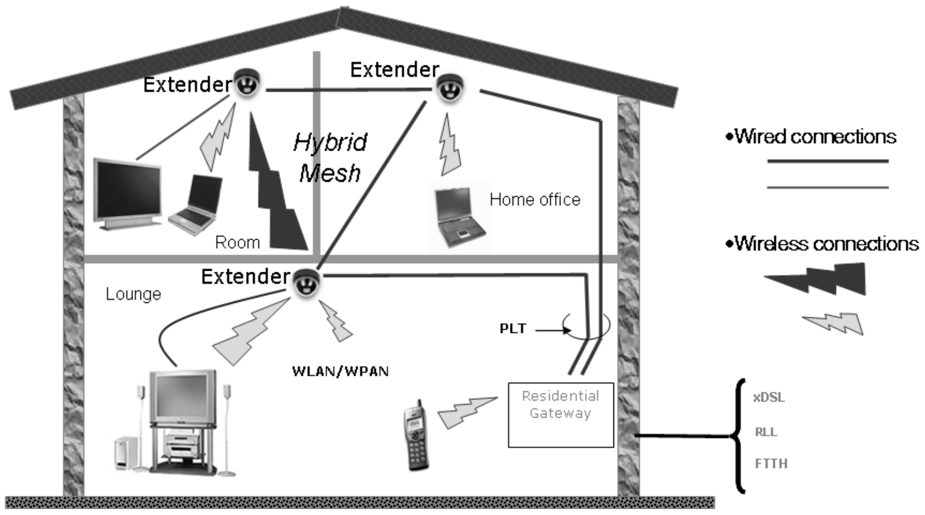


Figure 7 - Hybridization of technologies inside the home network

This illustration highlights the interconnection of a wide range of terminals with a mesh network ensuring the coverage of the whole home area. These terminals can be classified in families or clusters, not completely disjoint:

- data communication terminals (computers, PDA, notebook, …);
- gaming cluster;
- voice/video communication terminals (analog/digital phones, videophones, mobile phones, …) ;
- entertainment consumer electronics audio/video terminals (STB, TV, MP3 player, HiFi equipment, …);
- domestic equipment (fridge, sensor networks, …).

In addition, the NGHN may also coexist with extensive legacy networks based on technologies with which it should ensure compatibility.

A Next Generation Home Network can be considered as a set of devices implementing the following capabilities ([11]): Gateway capability; Extender capability; End Device capability and Serve Legacy Device capability. They implement one or several specific functionalities in addition to the common set of mandatory NGHN device functionalities (including the Inter-MAC functionalities), also described in [11]. I summarize here the approach leading to the structure of these network capabilities:

- a gateway can be considered a capability implementing a WAN connectivity in addition to the common set of mandatory NGHN device functionalities;

- an end device can be considered as a capability implementing the functionality of user terminal device, where traffic can leave or enter the network, in addition to the common set of mandatory NGHN device functionalities;
- any NGHN device, implementing the basic set of mandatory NGHN functionalities, can be considered as an extender capability enabler, which is used to extend the Gigabit/s home network coverage or to interconnect different devices that cannot communicate directly.

It has also been stated that the NGHN should provide interfaces in order to interconnect to legacy devices or other networks. This is achieved by the serve legacy device capability, which provides a minimum set of functionalities to make the legacy device interoperate with the NGHN with the same experience as when it was used before. All things considered, the architecture reference model can be built around these four kinds of NGHN capabilities: the gateway, the end device, the extender, and the serve legacy device. Each of the related devices may have one or several interfaces (based on a 'no new wires' broadband technology) in order to connect to its neighbours. All these interfaces have in common the fact to be compliant with the Inter-MAC framework described in [10].

All of them can be named by the same term: the so called Ω-interface, which is therefore a multi-technology interface. Moreover the NGHN presents two natural external interfaces, the first one between the legacy device and the home network device achieving its interconnection (R interface), and the second one between the access network and the NGHN gateway (U interface).

The set of devices constituting the NGHN is organized in the form of a mesh architecture bringing in the advantages of multi-path capabilities for traffic reconfiguration. Their association can be represented under the global name of "NGHN Device", maintaining apart the Gateway in order to highlight the interface with the Access Network.

This leads to the following Next Generation Home Network Architecture Reference Model ([12]):

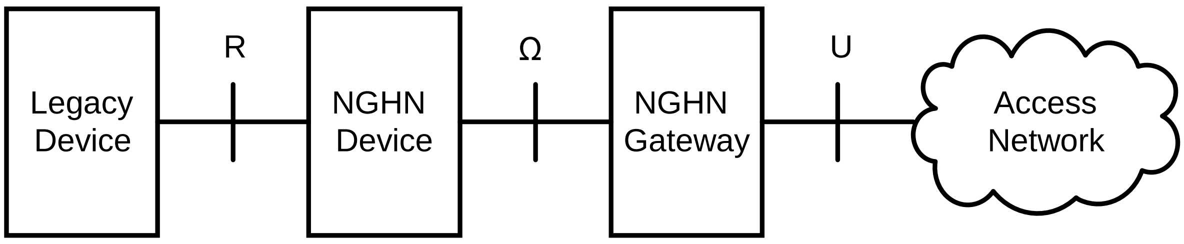


Figure 8 - NGHN Architecture Reference Model

In a real network several end devices, extenders and legacy device adapters can be interconnected in a ramified and extensive way. The multi-homing scenario, where more than one interface to external networks exists, is also possible. The following figure shows a possible implementation of the NGHN architecture with real devices:

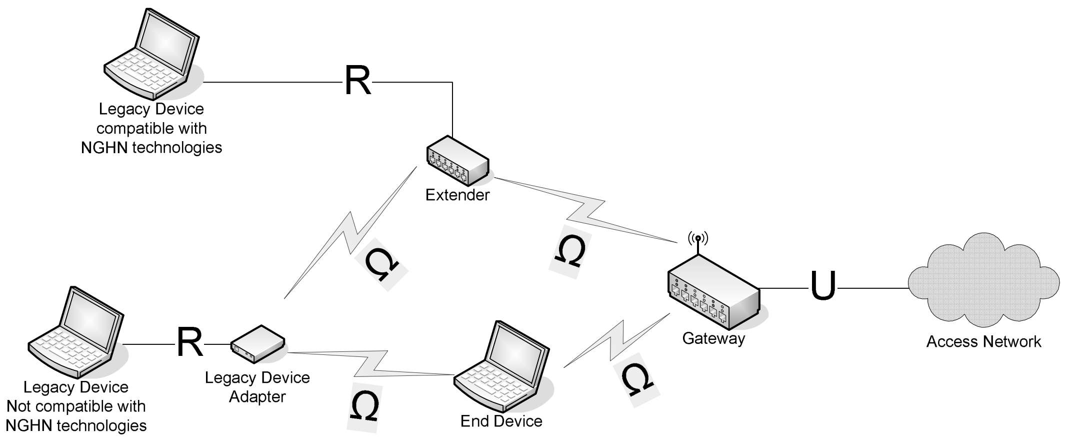

Legacy Device compatible with NGHN technologies
R
Extender
Ω
Ω
Gateway
U
Access Network
Legacy Device Not compatible with NGHN technologies
R
Legacy Device Adapter
Ω
End Device
Ω

Figure 9 - A typical NGHN architecture configuration

Figure 9 illustrates the mesh structure of a NGHN and the generic feature of the Ω interface. It also illustrates the fact that the interfacing of legacy devices can be achieved by different kinds of devices.

A lot of research open issue are still open in order to make the NGHN architecture model presented above ready to be deployed and commercialized. Among them I mention the problems related to the remote management of the network by the operator, the management and the control of multimedia service provisioning, the security, the efficiency in energy consumption and the management and control of the Quality of Service (QoS). Thus, new solutions for connection admission control, routing and path selection, load balancing, congestion control, scheduling and so on have to be studied in order to exploit in the most effective and efficient way the capabilities and the capacity offered by NGHNs.

This work intends to propose a new solution to the routing problem in NGHNs. The peculiaritites of the routing problem in NGHNs is presented in the next section, while the proposed solution is descrive in Chapter 5.

II.3.1 Routing in NGHNs

As described in the previous section, to support a variety of high capacity demanding applications (data, audio, video), next generation home networks will be realized through the integration of heterogeneous wired (e.g., Ethernet, Power Line Communication (PLC), Optical Fiber (OPT)) and wireless (e.g., Wi-Fi, Ultra Wide Band, Hybrid Wireless Optic) telecommunication technologies.

Since we are dealing with hybrid (i.e., meshed wireless and wired) networks, we have to consider frequent topology changes due to the scarce robustness of some technologies, which cause the link availability to be time-varying. In fact, due to their nature, wireless and PLC technologies are characterized by high probability of link faults (i.e., links becoming unavailable): for example, PLC systems suffer from interference due to the use of electrical power by home appliance ([13]); Wi-Fi communication systems suffer from interference due to other communication systems using the same frequency spectrum ([14]).

The objective of these high-capacity home networks is to provide new multimedia services (such as High-Definition TV (HDTV) on-demand or high-quality Video-conference) characterized by high-bitrate, long flow duration and tight Quality of Service (QoS) constraints (e.g., in terms of delay and delay variation – or jitter). To guarantee the required QoS to these flows, they are subject to an admission control procedure, in charge of deciding if the flow can be supported by the network based on current traffic and network conditions, and to a routing protocol, which decides the path to be used for the transmission[1]. In a home network, the number of high-quality flows is likely to be small (at maximum, 5-10 simultaneous flows). Therefore, given the scarce robustness of wireless and PLC technologies, the link availability dynamics due to link

[1] Standard home services, such as web browsing, emails, P2P, are low-bitrate services and/or ‘elastic’ services (i.e., they adapt their transmission bitrate to the available capacity), and do not have strict QoS requirements. Thus they are less impaired by link faults, and are regarded as background traffic with lower priority with respect to the high-quality flows.

faults become even faster than the high-quality traffic dynamics (i.e., birth and termination of high-quality flows).

In this scenario, the routing algorithm has to be fault-tolerant, in the sense that it should be able to rapidly re-route active flows as soon as a link become unavailable in the path: in fact, as a link becomes unavailable, all the flows crossing that link have to be re-routed on other paths. This re-routing event should be avoided as far as possible, because

i) during the re-routing process, some packets are likely to be lost (affecting the QoS of the flow)

ii) the re-routing process involves additional control communications, which reduce the capacity available to data communications.

If the network supports classes of service to offer QoS guarantees, decisions upon the re-routing of flows should be based also on their classes of service. For instance, re-routing a flow is likely to cause jitters in the flow transmission (i.e., a variation in the transmission delay of flow packets): such jitters are insignificant in case of data flows, whereas in case of video flows they affect the quality experienced by users.

Existing routing algorithms are classified either as proactive (e.g., [16]-[18]) or as reactive (e.g., [19], [20]). The former algorithms continuously update path information, which is then available at algorithm decision time; the drawback is that these algorithms require the knowledge of the topology of the whole network. Reactive algorithms performs a route discovery procedure on demand, i.e., only at routing decision time: on the one hand, they generate less control information since they must not continuously update topology information; on the other hand, they delay the actual data transmission until the path is discovered.

Clearly, the proactive approach is preferred in the considered home network scenario due to the fast re-routing requirements and to its limited topology width which makes the updating process fast.

To conclude, the aim of the proactive algorithm developed in this work is then twofold: on one side it has to minimize re-routing occurrences; on the other side it has to be be able of provide a fast re-routing since we are dealing with scenarios characterized by highly variable topology.

Chapter III
MDP control framework

III.1. Introduction

The fault-tolerant routing algorithm is based on the Markov Decision Process (MDP) control framework, which is presented in this chapter. MDP is a stochastic control framework where decisions need to take into account uncertainty about many future events. This chapter begins with the presentation of probability models for processes that evolve over time in a probabilistic manner. Such processes are called stochastic processes. After briefly introducing general stochastic processes, the reminder of the chapter focuses on a special kind called Markov chain. Markov chains have the special property that probabilities involving how the process will evolve in the future depend only on the present state of the process, and so are independent of events in the past. After that, Markov Decision Processes are presented as they allow to control the behavior of a system modeled as a markov chain. In fact, rather than passively accepting the design of the Markov chain, MDP allows to make a decision on how the system should evolve by controlling the transition from a state to the following one. The objective of MDP is to choose the optimal action for each state that minimize the cost associated for the system in being in each state, considering both immediate and subsequent costs.

III.2. Stochastic process

A *stochastic process* is defined to be an indexed collection of random variables $\{X_t\}$, where the index *t* runs through a given set *T*. Often *T* is taken to be the set of non-negative integers, and X_t represents a measurable characteristic of interest at time *t*. Stochastic processes are of interest for describing the behaviour of a system operating over some period of time. The current status of the system can fall into anyone of the *M* + 1 mutually exclusive categories called **states**. For notational convenience, in this chapter these states are labelled 0,1,…,*M*. The random variable X_t represents the state of the system at time *t*, so its only possible values are 0,1,…,*M*. The system is observed at particular points of time, labelled *t*=0,1,…. Thus, the stochastic process $\{X_t\} = \{X_0, X_1, X_2, ...\}$provides a mathematical representation of how the status of the physical system evolves over time. This kind of processes is referred to as being a *discrete time* stochastic process with *finite state space*.

III.3. Markov chains

Assumptions regarding the joint distribution of $X_0, X_1, ...$ are necessary to obtain analytical results. One assumption that leads to analytical tractability is that the stochastic process is a Markov chain, which has the following key property: "a stochastic process X_t is said to have the Markovian property if:

$P\{X_{t+1} = j \mid X_0 = k_0, X_1 = k_1, ..., X_{t-1} = k_{t-1}, X_t = i\} = P\{X_{t+1} = j \mid X_t = i\}$, for *t* = 0,1,… and every sequence *i*, *j*, k_0, k_1,…, k_{t-1}.

In words, this Markovian property says that the conditional probability of any future "event", given any past "event" and the present state $X_t = i$, is *independent* of any past event and depends only upon the present state.

A stochastic process $\{X_t\}$ (*t* = 0,1,2,…) is a Markov chain if it has the *Markovian property*.

The conditional probabilities $P\{X_{t+1} = j \mid X_t = i\}$ for a Markov chain are called (one-step) transition probabilities. If, for each *i* and *j*, $P\{X_{t+1} = j \mid X_t = i\} = P\{X_1 = j \mid X_0 = i\}$, for all *t* = 0,1,2,… then the (one-step) transition probabilities are said to be *stationary*. Thus, having stationary transition probabilities implies that the transition probabilities do not change over time. The existence of stationary (one-step) transition probabilities also implies that, for each *i*, *j*, and *n* (*n* =0,1,2,…), $P\{X_{t+n} = j \mid X_t = i\} = P\{X_n = j \mid X_0 = i\}$ for all *t* = 0,1,…. These conditional probabilities are called *n*-step transitional probabilities.

To simplify notation with stationary transition probabilities, let:

$$p_{ij} = P\{X_{t+1} = j \mid X_t = i\},$$

$$p_{ij}^{(n)} = P\{X_{t+n} = j \mid X_t = i\}.$$

Thus, the *n*-step transition probabilitiy $p_{ij}^{(n)}$ is just the conditional probability that the system will be in state *j* after exact *n* steps (unit of time), given it starts in state *i* at any time *t*.

Because the $p_{ij}^{(n)}$ are conditional probabilities, they must be non negative, and since the process must make a transition into some state, they must satisfy the properties:

$$p_{ij}^{(n)} \geq 0, \qquad \text{for all } i \text{ and } j;\ n = 0,1,2,\ldots,$$

$$\sum_{j=0}^{M} p_{ij}^{(n)} = 1, \qquad \text{for all } i;\ n = 0,1,2,\ldots$$

A convenient way to show all the *n*-step transition probabilities is the *n-step transition matrix*:

$$P^{(n)} = \begin{array}{cccc} p_{00}^{(n)} & p_{01}^{(n)} & \cdots & p_{0M}^{(n)} \\ p_{10}^{(n)} & p_{11}^{(n)} & \cdots & p_{1M}^{(n)} \\ \cdots & \cdots & \cdots & \cdots \\ p_{M0}^{(n)} & p_{M1}^{(n)} & \cdots & p_{MM}^{(n)} \end{array} \qquad \text{for } n = 0,1,2,\ldots$$

Note that the transition probability in a particular row and column is for the transition from the row state to the column state. When n =1, we drop the superscript n and simply refer to this as the transition matrix.

The Markov chains considered in this work have the following properties:

1. a finite number of states.
2. stationary transition probabilities.

The following *Chapman-Kolmogorov equations* provide a method for computing the n-step transtion probabilities:

$$p_{ij}^{(n)} = \sum_{k=0}^{M} p_{ik}^{(m)} p_{kj}^{(n-m)}$$

for all *i* = 0,1,…,*M*; *j* =0,1,…,*M*; and any *m* =1,2,…, *n*-1; *n* = *m*+1, *m*+2,…

These equations point out that in going from state *i* to state *j* in *n* steps, the process will be in some state *k* after exactly *m* (less than *n*) states. This expression enable the *n*-step transition probabilities to be obtained from the one-step transition probabilities recursively. Thus, the *n*-step transition probability matrix $\mathbf{P}^n$ can be obtained by computing the *n*th power of the one-step transition matrox **P**: $\mathbf{P}^{(n)} = \mathbf{P}^n$.

III.3.1 Classification of states of a Markov chain

It is evident that the transition probabilities associated with the states play an important role in the study of Markov chains. To further describe the properties of Markov chains, it is necessary to present some concepts and definitions concerning these states.

State *j* is said to be accessible from state *i* if $p_{ij}^{(n)} > 0$ for some $n \geq 0$. Thus, state *j* being accessible from state *i* means that it is possible for the system to enter state *j* eventually when it starts from state *i*. In general, a sufficient condition for *all* states to be accessible is that there exists a value of *n* for which $p_{ij}^{(n)} > 0$ for all *i* and *j*.

If state *j* is accessible from state *i* and state *i* is accessible from state *j*, then states *i* and *j* are said to communicate. In general:

1. any state communicates with itself (because $p_{ii}^{(0)} = 1$);
2. if state *i* communicates with state *j*, then state *j* communicates with state *i*;
3. if state *i* communicates with state *j* and state *j* communicates with state *k*, then state *i* communicates with state *k*.

As a result of these properties of communication, the states may be partitioned into one or more separate class such that those states that communicate with each other are in the same class. If there is only one class, i.e., all the states communicate, the Markov chain is said to be irreducible.

It is often useful to talk about whether a process entering a state will ever return to this state. A state is said to be a transient state if, upon entering this state, the process *may never return* to this state again. Therefore, state *i* is transient if and only if there exists a state *j* ($j \neq i$) that is accessible from state *i* but not vice versa, that is, state *i* is not accessibile from state *j*. Thus, if state *i* is transient and the process visits this state, there is a positive probability (perhaps even a probability of 1) that the process will later move to state *j* and so will never return to state *i*. Consequently, a transient state will be visited only a finite number of times.

When starting in state *i*, another possibility is that the process *definitely* will return to this state. A state is said to be a recurrent state if, upon entering this state, the process *definitely will return* to this state again. Therefore, a state is recurrent if and only if it is not transient. Since a recurrent state definitely will be revisited after each visit, it will be visited infinitely often if the process continues forever.

If the process enters a certain state and then stays in this state at the next step, this is considered a *return* to this state. Hence, the following kind of state is a special type of recurrent state. A state is said to be an absorbing state if, upon entering this state, the process *never will leave* this state again. Therefore, state *i* is an absorbing state if and only if p_{ii} =1.

Recurrence is a class property. That is, all states in a class are either recurrent or transient. Furthermore, in a finite-state Markov chain, not all states can be transient. Therefore, all states in an irreducible finite-state Markov chain are recurrent.

Another useful property of Markov chains is *periodicities*. The periodo f state *i* is defined to be the integer (t > 1) such that $p_{ii}^{(n)} = 0$ for all the values of *n* other than *t*, 2*t*, 3*t*,…and *t* is the largest integer with this property. Just as recurrence is a class property, it can be shown that periodicity is a class property. That is, if state *i* in a class has period *t*, the all states in that class have period *t.*

In a finite-state Markov chain, recurrent states that are aperiodic are called ergodic states. A Markov chain is said to be *ergodic* if all its states are ergodic states.

III.3.2 Long run properties of Markov chains

For any irreducible ergodic Markov chain, $\lim_{n\to\infty} p_{ij}^{(n)}$ exists and is independent of *i*. Furthermore,

$$\lim_{n\to\infty} p_{ij}^{(n)} = \pi_j > 0,$$

where the π_j uniquely satisfy the following steady-state equations.

$$\pi_J = \sum_{i=0}^{M} \pi_i p_{ij} \qquad \text{for } j = 0,1,\ldots, \text{M}$$

$$\sum_{j=0}^{M} \pi_i = 1$$

The π_j are called steady-state probabilities of the Markov chain. The term *steady-state* probability means that the probability of finding the process in a certain state, say *j*, after a large number of transitions tends to the value *_j*, independent of the probability distribution of the initial state. It is important to note that the steady-state probability does not imply that the process settles down into one state. On the contrary, the process continues to make transitions from state to state, and at any step *n* the transition probability from state *i* to state *j* is still p_{ij}.

There are other important results concerning steady-state probabilities. In particular, if *i* and *j* are recurrent states belonging to different classes, then $p_{ij}^{(n)} = 0$ for all *n*. This result follows from the definition of a class.

Similarly, if *j* is a transient state, then $\lim_{n\to\infty} p_{ij}^{(n)} = 0$ for all *i*. Thus, the probability of finding the process in a transient state after a large number of transitions tends to zero.

If the requirement that the states be aperiodic is relaxed, then the limit $\lim_{n\to\infty} p_{ij}^{(n)}$ may not exist. However, the following limit always exists for an irreducible (finite-state) Markov chain:

$$\lim_{n\to\infty}\left(\frac{1}{n}\sum_{k=1}^{n} p_{ij}^{(k)}\right) = \pi_j$$

When the π_j satisfy the steady-state equations.

This result is important in computing the *long-run average cost per unit time* associated with a Markov chain. Suppose that a cost (or other penalty function) $C(X_t)$ is incurred when the process is in state X_t at time *t*, for $t = 0, 1, 2, \ldots$. Note that $C(X_t)$ is a random variable that takes on any one of the values $C(0), C(1), \ldots, C(M)$ and that the function $C(\bullet)$ is independent of *t*. The expected average cost incurred over the first *n* periods is given by

$$E\left[\frac{1}{n}\sum_{t=1}^{n} C(X_t)\right].$$

By using the result that

$$\lim_{n\to\infty}\left(\frac{1}{n}\sum_{k=1}^{n} p_{ij}^{(k)}\right)=\pi_j$$

it can be shown that the (long-run) *expected average cost per unit time* is given by

$$\lim_{n\to\infty} E\left[\frac{1}{n}\sum_{t=1}^{n} C(X_t)\right]=\sum_{j=0}^{M}\pi_j C(j)\,.$$

In addition, π_j can be interpreted as the (long-run) actual fraction of times the system is in state *j*.

III.3.3 Continuous time Markov chains

Until now it was assumed that the time parameter *t* was discrete (that is, *t* = 0,1,2,…). Such an assumption is suitable for many problems, but there are certain cases where a continuous time parameter (call it *t'*) is required, because the evolution of the process is being observed continuously over time. The definition of a Markov chain given before also extends to such continuous processes.

As before, I label the possible states of the system as 0, 1, . . . , *M*. Starting at time 0 and letting the time parameter *t'* run continuously for $t \geq 0$, I let the random variable *X*(*t'*) be the state of the system at time *t'*. Thus, *X*(*t'*) will take on one of its possible (*M* + 1) values over some interval, $0 \leq t' < t_1$, then will jump to another value over the next interval, $t_1 \leq t' < t_2$, etc., where these transit points (t_1, t_2, . . .) are random points in time (*not* necessarily integer).

Now consider the three points in time (1) *t'* = *r* (where $r \geq 0$), (2) *t'* = *s* (where *s* > *r*), and (3) *t'* = *s* + *t* (where *t* > 0), interpreted as follows:

t' = *r*	is a past time,
t' = *s*	is the current time,
t' = *s* + *t*	is *t* time units into the future.

Therefore, the state of the system now has been observed at times *t'* = *s* and *t'* = *r*. Label these states as *X*(*s*) = *i* and *X*(*r*) = *x*(*r*). Given this information, it now would be natural to seek the probability distribution of the state of the system at time *t'* = *s* + *t*:

$$P\{X(s+t)=j \mid X(s)=i, X(r)=x(r)\} \qquad \text{for } j = 0,1,\ldots, \text{M}.$$

Deriving this conditional probability often is very difficult. However, this task is considerably simplified if the stochastic process involved possesses the following key property: a continuous time stochastic process $\{X(t');t'\geq 0\}$ has the Markovian property if $P\{X(s+t)=j \mid X(s)=i, X(r)=x(r)\}=P\{X(t+s)=j \mid X(s)=i\}$, for all *i*, *j* = 0,1,…, M and for all $r \geq 0$, *s* > *r*, and *t* > 0.

Note that $P\{X(t+s)=j \mid X(s)=i\}$ is a transition probability**,** just like the transition probabilities for discrete time Markov chains considered above, where the only difference is that *t* now need not be an integer. If the transition probabilities are independent of *s*, so that $P\{X(s+t)=j \mid X(s)=i\}=P\{X(t)=j \mid X(0)=i\}$ for all *s* > 0, they are called stationary transition probabilities. To simplify notation, I shall denote these stationary transition probabilities by

$$p_{ij}(t)=P\{X(t)=j \mid X(0)=i\},$$

where $p_{ij}(t)$ is referred to as the continuous time transition probability function. It is assumed that

$$\lim_{t\to 0} p_{ij}(t)=\begin{cases}1 & if \quad i=j\\ 0 & if \quad i\neq j\end{cases}.$$

Now we are ready to define the continuous time Markov chains: a continuous time stochastic process $\{X(t');t'\geq 0\}$ is a continuous time Markov chain if it has the *Markovian property*.

In the analysis of continuous time Markov chains, one key set of random variables is the following: aach time the process enters state *i*, the amount of time it spends in that state before moving to a different state is a random variable T_i, where *i* = 0, 1, . . . , *M*. Suppose that the process enters state *i* at time *t'* = *s*. Then, for any fixed amount of time *t* > 0, note that $T_i > t$ if and only if *X*(*t'*) = *i* for all *t'* over the interval $s \leq t' \leq s+t$. Therefore, the Markovian property (with stationary transition probabilities) implies that

$$P\{T_i > t+s \mid T_i > s\}=P\{T_i > t\}.$$

This is a rather unusual property for a probability distribution to possess. It says that the probability distribution of the *remaining* time until the process transits out of a given state always is the same, regardless of how much time the process has already spent in that state. In effect, the random variable is memoryless; the process forgets its history. There is only one (continuous) probability distribution that possesses this property - the *exponential distribution.* The exponential distribution has a single parameter, call it q, where the mean is $1/q$ and the cumulative distribution function is

$$P\{T_i \leq t\} = 1 - e^{-qt}, \qquad \text{for } t \geq 0.$$

This result leads to an equivalent way of describing a continuous time Markov chain:

1. the random variable T_i ha san exponential distribution with a mean of $1/q_i$
2. when leaving state i, the process moves to a state j with probability p_{ij}, where p_{ij} satisfy the conditions

   $p_{ij} = 0$ for all i,

   $\sum_{j=o}^{M} p_{ij} = 1$ for all i
3. the next state visited after state i is independent of the time spent in state i.

Just as the transition probabilities for a discrete time Markov chain satisfy the Chapman-Kolmogorov equations, the continuous time transition probability function also satisfies these equations. Therefore, for any states i and j and nonnegative numbers t and s ($0 \leq s \leq t$),

$$p_{ij}(t) = \sum_{k=1}^{M} p_{ik}(s) p_{kj}(t-s).$$

A pair of states i and j are said to *communicate* if there are times t_1 and t_2 such that $p_{ij}(t_1) > 0$ and $p_{ji}(t_2) > 0$. All states that communicate are said to form a *class.* If all states form a single class, i.e., if the Markov chain is *irreducible* (hereafter assumed), then $p_{ij}(t) > 0$, for all $t > 0$ and all states i and j.

Furthermore, $\lim_{t\to\infty} p_{ij}(t) = \pi_j$ always exists and is independent of the initial state of the Markov chain, for j _ 0, 1, . . . , M. These limiting probabilities are commonly referred to as the steady-state probabilities (or *stationary probabilities*) of the Markov chain. The π_j satisfy the equations

$$\pi_j = \sum_{i=0}^{M} \pi_i p_{ij}(t) \qquad \text{for } j = 0,1,\ldots, \text{M and every } t \geq 0 .$$

III.4. Markov Decision Processes

Many important systems can be modelled as either a discrete time or continuous time Markov chain. It is often useful to describe the behaviour of such a system in order to evaluate its performances. However, it may be even more useful to design the operation of the system so as to optimize its performance. Therefore, rather than passively accepting the design of the Markov chain and the corresponding fixed transition matrix, it is possible to be proactive. In fact, for each possible state of the Markovian chain, it is possible to make a decision about which one of the several alternative actions should be taken in that state. The action chosen affects the transition probabilities as well as both the immediate costs and subsequent costs from operating the system. The objective is to choose the optimal actions for the respective states when considering both immediate and subsequent costs. The decision process for doing this is referred to as *Markov decision process*.

The model for the Markov decision process considered in this work can be summarized as follows:

1. The state i of a discrete time Markov chain is observed after each transition (i = 0,1,…, M).
2. After each observation, a decision (action) k is chosen from a set of K possible decisions (k = 1,2,…, K). (Note that some of the K decisions may not be relevant for some of the states).
3. If decision $d_i = k$ is made in state i, an immediate cost is incurred that has an expected value C_{ik}.
4. The decision $d_i = k$ in state i determines what the transition probabilities will be for the next transition from state i. Denote these transition probabilities by $p_{ij}(k)$, for j =0,1,…, M.
5. A specification of the decisions for the respective states (d_0, d_1,…, d_M) prescribes a *policy for the Markov decision process.*

6. The objective is to find an optimal policy according to some cost criterion which considers both immediate costs and subsequent costs that result from the future evolution of the process. One common criteria is to minimize the (long-run) expected average cost per unit time.

This general model qualifies to be a Markov decision process because it possesses the Markovian property that characterizes any Markov process. In particular, given the current state and decision, any probabilistic statement about the future of the process is completely unaffected by providing any information about the history of the process. This property holds here since (1) we are dealing with a Markov chain, (2) the new transition probabilities depend on only the current state and decision, and (3) the immediate expected cost also depends on only the current state and decision.

There exists several procedures to find the optimal policy. One of them is to use the exhaustive enumeration, but this one is appropriate only for tiny stationary and deterministic problems, where there are only few relevant policies. In many applications where there are many policies to be evaluated, this approach is not feasible. For such cases, algorithms that can efficiently find an optimal policy are needed. Some of them are described in the next sections.

III.4.1 Linear programming and optimal policies

Any stationary and deterministic policy R can be viewed as a rule that the prescribes decision $d_i(R)$ whenever the system is in state i, for each i = 0,1,…,M. Thus, R is characterized by the values

$$\{d_0(R), d_1(R), ..., d_M(R)\}.$$

Equivalently, R can be characterized by assigning values D_{ik} = 0 or 1 in the matrix

$$\begin{array}{cc} & \text{Decision } k \\ \text{State} & \begin{bmatrix} D_{01} & D_{02} & ... & D_{0K} \\ D_{11} & D_{12} & ... & D_{1K} \\ ... & ... & ... & ... \\ D_{M1} & D_{M1} & ... & D_{MK} \end{bmatrix} \end{array}$$

Where each D_{ik} (i = 0,1,…,M and k = 1,2,…,K) is defined as

$$D_{ik} = \begin{cases} 1 & if \;\; decision \;\; k \, is \, to \, be \, made \, in \, state \; i \\ 0 & otherwise \end{cases}$$

Therefore, each row in the matrix must contain a single 1 with the rest of the elements 0s.

Introducing D_{ik} provides motivation for a *linear programming formulation.* It is hoped that the expected cost of a policy can be expressed as a linear function of D_{ik} or a related variable, subject to linear constraints. Unfortunately, the D_{ik} values are integers (0 or 1), and continuous variables are required for a linear programming formulation. This requirement can be handled by expanding the interpretation of a policy. The previous definition calls for making the same decision every time the system is in state *i*. The new interpretation of a policy will call for determining a probability distribution for the decision to be made when the system is in state *i*. With this new interpretation, the D_{ik} now need to be redefined as

$$D_{ik} = P\{decision = k \mid state = i\}.$$

In other words, given that the system is in state *i*, variable D_{ik} is the *probability* of choosing decision *k* as the decision to be made. Therefore, (D_{i1}, D_{i2}, . . . , D_{ik}) is the *probability distribution* for the decision to be made in state *i*. This kind of policy using probability distributions is called **a randomized policy,** whereas the policy calling for D_{ik} = 0 or 1 is a *deterministic policy.* Randomized polizie can again be characterized by the matrix

$$\begin{array}{cc} & \text{Decision } k \\ \text{State} & \begin{bmatrix} D_{01} & D_{02} & \dots & D_{0K} \\ D_{11} & D_{12} & \dots & D_{1K} \\ \dots & \dots & \dots & \dots \\ D_{M1} & D_{M1} & \dots & D_{MK} \end{bmatrix} \end{array}$$

where each row sum sto 1, and now $0 \le D_{ik} \le 1$.

III.4.1.1. A linear programming formulation

The convenient decision variables (denoted here by y_{ik}) for a linear programming model are defined as follows. For each $i = 0,1,...,M$ and $k = 1,2,...,K$, let

y_{ik} be the steady-state unconditional probability that the system is in state i *and* decision k is made; i.e.,

$$y_{ik} = P\{ \text{ state} = i \text{ and decision} = k \ \}.$$

Each y_{ik} is closely related to the corresponding D_{ik} since, from the rules of conditional probability, $y_{ik} = \pi_i D_{ik}$, where π_i is the steady-state probability that the Markov chain is in state i. Furthermore,

$$\pi_i = \sum_{k=1}^{K} y_{ik} ,$$

so that

$$D_{ik} = \frac{y_{ik}}{\pi_i} = \frac{y_{ik}}{\sum_{k=1}^{K} y_{ik}}$$

There exist several constraints on y_{ik}:

1. $\sum_{i=1}^{M} \pi_i = 1$ so that $\sum_{i=0}^{M}\sum_{k=1}^{K} y_{ik} = 1$.

2. From results on steady-state probabilities $\pi_j = \sum_{i=0}^{M} \pi_i p_{ij}$ so that

$$\sum_{k=1}^{K} y_{jk} = \sum_{i=0}^{M}\sum_{k=1}^{K} y_{ik} p_{ij}(k), \qquad \text{for } j = 0,1,\ldots,M .$$

3. $y_{ik} \geq 0$, for $i = 0,1,\ldots,M$ and $k = 1,2,\ldots,K$.

The long-run expected average cost per unit time is given by

$$E(C) = \sum_{i=0}^{M}\sum_{k=1}^{K} \pi_i C_{ik} D_{ik} = \sum_{i=0}^{M}\sum_{k=1}^{K} C_{ik} y_{ik} .$$

Hence, the linear programming model is to choose the y_{ik} so as to

Minimize $Z = \sum_{i=0}^{M}\sum_{k=1}^{K} C_{ik} y_{ik}$, subject to the constraints:

(1) $$\sum_{i=0}^{M}\sum_{k=1}^{K} y_{ik} = 1.$$

(2) $$\sum_{k=1}^{K} y_{jk} - \sum_{i=0}^{M}\sum_{k=1}^{K} y_{ik}\, p_{ij}(k) = 0, \qquad \text{for } j = 0,1,\ldots,M\ .$$

(3) $$y_{ik} \geq 0, \qquad \text{for } i = 0,1,\ldots,M; \quad k = 1,2,\ldots,K.$$

Thus, this model has $M+2$ functional constraints and $K(M+1)$ decision variables. Assuming that the model is not too huge, it can be solved by the *simplex method*. Once the y_{ik} values are obtained, each D_{ik} is found from

$$D_{ik} = \frac{y_{ik}}{\sum_{k=1}^{K} y_{ik}}.$$

The optimal solution obtained by the simplex method has some interesting properties. It will contain $M+1$ basic variables $y_{ik} \geq 0$. It can be shown that $y_{ik} > 0$ for at least one $k = 1,2,\ldots,K$, for each $i = 0,1,\ldots,M$. Therefore, it follows that $y_{ik} > 0$ for only *one k* for each $i = 0,1,\ldots,M$. Consequently, each $D_{ik} = 0$ or 1.

The key conclusion is that the optimal policy found by the simplex method is *deterministic* rather than randomized. Thus, allowing policies to be randomized does not help at all in improving the final policy. However, it serves an extremely useful role in this formulation by converting integer variables (the D_{ik}) to continuous variables so that linear programming (LP) can be used.(The analogy in *integer programming* is to use the *LP relaxation* so that the simplex method can be applied and then to have the *integer solutions property* hold so that the optimal solution for the LP relaxation turns out to be integer anyway.)

Linear programming can be thus used to solve vastly large problems, and software packages for the simplex method are very widely available.

III.4.2 Policy improvement algorithm

After the presentation of the exhaustive enumeration and the linear programming techniques, hereafter I present a third popular method to derive an optimal policy for

Markov decision processes called policy improvement algorithm. The key advantage of this method is that it tends to be very efficient, because it usually reaches an optimal policy in a relatively small number of iterations.

As a joint result of the current state *i* of the system and the decision $d_i(R) = k$ when operating under policy *R*, two things occur. An (expected) cost C_{ik} is incurred that depends upon only the observed state of the system and the decision made. The system moves to state *j* at the next observed time period, with transition probability given by $p_{ij}(k)$. If, in fact, state *j* influences the cost that has been incurred, then C_{ik} is calculated as follows. Denote by $q_{ij}(k)$ the (expected) cost incurred when the system is in state *I* and decision *k* is made and then it evolves to state *j* at the next observed time period. Then

$$C_{ik} = \sum_{j=0}^{M} q_{ij}(k) p_{ij}(k).$$

It is possible to show that, for any given policy *R*, there exist values *g(R),* $v_0(R),\ v_1(R),\ \ldots,\ v_M(R)$ that satisfy

$$g(R) + v_i(R) = C_{ik} + \sum_{j=0}^{M} p_{ij}(k) v_j(R), \qquad \text{for } i = 0,1,2,...,M.$$

Denote by $v_i^n(R)$ the total expected cost of a system starting in state *I* (beginning the first observed time period) and evolving for *n* time periods. Then $v_i^n(R)$ has two components: C_{ik}, the cost incurred during the first observed time period, and $\sum_{j=0}^{M} p_{ij}(k) v_j^{n-1}(R),$ the total expected cost of the system evolving over the remaining $n-1$ time periods. This gives the *recursive equation*

$$v_i^n(R) = C_{ik} + \sum_{j=0}^{M} p_{ij}(k) v_j^{n-1}(R), \qquad \text{for } i = 0,1,2,...,M,$$

where $v_i^1(R) = C_{ik}$ for all *i*.

It will be useful to explore the behaviour of $v_i^n(R)$ as *n* grows large. Recall that the (long-run) expected average cost per unit time following any policy *R* can be expressed as

$$g(R)=\sum_{i=0}^{M}\pi_i C_{ik},$$

which is independent of the starting state *i*. Hence, $v_i^n(R)$ behaves approximately as *n g(R)* for large *n*. In fact, if we neglect small fluctuations, $v_i^n(R)$ can be expressed as the sum of two components: $v_i^n(R)\approx ng(R)+v_i(R),$ where the first component is independent of the initial state and the second is dependent upon the initial state. Thus, $v_i(R)$ can be interpreted as the effect on the total expected cost due to starting in state *i*. Consequently, $v_i^n(R)-v_j^n(R)\approx v_i(R)-v_j(R),$ so that $v_i(R)-v_j(R)$ is a measure of the effect of starting in state *i* rather than state *j*. Letting *n* grow large, it is then possible to substitute $v_i^n(R)=ng(R)+v_i(R)$ and $v_j^{n-1}(R)=(n-1)g(R)+v_j(R)$ into the *recursive equation.* This leads to the system of equations given in the opening paragraph of this subsection.

Note that this system has *M*+1 equations with *M*+2 unknowns, so that one of these variables may be chosen arbitrarily. By convention, $v_M(R)$ will be chosen equal to zero. Therefore, by solving the system of linear equations, I can obtain $g(R),$ the (long-run) expected average cost per unit time when policy *R* is followed. In principle, all policies can be enumerated and that policy which minimizes $g(R)$ can be found. However, even for a moderate number of states and decisions, this technique is cumbersome. Fortunately, there exists an algorithm that can be used to evaluate policies and find the optimal one without complete enumeration, as described next.

III.4.2.1. The Policy Improvement Algorithm

The algorithm begins by choosing an arbitrary policy R_1. It then solves the system of equations to find the values of $g(R_1),v_0(R),v_1(R),...,v_{M-1}(R)$ [with $v_M(R)=0$]. This step is called *value determination.* A better policy, denoted by R_2, is then constructed. This step is called *policy improvement.* These two steps constitute an iteration of the algorithm. Using the new policy R_2, we perform another iteration. These iterations continue until two successive iterations lead to identical policies, which signifies that the optimal policy has been obtained. The details are outlined below:

Initialization: Choose an arbitrary initial trial policy R_1. Set $n = 1$

Iteration n:

Step 1 - Value determination: For policy R_n, use $p_{ij}(k)$, C_{ik}, and $v_M(R_n) = 0$ to solve the system of *M* + 1 equations

$$g(R_n) = C_{ik} + \sum_{j=0}^{M} p_{ij}(k) v_j(R_n) - v_i(R_n), \qquad \text{for } i = 0,1,\ldots,M,$$

for all *M* + 1 unknown values of $g(R_n), v_0(R_n), v_1(R_n), \ldots, v_{M-1}(R_n)$.

Step 2 - Policy improvement: Using the current values of $v_i(R_n)$ computed for policy R_n, find the alternative policy R_{n+1} such that, for each state $i, d_i(R_{n+1}) = k$ is the decision that minimizes

$$C_{ik} + \sum_{j=0}^{M} p_{ij}(k) v_j(R_n) - v_i(R_n)$$

i.e., for *each* state *i*,

$$\underset{k=1,2,\ldots,k}{Minimize} \quad \left[C_{ik} + \sum_{j=0}^{M} p_{ij}(k) v_j(R_n) - v_i(R_n) \right],$$

and then set $d_i(R_{n+1})$ equal to the minimizing value of *k*. This procedure defines a new policy R_{n+1}.

Optimality test: The current policy R_{n+1} is optimal if this policy is identical to policy R_n. If it is, stop. Otherwise, reset $n = n + 1$ and perform another iteration.

Two key properties of this algorithm are

1. $g(R_{n+1}) \le g(R_n), \qquad \text{for } n = 1,2,\ldots$

2. The algorithm terminates with an optimal policy in a finite number of iterations.

III.4.3 Discounted cost criterion

Up to now, policies were measured on the basis of their (long-run) expected average cost per unit time. Now I turn to an alternative measure of performance, namely, the *expected total discounted cost*.

This measure uses a *discount factor* α, where $0 < \alpha < 1$. The discount factor α can be interpreted as equal to $1/(1+ i)$, where i is the current interest rate per period. Thus, α is the *present value* of one unit of cost one period in the future. Similarly, α^m is the *present value* of one unit of cost m periods in the future.

This *discounted cost criterion* becomes preferable to the *average cost criterion* when the time periods for the Markov chain are sufficiently long that the *time value of money* should be taken into account in adding costs in future periods to the cost in the current period. Another advantage is that the discounted cost can readily be adapted to dealing with a *finite-period* Markov decision process where the Markov chain will terminate after a certain number of periods.

Both the policy improvement technique and the linear programming approach still can be applied here with relatively minor adjustments from the average cost case, as I describe next. Then I will present another technique, called the *method of successive approximations,* for quickly approximating an optimal policy.

III.4.3.1. A Policy Improvement Algorithm

To derive the expression needed for the value determination and policy improvement steps of the algorithm, I now adopt the viewpoint of *probabilistic dynamic programming.* In particular, for each state i ($i = 0,1,\ldots,M$) of a Markov decision process operating under policy R, let $V_i^n(R)$ be the *expected total discounted cost* when the process starts in state i (beginning the first observed time period) and evolves for n time periods. Then $V_i^n(R)$ has two components: C_{ik} , the cost incurred during the first observed time period, and $\alpha\sum_{j=0}^{M} p_{ij}(k)V_j^{n-1}(R)$, the expected total discounted cost of the process evolving over the remaining $n-1$ time periods. For each $i = 0,1,...,M$, this yields the recursive equation

$$V_i^n(R) = C_{ik} + \alpha \sum_{j=0}^{M} p_{ij}(k) V_j^{n-1}(R),$$

As *n* approaches infinity, this recursive equation converges to

$$V_i(R) = C_{ik} + \alpha \sum_{j=0}^{M} p_{ij}(k) V_j(R), \qquad \text{for } i = 0,1,\dots,M \text{ ,}$$

where $V_i(R)$ can now be interpreted as the expected total discounted cost when the process starts in state I and continues indefinitely. There are $M+1$ equations and $M+1$ unknowns, so the simultaneous solution of this system of equations yields the $V_i(R)$.

This system of equations provides the expressions needed for a policy improvement algorithm. After summarizing this algorithm in general terms, we shall use it to check whether this particular policy still is optimal under the discounted cost criterion.

Summary of the Policy Improvement Algorithm (Discounted Cost Criterion):

Initialization: Choose an arbitrary initial trial policy R_1. Set *n*=1.

Iteration n:

Step 1: Value determination: For policy R_n, use $p_{ij}(k)$ and C_{ik} to solve the system of $M+1$ equations

$$V_i(R_n) = C_{ik} + \alpha \sum_{j=0}^{M} p_{ij}(k) V_j(R_n), \qquad \text{for } i = 0, 1, \dots, M,$$

for all *M*+1 unknown values of $V_0(R_n), V_1(R_n), \dots, V_M(R_n)$.

Step 2: Policy improvement: Using the current values of the $V_i(R_n)$, find the alternative policy R_{n+1} such that, for each state *i*, $d_i(R_{n+1}) = k$ is the decision that minimizes

$$C_{ik} + \alpha \sum_{j=0}^{M} p_{ij}(k) V_j(R_n)$$

i.e., for *each* state *i*,

$$\underset{k=1,2,\ldots,K}{Minimize} \left[C_{ik} + \alpha \sum_{j=0}^{M} p_{ij}(k) V_j(R_n) \right],$$

and then set $d_i(R_{n+1})$ equal to the minimizing value of *k*. This procedure defines a new policy R_{n+1}.

Optimality test: The current policy R_{n+1} is optimal if this policy is identical to policy R_n. If it is, stop. Otherwise, reset $n = n + 1$ and perform another iteration.

Three key properties of this algorithm are as follows:

1. $V_i(R_{n+1}) \le V_i(R_n)$, for i = 0, 1, …,M and n = 1, 2,….

2. The algorithm terminates with an optimal policy in a finite number of iterations.

3. The algorithm is valid without the assumption (used for the average cost case) that the Markov chain associated with every transition matrix is irreducible.

III.4.3.2. Linear Programming Formulation

The linear programming formulation for the discounted cost case is similar to that for the average cost case. However, we no longer need the first constraint given before; but the other functional constraints do need to include the discount factor α.

The other difference is that the model now contains constants β_j for *j* = 0, 1, …, *M.*

These constants must satisfy the conditions

$$\sum_{j=0}^{M} \beta_j = 1, \qquad \beta_j > 0 \quad \text{for } j = 0, 1, \ldots, M,$$

but otherwise they can be chosen arbitrarily without affecting the optimal policy obtained from the model.

The resulting model is to choose the values of the *continuous* decision variables y_{ik} so as to

$$\text{Minimize } Z = \sum_{i=0}^{M} \sum_{k=1}^{K} C_{ik} y_{ik} ,$$

subject to the constraints

(1) $\sum_{k=1}^{K} y_{jk} - \alpha \sum_{i=0}^{M} \sum_{k=1}^{K} y_{ik} p_{ij}(k) = \beta_j$, for $j = 0, 1, \ldots, M$,

(2) $y_{ik} \geq 0$, for $i = 0, 1, \ldots, M$; $k = 1, 2, \ldots, K$.

Once the simplex method is used to obtain an optimal solution for this model, the corresponding optimal policy then is defined by

$$D_{ik} = P\,\{\text{decision} = k \text{ and state} = i\} = \frac{y_{ik}}{\sum_{k=1}^{K} y_{ik}}.$$

The y_{ik} now can be interpreted as the *discounted* expected time of being in state *i* and making decision *k*, when the probability distribution of the *initial state* (when observations begin) is $P\{X_0 = j\} = \beta_j$ for *j*= 0, 1, …, *M*. In other words, if $z_{ik}^{n} = P$ {at time *n*, state = *i* and decision = *k*}, then

$$y_{ik} = z_{ik}^{0} + \alpha z_{ik}^{1} + \alpha^2 z_{ik}^{2} + \alpha^3 z_{ik}^{3} + \cdots.$$

With the interpretation of the β_j as *initial state probabilities* (with each probability greater than zero), *Z* can be interpreted as the corresponding expected total discounted cost. Thus, the choice of β_j affects the optimal value of *Z* (but not the resulting optimal policy).

It again can be shown that the optimal policy obtained from solving the linear programming model is deterministic; that is, $D_{ik} = 0$ or 1. Furthermore, this technique is valid without the assumption (used for the average cost case) that the Markov chain associated with every transition matrix is irreducible.

III.4.3.3. Finite-Period Markov Decision Processes and the Method of Successive Approximations

I now turn our attention to an approach, called the *method of successive approximations,* for *quickly* finding at least an *approximation* to an optimal policy.

We have assumed that the Markov decision process will be operating indefinitely, and we have sought an optimal policy for such a process. The basic idea of the method of successive approximations is to instead find an optimal policy for the decisions to make in the first period when the process has only *n* time periods to go before termination, starting with $n = 1$, then $n = 2$, then $n = 3$, and so on. As *n* grows large, the corresponding optimal policies will converge to an optimal policy for the infinite-period problem of interest. Thus, the policies obtained for $n = 1, 2, 3, \ldots$ provide *successive approximations* that lead to the desired optimal policy.

The reason that this approach is attractive is that we already have a quick method of finding an optimal policy when the process has only *n* periods to go, namely, probabilistic dynamic programming.

In particular, for $i = 0, 1, \ldots, M$, let V_i^n be the expected total discounted cost of following an optimal policy, given that process starts in state *i* and has only *n* periods to go.

By the *principle of optimality* for dynamic programming, the V_i^n are obtained from the recursive relationship

$$V_i^n = \min_k \left\{ C_{ik} + \alpha \sum_{j=0}^{M} p_{ij}(k) V_j^{n-1} \right\}, \qquad \text{for } i = 0, 1, \ldots, M.$$

The minimizing value of *k* provides the optimal decision to make in the first period when the process starts in state *i*.

To get started, with $n = 1$, all the $V_i^0 = 0$ so that

$$V_i^1 = \min_K \{C_{ik}\}, \qquad \text{for } i = 0, 1, \ldots, M.$$

Although the method of successive approximations may not lead to an optimal policy for the infinite-period problem after only a few iterations, it has one distinct advantage over the policy improvement and linear programming techniques. It never requires solving a system of simultaneous equations, so each iteration can be performed simply and quickly.

Furthermore, if the Markov decision process actually does have just *n* periods to go, *n* iterations of this method definitely will lead to an optimal policy. (For an *n*-period problem, it is permissible to set $\alpha = 1$, that is, no discounting, in which case the objective is to minimize the expected total cost over *n* periods.)

Chapter IV
Reinforcement Learning

IV.1. Introduction

Due to well known scalability problems with MDP control framework, a MDP algorithms is not suitable to be implemented in real time systems, as the one considered in this work. In fact, the path selection engine in the NGHN QoS controller is in charge of deciding the path for a new flow as soon as it receives a new flow request. In addition, due to the frequent link faults in the considered home networks, also re-routing has to be calculated in real time to avoid loss of packet during the handover from the old path to the new path.

Reinforcement learning (RL) is a control framework that can be easily built on a MDP control framework of a system and produces interesting results that can be obtained with low computation complexity. For this reason, the RL approach is presented here as it is used in this work to derive, form the general MDP control framework, a RL new algorithm that can be implemented in real time NGHN controllers and provides, at the same time, results that are very close to the ones that are obtained with the optimal MDP controller.

IV.2. An introduction to Reinforcement Learning

Reinforcement learning [15] is learning what to do so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics (namely trial-and-error search and delayed reward) are the two most important distinguishing features of reinforcement learning.

Reinforcement learning is defined not by characterizing learning methods, but by characterizing a learning *problem*. Any method that is well suited to solving that problem, we consider to be a reinforcement learning method. A full specification of the reinforcement learning problem in terms of optimal control of Markov decision processes is presented later, but the basic idea is simply to capture the most important aspects of the real problem facing a learning ***agent*** interacting with its ***environment*** to achieve a goal. Clearly, such an agent must be able to sense the state of the environment to some extent and must be able to take actions that affect the state. The agent also must have a goal or goals relating to the state of the environment. The formulation is intended to include just these three aspects (sensation, action, and goal) in their simplest possible forms without trivializing any of them.

One of the challenges that arise in reinforcement learning and not in other kinds of learning is the trade-off between exploration and exploitation. To obtain a lot of reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing reward. But to discover such actions, it has to try actions that it has not selected before. The agent has to *exploit* what it already knows in order to obtain reward, but it also has to *explore* in order to make better action selections in the future. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task. The agent must try a variety of actions *and* progressively favor those that appear to be best. On a stochastic task, each action must be tried many times to gain a reliable estimate its expected reward. The exploration-exploitation dilemma has been intensively studied by mathematicians for many decades.

Another key feature of reinforcement learning is that it explicitly considers the *whole* problem of a goal-directed agent interacting with an uncertain environment. All reinforcement learning agents have explicit goals, can sense aspects of their environments, and can choose actions to influence their environments. Moreover, it is usually assumed from the beginning that the agent has to operate despite significant uncertainty about the environment it faces. When reinforcement learning involves planning, it has to address the interplay between planning and real-time action selection, as well as the question of how environmental models are acquired and improved. When reinforcement learning involves supervised learning, it does so for specific reasons that determine which capabilities are critical and which are not. For learning research to make progress, important subproblems have to be isolated and studied, but they should be subproblems that play clear roles in complete, interactive, goal-seeking agents, even if all the details of the complete agent cannot yet be filled in.

IV.2.1 Elements of RL

Beyond the agent and the environment, one can identify four main subelements of a reinforcement learning system: a *policy*, a *reward function*, a *value function*, and, optionally, a *model* of the environment.

A *policy* defines the learning agent's way of behaving at a given time. Roughly speaking, a policy is a mapping from perceived states of the environment to actions to be taken when in those states. It corresponds to what in psychology would call a set of stimulus-response rules or associations. In some cases the policy may be a simple function or lookup table, whereas in others it may involve extensive computation such as a search process. The policy is the core of a reinforcement learning agent in the sense that it alone is sufficient to determine behavior. In general, policies may be stochastic.

A *reward function* defines the goal in a reinforcement learning problem. Roughly speaking, it maps each perceived state (or state-action pair) of the environment to a single number, a *reward*, indicating the intrinsic desirability of that state. A reinforcement learning agent's sole objective is to maximize the total reward it receives in the long run. The reward function defines what are the good and bad events for the agent. In a biological system, it would not be inappropriate to identify rewards with pleasure and pain. They are the immediate and defining features of the problem faced by the agent. As such, the reward function must necessarily be unalterable by the agent. It may, however, serve as a basis for altering the policy. For example, if an action selected by the policy is followed by low reward, then the policy may be changed to select some other action in that situation in the future. In general, reward functions may be stochastic.

Whereas a reward function indicates what is good in an immediate sense, a *value function* specifies what is good in the long run. Roughly speaking, the *value* of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. Whereas rewards determine the immediate, intrinsic desirability of environmental states, values indicate the *long-term* desirability of states after taking into account the states that are likely to follow, and the rewards available in those states. For example, a state might always yield a low immediate reward but still have a high value because it is regularly followed by other states that yield high rewards. Or the reverse could be true. To make a human analogy, rewards are like pleasure (if high) and pain (if low), whereas values correspond to a more refined and farsighted judgment of how pleased or displeased we are that our environment is in a particular state. Expressed this way, it is clear that value functions formalize a basic and familiar idea.

Rewards are in a sense primary, whereas values, as predictions of rewards, are secondary. Without rewards there could be no values, and the only purpose of estimating values is to achieve more reward. Nevertheless, it is values with which we are most concerned when making and evaluating decisions. Action choices are made based on value judgments. We seek actions that bring about states of highest value, not highest reward, because these actions obtain the greatest amount of reward for us over the long run. In decision-making and planning, the derived quantity called value is the one with which we are most concerned. Unfortunately, it is much harder to determine values than it is to determine rewards. Rewards are basically given directly by the environment, but values must be estimated and reestimated from the sequences of observations an agent makes over its entire lifetime. In fact, the most important component of almost all reinforcement learning algorithms is a method for efficiently estimating values. The central role of value estimation is arguably the most important thing we have learned about reinforcement learning over the last few decades.

The fourth and final element of some reinforcement learning systems is a *model* of the environment. This is something that mimics the behavior of the environment. For example, given a state and action, the model might predict the resultant next state and next reward. Models are used for *planning*, by which we mean any way of deciding on a course of action by considering possible future situations before they are actually experienced. The incorporation of models and planning into reinforcement learning systems is a relatively new development. Early reinforcement learning systems were explicitly trial-and-error learners; what they did was viewed as almost the *opposite* of planning. Nevertheless, it gradually became clear that reinforcement learning methods are closely related to dynamic programming methods, which do use models, and that they in turn are closely related to state-space planning methods. Modern reinforcement learning spans the spectrum from low-level, trial-and-error learning to high-level, deliberative planning.

IV.2.2 Evaluative feedback

The most important feature distinguishing reinforcement learning from other types of learning is that it uses training information that *evaluates* the actions taken rather than *instructs* by giving correct actions. This is what creates the need for active exploration, for an explicit trial-and-error search for good behavior. Purely evaluative feedback indicates how good the action taken is, but not whether it is the best or the worst action possible. Evaluative feedback is the basis of methods for function optimization, including evolutionary methods. Purely instructive feedback, on the other hand, indicates the correct action to take, independently of the action actually taken. Thus, evaluative feedback depends entirely on the action taken, whereas instructive feedback is independent of the action taken.

Let’s consider the following learning problem. You are faced repeatedly with a choice among *n* different options, or actions. After each choice you receive a numerical reward chosen from a stationary probability distribution that depends on the action you selected. Your objective is to maximize the expected total reward over some time period. Each action selection is called a *play*.

This is the original form of the *n-armed bandit problem*. In this *n*-armed bandit problem, each action has an expected or mean reward given that that action is selected; let’s call this the *value* of that action. If you knew the value of each action, then it would be trivial to solve the *n*-armed bandit problem: you would always select the action with highest value. It is assumed here that you do not know the action values with certainty, although you may have estimates.

If you maintain estimates of the action values, then at any time there is at least one action whose estimated value is greatest. This is called a *greedy* action. If you select a greedy action, you are *exploiting* your current knowledge of the values of the actions. If instead you select one of the nongreedy actions, then you are *exploring* because this enables you to improve your estimate of the nongreedy action's value. Exploitation is the right thing to do to maximize the expected reward on the one play, but exploration may produce the greater total reward in the long run. For example, suppose the greedy action's value is known with certainty, while several other actions are estimated to be nearly as good but with substantial uncertainty. The uncertainty is such that at least one of these other actions probably is actually better than the greedy action, but you don't know which one. If you have many plays yet to make, then it may be better to explore the nongreedy actions and discover which of them are better than the greedy action. Reward is lower in the short run, during exploration, but higher in the long run because after you have discovered the better actions, you can exploit *them*. Because it is not possible both to explore and to exploit with any single action selection, one often refers to the "conflict" between exploration and exploitation.

In any specific case, whether it is better to explore or exploit depends in a complex way on the precise values of the estimates, uncertainties, and the number of remaining plays. There are many sophisticated methods for balancing exploration and exploitation for particular mathematical formulations of the *n*-armed bandit and related problems. However, most of these methods make strong assumptions about stationarity and prior knowledge that are either violated or impossible to verify in applications and in the full reinforcement learning problem that we consider in subsequent chapters. The guarantees of optimality or bounded loss for these methods are of little comfort when the assumptions of their theory do not apply.

Let’sdenote the true (actual) value of action *a* as $Q^*(a)$, and the estimated value at the *t* th play as $Q_t(a)$. Recall that the true value of an action is the mean reward received when that action is selected. One natural way to estimate this is by averaging the rewards actually received when the action was selected. In other words, if at the *t*th play action *a* has been chosen k_a times prior to *t*, yielding rewards $r_1 + r_2 + ... + r_{k_a}$, then its value is estimated to be

$$Q_t(a) = \frac{r_1 + r_2 + ... + r_{k_a}}{k_a} . \qquad (4.1)$$

If $k_a = 0$, then it is possible to define $Q_t(a)$ instead as some default value, such as $Q_0(a) = 0$. As $k_a \to \infty$, by the law of large numbers $Q_t(a)$ converges to $Q^*(a)$. This is called the *sample-average* method for estimating action values because each estimate is a simple average of the sample of relevant rewards. Of course this is just one way to estimate action values, and not necessarily the best one. Nevertheless, for now let’s stay with this simple estimation method and turn to the question of how the estimates might be used to select actions.

The simplest action selection rule is to select the action (or one of the actions) with highest estimated action value, that is, to select on play *t* one of the greedy actions, a^*, for which $Q_t(a^*) = \max_a Q_t(a)$. This method always exploits current knowledge to maximize immediate reward; it spends no time at all sampling apparently inferior actions to see if they might really be better. A simple alternative is to behave greedily most of the time, but every once in a while, say with small probability ε, instead select an action at random, uniformly, independently of the action-value estimates. The methods using this near-greedy action selection rule are called ε*-greedy* methods. An advantage of these methods is that, in the limit as the number of plays increases, every action will be sampled an infinite number of times, guaranteeing that $k_a \rightarrow \infty$ for all *a*, and thus ensuring that all the $Q_t(a)$ converge to $Q^*(a)$. This of course implies that the probability of selecting the optimal action converges to greater than 1-ε, that is, to near certainty. These are just asymptotic guarantees, however, and say little about the practical effectiveness of the methods.

The advantage of ε*-greedy* over greedy methids depends on the task. In general we can say that Reinforcement Learning requires a balance between exploration and exploitation.

IV.2.3 Incremental Implementation

The action-value methods discussed so far all estimate action values as sample averages of observed rewards. The obvious implementation is to maintain, for each action *a*, a record of all the rewards that have followed the selection of that action. Then, when the estimate of the value of action a is needed at time *t*, it can be computed according to (4.1). A problem with this straightforward implementation is that its memory and computational requirements grow over time without bound. That is, each additional reward following a selection of action *a* requires more memory to store it and results in more computation being required to determine $Q_t(a)$.

As you might suspect, this is not really necessary. It is easy to devise incremental update formulas for computing averages with small, constant computation required to process each new reward. For some action, let Q_k denote the average of its first *k* rewards (not to be confused with $Q_k(a)$, the average for action *a* at the *k*th *play*). Given this average and a (*k*+1)st reward, r_{k+1}, then the average of all *k*+1 rewards can be computed by:

$$Q_{k+1} = Q_k + \frac{1}{k+1}\left[r_{k+1} - Q_k\right] \qquad (4.2)$$

which holds even for k = 0, obtaining $Q_1 = r_1$ for arbitrary Q_0. This implementation requires memory only for Q_k and k, and only a small computation for each new reward. The general form for the update rule is:

$$NewEstimate \leftarrow OldEstimate + StepSize[Target - OldEstimate] \qquad (4.3)$$

The expression $[Target - OldEstimate]$ is an *error* in the estimate. It is reduced by taking a step toward the "Target." The target is presumed to indicate a desirable direction in which to move, though it may be noisy. In the case above, for example, the target is the (k+1)st reward.

Note that the step-size parameter (*StepSize*) used in the incremental method described above changes from time step to time step. In processing the kth reward for action a, that method uses a step-size parameter of $\frac{1}{k}$. In this work I denote the step-size parameter by the symbol α or, more generally, by $\alpha_k(a)$. For example, the above incremental implementation of the sample-average method is described by the equation $\alpha_k(a) = \frac{1}{k_a}$. Accordingly, I sometimes use the informal shorthand $\alpha(a) = \frac{1}{k}$ to refer to this case, leaving the action dependence implicit.

IV.2.4 Tracking a Nonstationary problem

The averaging methods discussed so far are appropriate in a stationary environment, but not if the bandit is changing over time. But we may often encounter reinforcement learning problems that are effectively nonstationary. In such cases it makes sense to weight recent rewards more heavily than long-past ones. One of the most popular ways of doing this is to use a constant step-size parameter. For example, the incremental update rule (4.3) for updating an average Q_k of the $k$past rewards is modified to be:

$$Q_{k+1} = Q_k + \alpha[r_{k+1} - Q_k] \qquad (4.4)$$

Where the step-size parameter, α, $0 < \alpha \leq 1$, is costant. This results in Q_k being a weighted average of past reward and the initial estimate Q_0.

IV.3. The Reinforcement Learning problem

The reinforcement learning problem is meant to be a straightforward framing of the problem of learning from interaction to achieve a goal. The learner and decision-maker is called the *agent*. The thing it interacts with, comprising everything outside the agent, is called the *environment*. These interact continually, the agent selecting actions and the environment responding to those actions and presenting new situations to the agent. The environment also gives rise to rewards, special numerical values that the agent tries to maximize over time. A complete specification of an environment defines a *task*, one instance of the reinforcement learning problem.

More specifically, the agent and environment interact at each of a sequence of discrete time steps, t = 0,1,2,3,… . At each time step t, the agent receives some representation of the environment's *state*, $S_t \in S$, where S is the set of possible states, and on that basis selects an *action*, $a_t \in A(s_t)$, where $A(s_t)$ is the set of actions available in state s_t. One time step later, in part as a consequence of its action, the agent receives a numerical *reward*, $r_{t+1} \in R$, and finds itself in a new state, s_{t+1}. Figure 10 diagrams the agent-environment interaction.

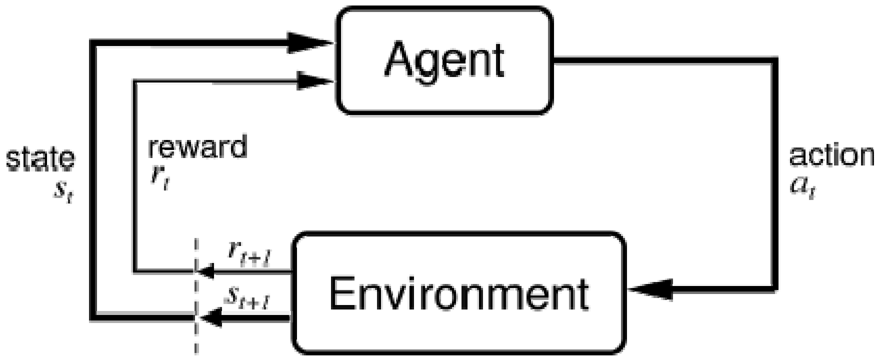


Figure 10 - The agent-environment interaction in RL

At each time step, the agent implements a mapping from states to probabilities of selecting each possible action. This mapping is called the agent's *policy* and is denoted π_t , where $\pi_t(s,a)$ is the probability that $a_t = a$ if $s_t = s$. Reinforcement learning methods specify how the agent changes its policy as a result of its experience. The agent's goal, roughly speaking, is to maximize the total amount of reward it receives over the long run. This means maximizing not immediate reward, but cumulative reward in the long run.

IV.3.1 Returns

So far we have been imprecise regarding the objective of learning. We have said that the agent's goal is to maximize the reward it receives in the long run. How might this be formally defined? If the sequence of rewards received after time step *t* is denoted $r_{t+1}, r_{t+2}, r_{t+3}, \ldots$, then what precise aspect of this sequence do we wish to maximize? In general, we seek to maximize the *expected return*, where the return, R_t, is defined as some specific function of the reward sequence. In the simplest case the return is the sum of the rewards:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_T \qquad (4.5)$$

where *T* is a final time step. This approach makes sense in applications in which there is a natural notion of final time step, that is, when the agent-environment interaction breaks naturally into subsequences, which I call *episodes*, such as plays of a game, trips through a maze, or any sort of repeated interactions. Each episode ends in a special state called the *terminal state*, followed by a reset to a standard starting state or to a sample from a standard distribution of starting states. Tasks with episodes of this kind are called *episodic tasks*. In episodic tasks we sometimes need to distinguish the set of all nonterminal states, denoted *S*, from the set of all states plus the terminal state, denoted S^+.

On the other hand, in many cases the agent-environment interaction does not break naturally into identifiable episodes, but goes on continually without limit. For example, this would be the natural way to formulate a continual process-control task, or an application to a robot with a long life span. I call these *continuing tasks*. The return formulation (4.5) is problematic for continuing tasks because the final time step would be $T = \infty$, and the return, which is what we are trying to maximize, could itself easily be infinite. (For example, suppose the agent receives a reward of +1 at each time step.) Thus, in this work I usually use a definition of return that is slightly more complex conceptually but much simpler mathematically.

The additional concept that I need to introduce is that of *discounting*. According to this approach, the agent tries to select actions so that the sum of the discounted rewards it receives over the future is maximized. In particular, it chooses a_t to maximize the expected *discounted return*:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (4.6)$$

where γ is a parameter, $0 \le \gamma \le 1$, called the *discount rate*.

The discount rate determines the present value of future rewards: a reward received *k* time steps in the future is worth only γ^{k-1} times what it would be worth if it were received immediately. If $\gamma < 1$, the infinite sum has a finite value as long as the reward sequence $\{r_k\}$ is bounded. If $\gamma = 0$, the agent is "myopic" in being concerned only with maximizing immediate rewards: its objective in this case is to learn how to choose a_t so as to maximize only r_{t+1}. If each of the agent's actions happened to influence only the immediate reward, not future rewards as well, then a myopic agent could maximize (4.6) by separately maximizing each immediate reward. But in general, acting to maximize immediate reward can reduce access to future rewards so that the return may actually be reduced. As γ approaches 1, the objective takes future rewards into account more strongly: the agent becomes more farsighted.

IV.3.1.1. Unified notation for episodic and continuing tasks

As described previously, there are two kinds of reinforcement learning tasks, one in which the agent-environment interaction naturally breaks down into a sequence of separate episodes (episodic tasks), and one in which it does not (continuing tasks). The former case is mathematically easier because each action affects only the finite number of rewards subsequently received during the episode. It is therefore useful to establish one notation that enables us to talk precisely about both cases simultaneously.

To be precise about episodic tasks requires some additional notation. Rather than one long sequence of time steps, we need to consider a series of episodes, each of which consists of a finite sequence of time steps. We number the time steps of each episode starting anew from zero. Therefore, we have to refer not just to s_t, the state representation at time *t*, but to $s_{t,i}$, the state representation at time *t* of episode *i* (and similarly for $a_{t,i}$, $r_{t,i}$, $\pi_{t,i}$, T_i, etc.). However, it turns out that, when we discuss episodic tasks we will almost never have to distinguish between different episodes. We will almost always be considering a particular single episode, or stating something that is true for all episodes. Accordingly, in practice we will almost always abuse notation slightly by dropping the explicit reference to episode number. That is, I will write s_t to refer to $s_{t,i}$, and so on.

We need one other convention to obtain a single notation that covers both episodic and continuing tasks. We have defined the return as a sum over a finite number of terms in one case (4.5) and as a sum over an infinite number of terms in the other (4.6). These can be unified by considering episode termination to be the entering of a special *absorbing state* that transitions only to itself and that generates only rewards of zero.

IV.4. Modelling the environment as a Markov chain

In the reinforcement learning framework, the agent makes its decisions as a function of a signal from the environment called the environment's *state*. By "the state" we mean whatever information is available to the agent. We assume that the state is given by some preprocessing system that is nominally part of the environment. The state signal should not be expected to inform the agent of everything about the environment, or even everything that would be useful to it in making decisions.

What we would like, ideally, is a state signal that summarizes past sensations compactly, yet in such a way that all relevant information is retained. This normally requires more than the immediate sensations, but never more than the complete history of all past sensations. A state signal that succeeds in retaining all relevant information is said to be *Markov*, or to have *the Markov property*.

If an environment has the Markov property, then its one-step dynamics allow to predict the next state and expected next reward given the current state and action. One can show that, by iteration, one can predict all future states and expected rewards from knowledge only of the current state as well as would be possible given the complete history up to the current time. It also follows that Markov states provide the best possible basis for choosing actions. That is, the best policy for choosing actions as a function of a Markov state is just as good as the best policy for choosing actions as a function of complete histories.

A reinforcement learning task that satisfies the Markov property is called a *Markov decision process*, or *MDP*. If the state and action spaces are finite, then it is called a *finite Markov decision process (finite MDP).* Finite MDPs are particularly important to the theory of reinforcement learning.

A particular finite MDP is defined by its state and action sets and by the one-step dynamics of the environment. Given any state and action, *s* and *a*, the probability of each possible next state, *s'*, is

$$P_{ss'}^{a} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\} \qquad (4.7)$$

These quantities are called *transition probabilities*. Similarly, given any current state and action, *s* and *a*, together with any next state, *s'*, the expected value of the next reward is

$$R_{ss'}^{a} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\} \qquad (4.8)$$

These quantities, $P_{ss'}^{a}$ and $R_{ss'}^{a}$, completely specify the most important aspects of the dynamics of a finite MDP (only information about the distribution of rewards around the expected value is lost).

Almost all reinforcement learning algorithms are based on estimating *value functions*--functions of states (or of state-action pairs) that estimate *how good* it is for the agent to be in a given state (or how good it is to perform a given action in a given state). The notion of "how good" here is defined in terms of future rewards that can be expected, or, to be precise, in terms of expected return. Of course the rewards the agent can expect to receive in the future depend on what actions it will take. Accordingly, value functions are defined with respect to particular policies.

Recall that a policy, π, is a mapping from each state, $s \in S$, and action, $a \in A(s)$, to the probability $\pi(s,a)$ of taking action *a* when in state *s*. Informally, the *value* of a state *s* under a policy π, denoted $V^{\pi}(s)$, is the expected return when starting in *s* and following π thereafter. For MDPs, we can define $V^{\pi}(s)$ formally as

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_{t=s}\right\} \qquad (4.9)$$

where $E_\pi\{\ \}$ denotes the expected value given that the agent follows policy π, and *t* is any time step. Note that the value of the terminal state, if any, is always zero. We call the function V^π the *state-value function for policy* π.

Similarly, we define the value of taking action *a* in state *s* under a policy π, denoted $Q^\pi(s,a)$, as the expected return starting from *s*, taking the action *a*, and thereafter following policy π:

$$Q^\pi(s,a) = E_\pi\{R_t \mid s_t = s, a_t = a\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\} \qquad (4.10)$$

We call Q^π the *action-value function for policy* π.

The value functions V^π and Q^π can be estimated from experience.

A fundamental property of value functions used throughout reinforcement learning and dynamic programming is that they satisfy particular recursive relationships. For any policy π and any state *s*, the following consistency condition holds between the value of *s* and the value of its possible successor states:

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P_{ss'}^a \left[R_{ss'}^a + \gamma V^\pi(s')\right] \qquad (4.11)$$

where it is implicit that the actions, *a*, are taken from the set *A*(*s*), and the next states, *s*', are taken from the set *S*, or from S^+ in the case of an episodic problem. Equation (4.11) is the *Bellman equation for* V^π. It expresses a relationship between the value of a state and the values of its successor states. The value function V^π is the unique solution to its Bellman equation.

IV.4.1 Optimal value functions and approssimations

Solving a reinforcement learning task means, roughly, finding a policy that achieves a lot of reward over the long run. For finite MDPs, we can precisely define an optimal policy in the following way. Value functions define a partial ordering over policies. A policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states. In other words, π > π' if and only if $V^\pi(s) \geq V^{\pi'}(s)$ for all $s \in S$. There is always at least one policy that is better than or equal to all other policies. This is an *optimal policy*. Although there may be more than one, we denote all the optimal policies by π*. They share the same state-value function, called the *optimal state-value function*, denoted V*, and defined as

$$V^*(s) = \max_{\pi} V^{\pi}(s) \qquad (4.12)$$

for all $s \in S$.

Optimal policies also share the same *optimal action-value function,* denoted Q*, and defined as

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \qquad (4.13)$$

for all $s \in S$ and $a \in A(s)$. For the state-action pair (*s*,*a*), this function gives the expected return for taking action *a* in state *s* and thereafter following an optimal policy. Thus, we can write Q* in terms of V* as follows

$$Q^*(s,a) = E\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\} \qquad (4.14)$$

Because V* is the value function for a policy, it must satisfy the self-consistency condition given by the Bellman equation for state values (4.11). Because it is the optimal value function, however, V* 's consistency condition can be written in a special form without reference to any specific policy. This is the Bellman equation for V*, or the *Bellman optimality equation*. Intuitively, the Bellman optimality equation expresses the fact that the value of a state under an optimal policy must equal the expected return for the best action from that state:

$$V^*(s) = \max_{a} E\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\} \qquad (4.15)$$

and

$$V^*(s) = \max_{a \in A(s)} \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma V^*(s')\right] \qquad (4.16)$$

The last two equations are two forms of the Bellman optimality equation for V*. The Bellman optimality equation for Q* is:

$$Q^*(s,a) = E\left\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1},a) \mid s_t = s, a_t = a\right\} = \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + \gamma \max_{a'} Q^*(s',a')\right]$$
(4.17)

For finite MDPs, the Bellman optimality equation (4.16) has a unique solution independent of the policy. The Bellman optimality equation is actually a system of equations, one for each state, so if there are N states, then there are N equations in N unknowns. If the dynamics of the environment are known ($R^a_{ss'}$ and $P^a_{ss'}$), then in principle one can solve this system of equations for V* using any one of a variety of methods for solving systems of nonlinear equations. One can solve a related set of equations for Q*.

Once one has V*, it is relatively easy to determine an optimal policy. For each state *s*, there will be one or more actions at which the maximum is obtained in the Bellman optimality equation. Any policy that assigns nonzero probability only to these actions is an optimal policy. You can think of this as a one-step search. If you have the optimal value function, V*, then the actions that appear best after a one-step search will be optimal actions. Another way of saying this is that any policy that is *greedy* with respect to the optimal evaluation function V* is an optimal policy. The term greedy is used in computer science to describe any search or decision procedure that selects alternatives based only on local or immediate considerations, without considering the possibility that such a selection may prevent future access to even better alternatives. Consequently, it describes policies that select actions based only on their short-term consequences. The beauty of V* is that if one uses it to evaluate the short-term consequences of actions--specifically, the one-step consequences--then a greedy policy is actually optimal in the long-term sense in which we are interested because V* already takes into account the reward consequences of all possible future behavior. By means of V*, the optimal expected long-term return is turned into a quantity that is locally and immediately available for each state. Hence, a one-step-ahead search yields the long-term optimal actions.

Having Q* makes choosing optimal actions still easier. With Q*, the agent does not even have to do a one-step-ahead search: for any state *s*, it can simply find any action that maximizes $Q^*(s,a)$. The action-value function effectively caches the results of all one-step-ahead searches. It provides the optimal expected long-term return as a value that is locally and immediately available for each state-action pair. Hence, at the cost of representing a function of state-action pairs, instead of just of states, the optimal action-value function allows optimal actions to be selected without having to know anything about possible successor states and their values, that is, without having to know anything about the environment's dynamics.

Explicitly solving the Bellman optimality equation provides one route to finding an optimal policy, and thus to solving the reinforcement learning problem. However, this solution is rarely directly useful. It is akin to an exhaustive search, looking ahead at all possibilities, computing their probabilities of occurrence and their desirabilities in terms of expected rewards. This solution relies on at least three assumptions that are rarely true in practice: (1) we accurately know the dynamics of the environment; (2) we have enough computational resources to complete the computation of the solution; and (3) the Markov property. For the kinds of tasks in which we are interested, one is generally not able to implement this solution exactly because various combinations of these assumptions are violated.

We have defined optimal value functions and optimal policies. Clearly, an agent that learns an optimal policy has done very well, but in practice this rarely happens. For the kinds of tasks in which we are interested, optimal policies can be generated only with extreme computational cost. As we discussed above, even if we have a complete and accurate model of the environment's dynamics, it is usually not possible to simply compute an optimal policy by solving the Bellman optimality equation.

A critical aspect of the problem facing the agent is always the computational power available to it, in particular, the amount of computation it can perform in a single time step.

The memory available is also an important constraint. A large amount of memory is often required to build up approximations of value functions, policies, and models.

Our framing of the reinforcement learning problem forces us to settle for approximations. The on-line nature of reinforcement learning makes it possible to approximate optimal policies in ways that put more effort into learning to make good decisions for frequently encountered states, at the expense of less effort for infrequently encountered states. This is one key property that distinguishes reinforcement learning from other approaches to approximately solving MDPs.

IV.5. RL solutions methods

There exist three fundamental classes of methods for solving the reinforcement learning problem:

- Dynamic programming;
- Monte Carlo methods;
- Temporal-Difference learning.

Each class of methods has its strengths and weaknesses. Dynamic programming methods are well developed mathematically, but require a complete and accurate model of the environment. Monte Carlo methods don't require a model and are conceptually simple, but are not suited for step-by-step incremental computation. Finally, temporal-difference methods require no model and are fully incremental, but are more complex to analyze. The methods also differ in several ways with respect to their efficiency and speed of convergence.

In the following sections I introduce all these methods, but I will focus in particular on Temporal-Difference learning, which is the one used for the fault-tolerant routing algorithm object of the present work.

IV.5.1 Dynamic Programming

The term dynamic programming (DP) refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process. Classical DP algorithms are of limited utility in reinforcement learning both because of their assumption of a perfect model and because of their great computational expense, but they are still important theoretically. DP provides an essential foundation for the understanding of the other two methods presented in this chapter. In fact, all of these methods can be viewed as attempts to achieve much the same effect as DP, only with less computation and without assuming a perfect model of the environment.

The key idea of DP, and of reinforcement learning generally, is the use of value functions to organize and structure the search for good policies.

The basic ideas and algorithms of dynamic programming as they relate to solving finite MDPs are:

- *Policy evaluation:* refers to the (typically) iterative computation of the value functions for a given policy.
- *Policy improvement:* refers to the computation of an improved policy given the value function for that policy.
- Putting these two computations together, we obtain *policy iteration* and *value iteration,* the two most popular DP methods. Either of these can be used to reliably compute optimal policies and value functions for finite MDPs given complete knowledge of the MDP
- Insight into DP methods and, in fact, into almost all reinforcement learning methods, can be gained by viewing them as *generalized policy iteration* (GPI). GPI is the general idea of two interacting processes revolving around an approximate policy and an approximate value function. One process takes the policy as given and performs some form of policy evaluation, changing the value function to be more like the true value function for the policy. The other process takes the value function as given and performs some form of policy improvement, changing the policy to make it better, assuming that the value function is its value function. Although each process changes the basis for the other, overall they work together to find a joint solution: a policy and value function that are unchanged by either process and, consequently, are optimal

DP may not be practical for very large problems, but compared with other methods for solving MDPs, DP methods are actually quite efficient. If we ignore a few technical details, then the (worst case) time DP methods take to find an optimal policy is polynomial in the number of states and actions. Linear programming methods can also be used to solve MDPs, and in some cases their worst-case convergence guarantees are better than those of DP methods. But linear programming methods become impractical at a much smaller number of states than do DP methods (by a factor of about 100). For the largest problems, only DP methods are feasible. DP is sometimes thought to be of limited applicability because of the *curse of dimensionality,* the fact that the number of states often grows exponentially with the number of state variables. Large state sets do create difficulties, but these are inherent difficulties of the problem, not of DP as a solution method. In fact, DP is comparatively better suited to handling large state spaces than competing methods such as direct search and linear programming.

IV.5.2 Monte Carlo methods

Here I consider the first learning methods for estimating value functions and discovering optimal policies. Unlike the previous section, here we do not assume complete knowledge of the environment. Monte Carlo methods require only *experience*-sample sequences of states, actions, and rewards from on-line or simulated interaction with an environment. Learning from *on-line* experience is striking because it requires no prior knowledge of the environment's dynamics, yet can still attain optimal behavior. Learning from *simulated* experience is also powerful. Although a model is required, the model need only generate sample transitions, not the complete probability distributions of all possible transitions that is required by dynamic programming (DP) methods. In surprisingly many cases it is easy to generate experience sampled according to the desired probability distributions, but infeasible to obtain the distributions in explicit form.

Monte Carlo methods are ways of solving the reinforcement learning problem based on averaging sample returns. To ensure that well-defined returns are available, we define Monte Carlo methods only for episodic tasks. That is, we assume experience is divided into episodes, and that all episodes eventually terminate no matter what actions are selected. It is only upon the completion of an episode that value estimates and policies are changed. Monte Carlo methods are thus incremental in an episode-by-episode sense, but not in a step-by-step sense.

As for DP algorithms, Monte Carlo method is used to compute policy evaluation, policy improvement and generalized policy iteration. Each of these ideas taken from DP is extended to the Monte Carlo case in which only sample experience is available.

In addition, we can distinghuish among *on-policy* methods, which attempt to evaluate or improve the policy that is used to make decisions, and *off.policy* methods, in which the policy used to generate behavior, called the *behavior* policy, may in fact be unrelated to the policy that is evaluated and improved, called the *estimation* policy. An advantage of this separation is that the estimation policy may be deterministic (e.g., greedy), while the behavior policy can continue to sample all possible actions.

To conclude, we can say that Monte Carlo methods learn value functions and optimal policies from experience in the form of *sample episodes*. This gives them at least three kinds of advantages over DP methods. First, they can be used to learn optimal behavior directly from interaction with the environment, with no model of the environment's dynamics. Second, they can be used with simulation or *sample models*. For surprisingly many applications it is easy to simulate sample episodes even though it is difficult to construct the kind of explicit model of transition probabilities required by DP methods. Third, it is easy and efficient to *focus* Monte Carlo methods on a small subset of the states. A region of special interest can be accurately evaluated without going to the expense of accurately evaluating the rest of the state set.

IV.5.3 Temporal-Difference Learning

TD learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas. Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environment's dynamics. Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome (they bootstrap).

TD methods have an advantage over DP methods in that they do not require a model of the environment, of its reward and next-state probability distributions.

The next most obvious advantage of TD methods over Monte Carlo methods is that they are naturally implemented in an on-line, fully incremental fashion. With Monte Carlo methods one must wait until the end of an episode, because only then is the return known, whereas with TD methods one need wait only one time step. Surprisingly often this turns out to be a critical consideration. Some applications have very long episodes, so that delaying all learning until an episode's end is too slow. Other applications are continuing tasks, like the one considered in this work, and have no episodes at all. Finally, as we noted in the previous section, some Monte Carlo methods must ignore or discount episodes on which experimental actions are taken, which can greatly slow learning. TD methods are much less susceptible to these problems because they learn from each transition regardless of what subsequent actions are taken.

But are TD methods sound? Certainly it is convenient to learn one guess from the next, without waiting for an actual outcome, but can we still guarantee convergence to the correct answer? Happily, the answer is yes. For any fixed policy π, the TD algorithm described above has been proved to converge to V^{π}, in the mean for a constant step-size parameter if it is sufficiently small, and with probability 1 if the step-size parameter decreases according to the usual stochastic approximation conditions. If both TD and Monte Carlo methods converge asymptotically to the correct predictions, then a natural next question is "Which gets there first?" At the current time this is an open question in the sense that no one has been able to prove mathematically that one method converges faster than the other. In practice, however, TD methods have usually been found to converge faster than constant-α MC methods on stochastic tasks.

The methods presented in the following sub-sections are today the most widely used reinforcement learning methods. This is probably due to their great simplicity: they can be applied on-line, with a minimal amount of computation, to experience generated from interaction with an environment; they can be expressed nearly completely by single equations that can be implemented with small computer programs.

IV.5.4 TD prediction

Given some experience following a policy π, both methods update their estimate V of V^{π}. If a nonterminal state s_t is visited at time t, then both methods update their estimate $V(s_t)$ based on what happens after that visit. Roughly speaking, Monte Carlo methods wait until the return following the visit is known, then use that return as a target for $V(s_t)$. A simple every-visit Monte Carlo method suitable for nonstationary environments is

$$V(s_t) \leftarrow V(s_t) + \alpha\left[R_t - V(s_t)\right] \qquad (4.18)$$

where R_t is the actual return following time t and α is a constant step-size parameter.

Let us call this method *constant*-α *MC*. Whereas Monte Carlo methods must wait until the end of the episode to determine the increment to $V(s_t)$ (only then is R_t known), TD methods need wait only until the next time step. At time t+1 they immediately form a target and make a useful update using the observed reward r_{t+1} and the estimate $V(s_{t+1})$. The simplest TD method, known as *TD(0)*, is

$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right] \qquad (4.19)$$

Because the TD method bases its update in part on an existing estimate, we say that it is a *bootstrapping* method, like DP. We know

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} \qquad (4.20)$$

and

$$V^{\pi}(s) = E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) \mid s_t = s\right\} \qquad (4.21)$$

Roughly speaking, Monte Carlo methods use an estimate of (4.20) as a target, whereas DP methods use an estimate of (4.21) as a target. The Monte Carlo target is an estimate because the expected value in (4.20) is not known; a sample return is used in place of the real expected return. The DP target is an estimate not because of the expected values, which are assumed to be completely provided by a model of the environment, but because $V^{\pi}(s_{t+1})$ is not known and the current estimate, $V_t(s_{t+1})$, is used instead. The TD target is an estimate for both reasons: it samples the expected values in (4.21) and it uses the current estimate V_t instead of the true V^{π}. Thus, TD methods combine the sampling of Monte Carlo with the bootstrapping of DP. As we shall see, with care and imagination this can take us a long way toward obtaining the advantages of both Monte Carlo and DP methods.

IV.5.4.1. Sarsa: on-policy TD control

The first step is to learn an action-value function rather than a state-value function. In particular, for an on-policy method we must estimate $Q^{\pi}(s,a)$ for the current behavior policy π and for all states *s* and actions *a*. This can be done using essentially the same TD method described above for learning V^{π}. Recall that an episode consists of an alternating sequence of states and state-action pairs:

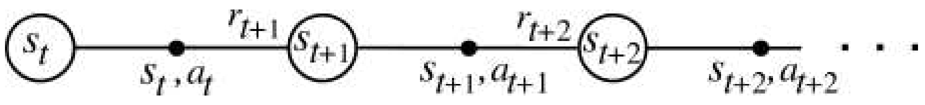


Figure 11 - State and state-action pairs sequence

Now we consider transitions from state-action pair to state-action pair, and learn the value of state-action pairs. Formally these cases are identical: they are both Markov chains with a reward process. The theorems assuring the convergence of state values under TD(0) also apply to the corresponding algorithm for action values:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right] \qquad (4.22)$$

This update is done after every transition from a nonterminal state s_t. If s_{t+1} is terminal, then $Q(s_{t+1}, a_{t+1})$ is defined as zero. This rule uses every element of the quintuple of events, $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, that make up a transition from one state-action pair to the next. This quintuple gives rise to the name *Sarsa* for the algorithm.

It is straightforward to design an on-policy control algorithm based on the Sarsa prediction method. As in all on-policy methods, we continually estimate Q^{π} for the behavior policy π, and at the same time change π toward greediness with respect to Q^{π}.

IV.5.4.2. Q-Learning: off-policy TD control

One of the most important breakthroughs in reinforcement learning was the development of an off-policy TD control algorithm known as *Q-learning*. Its simplest form, *one-step Q-learning*, is defined by

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a} Q(s_{t+1}, a) - Q(s_t, a_t) \right] \qquad (4.23)$$

where α is the learning rate, γ is the discount factor and r_{t+1} is the cost associated to s_{t+1}.

In particular, the learning rate α determines to what extent the newly acquired information will override the old information. A factor of 0 will make the agent not learn anything, while a factor of 1 would make the agent consider only the most recent information. The discount factor γ determines the importance of future rewards. A factor of 0 will make the agent "opportunistic" by only considering current rewards, while a factor approaching 1 will make it strive for a long-term high reward. If the discount factor meets or exceeds 1, the *Q* values will diverge.

In this case, the learned action-value function, *Q*, directly approximates *Q**, the optimal action-value function, independent of the policy being followed. This dramatically simplifies the analysis of the algorithm and enabled early convergence proofs. The policy still has an effect in that it determines which state-action pairs are visited and updated. However, all that is required for correct convergence is that all pairs continue to be updated. Under this assumption and a variant of the usual stochastic approximation conditions on the sequence of step-size parameters, Q_t has been shown to converge with probability 1 to *Q**. The Q-learning algorithm shown in procedural form is:

```
Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
    Initialize $s$
    Repeat (for each step of episode):
    Chose $a$ from $s$ using policy derived from $Q$ (e.g., ε-greedy)
    Take action $a$, observe $r$, $s$’
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$
until $s$ is terminal
```

IV.5.4.3. R-Learning for undiscounted continuing tasks

R-learning is an off-policy control method for the advanced version of the reinforcement learning problem in which one neither discounts nor divides experience into distinct episodes with finite returns. In this case one seeks to obtain the maximum reward per time step. The value functions for a policy, π, are defined relative to the average expected reward per time step under the policy, ρ^{π}:

$$\rho^{\pi} = \lim_{n\to\infty} \frac{1}{n} \sum_{t=1}^{n} E_{\pi}\{r_t\} \qquad (4.24)$$

assuming the process is ergodic (nonzero probability of reaching any state from any other under any policy) and thus that ρ^{π} does not depend on the starting state. From any state, in the long run the average reward is the same, but there is a transient. From some states better-than-average rewards are received for a while, and from others worse-than-average rewards are received. It is this transient that defines the value of a state:

$$\tilde{V}^{\pi}(s) = \sum_{k=1}^{\infty} E_{\pi}\left\{r_{t+k} - \rho^{\pi} \mid s_t = s\right\} \qquad (4.25)$$

and the value of a state-action pair is similarly the transient difference in reward when starting in that state and taking that action:

$$\tilde{Q}^{\pi}(s,a) = \sum_{k=1}^{\infty} E_{\pi}\left\{r_{t+k} - \rho^{\pi} \mid s_t = s, a_t = a\right\} \qquad (4.26)$$

We call these *relative values* because they are relative to the average reward under the current policy.

There are subtle distinctions that need to be drawn between different kinds of optimality in the undiscounted continuing case. Nevertheless, for most practical purposes it may be adequate simply to order policies according to their average reward per time step, in other words, according to their ρ^{π}. For now let us consider all policies that attain the maximal value of ρ^{π} to be optimal.

Other than its use of relative values, R-learning is a standard TD control method based on off-policy GPI, much like Q-learning. It maintains two policies, a behavior policy and an estimation policy, plus an action-value function and an estimated average reward. The behavior policy is used to generate experience; it might be the ε-greedy policy with respect to the action-value function. The estimation policy is the one involved in GPI. It is typically the greedy policy with respect to the action-value function. If π is the estimation policy, then the action-value function, Q, is an approximation of Q^{π} and the average reward, ρ, is an approximation of ρ^{π}. There has been little experience with this method and it should be considered experimental.

Chapter V

Fault-tolerant routing in Next Generation Home Networks

V.1. Introduction

This chapter deals with the fault-tolerant routing problem in Next Generation Home Networks. At the beginning, some state of the art algorithms are presented, together with an explaination of the limitations of their application in the new scenario considered in this work.

After that, the fault-tolerant routing algorithm, object of this work, is described. First of all, the MDP formulation of the problem is described, with the definition of the state space, the action space, the transition matrix and the cost function. Then, the main scalability problem related to the implementation of the optimal MDP controller are described in order to explain the need to derive a new algorithm based on Reinforcement Learning. This explaination opens the way to the presentation of the Q-Learning fault-tolerant algorithm.

V.2. State of the art routing algorithms

Existing routing algorithms are classified either as proactive (e.g., [16]-[18]), as reactive (e.g., [19], [20]) or as *hybrid* (e.g., [21], [22]). The proactive algorithms continuously update path information, which is then available at algorithm decision time; the drawback is that these algorithms require the knowledge of the topology of the whole network. Reactive algorithms performs a route discovery procedure on demand, i.e., only at routing decision time: on the one hand, they generate less control information since they must not continuously update topology information; on the other hand, they delay the actual data transmission until the path is discovered. Hybrid protocols use a combination of these two ideas.

Clearly, the proactive approach is preferred in the considered home network scenario due to the fast re-routing requirements and to its limited topology width which makes the updating process fast. Proactive routing problems have been successfully modeled as Markov Decision Processes (MDP), with the objective of maximizing the number of active flows supported by the network (e.g., [23]-[28]). MDPs are stochastic control processes, and provide a mathematical framework for optimization problems involving both random events and decision makers ([29]). However, in the MDP formulations introduced so far, the topology of the network is considered as static, and the dynamics of the MDP is driven by traffic events (e.g., acceptance of new flows, flow terminations, flow variations); the routing problem is then to decide the optimal paths of the active flows. As topology events such as link faults occur, the MDP must be re-defined and the optimal policy must be re-computed. This approach is then not suitable to provide fast re-routing.

Fault-tolerant routing algorithms have been proposed in the mobile ad-hoc networks scenario. In [30]-[34], robustness is achieved by redundancy: the source node sends the same packets along all the different paths available between the source and the destination; these multipath routing mechanisms are not suitable for the scenario considered in this work, since sending multiple copies of high-bitrate flows over different paths would rapidly flood the network. Also in [35], a multipath routing algorithm is proposed, which is capable of significantly reducing the packet overhead by dynamically identifying unavailable paths via end-to-end path performance measurements. In [36], a stochastic learning-based weak estimation procedure is used to minimize the overhead while guaranteeing a certain level of packet delivery. By the way, since also both [35] and [36] use duplicate packets to achieve robustness to faults, they are not efficient in case of high-bitrate flows.

V.3. MDP fault-tolerant routing in NGHNs

The aim of the proactive algorithm developed in this work is twofold:

1. Minimization of re-routing occurrences;
2. Fast re-routing in scenarios characterized by highly variable topology.

To achieve these objectives (recalling that, in the considered home network scenario, topology dynamics are faster than traffic dynamics), the proposed MDP algorithm considers the traffic as static, and MDP dynamics are driven by topology events. In this case, the optimal re-routing policy is computed by taking into consideration the probabilities that the paths can become unavailable in the future, and explicitly specifies the new path in case of link faults. Even if this approach is capable of guaranteeing fast re-routing decisions, as traffic events occur the MDP must be re-defined and the optimal policy must be re-computed. Note that the solution of the new MDP (i.e., the MDP defined after a traffic event) allows also the determination of the optimal initial routing after the traffic event.

V.3.1 Finite-Horizon MDP definition

Under the markovian[2] and stationarity assumptions, a MDP is defined by a finite state space S, a finite set of available control actions $A(\mathbf{s})$ associated to each state $\mathbf{s} \in S$, a cost $c(\mathbf{s},\mathbf{s}',\mathbf{u})$ which is incurred by the system when it is in state $\mathbf{s}$, action $\mathbf{u}$ is chosen, and the system transitions to state $\mathbf{s}'$, and the transition probability $t(\mathbf{s},\mathbf{s}',\mathbf{u})$ that, in the next stage, the system will be in state $\mathbf{s}'$ when action $\mathbf{u}$ in state $\mathbf{s}$ is chosen. The transition probabilities $t(\mathbf{s},\mathbf{s}',\mathbf{u})$ constitute the transition matrix $\mathbf{T}$.

In finite-horizon MDPs, the system is observed for n stages. A policy is a function $\pi(\mathbf{s};t)$ which at stage t maps every state $\mathbf{s} \in S$ to a unique control action $\mathbf{u} \in A(\mathbf{s})$. When the system operates under a policy $\pi(\mathbf{s};t)$, the system incurs in the following expected total (undiscounted) cost:

$$C_\pi = E_\pi\left\{\sum_{t=1}^{n} c\left[\mathbf{s}(t),\mathbf{u}(t),\mathbf{s}(t+1)\right]\right\} \tag{5.1}$$

where the subscript π specifies that the controller operates under policy π and $c[\mathbf{s}(t), \mathbf{u}(t), \mathbf{s}(t+1)]$ is the cost incurred at stage t when the system is in state $\mathbf{s}(t)$. The MDP problem is to determine the optimal policy π^* minimizing (5.1).

A standard algorithm for finite-horizon MDPs is the *successive approximation* algorithm ([42]), which returns i) the optimal policy $\pi^*(\mathbf{s};t)$ to be applied at stage t, $t = 1,\dots,n$, $\mathbf{s} \in S$; ii) the optimal value function $V^*(\mathbf{s})$, which represents the expected cost of applying the optimal policy $\pi^*(\mathbf{s};t)$ for stages $t = 1,\dots,n$, starting from state $\mathbf{s} \in S$.

[2] A stochastic process has the Markov property if the conditional probability distribution of the next state of the process depends only upon the present state and is conditionally independent of past states

V.3.2 MDP fault-tolerant routing

In this section, the fault-tolerant routing problem is formulated as a finite-horizon MDP. For the sake of comprehension, Table 1 summarizes part of the notation which will be used in this section.

Element	Set	Cardinality
Routing table **r**	P is the set of all possible routing tables	card(P) = R
Path status **x**	Ξ is the set of all possible path status vectors	card(Ξ) = X
State **s**	S is the state space	card(S) $\leq PR$
	$S_{\mathbf{x}}$ is the set of states with path status **x**	card($S_{\mathbf{x}}$) $\leq$ card(S)
Flow f	Φ is the set of flows	card(Φ) = F
Link l	Λ is the set of links	card(Λ) = L
	Λ_p is the set of the links of path p	card(Λ_p) $\leq L$
	$\Lambda_{(\mathbf{x},\mathbf{x}')}$ is the set of the links which cause the transitions between **x** and **x**' as they change link state	card($\Lambda_{(\mathbf{x},\mathbf{x}')}$) $\leq L$
Path p	Π is the set of paths	card(Π) = P
	Π_l is the set of the paths which include link l	card(Π_l) $\leq P$
	Π_f is the set of the paths which are available to flow f	card(Π_f) $\leq P$
	$\Pi_{\mathbf{r}}$ is the set of the paths in use by routing table **r**	card($\Pi_{\mathbf{r}}$) $\leq P$

Table 1 - Definitions of flow, path and link sets

Let us consider a network supporting K classes of services and characterized by a set of links Λ, with cardinality L. Each link l is characterized by its capacity b_l, expressed in [Mbps]. Let us define a generic flow f as a triple (source, destination, class of service). Let the set of flows be Φ and let F be the total number of flows. Each flow f is characterized by a bitrate b_f, expressed in [Mbps].

Different paths are available to route each flow $f \in \Phi$ (i.e., different paths exist from source to destination of flow f); let Π be the set of paths, with cardinality P, and let $\Pi_f \subseteq \Pi$ be the set of paths available to route flow f. Moreover, each path $p \in \Pi$ is constituted by a set of links $\Lambda_p \subseteq \Lambda$. Clearly, the generic link l can be included in more than one path: let $\Pi_l \subseteq \Pi$ be the set of paths including link l, $l = 1,\ldots,L$.

The network routing table $\mathbf{r}$ is a vector with F elements r_f, $f = 1,\ldots,F$; r_f is equal to the path p assigned to flow f. The set of routing tables is then:

$$\mathrm{P} = \left\{ \mathbf{r} = (r_1, r_2, \ldots, r_F) \middle| r_f = p \in \Pi_f,\ f = 1,2,\ldots,F \right\} \tag{5.2}$$

Let R be the number of possible routing tables, and let $\Pi_{\mathbf{r}} \subseteq \Pi$ be the set of paths used by routing table $\mathbf{r}$ (i.e., the set of paths p such that $r_f = p$ for at least one f).

In my purposes, as specified in before, network traffic is considered static between two traffic events, in the sense that the number and the characteristics of the flows remains the same in the period between two traffic events: in other words, the MDP is defined between two traffic events. Traffic events are: *new flow acceptance, flow termination* and *flow variation*. In this work (as in the OMEGA project) I assume that an admission controller is in charge of admitting high-demanding flows in the network. Thus, the new flow acceptance event corresponds to the establishment of a new flow in the network; the flow termination event corresponds to the end of transmission of an on-going flow; the flow variation event corresponds to the variation of the bitrate of an already accepted flow (after re-negotiation of flow parameters with the admission controller).

The MDP must be re-initialized at every traffic event. The mean interval between two traffic events is considered as the duration of the finite-horizon MDP.

The control action is relevant whenever a path currently used by a flow becomes unavailable due to a link fault. The controller must then decide where to re-route the flows, i.e., which paths to select among the available ones. It is also possible that one or more flows cannot be routed anymore in the new link conditions: in this case, the admission controller must decide upon the dropping of one or more flows. From the routing point of view, the decision to drop a flow is equivalent to the flow termination traffic event, which entails the definition of a new MDP.

In the following, it is introduced the link model, the overall framework and finally how the proposed algorithm is used to take routing decisions.

V.3.2.1. Link model

In this work, I consider the possibility of incurring in link faults. The dynamics of each link $l \in \Lambda$ is modeled by a two-state Markov chain: in the *unavailable* state, the link cannot be used to transmit data, i.e., its capacity is 0; in the *available* state, the link can be used to transmit data, i.e., its capacity is b_l[3]. I assume that both the transition frequency between the available state and the unavailable state and the transition frequency between the unavailable state and the available state are distributed according to Poisson processes with mean frequencies μ_l and λ_l, respectively, expressed in [min^{-1}]. A given path p is available only if all the links of the set Λ_p are available. Then, if a link l becomes unavailable, all the paths $p \in \Pi_l$ becomes unavailable. From standard Markov chain theory ([37]), the probability that link l is in the available and in the unavailable states is computed as $\lambda_l / (\lambda_l + \mu_l)$ and $\mu_l / (\lambda_l + \mu_l)$, respectively.

Link state changes trigger topology events, which drive the MDP dynamics. Link statistics are easily available in home networks (for example, in OMEGA link statistics are collected by any device in charge of controlling the network).

V.3.2.2. Fault-tolerant MDP routing

The MDP is defined by the state space S, the action space A, the transition probability matrix $\mathbf{T}$ and the cost function c.

1. *State space S*

The *path status* $\mathbf{x}$ is a vector with P elements x_p, $p = 1,\ldots,P$, such that $x_p = 1$ if path p is available (i.e., if all links $l \in \Lambda_p$ are available), $x_p = 0$ otherwise. The set of path status vectors is then:

$$\Xi = \left\{ \mathbf{x} = (x_1, x_2, \ldots, x_P) \middle| x_p \in \{0,1\},\ p = 1,2,\ldots,P \right\}. \qquad (5.3)$$

The number X of possible path status vectors is 2^P. In the following, considering two path status vectors $\mathbf{x}, \mathbf{x}' \in \Xi$, I will write $\mathbf{x} > \mathbf{x}'$ if $x_p \geq x_p\ \forall\ p \in \Pi$ and $x_p > x_p$ for at least one path $p \in \Pi$.

[3] Note that, for some links, a two-state link model might be insufficient. The proposed framework can be extended to include also links modeled by N-state Markov chain by following the rationale in [38].

The system state is given by the path status vector and by the current routing table; the generic state **s** is then the (F+P)-vector $\mathbf{s} = (\mathbf{r},\mathbf{x})$, with $\mathbf{r} \in \Pi$ and $\mathbf{x} \in \Xi$. Clearly, not all the couples (**r**,**x**) are feasible; specifically, the state **s** = (**r**,**x**) is feasible only if the following two *feasibility conditions* hold:

$$x_p = 1, \forall p \in \Pi_{\mathbf{r}}; \qquad (5.4)$$

$$\sum_{f \in \Phi \left| r_f = p \text{ and } l \in \Lambda_p \right.} b_f \le b_l \,,\ l \in \Lambda . \qquad (5.5)$$

The first feasibility condition (5.4) states that all the paths used by **r** must be available in **x**; the second condition (5.5) states that, for each link $l \in \Lambda$, the link capacity b_l must be greater than or equal to the load of link l, computed as the sum of the bitrates of all the flows routed by **r** on paths including link l.

In addition to the states identified by equations (5.4) and (5.5), I add an absorbing state $\mathbf{s}_{abs}$, where the system transitions whenever no other feasible state exist (in brief, the absorbing state can be considered as an aggregate of all the states (**r**,**x**) which does not meet the two feasibility conditions).

The state space is then defined as follows:

$$S = \left\{ \mathbf{s} = (\mathbf{r},\mathbf{x}) \middle| \mathbf{r} = (r_1, r_2, ..., r_F) \in \mathrm{P}; \mathbf{x} = (x_1, x_2, ..., x_P) \in \Xi; x_p = 1, \forall p \in \Pi_{\mathbf{r}}; \right.$$

$$\left. ; \sum_{f \in \Phi \left| r_f = p \text{ and } l \in \Lambda_p \right.} b_f \le b_l, \forall\, l \in \Lambda \right\} \cup \{\mathbf{s}_{abs}\}. \qquad (5.6)$$

Finally, the following sets are defined:

- Let $\Lambda_{(\mathbf{x},\mathbf{x}')}$ be the set of links which are available when the path status is **x** and whose transition to the unavailable state lead the path status from **x** to $\mathbf{x} > \mathbf{x}'$ (generally, there are different links which causes the same change of path status). The same set of links is clearly involved in the transition from **x**' to **x**: in this case, the transition occurs when a given link $l \in \Lambda_{(\mathbf{x},\mathbf{x}')}$ transitions from the unavailable to the available state and all the other links $l' \in \Lambda_{(\mathbf{x},\mathbf{x}')} \backslash \{l\}$ are already available. $\Lambda_{(\mathbf{x},\mathbf{x}')}$ is defined as follows:

$$\Lambda_{(\mathbf{x},\mathbf{x}')} = \left\{ l \in \Lambda \middle| x_p = x'_p \text{ if } p \notin \Pi_l, x_p \ne x'_p \text{ otherwise} \right\} \qquad (5.7)$$

- Let $S_{\mathbf{x}} \subseteq S$ be the set of states associated to path status **x**:

$$S_{\mathbf{x}} = \{\mathbf{s} \in S | \mathbf{s} = (\mathbf{r}, \mathbf{x}), \mathbf{x} \in \Xi\} \qquad (5.8)$$

  (note that $S_{\mathbf{x}}$ might be empty for some $\mathbf{x} \in \Xi$, and that $S = \bigcup_{\mathbf{x} \in \Xi} S_{\mathbf{x}} \cup \{\mathbf{s}_{abs}\}$).

2. *Action Space A*

In the generic state $\mathbf{s} = (\mathbf{r},\mathbf{x}) \in S$, if i) link $l \in \Lambda_{(\mathbf{x},\mathbf{x}')}$ is unavailable, ii) all the other links $l' \in \Lambda_{(\mathbf{x},\mathbf{x}')} \backslash \{l\}$ are already available, and iii) l becomes available, one ore more paths which are not available in **x** (i.e., all the paths $p \in \Pi_l$ such that $x_p = 0$) become available. In this case, since the path status changes from **x** to $\mathbf{x}' \in \Xi$ with $\mathbf{x}' > \mathbf{x}$, all the paths which were feasible in **s** are still feasible, and the system transitions from $\mathbf{s} = (\mathbf{r},\mathbf{x})$ to the new (feasible) state $\mathbf{s}' = (\mathbf{r},\mathbf{x}') \in S$ without requiring any control action.

On the other hand, if a link $l \in \Lambda_{(\mathbf{x},\mathbf{x}')}$, available in **x**, transitions to the unavailable state, it renders unavailable one or more available paths (i.e, all the available paths $p \in \Pi_l$ such that $x_p = 1$). In this case, the path status vector changes from **x** to $\mathbf{x}' \in \Xi$. If $\Pi_{\mathbf{r}} \cap \Pi_l = \varnothing$ (i.e., if all the paths $p \in \Pi_l$ are not used by the current routing table **r**), the system transitions from $\mathbf{s} = (\mathbf{r},\mathbf{x})$ to the new state $\mathbf{s}' = (\mathbf{r},\mathbf{x}') \in S$ without requiring any control action.

Conversely, if $\Pi_{\mathbf{r}} \cap \Pi_l \neq \varnothing$ (i.e., if one or more paths used by **r** become unavailable), the controller must change the routing table. If $S_{\mathbf{x}'} = \varnothing$ (i.e., if no routing table $\mathbf{r}' \in \mathrm{P}$ exist such that $(\mathbf{r}',\mathbf{x}') \in S$), the system transitions to the state $\mathbf{s}_{abs}$, and the admission controller is triggered. Otherwise, the controller must decide which routing table to choose among the routing tables which are feasible with respect to **x'**.

Let us consider the generic state $\mathbf{s} = (\mathbf{r},\mathbf{x}) \in S_{\mathbf{x}}$, and let us assume that a transition occurs from the available to the unavailable state of a link $l \in \Lambda_{(\mathbf{x},\mathbf{x}')}$. When this event occurs, the decision to change the routing table from **r** to $\mathbf{r}' \in \mathrm{P}$ is denoted with $u(\mathbf{s},\mathbf{s}')$, where $\mathbf{s}' = (\mathbf{r}',\mathbf{x}')$:

- if $\mathbf{s}' \in S_{\mathbf{x}'}$ and the controller decides to enforce the routing table **r'**, then $u(\mathbf{s},\mathbf{s}') = 1$;
- if $\mathbf{s}' \in S_{\mathbf{x}'}$ but the controller decides to enforce another routing table, $u(\mathbf{s},\mathbf{s}') = 0$;
- finally, if $\mathbf{s}' \notin S$, then $u(\mathbf{s},\mathbf{s}')$ is not an available decision in **s**.

Clearly, the controller must decide to enforce exactly one routing table.

In conclusion, the action space when the system is in state $\mathbf{s} \in S_{\mathbf{x}}$ is then defined as follows:

$$A(\mathbf{s}) = \left\{ \mathbf{u} = [u(\mathbf{s},\mathbf{s}')]_{\mathbf{s}' \in S_{\mathbf{x}'}} \middle| \Lambda_{(\mathbf{x},\mathbf{x}')} \neq \emptyset, \sum_{\mathbf{s}' \in S_{\mathbf{x}'}} u(\mathbf{s},\mathbf{s}') = 1, u(\mathbf{s},\mathbf{s}') = 1 \text{ if } \mathbf{s}' = (\mathbf{r},\mathbf{x}'), u(\mathbf{s},\mathbf{s}') \in \{0,1\} \text{ otherwise} \right\} \tag{5.9}$$

where $\mathbf{u}$ is the vector of possible controller actions when the system is in state $\mathbf{s} \in S_{\mathbf{x}}$ and a link $l \in \Lambda_{(\mathbf{x},\mathbf{x}')}$ becomes unavailable.

The controller policy is the function π: $S \times [1,n] \to A$ defined by setting a feasible action vector $\mathbf{u} \in A(\mathbf{s})$ for each state $\mathbf{s} \in S$ and for each stage $t \in [1,n]$. The policy space O is the set of the feasible policies:

$$\mathrm{O} = \{\pi(\mathbf{s},t) = \mathbf{u} | \mathbf{s} \in S, \mathbf{u} \in A(\mathbf{s}), t = 1,\dots,n\} \tag{5.10}$$

3. *Transition matrix*

The transition frequencies between states can be inferred from the link transition frequencies (between their available and unavailable states) and from the above-defined action space.

Let us consider two generic states $\mathbf{s} = (\mathbf{r},\mathbf{x}) \in S_{\mathbf{x}}$ and $\mathbf{s}' = (\mathbf{r},\mathbf{x}') \in S_{\mathbf{x}'}$, with the same routing table $\mathbf{r}$ and such that $\mathbf{x}' > \mathbf{x}$. I recall that when the path status is $\mathbf{x}$, it changes to $\mathbf{x}'$ if the following conditions hold:

i) a given link $l \in \Lambda_{(\mathbf{x},\mathbf{x}')}$ is in the unavailable state;

ii) all the other links $l' \in \Lambda_{(\mathbf{x},\mathbf{x}')} \backslash \{l\}$ are already available;

iii) link l transitions to the available state.

Since the path status is $\mathbf{x}$, the probability that all links $l \in \Lambda_{(\mathbf{x},\mathbf{x}')}$ are available is null (otherwise the path status would be $\mathbf{x}'$): thus, condition ii) implies condition i). From the link dynamic model, it follows that the probability of ii) is $\prod_{l' \in \Lambda_{(\mathbf{x},\mathbf{x}')} \backslash l} [\lambda_{l'} / (\lambda_{l'} + \mu_{l'})]$, and that the mean frequency of event iii) is λ_l. As specified above, no action is required in this case, and the system transits from state **s** to state **s**' with the following total mean frequency:

$$\phi(\mathbf{s},\mathbf{s}') = \sum_{l\in\Lambda_{(\mathbf{x},\mathbf{x}')}} \left\{ \prod_{l'\in\Lambda_{(\mathbf{x},\mathbf{x}')}\setminus l} \frac{\lambda_{l'}}{\lambda_{l'}+\mu_{l'}} \lambda_l \right\}, \mathbf{s} = (\mathbf{r},\mathbf{x}) \in S_{\mathbf{x}}, \mathbf{s}' = (\mathbf{r},\mathbf{x}') \in S_{\mathbf{x}'}, \mathbf{x}' > \mathbf{x}. \qquad (5.11)$$

where in the summation I exploited the hypothesis of Poisson transition frequencies between link states.

Let us consider two generic states $\mathbf{s} = (\mathbf{r},\mathbf{x}) \in S_{\mathbf{x}}$ and $\mathbf{s}' = (\mathbf{r}',\mathbf{x}') \in S_{\mathbf{x}'}$ such that $\mathbf{x} > \mathbf{x}'$ and $S_{\mathbf{x}'} \neq \emptyset$. I recall that when the path status is **x**, it changes to **x**' if a given link $l \in \Lambda_{(\mathbf{x},\mathbf{x}')}$ transitions to the unavailable state (note that all links $l \in \Lambda_{(\mathbf{x},\mathbf{x}')}$ are always available when the path status is **x**). From the link dynamic model, it follows that the mean frequency of this event is μ_l. In this case, the mean frequency of the transition from state **s** to state **s'** depends also on the re-routing decision of the controller $u(\mathbf{s},\mathbf{s}')$:

$$\phi(\mathbf{s},\mathbf{s}') = u(\mathbf{s},\mathbf{s}') \sum_{l\in\Lambda_{(\mathbf{x},\mathbf{x}')}} \mu_l, \mathbf{s} \in S_{\mathbf{x}}, \mathbf{s}' \in S_{\mathbf{x}'}, \mathbf{x} > \mathbf{x}'. \qquad (5.12)$$

Finally, let us consider the generic state $\mathbf{s} = (\mathbf{r},\mathbf{x}) \in S_{\mathbf{x}}$ and a path status **x**' such that $S_{\mathbf{x}'} = \emptyset$. In this case, the system transitions from state **s** to state $\mathbf{s}_{abs}$ with the following transition frequency:

$$\phi(\mathbf{s},\mathbf{s}_{abs}) = \sum_{l\in\Lambda_{(\mathbf{x},\mathbf{x}')}} \mu_l, \mathbf{s} \in S_{\mathbf{x}}, S_{\mathbf{x}'} = \emptyset, \mathbf{x} > \mathbf{x}'. \qquad (5.13)$$

To obtain the transition probabilities $t_\pi(\mathbf{s},\mathbf{s}')$ (where the sub-index π highlights that some transition probabilities depend on the adopted policy), I apply a standard *uniformization* procedure ([39]):

i) compute the so-called uniformization constant, which is an upper-bound of the total outgoing frequency of each state:

$$\gamma > \max_{\mathbf{s}\in S} \left\{ \sum_{\mathbf{s}'\in S} \phi(\mathbf{s},\mathbf{s}') \right\} \qquad (5.14)$$

ii) divide the transition frequencies by γ:

$$t_\pi(\mathbf{s},\mathbf{s}') = \frac{1}{\gamma}\sum_{l\in\Lambda_{(\mathbf{x},\mathbf{x}')}}\lambda_l, \mathbf{s}\in S_\mathbf{x}, \mathbf{s}'\in S_{\mathbf{x}'}, \mathbf{x}'>\mathbf{x}; \tag{5.15}$$

$$t_\pi(\mathbf{s},\mathbf{s}') = \frac{1}{\gamma}u(s,s')\sum_{l\in\Lambda_{(\mathbf{x},\mathbf{x}')}}\mu_l, \mathbf{s}\in S_\mathbf{x}, \mathbf{s}'\in S_{\mathbf{x}'}, \mathbf{x}>\mathbf{x}'; \tag{5.16}$$

$$t_\pi(\mathbf{s},\mathbf{s}_{abs}) = \frac{1}{\gamma}\sum_{l\in\Lambda_{(\mathbf{x},\mathbf{x}')}}\mu_l, \mathbf{s}\in S_\mathbf{x}, S_{\mathbf{x}'}=\varnothing, \mathbf{x}>\mathbf{x}'. \tag{5.17}$$

iii) Add self-transitions to let the sum of the transitions leaving each state be equal to 1:

$$t_\pi(\mathbf{s},\mathbf{s}) = 1-\sum_{\substack{\mathbf{s}'\in S\\ \mathbf{s}'\neq\mathbf{s}}} t_\pi(\mathbf{s},\mathbf{s}'),\ \mathbf{s}\in S\ . \tag{5.18}$$

Note that γ is expressed in [min^{-1}] and that, since no transition outgoing from $\mathbf{s}_{abs}$ exists but the self-transition (5.18), its self-transition probability $t_\pi(\mathbf{s}_{abs}, \mathbf{s}_{abs})$ is 1 ($\mathbf{s}_{abs}$ is in fact an absorbing state).

The transition probabilities $t_\pi(\mathbf{s},\mathbf{s}')$ constitute the transition matrix **T**.

Remark 1

Note that when the system is in a given state in the subset $S_\mathbf{x}$ and a topology event causes the path status vector to transition from **x** to **x**', the transition probability t(**x**,**x**') between the subsets $S_\mathbf{x}$ and $S_{\mathbf{x}'}$ is uncontrolled and, thus, does not depend on the routing policy. In fact, from equations (5.15)-(5.17), the following transition probabilities between subsets $S_\mathbf{x}$ and $S_{\mathbf{x}'}$ are obtained:

$$t(\mathbf{x},\mathbf{x}') = \frac{1}{\gamma}\sum_{l\in\Lambda_{(\mathbf{x},\mathbf{x}')}}\left\{\prod_{l'\in\Lambda_{(\mathbf{x},\mathbf{x}')}\backslash l}\frac{\lambda_{l'}}{\lambda_{l'}+\mu_{l'}}\lambda_l\right\}, S_\mathbf{x}\neq\varnothing, S_{\mathbf{x}'}\neq\varnothing, \mathbf{x}'>\mathbf{x}; \tag{5.19}$$

$$t(\mathbf{x},\mathbf{x}') = \frac{1}{\gamma}\sum_{l\in\Lambda_{(\mathbf{x},\mathbf{x}')}}\mu_l, S_\mathbf{x}\neq\varnothing, S_{\mathbf{x}'}\neq\varnothing, \mathbf{x}>\mathbf{x}'; \tag{5.20}$$

$$t(\mathbf{x},\mathbf{s}_{abs}) = \frac{1}{\gamma}\sum_{l\in\Lambda_{(\mathbf{x},\mathbf{x}')}}\mu_l, S_\mathbf{x}\neq\varnothing, S_{\mathbf{x}'}=\varnothing, \mathbf{x}>\mathbf{x}'. \tag{5.21}$$

where, to compute equation (5.20), I considered that the sum of the decisions u(**s**,**s**') must be 1 (see definition (5.9)).

In conclusion, if the system is in the generic state $\mathbf{s} \in S_{\mathbf{x}}$, the role of the controller is then just to decide which state **s**' among the ones in $S_{\mathbf{x}'}$ to choose when the path status vector transitions from **x** to **x**', with $\mathbf{x} > \mathbf{x}'$ and $S_{\mathbf{x}'} \neq \emptyset$. If the topology event is such that $\mathbf{x}' > \mathbf{x}$ or $S_{\mathbf{x}'} = \emptyset$, no control decision is required.

4. *Cost function c*

The main objective of the fault-tolerant routing policy is to minimize the number of path changes, and in case of path changes it is desirable to minimize the link changes (i.e., the number of links affected by re-routing changes). Moreover, if the network supports classes of service to offer QoS guarantees, the cost of changing paths is also weighted by the class of service of the re-routed flows.

To reflect these objectives, the cost function associated to state **s** = (**r**,**x**) and next state **s**' = (**r**',**x**') is defined as follows:

$$c(\mathbf{s},\mathbf{s}') = \sum_{k=1}^{K} \left[w_{path}^{(k)} \Delta_{path}^{(k)}(\mathbf{r},\mathbf{r}') \right] + w_{link} \Delta_{link}(\mathbf{r},\mathbf{r}'), \qquad (5.22)$$

where: $\Delta_{path}^{(k)}(\mathbf{r},\mathbf{r}')$ is the number of re-routed flows of class k when the routing table changes from **r** to **r**'; $\Delta_{link}(\mathbf{r},\mathbf{r}')$ is the number of links which in **r**' support different paths with respect to **r**; $w_{path}^{(k)}$ is the weight associated to the re-routing of a class k flow; w_{link} is the weight associated to the link changes. I consider that the last part of the cost (related to the link changes) is used just to decide among two or more new routing tables which have the same path cost; thus, I will set $w_{link} << w_{path}^{(k)}$, $k = 1,\ldots,K$[4].

V.3.2.3. MDP algorithm outcomes

As mentioned in before, a standard algorithm to find the optimal solution of a finite-horizon MDP problem is the Successive Approximation algorithm, which returns, at each stage t, the optimal stage-per-stage policy $\pi^*(\mathbf{s},t)$ and the coupled optimal value

[4] The path and link weights could also be setup according to the technologies of the involved networks.

function $V^*(\mathbf{s})$. The total number of stages n is computed by taking the upper integer value of the mean time interval between two traffic events (which is considered as the duration of the finite-horizon MDP) times the uniformization constant γ. Note that the new traffic event might occur before or after the final stage n: in the former case, the MDP is re-initialized before the considered finite-horizon; in the latter case, the controller keeps on using the final policy. In both cases, since the actual MDP duration is different with respect to the considered finite horizon, the policy $\pi^*(\mathbf{s},t)$ is sub-optimal.

The results of the algorithm are exploited in two ways, as analyzed in the following paragraphs: 1) to define the optimal re-routing policy in case of link faults; 2) to identify the optimal initial state.

1. *Optimal re-routing policy in case of link faults*

At each stage $t = 1,\ldots,n$, the optimal stage-to-stage policy $\pi^*(\mathbf{s},t)$ conveys the re-routing actions in case of link faults: let us assume that at stage t the system is in state $\mathbf{s} = (\mathbf{r},\mathbf{x})$, and that link l becomes unavailable causing the path status to change from $\mathbf{x}$ to $\mathbf{x}'$, with $\mathbf{x} > \mathbf{x}'$; thanks to the action space defined in equations (5.3) and (5.9), in the optimal policy there is exactly one state $\mathbf{s}^* = (\mathbf{r}^*,\mathbf{x}') \in S_{\mathbf{x}'}$ such that $u^*(\mathbf{s},\mathbf{s}^*) = 1$, whereas the other decisions $u^*(\mathbf{s},\mathbf{s}')$ are equal to 0 for each state $\mathbf{s}' \in S_{\mathbf{x}'}$ such that $\mathbf{s}' \neq \mathbf{s}^*$. Thus, the controller decision is to change the routing table from $\mathbf{r}$ to $\mathbf{r}^*$, entailing the system transition from state $\mathbf{s}$ to state $\mathbf{s}^*$.

2. *Optimal initial state identification*

The optimal value function $V^*(\mathbf{s})$ is used to decide the optimal initial state $\mathbf{s}^*_{initial} = (\mathbf{r}^*_{initial}, \mathbf{x})$ after a traffic event. Given the path status $\mathbf{x}$, there can be either no feasible states (i.e., $S_{\mathbf{x}} = \emptyset$), or there exist one or more *candidate initial states*, identified by all the routing tables $\mathbf{r}$ such that $\mathbf{s} = (\mathbf{r},\mathbf{x}) \in S$. In the former case, which is relevant either in case of new incoming flows or in case of flow variation, the new MDP starts directly in the absorbing state $\mathbf{s}_{abs}$, which means in practice that the admission controller must block/drop one or more flows, and that then the MDP must be re-initialized. In the latter case, the most appropriate initial state must be chosen.

To choose the optimal initial state, I simply extend the definition of cost (5.22) to compute the cost of each candidate initial state $\mathbf{s}$, hereafter referred to as $c_{initial}(\mathbf{s})$. As a traffic event occurs, a new MDP $\{S,A,\mathbf{T},C\}$ is defined, and the controller has to decide the initial state, based on the last routing table of the former MDP, $\mathbf{r}_{old}$, and on the current path status $\mathbf{x}$. Three traffic events are considered:

a) Flow variation

In this case, the new routing table r will have the same number of flows with respect to the past one $\mathbf{r}_{old}$. Cost definition (5.22) is thus seamlessly applicable to compute the cost $c_{initial}(\mathbf{s})$ of transiting from $(\mathbf{r}_{old},\mathbf{x})$ to the candidate initial states $\mathbf{s} \in S_{\mathbf{x}}$: $c_{initial}(\mathbf{s}) = c[(\mathbf{r}_{old},x), \mathbf{s}]$. Note that if $(\mathbf{r}_{old},\mathbf{x})$ is still feasible (i.e., $(\mathbf{r}_{old},\mathbf{x}) \in S$), no re-routing is necessary.

b) New flow acceptance

In this case, the new routing table $\mathbf{r}$ will have one more flow with respect to the past one $\mathbf{r}_{old}$. Let F be the number of flows of the new MDP; then, $\mathbf{r}_{old}$ has (F – 1) flows. Without loss of generality, let us assume that the first (F – 1) flows of the new MDP are the same flows of the past routing table; then, I extend $\mathbf{r}_{old}$ by adding a null F-th element. Cost definition (5.22), is subsequently applicable to compute the cost $c_{initial}(\mathbf{s})$ of transiting from $[(\mathbf{r}_{old},0),\mathbf{x}]$ to the candidate initial states $s \in S_{\mathbf{x}}$: $c_{initial}(\mathbf{s}) = c[((\mathbf{r}_{old},0),\mathbf{x}), \mathbf{s}]$.

c) Flow termination (or dropping)

In this case, the new routing table $\mathbf{r}$ will have one less flow with respect to the past one $\mathbf{r}_{old}$. Let F be the number of flows of the new MDP; then, $\mathbf{r}_{old}$ has (F + 1) flows. Without loss of generality, let us assume that the first F flows of the old routing table are the same flows of the new routing table; then, I extend $\mathbf{r}$ by adding a null (F+1)-th element. Cost definition (5.22), is subsequently applicable to compute the cost $c_{initial}(\mathbf{s})$ of transiting from $[\mathbf{r}_{old},\mathbf{x}]$ to the extended candidate initial states $\mathbf{s}_{ext} = [(\mathbf{r},0),\mathbf{x}]$, such that $(\mathbf{r},\mathbf{x}) \in S_{\mathbf{x}}$: $c_{initial}(\mathbf{s}) = c[(\mathbf{r}_{old},\mathbf{x}), \mathbf{s}_{ext}]$.

To finally choose the optimal initial state, the cost $c_{initial}(\mathbf{s})$ of choosing the candidate initial state $\mathbf{s}$ is added to the n-stage cost incurred by the system starting from $\mathbf{s}$. The expected cost of starting from a candidate state $\mathbf{s} \in S_{\mathbf{x}}$ when the current path status is $\mathbf{x}$ is then the following:

$$E_{initial}(\mathbf{s}) = c_{initial}(\mathbf{s}) + V^{*}(\mathbf{s}),\ \mathbf{s} \in S_{\mathbf{x}}. \qquad (5.23)$$

The optimal initial state $\mathbf{s}^{*}_{initial}$ is then chosen as the candidate initial state $\mathbf{s} \in S_{\mathbf{x}}$ which has the lowest expected cost (5.23).

From this discussion, it appears clear that the validity of a given policy is limited in time by traffic events, i.e., traffic changes. This limitation and the well-known scalability problems of the MDP approach ([29]) are lightened since, as above discussed, I consider 1) home networks with a limited number of nodes, 2) a limited number of high-bitrate flows with long duration, and 3) I consider sporadic and low-bitrate flows as uncontrolled background traffic with low-priority.

In any case, the scalability problem of the proposed MDP approach renders it unsuitable in future home networks, which are expected to consists of tens (or even hundreds) of nodes. In this respect, the purpose of this MDP formulation is to define the fundamental theoretical framework which is necessary to analyze the fault-tolerant routing problem in time-varying network topology scenarios: then, the developed framework can be used to develop more practical algorithms based, for example, on Approximate Dynamic Programming ([39]) and Reinforcement Learning ([15]) approaches.

In the next section, a Reinforcement Learning formulation of the fault-tolerant routing described so far is presented.

V.4. Q-Learning formulation of the routing algorithm

A Reinforcement Learning formulation of the fault-tolerant MDP routing algorithm presented in the previous section has been realized in order to develop an algorithm that has low computational cost and then that can be easily implemented in real-time network control systems.

In particular, in this work the Q-Learning approach has been used to calculate both the 1) initial action and 2) the action to be taken in case of link fault.

1. *Initial action identification*

In the Q-Learning approach, the learned action-value function $Q(s_t, a_t)$ is used to determine the optimal action to be taken in each state.

At the beginning of the process, the $Q(s_t, a_t)$ matrix is initialized with a value that is related to the probability that the paths used by the new flows can be subject to a fault. In particular, considering that a path *p* is composed by a set of links *l* and that the probability of fault for a link *l* is $a_u(l)$, the probability that a path is not subject to a fault is given by

$$\prod_{l \in p} \left[(1 - a_u(l)) \right] \qquad (5.24)$$

Thus, the initial $Q(s_t, a_t)$ is set in the following way:

$$Q(s_t, a_t) = 1 - \prod_{l \in p} \left[(1 - a_u(l)) \right] \qquad (5.25)$$

The best initial action a_0 is thus chosen in order to minimize $Q(s_0, a_0)$ for each $a \in A$.

2. *Re-routing action in case of link faults*

Let us assume that at stage *t* the system is in state $\mathbf{s} = (\mathbf{r},\mathbf{x})$, and that link *l* becomes unavailable causing the path status to change from $\mathbf{x}$ to $\mathbf{x}'$, with $\mathbf{x} > \mathbf{x}'$. The controller decision is to change the routing table from $\mathbf{r}$ to $\mathbf{r}_{t+1}$, entailing the system transition from state $\mathbf{s}$ to state $\mathbf{s}_{t+1}$.

Following the Q-learning one-step action value optimization, I derive

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[c_{t+1} + \sigma \min_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \qquad (5.25)$$

where α is the learning rate, σ is the discount factor and c_{t+1} is the cost associated to s_{t+1}:

$$c(\mathbf{s}, \mathbf{s}') = \sum_{k=1}^{K} \left[w_{path}^{(k)} \Delta_{path}^{(k)}(\mathbf{r}, \mathbf{r}') \right] + w_{link} \Delta_{link}(\mathbf{r}, \mathbf{r}') \qquad (5.26)$$

In particular, the learning rate α determines to what extent the newly acquired information will override the old information. A factor of 0 will make the agent not learn anything, while a factor of 1 would make the agent consider only the most recent information. The discount factor σ determines the importance of future rewards. A factor of 0 will make the agent "opportunistic" by only considering current rewards, while a factor approaching 1 will make it strive for a long-term high reward. If the discount factor meets or exceeds 1, the *Q* values will diverge.

Following this rule it is possible to calculate, at each step, the best action to be taken in order to minimize the cost of passing from state s_t to state s_{t+1}.

A limitation in applying this rule is that, at each stage, the algorithm selects always the best possible action on the basis of the acquired knowledge (*greedy* policy). But in this case the exploration is never performed, thus I can say that the algorithm is myopic. In order to increment long-term performances of the algorithm, an ε- *greedy* policy is adopted so that, with probability ε, the action selected is chosen in a random way. This assures that the algorithm selects also not optimal actions that could instead lead to a lower long term costs.

Chapter VI
Results

VI.1. Introduction

In this chapter I present the results of a set of simulations done to understand the behaviour of the proposed algorithm. In order to perform the simulations, MATLAB simulation tool was used.

At first, a detailed description of the home network scenario used for the simulations is presented. In particular, I created a heterogeneous home network scenario where four different telecommunication technologies are used to compose the network, namely Ethernet, Otical fiber, PLC and WiFi.

The first serie of simulations were done to understand the behaviour of the MDP algorithm presented in section V.3. The ‘greedy’ policy, the ‘optimal’ policy and the ‘optimal QoS’ policy have been thus simulated and compared.

After that, also the behaviour of the Q_Learning algorithm presented in section V.4 has been simulated and its performances were compared with the ones of the ‘greedy’ and ‘optimal’ MDP algorithm.

As deeply described in this chapter, simulation results show that the MDP algorithm achieve better performances in respect to the Q-Learning algorithm. Anyway the results obtained demonstrate that Q-Learning performances are quite close to the MDP ones and thus it that it is possible to use a Q-learing algorithm in real-time applications as the one presented in this work.

VI.2. Scenario description

In order to simulate the behaviour of the proposed algorithms and to evaluate the performances, I consider the simple (for the sake of document comprehension) but meaningful (from the evaluation viewpoint) home network shown in Figure 12, where S*i* and D*j* denote the source of flow *i* and the destination of flow *j*, respectively, and the Home Gateway is the router interconnecting the home network and the Internet; flows 3 and 4, which comes from the Internet, are considered as originated by the Home Gateway.

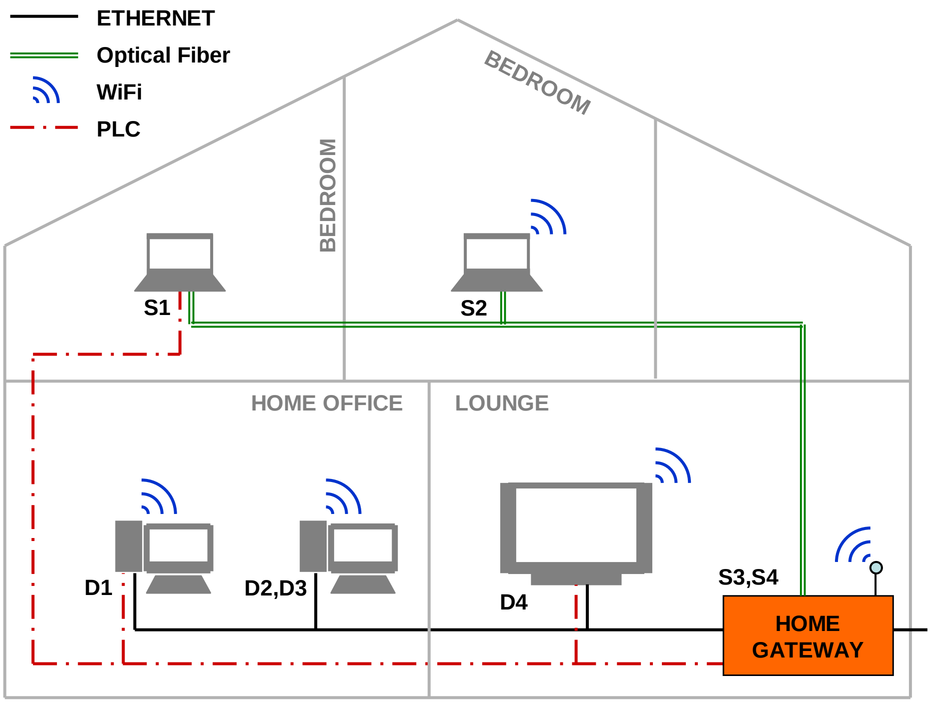


Figure 12 - Example home network

Considering that Ethernet, Wi-Fi, Power Line Communication (PLC) and Optical Fiber (OPT) are networks characterized by a shared medium (briefly, the network capacity is shared among all the users), each network is modeled as a single link: link 1 models the first-floor OPT network; link 2 models the PLC network; link 3 models the Wi-Fi network; link 4 models the ground-floor Ethernet network, which is connected to the first-floor network by the Home Gateway. Note that each network element (i.e., the PCs and the TV in Figure 12) can be source and/or destination of more than one flow, and that it is assumed that the same element is capable of using more than one technology. Figure 13 shows the scheme of the considered home network, where, for the sake of simplicity, D1 and D4 are collapsed in a single network entity.

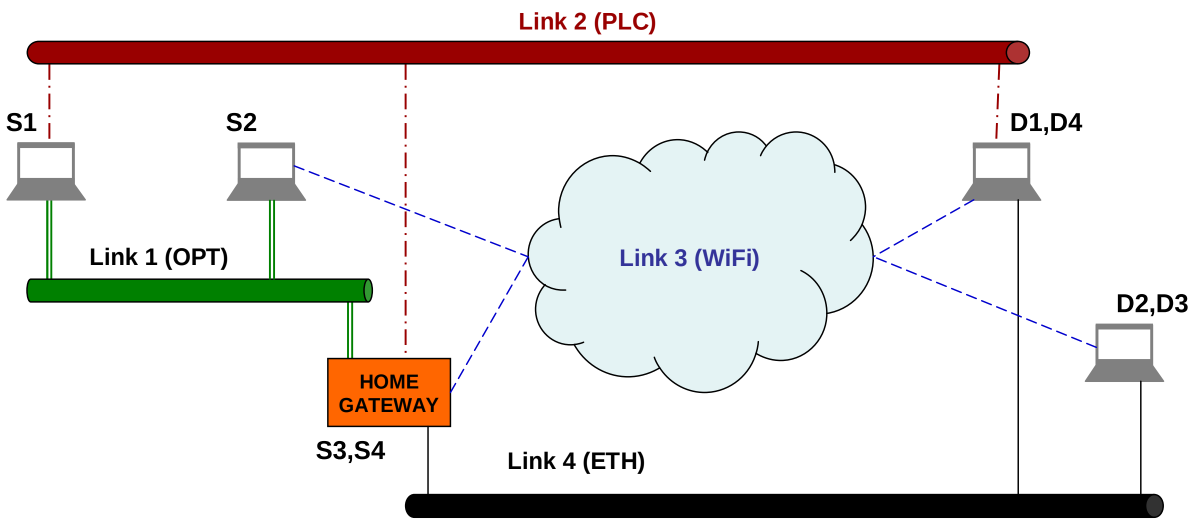


Figure 13 - Scheme of the exaple network

Link characteristics are strictly related to the network technologies. Table 2 shows the link parameters μ_l, λ_l and b_l, $l = 1,\ldots,4$, used in the simulations[5]. For the sake of simplicity, all link capacities b_l, were set equal to 10 Mbps, but the OPT capacity $b_1 = 100$ Mbps. By equation (5.14) and considering the link characteristics, the value $\gamma = 0.669$ min^{-1} was chosen.

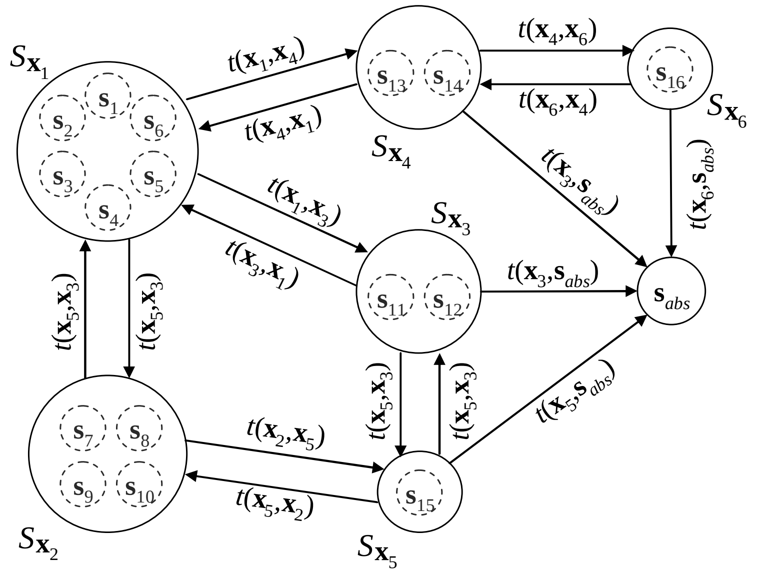
Link #	Technology	b_l [Mbps]	λ_l [min-1]	μ_l [min-1]
1	OPT	100	1/10	1e-4
2	PLC	10	1/60	1/20
3	WiFi	10	1/30	1/30
4	ETH	10	1/5	1e-4

Table 2 - Link characteristics

Four source-destination couples were considered, as shown in Figure 14 and Figure 15, each one modeled as a two-state Markov chain: in the ‘on’ state, the source transmits its flow; in the ‘off’ state, the source is silent. For the sake of simplicity, all transmission rates b_f, f = 1, 2, 3, were set equal to 4.5 Mbps. The transition frequencies from the ‘on’

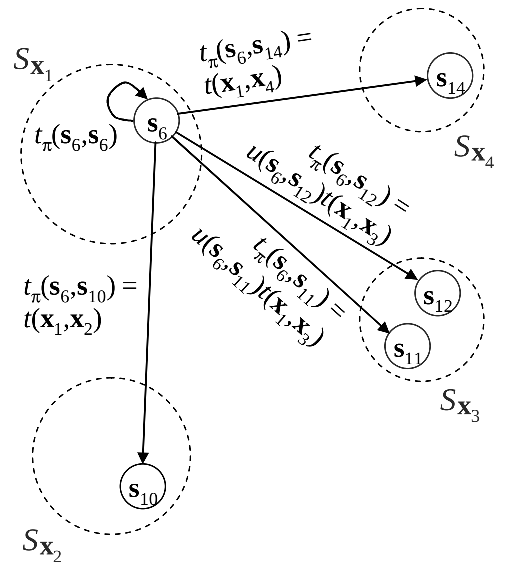
[5] Note that we considered Ethernet and Optical Fiber links as reliable links: in fact, the frequencies μ_l of links 1 and 4 are so small that, in practice, the algorithm results do not sensibly change if we consider them as always available, with the advantage of a state space reduction.

to the ‘off’ state and vice-versa, denoted with μ_f and λ_f, respectively, were set as in Table 3.



$S_{\mathbf{x}_1}$
$S_{\mathbf{x}_2}$
$S_{\mathbf{x}_3}$
$S_{\mathbf{x}_4}$
$S_{\mathbf{x}_5}$
$S_{\mathbf{x}_6}$
$t(\mathbf{x}_1,\mathbf{x}_4)$
$t(\mathbf{x}_4,\mathbf{x}_1)$
$t(\mathbf{x}_4,\mathbf{x}_6)$
$t(\mathbf{x}_6,\mathbf{x}_4)$
$t(\mathbf{x}_1,\mathbf{x}_3)$
$t(\mathbf{x}_3,\mathbf{x}_1)$
$t(\mathbf{x}_3,\mathbf{s}_{abs})$
$t(\mathbf{x}_6,\mathbf{s}_{abs})$
$t(\mathbf{x}_5,\mathbf{s}_{abs})$
$t(\mathbf{x}_2,\mathbf{x}_5)$
$t(\mathbf{x}_5,\mathbf{x}_2)$
$t(\mathbf{x}_5,\mathbf{x}_3)$
$\mathbf{s}_{abs}$

Figure 14 - MDP example: state space $S = \bigcup_{i=1}^{6} S_{\mathbf{x}_i} \cup \{\mathbf{s}_{abs}\}$ and transitions among subsets $S_{\mathbf{x}_i}$, *i* = 1,…,6



$S_{\mathbf{x}_1}$
$t_\pi(\mathbf{s}_6,\mathbf{s}_{14}) = t(\mathbf{x}_1,\mathbf{x}_4)$
$t_\pi(\mathbf{s}_6,\mathbf{s}_6)$
$S_{\mathbf{x}_4}$
$t_\pi(\mathbf{s}_6,\mathbf{s}_{12}) = u(\mathbf{s}_6,\mathbf{s}_{12})t(\mathbf{x}_1,\mathbf{x}_3)$
$t_\pi(\mathbf{s}_6,\mathbf{s}_{11}) = u(\mathbf{s}_6,\mathbf{s}_{11})t(\mathbf{x}_1,\mathbf{x}_3)$
$t_\pi(\mathbf{s}_6,\mathbf{s}_{10}) = t(\mathbf{x}_1,\mathbf{x}_2)$
$S_{\mathbf{x}_3}$
$S_{\mathbf{x}_2}$

Figure 15 - MDP example: transitions from state $\mathbf{s}_6$

Flow #	b_f [Mbps]	λ_f [min-1]	μ_f [min-1]
1	4.5	1/45	1/100
2	4.5	1/60	1/90
3	4.5	1/75	1/30
4	4.5	1/90	1/20

Table 3 - Flow characteristics

Finally, the paths available to each flow are shown in Table 4.

Flow #	Path #	Link sequence	Link technologies
1	1	{1,0,0,1}	OPT-ETH
1	2	{0,1,0,0}	PLC
1	3	{1,0,1,0}	OPT-WiFi
2	1	{1,0,0,1}	OPT-ETH
2	4	{0,0,1,0}	WiFi
3	5	{0,0,0,1}	ETH
3	4	{0,0,1,0}	WiFi
4	2	{0,1,0,0}	PLC
4	4	{0,0,1,0}	WiFi
4	5	{0,0,0,1}	ETH

Table 4 - Paths

To clarify the MDP framework developed in Chapter 5, I construct the MDP corresponding to the case in which only flows 3 and 4 are active ($F = 2$). The feasible states are listed in Table 5[6], where: the routing table vector elements denote the paths of flows 3 and 4, respectively; the path status vector elements denote the status of path 1 (OPT-ETH), of path 4 (WiFi) and of path 5 (ETH), respectively.

[6] Clearly, when 3 or 4 flows are active, the state space is considerably larger.

Routing table	Path status	State
$\mathbf{r}1 = (4,2)$	$\mathbf{x}1 = (1,1,1)$	$\mathbf{s}1 = (\mathbf{r}1,\mathbf{x}1)$
$\mathbf{r}_2 = (4,4)$	$\mathbf{x}_1 = (1,1,1)$	$\mathbf{s}_2 = (\mathbf{r}_2,\mathbf{x}_1)$
$\mathbf{r}_3 = (4,5)$	$\mathbf{x}_1 = (1,1,1)$	$\mathbf{s}_3 = (\mathbf{r}_3,\mathbf{x}_1)$
$\mathbf{r}_4 = (5,2)$	$\mathbf{x}_1 = (1,1,1)$	$\mathbf{s}_4 = (\mathbf{r}_4,\mathbf{x}_1)$
$\mathbf{r}_5 = (5,4)$	$\mathbf{x}_1 = (1,1,1)$	$\mathbf{s}_5 = (\mathbf{r}_5,\mathbf{x}_1)$
$\mathbf{r}_6 = (5,5)$	$\mathbf{x}_1 = (1,1,1)$	$\mathbf{s}_6 = (\mathbf{r}_6,\mathbf{x}_1)$
$\mathbf{r}_2 = (4,4)$	$\mathbf{x}_2 = (0,1,1)$	$\mathbf{s}_7 = (\mathbf{r}_2,\mathbf{x}_2)$
$\mathbf{r}_3 = (4,5)$	$\mathbf{x}_2 = (0,1,1)$	$\mathbf{s}8 = (\mathbf{r}3,\mathbf{x}2)$
$\mathbf{r}_5 = (5,4)$	$\mathbf{x}_2 = (0,1,1)$	$\mathbf{s}_9 = (\mathbf{r}_5,\mathbf{x}_2)$
$\mathbf{r}_6 = (5,5)$	$\mathbf{x}_2 = (0,1,1)$	$\mathbf{s}_{10} = (\mathbf{r}_6,\mathbf{x}_2)$
$\mathbf{r}_1 = (4,2)$	$\mathbf{x}_3 = (1,1,0)$	$\mathbf{s}_{11} = (\mathbf{r}_1,\mathbf{x}_3)$
$\mathbf{r}_2 = (4,4)$	$\mathbf{x}_3 = (1,1,0)$	$\mathbf{s}_{12} = (\mathbf{r}_2,\mathbf{x}_3)$
$\mathbf{r}_4 = (5,2)$	$\mathbf{x}_4 = (1,0,1)$	$\mathbf{s}_{13} = (\mathbf{r}_4,\mathbf{x}_4)$
$\mathbf{r}_6 = (5,5)$	$\mathbf{x}_4 = (1,0,1)$	$\mathbf{s}_{14} = (\mathbf{r}_6,\mathbf{x}_4)$
$\mathbf{r}_2 = (4,4)$	$\mathbf{x}_5 = (0,1,0)$	$\mathbf{s}_{15} = (\mathbf{r}_2,\mathbf{x}_5)$
$\mathbf{r}_6 = (5,5)$	$\mathbf{x}_6 = (0,0,1)$	$\mathbf{s}_{16} = (\mathbf{r}_6,\mathbf{x}_6)$
-	-	$\mathbf{s}_{abs}$

Table 5 - Feasible states with active flows 2 and 3

Note that there are 6 path status vectors $\mathbf{x}_i$ which lead to the non-empty sets $S_{\mathbf{x}_i}$, $i = 1,\ldots,6$. The (uncontrolled) transition probabilities between the subsets $S_{\mathbf{x}}$, given by equations (5.19)-(5.21), are shown in Figure 14.

To show an example of transition probabilities, Figure 15 represents the transition probabilities outgoing from state $\mathbf{s}_6 = (\mathbf{r}_6,\mathbf{x}_1)$. Beside the self-transition, defined by equation (5.18), the figure shows that:

- Two transitions $t_\pi(\mathbf{s}_6,\mathbf{s}_{10})$ and $t_\pi(\mathbf{s}_6,\mathbf{s}_{14})$ exists from $\mathbf{s}_2$ to the subsets $S_{\mathbf{x}_2}$ and $S_{\mathbf{x}_4}$, respectively. Since the two states $\mathbf{s}_{10} = (\mathbf{r}_6,\mathbf{x}_2)$ and $\mathbf{s}_{14} = (\mathbf{r}_6,\mathbf{x}_4)$ are such that no routing table change is required, these transitions are uncontrolled (see definition (5.9)) and are equal to $t(\mathbf{x}_1,\mathbf{x}_2)$ and $t(\mathbf{x}_1,\mathbf{x}_4)$, respectively.
- Two controlled transitions $t_\pi(\mathbf{s}_6,\mathbf{s}_{11})$ and $t_\pi(\mathbf{s}_6,\mathbf{s}_{12})$ from $\mathbf{s}_6$ to the subset $S_{\mathbf{x}_3}$ exists, given by equation (5.15), since (i) $\mathbf{x}_1 > \mathbf{x}_3$, and (ii) the state $(\mathbf{r}_6,\mathbf{x}_3)$ is not feasible.

According to equations (5.16) and (5.20), the two controlled transitions are equal to $t_\pi(\mathbf{s}_6,\mathbf{s}_{11}) = u(\mathbf{s}_6,\mathbf{s}_{11})t(\mathbf{x}_1,\mathbf{x}_3)$ and $t_\pi(\mathbf{s}_6,\mathbf{s}_{12}) = u(\mathbf{s}_6,\mathbf{s}_{12})t(\mathbf{x}_1,\mathbf{x}_3)$, respectively, with $u(\mathbf{s}_6,\mathbf{s}_{11})$, $u(\mathbf{s}_6,\mathbf{s}_{12}) \in \{0,1\}$ and $u(\mathbf{s}_6,\mathbf{s}_{11}) + u(\mathbf{s}_6,\mathbf{s}_{12}) = 1$ (see definition (5.9)).

VI.3. MDP simulation results

Numerical simulations were performed with the aim of evaluating the effectiveness of the proposed MDP approach. The example home network described in previous section was considered. Two simulations were set up. Both simulations share the same scenario depicted above.

Three policies were computed by properly setting the algorithm parameters, denoted as ‘greedy’, ‘optimal’ and ‘optimal QoS’. The ‘optimal’ policy is the policy aimed at minimizing the cost (5.22) of changing the routing tables, without differentiating among the classes of service. The ‘optimal QoS’ policy takes into account also prioritization among the different classes of service. For comparison purposes, the ‘greedy’ policy is also considered, which, after a topology or traffic event, chooses the new routing table as the one which entails the least number of path changes and, in sequence, the least number of link changes.

With the ‘greedy’ and ‘optimal’ policies, the flows are not differentiated by their class of service, and the weights $w_{path}^{(k)}$ associated to path changes of class k, $k = 1,\ldots,K$, are equal to 1. With the ‘optimal QoS’ policy, flow 1 has higher priority: accordingly, the weight $w_{path}^{(1)}$ was increased and set equal to 2. In all the policies, the weight w_{link}, associated to link changes, was set equal to 0.025. Note that the link weight is much smaller then the path weights since it is used only to choose between routing tables which involve the same number of path changes.

From the Markov chain modeling of the sources, the mean time interval between traffic events, regarded as the finite-horizon time of the MDP, is computed as $t_{fh} = \left(\sum_{f=1}^{4} \frac{2\lambda_f \mu_f}{\lambda_f + \mu_f} \right)^{-1}$ = 15.54 min. Thus, the number of stages of the MDP is computed as $n = \lceil \gamma\, t_{fh} \rceil = 11$. To obtain the ‘greedy’ policy it is sufficient to set $n = 1$.

Algorithm parameters are shown in Table 6.

Policy	$w_{path}^{(1)}$	$w_{path}^{(2)}$	$w_{path}^{(3)}$	$w_{path}^{(4)}$	w_{link}	n
Greedy	1	1	1	1	0.0025	1
Optimal	1	1	1	1	0.0025	33
Optimal QoS	2	1	1	1	0.0025	33

Table 6 – First simulation set: algorithm parameters

Simulation 1 was aimed at evaluating the overall algorithm performances. 10 simulation runs were performed. For each run, the link and flow parameters were used to generate an event list; the events can be traffic events, i.e., flow births or terminations, and topology events, i.e., link state variations. At each traffic event, the MDP algorithm is performed and the initial routing table is selected according to the theory presented in the previous chapter. At each topology event, the policy computed by the MDP algorithm is applied to decide upon state transitions. Each simulation run was executed three times: the first time with the 'greedy' policy, the second time with the 'optimal' policy, the third time with the 'optimal QoS' policy.

Simulation results are collected by Table 7 and Table 8 and by Figure 16. Table 7 shows the mean number (over the 10 simulation runs) of routing table, path and link changes due to flow re-routing (i.e., to the decision to change the path of already active flows), denoted with $N_{\mathbf{r}}$, N_p and N_l, respectively, whereas Table 8 shows the per-flow path changes, denoted with $N_p(i)$, $i = 1,\dots,4$. Figure 16 shows the ratio between the values obtained with the 'optimal' and 'optimal QoS' policies over the values obtained by the 'greedy' policy. The tables and the figure clearly show that:

I. the number of routing table, path and link changes are nearly halved thanks to the proposed MDP approach, both with the 'optimal' and with the 'optimal QoS' policies, with a slight advantage of the 'optimal' one;

II. the 'optimal QoS' policy manages to reduce the number of path changes experienced by flow 1, which is the flow with the highest priority (i.e., with the largest weight), both with respect to the 'greedy' and the 'optimal' policies; to achieve this result, the 'optimal QoS' policy increases the number of path changes experienced by the other flows (in particular, in this scenario, by flow 3).

Policy	N_r	N_p	N_l
Greedy	13.6	18.0	45.9
Optimal	7.3	10.0	23.7
Optimal QoS	7.3	11.1	25.9

Table 7 - Simulation 1: total routing table/path/link changes

Policy	$N_p(1)$	$N_p(2)$	$N_p(3)$	$N_p(4)$
Greedy	5.8	5.3	2.0	4.9
Optimal	4.5	1.0	1.5	3.0
Optimal QoS	1.1	3.5	3.2	3.3

Table 8 - Simulation 1: per-flow path changes

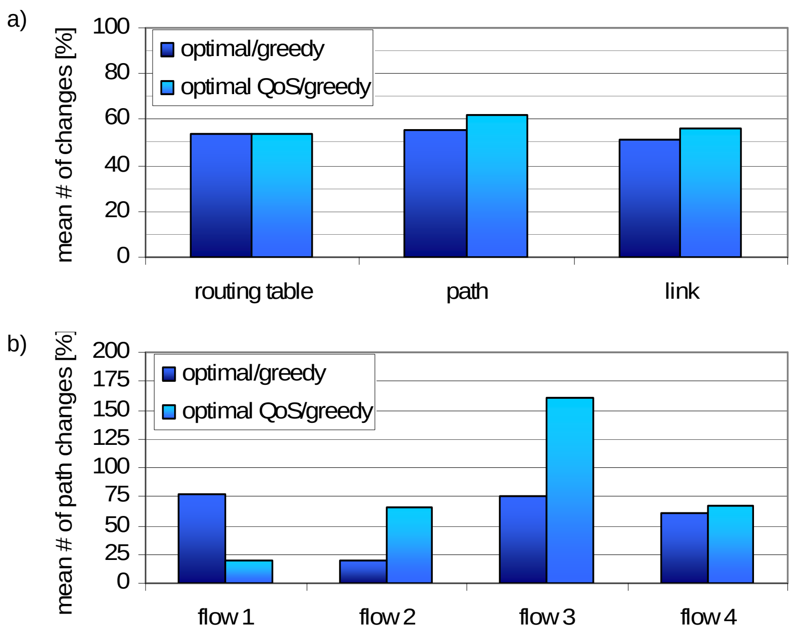


Figure 16 - Simulation 1 results

Simulation 2 was aimed at showing how the routing table is chosen in case of an acceptance of a new flow in the 'optimal' and in the 'optimal QoS' cases. To further emphasize the 'optimal QoS' behavior, the weight of flow 1 $w_{path}^{(1)}$ was increased to 5. In the example, initially all links are available, i.e., initial path status $\mathbf{x}$ is a P vector of ones; flows 2 and 3 are active and routed on paths 1 (OPT-ETH) and 5 (ETH), respectively. The algorithm is triggered by the acceptance of flow 1.

Table 9 and Table 10 and Figure 17 collect the algorithm results. Table 9and Figure 17 a) show i) the expected number of routing table changes $E_{\mathbf{r}}$ and the expected number of path changes E_p in the finite-horizon time t_{fh}, starting from the initial table[7], and ii) the probability $P_{\mathbf{r}}$ that the initial table is not changed in the finite-horizon time t_{fh}. Table 10 and Figure 17 b) show the expected number of path changes for flow, denoted with $E_{p(i)}$, $i = 1,\ldots,4$, in the finite-horizon time t_{fh}.

The resulting initial routing tables are [3 1 5] and [1 1 4] for the 'optimal' and 'optimal QoS' policies, respectively. The initial state obtained by 'optimal' policy, which is aimed at minimizing the total expected path changes, entails that the active flows 2 and 3 are not re-routed, and that flow 1 is routed on path 3 (OPT-WiFi); flow 1 cannot be routed on the more robust path 1 (OPT-ETH) since flows 2 and 3 already use the Ethernet link, whose capacity is not enough to support 3 flows. On the contrary, the 'optimal QoS' approach, which is aimed also at prioritizing flow 1, returns an initial state which implies to re-route flow 3 from path 5 (ETH) to path 4 (WiFi); in this way, flow 1, which is the highest priority flow, can be routed on the robust path 1 (OPT-ETH).

As shown by Table 9 and Figure 17 a), in the finite-horizon period t_{fh}, the 'optimal' routing policy achieves slightly lower $E_{\mathbf{r}}$ and $P_{\mathbf{r}}$, and significantly reduces E_p with respect to the 'optimal QoS' policy. Table 10 and Figure 17 b) shows that the 'optimal' policy addresses topology changes by re-routing flow 1, whereas the 'optimal QoS' policy re-routes flows 2 and 3: in this way, the 'optimal QoS' policy manages to minimize the expected number of path changes of the high-priority flow 1.

Policy	$E_{\mathbf{r}}$	E_p	$P_{\mathbf{r}}$
Optimal	0.529	0.558	0.419
Optimal QoS	0.581	2.163	0.450

Table 9 - Simulation 2: total expected routing table/path changes and probability of changing the initial routing table

[7] The collected statistics do not count the initial routing table change and the initial path change needed to route the new flow 2.

Policy	$E_p(1)$	$E_p(2)$	$E_p(3)$	$E_p(4)$
Optimal	0.513	0.030	0.015	0
Optimal QoS	0.033	0.581	1.549	0

Table 10 - Simulation 2: expected per-flow path changes

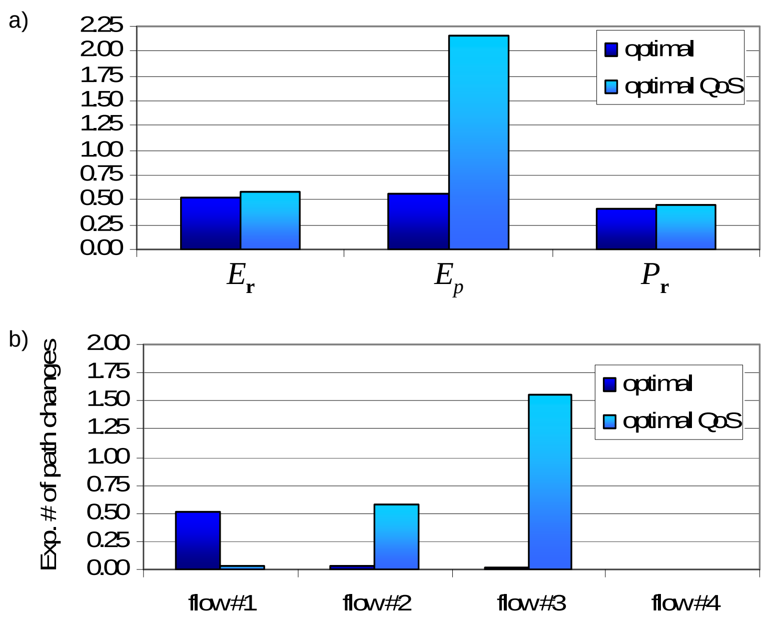


Figure 17 - Simulation 2 results

VI.4. Q-Learning simulation results

In this case, numerical simulations were performed with the aim of evaluating the effectiveness of the proposed Q-Learning approach. Again, the example home network described in previous section was considered. One simulation was set up in order to compare Q-Learning and MDP solutions performances.

In particular, the ‘optimal’ and the ‘greedy’ policies using the MDP approach has been compared with the “ε-greedy” policy using the Q-Learning approach. The ‘optimal’ MDP policy is the policy aimed at minimizing the cost (5.22) of changing the routing tables, without differentiating among the classes of service. The ‘greedy’ MDP policy is also considered, which, after a topology or traffic event, chooses the new routing table as the one which entails the least number of path changes and, in sequence, the least number of link changes. The “ε-greedy” Q-Learning instead is the policy aimed to minimizing the cost (5.22) of changing the routing tables on the basis of the acquired knowledge, but also performing exploration in order to increase and complete the knowledge about the system behaviour.

In this case I do not consider QoS, thus the flows are not differentiated by their class of service, and the weights $w_{path}^{(k)}$ associated to path changes of class k, $k = 1,\ldots,K$, are equal to 1.

Again, from the Markov chain modeling of the sources, the mean time interval between traffic events, regarded as the finite-horizon time of the MDP, is computed as $t_{fh} = \left(\sum_{f=1}^{4} \frac{2\lambda_f \mu_f}{\lambda_f + \mu_f} \right)^{-1}$ = 15.54 min. Thus, the number of stages of the MDP is computed as $n = \lceil \gamma\, t_{fh} \rceil = 11$. To obtain the 'greedy' policy it is sufficient to set $n = 1$

Algorithm parameters are shown in Table 11.

Policy	$w_{path}^{(1)}$	$w_{path}^{(2)}$	$w_{path}^{(3)}$	$w_{path}^{(4)}$	w_{link}	n	γ	α	ε
Greedy MDP	1	1	1	1	0.0025	1	-	-	-
Optimal MDP	1	1	1	1	0.0025	33	-	-	-
ε-Greedy QL	1	1	1	1	0.0025	-	0.95	0.9	0.1

Table 11 – Second simulation set: algorithm parameters

Two simulations were perfomed, the first one with 10 hours duration of network simulation, the second one with 20 hours duration of network simulation. Both the simulations was aimed at evaluating the overall Q-Learning algorithm performances and at comparing it bahviour with the MDP approach. For each simulation, the link and flow parameters were used to generate an event list; the events can be traffic events, i.e., flow births or terminations, and topology events, i.e., link state variations. At each traffic event, the MDP algorithm is performed and the initial routing table is selected according to the theory presented in the previous chapter. At each topology event, the policy computed by the MDP algorithm is applied to decide upon state transitions. Each simulation run was executed three times: the first time with the 'greedy' MDP policy, the second time with the 'optimal' MDP policy, the third time with the 'ε-greedy' Q-Learning policy. In addition, every 'ε-greedy' Q-Learning policy was simulated four times and the average routing table changes and re-routing table changes was calculated in order to avoid that exploration could have a too strong impact on the results.

Simulation 1 results (10 hours of network simulation) are collected by Table 12, by Figure 18 and Figure 19. Table 12 shows the number (the mean number over the 4 simulation runs for ‘ε-greedy’ Q-Learning policy) of routing table changes due both to routing and to flow re-routing (i.e., to the decision to change the path of already active flows), denoted with N_{r} and N_{re}, respectively.

The table and the figures clearly show that:

I. The performance of the QL algorithm in terms of number of routing table changes are better than the ones obtained with a greedy policy and at the same time are close to the ones obtained using the optimal MDP approach;

II. The same consideration ally also when considering the number of re-routing table changes.

Policy	N_{r}	N_{re}
Optimal MDP	130	35
Greedy MDP	135	40
ε-greedy QL	133.75	38.75

Table 12 - Simulation 1: total routing and re-routing table changes

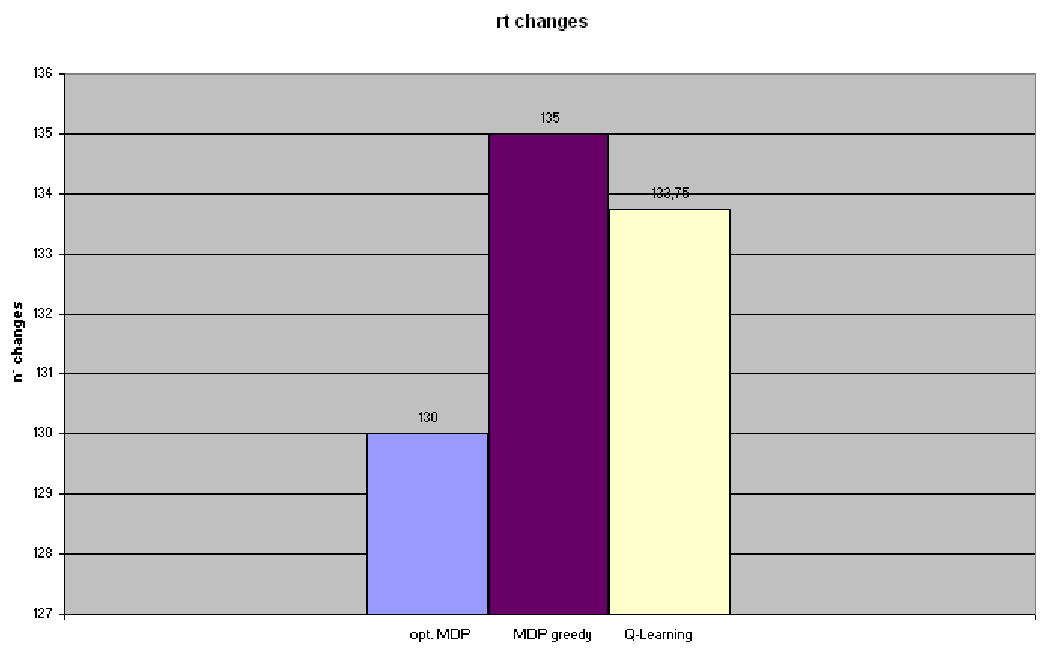


Figure 18 - Simulation 1 results (routing table changes)

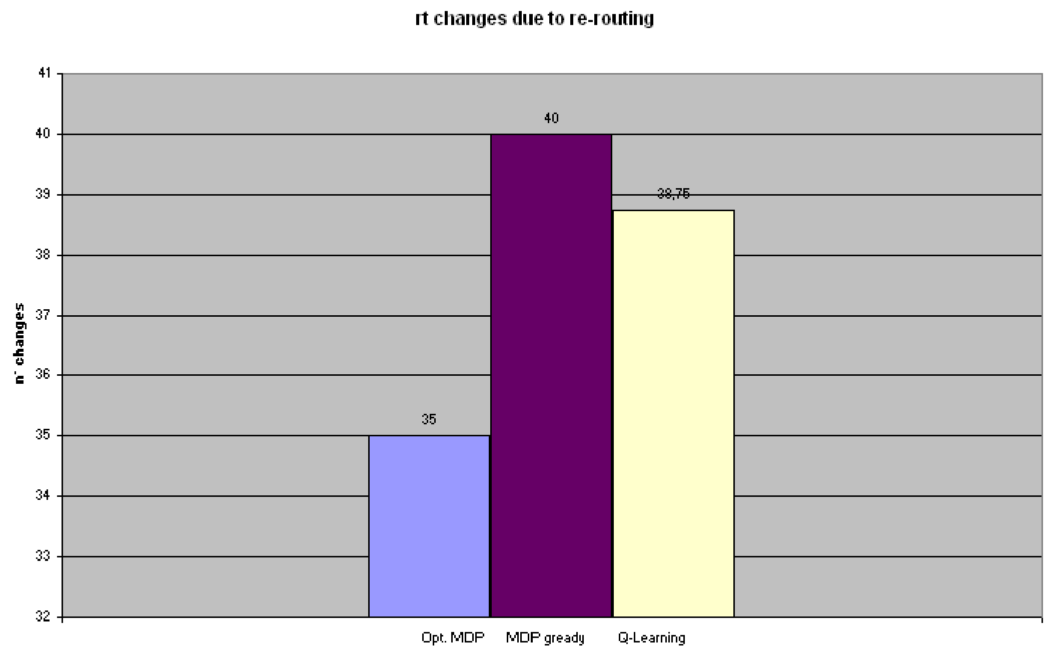


Figure 19 - Simulation 1 results (re-routing table changes)

Simulation 2 results (20 hours of network simulation) are collected by Table 13, by Figure 20 and Figure 21. Table 13 shows the number (the mean number over the 4 simulation runs for 'ε-greedy' Q-Learning policy) of routing table changes due both to routing and to flow re-routing (i.e., to the decision to change the path of already active flows), denoted with N_r and N_{re}, respectively.

The table and the figures clearly confirm the results obtained with the previous simulation. In particular, the proposed QL algorithm permits to achieve performances close to the optimal requiring less computational effort. From this consideration it appears that the proposed QL algorithm is suitable for real-time implementation as in the home network scenario depicted in Chapter 2.

Policy	N_r	N_{re}
Optimal MDP	54	6
Greedy MDP	66	18
ε-greedy QL	61.5	13.5

Table 13 - Simulation 2: total routing and re-routing table changes

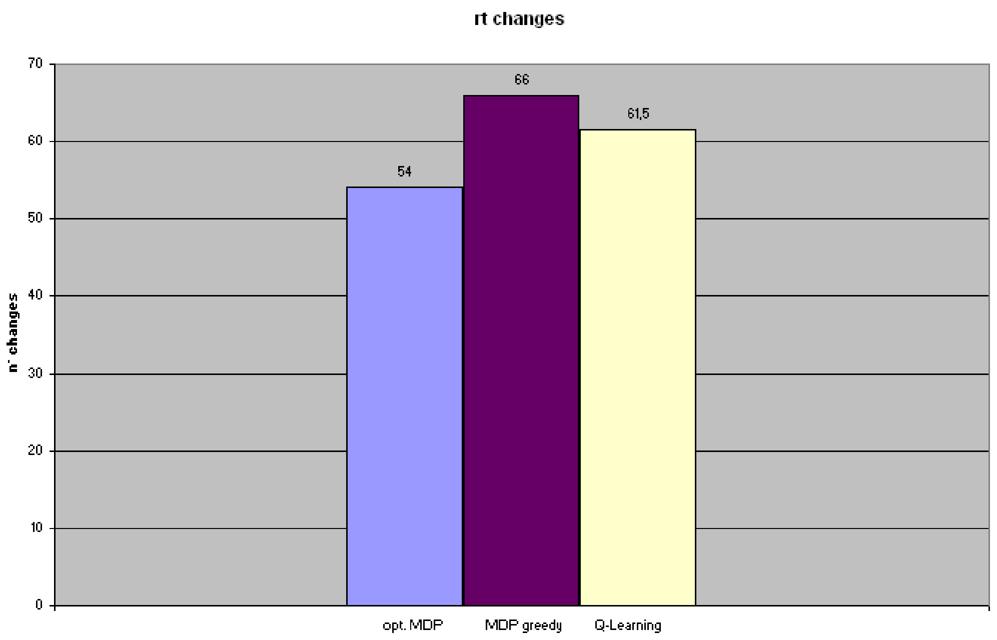


Figure 20 - Simulation 2 results (routing table changes)

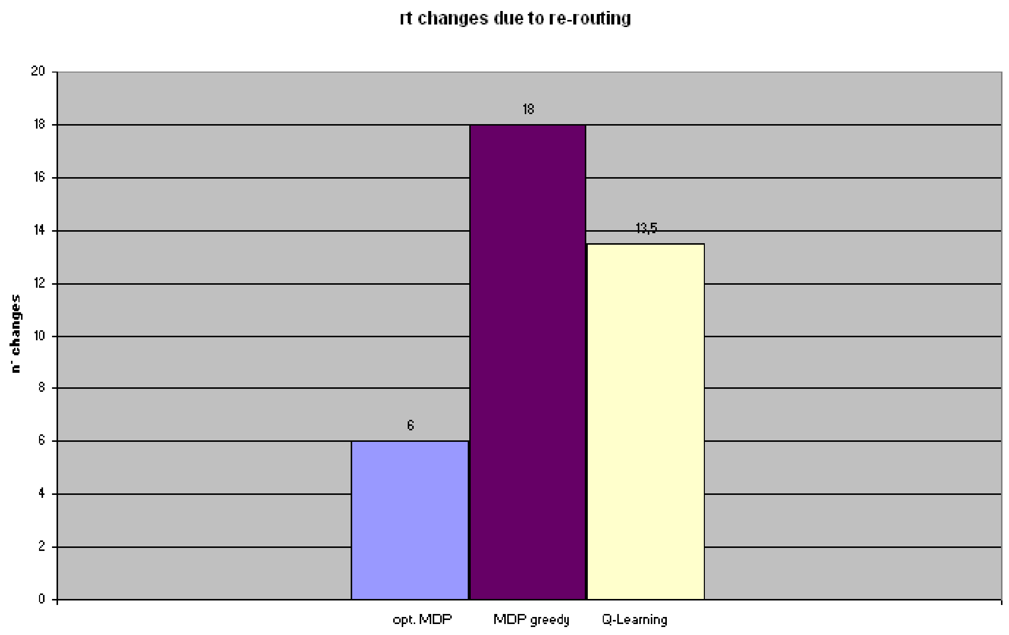


Figure 21 - Simulation 2 results (re-routing table changes)

Chapter VII
Conclusions

This thesis describes a fault-tolerant routing control algorithm for Next Generation Home Networks. The work has been done following two steps: i) definition of the MDP theoretical control framework, ii) definition of a Reinforcement Learning algorithm based on the control framework developed during the previous phase of the work.

The theoretical relevance of the first part of the work is that it defines an MDP framework for the fault-tolerant routing in communication networks characterized by time-varying path availabilities and supporting persistent multimedia flows; this is typically the case of heterogeneous home networks, where unreliable technologies such as Wi-Fi and Power Line Communications are used.

The innovative approach consists in i) considering the problem of re-routing flows between traffic events (e.g., flow acceptance/terminations); ii) defining the MDP dynamics depending on topology events (e.g., link faults). The resulting optimal fault-tolerant routing policy minimizes the re-routing occurrences, allows fast re-routing of flows in case of link faults, and also allows selecting the optimal initial state after a traffic event.

Numerical simulations validate the effectiveness of the proposed algorithm on a meaningful example of home network.

Moreover, the considered home network scenario, although representative of current home networks, is simpler than the future home network scenario considered in OMEGA, where tens or even hundreds of objects (from the TV to the washing machine) are inter-connected (the so-called Internet-of-Things (ITU Internet Reports, 2005)). In this scenario, the poor scalability of the MDP approach will prevent the use of the developed algorithm. Nonetheless, the proposed approach is still relevant since i) it provides a theoretical framework for developing more scalable Approximate Dynamic Programming and/or Reinforcement Learning algorithms, and ii) it provides an evaluation benchmark.

In the second part of the work, a Reinforcement Learning algorithm, based on the MDP theoretical framework built in the first phase, has been proposed to overcome scalability problems of MDP approach. This algorithm is suitable to be implemented in real-time environment and allows to achieve results that, has demonstrated by simulations, are close to optimal ones.

On-going work is aimed at a real network implementation of the proposed algorithm in the testbed under development within the European project OMEGA.

In addition, it is under study the possibility to extend the proposed MDP framework to include both traffic and topology dynamics.

References

[1] R. Holt et al., "Guide to Home Networks"; http://www.ce.org/networkguide/default.asp

[2] B. Rose, "Home networks: a standard perspective", IEEE Communication Magazine, December 2001

[3] ITU-T Reccommendation G.995.1; "Overview of digital subscriber line (DSL) Reccommendations", February 2001

[4] ITU-T Reccommendation G995.1; "Amendment 1: Overview of digital subscriber line (DSL) Recommendations", 2003

[5] DSL Forum Technical Report TR-058; "Multi-Service Architecture & Framework Requirements", 2003

[6] DSL Forum Technical Report TR-094; "Multi-Service Delivery Framework for Home Networks", 2004

[7] DSL Forum Technical Report TR-126; "Triple-play Services Quality of Experience (QoE) Requirements 13", 2006

[8] DSL Forum Technical Report TR-144; "Broadband Multi-Service Architecture & Framework Requirements", 2007

[9] OMEGA ICT project, http://www.ict-omega.eu/

[10] V. Suraci, F. Delli Priscoli, M. Castrucci, G. Tamea, W. De Vecchis, G. Di Pilla, "Inter-MAC: Convergence at MAC layer in Home Gigabit Network", ICT Mobile Summit 2008, Stockholm, Sweden, June 2008

[11] M. Castrucci, C. Liberatore, G. Tamea, P. Jaffrè, M. Bahr, "Functional analysis for Next Generation Home Networks", ICT Mobile Summit 2009, Santander, Spain, June 2009

[12] P. Jaffrè, J-P Javaudin, M. Castrucci, G. Tamea, C. Liberatore, M. Bahr, "Architecture reference model for Next Generation Home Networks", ICT Mobile Summit 2009, Santander, Spain, June 2009

[13] HomePlug Alliance Homeplug AV White Paper, 2005

[14] IEEE, "IEEE Standard for Information technology—Telecommunications and information exchange between systems— Local and metropolitan area networks—Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", 2003

[15] R. S. Sutton, A. G. Barto, "Reinforcement Learning: An Introduction", the MIT Press, 1988

[16] T. Clausen, P.Jacquet, "Optimized Link State Routing Protocol (OLSR)", IETF RFC 3626, October 2003

[17] D. F. Macedo, L. H. A. Correia, A. L. dos Santos, A. A. F. Loureiro, J. M. S. Nogueira, "A pro-active routing protocol for continuous, data dissemination in

wireless sensor networks", in proc. 10th IEEE Symposium on Computers and Communications, La Manga del Mar Menor, Spain, 2005

[18] L. Villasenor-Gonzalez, Y. Ge, L. Lamont, "HOLSR: a Hierarchical proactive routing mechanism for mobile ad hoc network", IEEE Communication Magazine, pp.118-125, July 2005

[19] C. Perkins, E. Belding-Royer, S. Das, "Ad hoc on-demand distance vector (AODV) routing", IETF RFC 3561, July 2003

[20] D. Johnson, Y.Hu, D. Maltz, "The dynamic source routing protocol (DSR) for mobile ad hoc networks for IPv4", IETF RFC 4728, February 2008

[21] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed Diffusion or Wireless Sensor Networking," IEEE/ACM Transactions on Networking, vol. 11, pp. 2–16, Feb 2003.

[22] D. Braginsky and D. Estrin, "Rumor Routing Algorithm for Sensor Networks," in Proc. of the 1st ACM International Workshop on Wireless Sensor Networks and Applications, Atlanta, Georgia, USA, 2002.

[23] Z. Dziong et al., "On Adaptive Call Routing Strategy for Circuit Switched Networks – Maximum Reward Approach," in Twelfth International Teletraffic Congress, (Torino), June 1988

[24] F. P. Kelly, "Routing in Circuit Switched Networks: Optimization, Shadow Prices and Decentralization," Advanced Applied Probability, vol. 20, pp. 112–144, 1988

[25] E. Nordstrom, J. Carlstrom, "A new reward model for MDP state aggregation with application to CAC and routing", European Transaction on Telecommunications, No. 16, pp 495-508, 2005

[26] E. Nordstrom, Z. Dziong, "CAC and routing for multi-service networks with blocked wide-band calls delayed, part I: exact link MDP framework", European Transaction on Telecommunications, No. 17, pp 21-36, 2006

[27] E. Nordstrom, Z. Dziong, "CAC and routing for multi-service networks with blocked wide-band calls delayed, part II: approximative link MDP framework", European Transaction on Telecommunications, No. 18, pp 13-33, 2007

[28] R.-H. Hwang, J. Kurose, D. Towsley, "MDP routing for multirate loss networks", Computer Networks, Vo.34, No.2, 2000

[29] F.S. Hillier, G.J. Lieberman, "Introduction to Operations Research", Sixth Edition. New York: McGraw Hill, ch. 21, 1995

[30] M.K. Marina, S.R. Das, "On-demand multipath distance vector routing in ad-hoc networks", 9th International Conference on Network Protocols, Riverside, California, 2001

[31] S. Nelakuditi, Z. –L. Zhang, "On Selection of Paths for Multipath Routing", 9th International Workshop on Quality of Service, LNCS, Vol. 2092, Springer-Werlag, London, 2001

[32] A. Tsirigos, Haas Z. J., "Multipath routing in the presence of frequent topological changes, IEEE Communication Magazine, November 2001

[33] K. Wu, J. Harms, "On-Demand Multipath Routing for Mobile Ad-Hoc Networks", Proceedings of EMPCC, Vienna, February 2001

[34] J. Tsai, T. Moors, “A Review of Multipath Routing Protocols: From Wireless Ad Hoc to Mesh Networks”, ACoRN Early Career Researcher Workshop on Wireless Multihop Networking, July 2006

[35] Y. Xue, K. Nahrstedt, “Fault tolerant routing in mobile ad-hoc networks”, in proc. IEEE Wireless Communication Networking conference, pp.1174-1179, March 2003

[36] J. Oommen, S. Misra, “A fault-tolerant routing algorithm for mobile ad hoc networks using a stochastic learning-based weak estimation procedure”, IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, June 2006

[37] F.S. Hillier, G.J. Lieberman, “Introduction to Operations Research”, Sixth Edition. New York: McGraw Hill, ch. 21, 1995

[38] A. Pietrabissa “Optimal call admission and call dropping control in links with variable capacity”, European Journal of Control, Vol. 15, N. 1, pp. 56-57, 2009

[39] D.P. Bertsekas, Dynamic Programming: Deterministic and Stochastic Models. Prentice-Hall, Englewood Cliffs, NJ, 1987.

[40] Heyman, D., and M. Sobel, “Stochastic Models in Operations Research”, vol. 1, McGraw-Hill, New York, 1982

[41] Ross, S., “Stochastic Processes”, 2d ed., Wiley, New York, 1995

[42] Puterman, M. L., “Markov Decision Processes: Discrete Stochastic Dynamic Programming”, Wiley, New York, 1994

[43] Feinberg, E. A., and A. Shwartz “Markov Decision Processes: Research Directions and Applications”, Kluwer Academic Publishers, Boston, 2001

[44] Whittle, P. “Optimization over Time: Dynamic Programming and Stochastic Control”, Wiley, New York, vol. 1, 1982; vol. 2, 1983

[45] R.E. Bellman, “On a Routing Problem”, Qtrly. Applied Mathematics, Vol. 16, pp. 87-90, 1958.

[46] ETSI, European Telecommunication Standard Institute, www.etsi.org

[47] ITU-T, International Telecommunication Union – Telecommunication Standardization Section, www.itu.int/ITU-T

[48] WiMAX Forum, www.wimaxforum.org

[49] BroadBand Forum, www.broadband-forum.org

[50] Home Gateway Initiative, www.homegatewayinitiative.org

[51] S. Waharte, R. Boutaba, Y. Iraqi, B. Ishibashi, “Routing protocols in wireless mesh networks: challenges and design considerations”, Multimedia Tools Applications, Vol. 29, pp. 285-303, 2006

[52] U. Lee, S. F. Midkiff, J. S. Park, “A Proactive Routing Protocol for Multi-Channel Wireless Ad-hoc Networks (DSDV-MC)”, International Conference on Information Technology: Coding and Computing, ITTC, 2005

[53] I. F. Akyildiz, X. Wang, W. Wang, “Wireless mesh networks: a survey”, Computer Networks, Vol. 47, pp. 445-487, 2005

[54] Y. Bejerano, S.-J. Han, A. Kumar, “Efficient load-balancing routing for wireless mesh networks”, Computer Networks, Vol. 51, pp. 2450-2466, 2007

[55] C. J. C. H. Watldns, P. Dayan, “Q-learning”, Machine Learning, Vol. 8, pp. 279-292, 1989