Imperial College London
Department of Electrical and Electronic Engineering

Traffic and Task Allocation in Networks and the Cloud

Lan Wang

January 2017

Supervised by Prof. Erol Gelenbe

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Electrical and Electronic Engineering of Imperial College
London and the Diploma of Imperial College London

Abstract

Communication services such as telephony, broadband and TV are increasingly migrating into Internet Protocol(IP) based networks because of the consolidation of telephone and data networks. Meanwhile, the increasingly wide application of Cloud Computing enables the accommodation of tens of thousands of applications from the general public or enterprise users which make use of Cloud services on-demand through IP networks such as the Internet. Real-Time services over IP (RTIP) have also been increasingly significant due to the convergence of network services, and the real-time needs of the Internet of Things (IoT) will strengthen this trend. Such Real-Time applications have strict Quality of Service (QoS) constraints, posing a major challenge for IP networks. The Cognitive Packet Network (CPN) has been designed as a QoS-driven protocol that addresses user-oriented QoS demands by adaptively routing packets based on online sensing and measurement. Thus in this thesis we first describe our design for a novel "Real-Time (RT) traffic over CPN" protocol which uses QoS goals that match the needs of voice packet delivery in the presence of other background traffic under varied traffic conditions; we present its experimental evaluation via measurements of key QoS metrics such as packet delay, delay variation (jitter) and packet loss ratio. Pursuing our investigation of packet routing in the Internet, we then propose a novel Big Data and Machine Learning approach for real-time Internet scale Route Optimisation based on Quality-of-Service using an overlay network, and evaluate is performance. Based on the collection of data sampled each 2 minutes over a large number of source-destinations pairs, we observe that intercontinental Internet Protocol (IP) paths are far from optimal with respect to metrics such as end-to-end round-trip delay. On the other hand, our machine learning based overlay network routing scheme exploits large scale data collected from communicating node pairs to select overlay paths, while it uses IP between neighbouring overlay nodes. We report measurements over a week long experiment with several million data points shows substantially better end-to-end QoS than is observed with pure IP routing. Pursuing the machine learning approach, we then address the challenging problem of dispatching incoming tasks to servers in Cloud systems so as to offer the best QoS and reliable job execution; an experimental system (the Task Allocation Platform) that we have developed is presented and used to compare several task allocation schemes, including a model driven algorithm, a reinforcement learning based scheme, and a "sensible allocation algorithm that assigns tasks to sub-systems that are observed to provide lower response time. These schemes are compared via measurements both among themselves and against a standard round-robin scheduler, with two architectures (with homogenous and heterogenous hosts having different processing capacities) and the conditions under which the different schemes offer better QoS are discussed. Since Cloud systems include both locally based servers at user premises and remote servers and multiple Clouds that can be reached over the Internet, we also describe a smart distributed system that combines local and remote Cloud facilities, allocating tasks dynamically to the service that offers the best overall QoS, and it includes a routing overlay which minimizes network delay for data transfer between Clouds. Internet-scale experiments that we report exhibit the effectiveness of our approach in adaptively distributing workload across multiple Clouds.

Acknowledgements

I would like to express my sincere appreciation to my supervisor, Professor Erol Gelenbe, for his invaluable academic and financial support with which I was able to start my research journey. His talent and passion on research have been always inspiring me. I also wish to thank Dr. Olivier Brun for the productive co-operation with him and his research group in the FP7 project during my PhD study. I am also grateful to my colleagues for the warm and friendly atmosphere in the Intelligent System and Network Group. Special thanks to Dr. Ricardo Lent, Gokce and Omer for their selfless help and support whenever I needed for my research. Many Thanks to Patrick Kelly for his administration work which made my PhD life much easier. I also thank my viva examiners Professor Yiannis Demiris and Dr. Toktam Mahmoodi for their effort in reviewing this thesis and their valuable comments. They gave me a very nice and memorable viva experience and I enjoyed discussing my research with them. Great thanks to my doctors Dr. Shirley D'SA, Professor Michael LUNN and Aviva Cerner who have been taking care of my health and really care about me like my family in the country thousands of miles away from my motherland. Most of all, I want to thank my family for allowing me to pursue my academic dream which I have had since I was a child. I would like to send what I've achieved as a gift to my father who is a talented scientist in material science and who is my spiritual support, and to Nianxi who I love most in this world.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Internet was initially designed to transfer files, and then later developed into a major resource for email and then for access to (and downloads from) web sites. The interaction with web sites offered by the Internet has enables e-commerce, Internet banking and other applications. Today it has also become a major means of face-to-face real-time communications and voice communications. Perhaps in the future it may also be increasingly used for on-line virtual reality, which will also requires real-time capabilities, and doubtless we will continue seeing new forms of communication that exploit the Internet. In addition to communications, the needs for web access has meant that web services and access to large data sets for social media and Google, now require Cloud computing to be integrated into the Internet. At the same time, the management of mobile networks and the Internet itself is transitioning into Cloud and smaller distributed Fog services.

In the meanwhile the Internet of Things (IoT) will increasingly use the Internet backbone with an obvious need for real-time capabilities to meet the sensor and actuator capabilities of the IoT. These real-time applications, such as Voice-over-IP, face-to-face communications and the IoT all require some level of Quality of Service (QoS) which is not generally imbedded into the Internet infrastructure. Real-Time implies that packets should be conveyed within a predetermined delay; furthermore the order in which the packets are sent should correspond to the order in which they are received.

Thus this thesis focuses on how we can dynamically and in a distributed manner create the means for QoS that is adapted to various specific uses of the Internet. We follow an approach: that combines design and experimentation, so that the results we present are based on experimental studies conducted throughout our work, where we have used laboratory test-beds as well as Internet scale experiments. Because we cannot expect that the schemes that we propose can be imbedded into the commercial Internet infrastructure which needs to meet wide ranging general purpose requirements, we propose to improve QoS using the dynamic control of paths in the Internet, and the dynamic control of performance for individual tasks in the Cloud.

Regarding network routing, we use the Cognitive Packet Network [Gel09] that implements a form of reinforcement learning [SB98]. We also use an experimental test-bed that implements this routing protocol [GLN04]. CPN acts on the basis of a QoS objective of goal G which incorporates measured quantities, and seeks paths in the network that will improve (i.e. minimise) this value G. CPN has been used in other recent research. For instance in [SG10] it has been used to defend a network against worm attacks, in [GLL06] it is used in conjunction with a genetic algorithm to optimise paths in a multi-hop network, while in [GG07] it is shown that path switching can be beneficial to achieving improved QoS using the CPN protocol.

Thus, based on CPN we design and test the real-time CPN protocol (RTCPN), and evaluate the improvements it can bring in terms of Delay, Jitter and Packet Loss, which are the QoS parameters that are often used for communication needs that require a real-time capability [WG15b, WG16].

Since such schemes can be used more easily at the overlay level, we also modify CPN so as to use it as an overlay network routing capability. We show that if the number of overlay hops is limited, a big data approach to CPN with data collected over long periods of time can result in significant improvements in end-to-end delay [BWG16].

Finally we exploit the machine-learning/reinforcement-learning approach to design an algorithm for real-time task allocation in Cloud servers, and provide experimental data on a laboratory test-bed [WG18].

1.1 Contributions

Real-Time services over IP (RTIP) have been increasingly significant due to the convergence of telephone and data networks worldwide around the IP standard, and the popularisation of the Internet. The real-time application we study is Voice-over-IP(VOIP) because of the dominant role of the voice communication. In order to improve the packet delay and delay variation which are of great importance to quality of voice communications, we design a "Real-Time (RT) traffic over CPN" protocol which uses QoS goals that match the needs of voice packet delivery in the presence of other background traffic under varied traffic conditions. The experimental results show that (1) using Jitter Minimisation as the QoS goal for routing all of the traffic flows in the network, namely the voice traffic and the background (other) traffic, one can minimise jitter but also delay and loss for the voice traffic, (2) packet loss, that can be provoked by path switching and other effects in a network, also results in further buffer overflows and hence further losses in the output resequencing buffers for VOIP codecs.

Cloud computing is now widely integrated into the Internet. It enables the consolidation of an increasing number of applications from the general public or enterprise users which generate diverse sets of workloads in terms of resource demands and performance requirements [DK13]. Moreover, the heterogeneity in the hardware configuration of physical servers or virtual machines in terms of the specific speeds and capacities of the processor, memory, storage, and networking subsystems further complicates the matching of applications to available machines. In order to address the challenge for Cloud service providers to dispatch incoming tasks to servers with the assurance of the quality and reliability of the job execution required by end users while also improving efficiency in the usage of resources, we propose an approach both for IP-based networks and Clouds which uses Reinforcement Learning with the Random Neural Network to make fast, judicious and efficient decisions on the best node or path based on the knowledge learned from the past observations, while adapting to changes in workload and on-going performance of the network and Cloud environment. The approach benefits from limited measurement overhead and can be easily deployed over a large population of machines.

We exploit the approach to design an algorithm for real-time task-to-resource allocation in cloud servers and an overlay routing algorithm to improve the overall QoS of Internet connections for data transfer between clouds, while still exploiting the exsiting IP protocol with path variations offered by an overlay network. We've evaluated the proposed algorithms on a laboratory test-bed and validated the RNN-based algorithm outperforms the Round Robin and sensible algorithm in adaptively distributing workloads across available servers within a cloud in response to user required QoS when there is a great diversity in the types of jobs, the class of QoS goals and the resources which are required by workloads and which are possessed by server.

The proposed overlay routing algorithm has been implemented and tested in an intercontinental overlay network that includes Europe, Asia, North and South America, and Australia and we have observed that with at no more than two overlay nodes in each connection, round trip packet delays are generally very close within a few percent, to the round trip delays observed with three or four overlay nodes per connection. We further observe that significant improvements can also be obtained when the RNN based adaptation uses no more than two alternate paths which emerge as the best, as a result of a wider search. In this way, we reduce the measurement overhead used by a N node overlay network from $O(N^2)$ to $O(N)$.

1.2 Overview of the Thesis

In chapter 2 we describe our design for the "Real-Time (RT) traffic over CPN" protocol which uses QoS goals that match the needs of real-time packet delivery in the presence of other background traffic under varied traffic conditions, and we present its experimental evaluation via measurements of the key QoS metrics which are packet delay, delay variation (jitter) and packet loss ratio. The work in this chapter was published in our papers[WG16].

Pursing our investigation of packet routing in the Internet, chapter 3 uses Big Data and Machine Learning for the real-time Internet scale Quality-of-Service Route Optimisation using an overlay network. Based on the collection of data sampled each 2 minutes over a large number of source-destinations pairs, we show that intercontinental Internet Protocol (IP) paths are far from optimal with respect to Quality of Service (QoS) metrics such as end-to-end round-trip delay. We therefore develop a machine learning based scheme that exploits large scale data collected from communicating node pairs in a multi-hop overlay network. IP is used between the overlay nodes, but the overlay nodes select multi-hop overlay paths that provide substantially better QoS than IP. The routing scheme is developed on a 20-node intercontinental overlay network that collects some 2×10^6 measurements per week, and makes scalable distributed routing decisions. Experimental results show that this approach improves QoS significantly and at low computational cost. This work was published in[BWG16].

Chapter 4 presents an experimental system that can exploit a variety of online QoS aware adaptive task allocation schemes, and three such schemes are designed and compared. These are a measurement driven algorithm that uses reinforcement learning, secondly a “sensible” allocation algorithm that assigns tasks to sub-systems that are observed to provide a lower response time, and then an algorithm that splits the task arrival stream into sub-streams at rates computed from the hosts’ processing capabilities. All of these schemes are compared via measurements among themselves and with a standard round-robin scheduler, on two experimental test-beds with homogenous and heterogenous hosts having different processing capacities. We also present the study of the effectiveness of TAP when there is greater diversity both in the types of tasks, and in the type of QoS criteria and the SLA that they request. This work is presented in our publications[WG18, GW15, WG15a].

Cloud systems include both locally based servers at user premises and remote servers and multiple Clouds that can be reached over the Internet. Thus chapter 5 describes a smart distributed system that combines local and remote Cloud facilities. It operates with a decision system that allocates tasks dynamically to the service that offers the best overall Quality of Service, and includes the effect of a routing overlay which minimizes network delay for data transfer between clouds. Internet-scale experiments that we report exhibit the effectiveness of our approach in adaptively distributing workload across multiple clouds. This work was reported in our publication[WBG16].

1.3 The Random Neural Network

The random neural network was first introduced in [Gel89a, Gel89b] and has been used in numerous machine learning based applications [GT08] including in image processing. In our work we use it as a machine learning based decision making tool, both for the Real-Time CPN routing algorithm, as well as the overlay network routing and the Cloud task allocation work. As a decision making unit, the RNN stores recent measurement data and provides an indication of which decision is best (which path to choose, or which server to allocate a task) based on a combination of recent measurements.

The random neural network (RNN) [Gel93, Gel00] is a recurrent model, i.e. it can contain feedback loops as in the present work, but it can also be used in feedforward mode as with conventional Artificial Neural Networks. It has a finite number of n interconnected neurons. Its state is a vector $\mathbf{k}(t) = [\mathbf{k}_1(t), \mathbf{k}_2(t), \ldots, \mathbf{k}_n(t)]$, where $\mathbf{k}_i(t)$ is a non-negative integer valued *random variable* representing the "potential" of the i-th neuron at time t. The probability of the state of the RNN is denoted by $p(k,t) \equiv Pr[\mathbf{k}(t) = k]$, where $k = (k_1, \ldots k_n)$ and the $k_i \geq 0$ are integers. Its stationary probability distribution function, when it exists, is $p(k) \equiv \lim_{t\to\infty} p(k,t)$.

A neuron i of the RNN is said to be excited whenever $\mathbf{k}_i(t) > 0$, in which case it can fire and send signals at an average rate r_i, with exponential, independent and identically distributed inter-spike intervals. Spikes will go from neuron i to neuron j with probability $p^+_{i,j}$ as excitatory spikes, and with probability $p^-_{i,j}$ as inhibitory spikes, where $\sum_{j=1}^{n} p^+_{i,j} + p^-_{i,j} + d_i = 1, \quad i = 1, \ldots, n$, and d_i is the probability that the fired spike is lost in the network or that it leaves the network towards some external system.

Let $w^+_{i,j} = r(i)p^+_{i,j} \geq 0$ and $w^-_{i,j} = r(i)p^-_{i,j} \geq 0$; they denote the emission rates of excitation and inhibition signals from neuron i to neuron j. In addition, for any neuron i, exogenous excitatory, and inhibitory spikes, can enter the neuron from outside the network at Poisson rates denoted by Λ_i and λ_i, respectively. By its no-linearity, the mathematical structure of a RNN differs from that of widely used queuing systems such as the Jackson network or the BCMP model [GM76, GT08].

However, despite this major difference (and the non-existence of properties such as quasi-reversibility of the RNN model equations), it has been shown to have a product form solution [Gel93] given by:

$$p(k) = \prod_{i=1}^{n} (1 - q_i)\, q_i^{k_i},\ q_i = \frac{\lambda^+_i}{r(i) + \lambda^-_i}, \tag{1.1}$$

where $q_i \lim_{t\to\infty} Pr[\mathbf{k}_i(t) > 0]$ is the stationary probability that neuron i is excited, and λ^+_i and λ^-_i, represent the total flows of excitatory and inhibitory spikes arriving at neuron i, and satisfy a system of nonlinear simultaneous equations:

$$\lambda_i^+ = \sum_j q_j w_{j,i}^+ + \Lambda_i,\ \lambda_i^- = \sum_j q_j w_{j,i}^- + \lambda_i.$$

and q_i is obtained from the solution of the following system of non-linear equations:

$$q_i = \frac{\lambda_i^+}{r_i + \lambda_i^-}. \tag{1.2}$$

Since it has been proved in [Gel93] that these equations have an unique solution with all $q_i \in [0,1]^n$, the non-linear system (1.2) can be solved efficiently using the simple fixed-point iteration:

$$q_i^{k+1} \leftarrow min\big[1, \frac{\sum_j q_j^k w_{j,i}^+ + \Lambda_i}{r_i + \sum_j q_j^k w_{j,i}^- + \lambda_i}\big], \tag{1.3}$$

starting with the initial values $q_i^0 = 0.5$ for all the $i = 1,\ ..\ , n$.

1.4 The Cognitive Packet Network (CPN)

CPN is a QoS-driven routing protocol used for routing traffic in a network where users (applications) can declare their desired QoS requirements. CPN adaptively routes their traffic by means of online sensing and measurement so as to provide the best possible QoS requested by the users [GLX01, GLMX00, GLMX02, Gel04, GGL$^+$04]. CPN distributes the intelligence at each node of a network and makes fast routing decisions at low computational cost using Reinforcement Leaning algorithm. It performs self-improvement so as to make routing decisions for improving QoS by learning from previous observations to adapt to the network conditions which change over time.

QoS requirements specified by users, such as $Delay$, $Loss$, $EnergyConsumption$, or a combination of the above, are incorporated in the ”goal” function which is used for the CPN routing algorithm. Three types of packets are used by CPN: smart packets (SPs), dumb packets (DPs) and acknowledgments (ACKs). SPs explore the route for DPs and collect measurements; DPs carry payload and also conduct measurements; ACKs bring back the information that has been discovered by the SPs and

DPs including route information and measurement results. SPs discover the route using random neural network (RNN)-based reinforcement learning (RL) [Gel00, GLN04] which resides in each node. Each RNN in a node corresponds to a QoS class and destination pair and each neuron of a RNN represents the choice to forward a given packet over a specific outgoing link from the node. The arrival of an SP for a specific QoS class at a node triggers the execution of the RNN-based algorithm.

During this process, the weights of the corresponding RNN are updated by Reinforcement Learning (RL) using QoS goal-based measurements that are collected by SPs and brought back by ACKs and stored in a mailbox at the node. Following that, the output link corresponding to the most excited neuron is chosen as the routing decision. More detail on CPN routing is available in [GLX01]. The paths explored by the SPs for the desired QoS goal are brought back to the source node by an ACK packet and stored in a list of possible paths. Among these, DPs will select one which is recent and has the best QoS goal; as DPs are sent forward, they will also create returning ACKs which can be used to update the performance (e.g. end-to-end delay or jitter) that is needed by the application or user that is forwarding the packets.

As indicated in Section1.1, the task allocation algorithm that we design and evaluate for cloud servers, is also inspired by CPN.

1.5 Statement of Originality

This thesis is submitted for the degree of Doctor in Philosophy in the Department of Electrical and Electronic Engineering at Imperial College London. I certify that all material in this thesis which is not my own is acknowledged.

1.6 Copyright

1.7 Publications

Journal Publications

- Lan Wang and Erol Gelenbe. "Adaptive dispatching of tasks in the cloud." IEEE Transactions on Cloud Computing, PP(99):11, 2015.
- Olivier Brun, Lan Wang, and Erol Gelenbe "Big data for autonomic intercontinental overlays." IEEE Journal on Selected Areas in Communications, 34(3):575583, March 2016.
- Lan Wang. The Random Neural Network for Cognitive Traffic Routing and Task Allocation in Networks and the Cloud. Probability in the Engineering and Informational Sciences, 31(4), 540-560, 2017.

Conference Publications

- Lan Wang and Erol Gelenbe. "Experiments with smart workload allocation to cloud servers." In 2015 IEEE Fourth Symposium on Network Cloud Computing and Applications (NCCA), pages 3135, June 2015.
- Lan Wang and Erol Gelenbe. "Real-time traffic over the cognitive packet network." In Proceedings of 23rd International Science Conference on Computer Networks (CN2016). Official Distinction Award at the 23th Computer Network Conference, Poland, June 2016.
- Lan Wang, Olivier Brun, and Erol Gelenbe. "Adaptive workload distribution for local and remote clouds." In Proceedings of IEEE International Conference on SYSTEMS, MAN, AND CYBERNETICS (SMC2016). Budapest, Hungary, October 2016.

- Lan Wang. “Online Work Distribution to Clouds.” In IEEE 24th International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS’16). (Accepted for Publication)
- Erol Gelenbe and Lan Wang. “TAP: A task allocation platform for the EU FP7 PANACEA project.” In The proceedings of the EU projects track, September 2015.
- Lan Wang and Erol Gelenbe. “Demonstrating voice over an autonomic network.” In 2015 IEEE International Conference on Autonomic Computing, Grenoble, France, July 7-10, 2015, pages 139140, 2015.

Chapter 2

Real-Time Traffic over the Cognitive Packet Network

Real-Time services over IP (RTIP) have been increasingly significant due to the convergence of data networks worldwide around the IP standard, and the popularisation of the Internet. Real-Time applications have strict Quality of Service (QoS) constraint, which poses a major challenge to IP networks. The Cognitive Packet Network (CPN) has been designed as a QoS-driven protocol that addresses user-oriented QoS demands by adaptively routing packets based on online sensing and measurement, and in this chapter we design and experimentally evaluate the "Real-Time (RT) over CPN" protocol which uses QoS goals that match the needs of real-time packet delivery in the presence of other background traffic under varied traffic conditions. The resulting design is evaluated via measurements of packet delay, delay variation (jitter) and packet loss ratio.

This chapter is based on the following papers that we have published:[WG16].

2.1 Introduction

Communication services such as telephone, broadband and TV are increasingly migrating into the IP network with the worldwide deployment and operation of Next-Generation Networks which consolidate telephone and data networks into common IP infrastructures [FCC13, BMMV08]. Applications such as remote security [Gel10] and the needs of cyber-physical systems [GW12] are also pushing the bounds in this direction. However, our IP networks are not well designed to guarantee Quality of Service (QoS) for such diverse communications [SS04], including voice (VOIP) [JVK05] and web based search, video and multimedia [DH11, DHC11, SPA12, BKC13]. For example, voice service constraints [TCAXEVTR11] could be well assured in dedicated circuit-switched connections, but are difficult to cover with IP networks [RAS05]. To achieve the same quality of voice transmission as basic telephone services, VOIP transmissions must meet the the following standards[cis]:

- The default G.729 codec requires packet loss far less than 1 percent to avoid audible errors.
- The ITU G.114 specification recommends less than 150 millisecond (ms) one-way end-to-end delay for high-quality real-time traffic such as voice.
- Jitter buffers (used to reorder packets and buffer them to reduce jitter) further add to the end-to-end delay, and are usually only effective on delay variations less than 100 ms.

Furthermore, industrial distributed systems and industrial networks [CC10, CMA$^+$09, FJG13, GJF13] are pushing the curve with respect to real-time communication needs. Real-time needs for enhanced reality and telepresence [GHK05] are also creating specific needs in on-line QoS management for multimedia systems [SBAMP11]. Thus in the past decade, many QoS approaches have been proposed including IntServ & RSVP, DiffServ and MPLS, which specify a coarse-grained mechanism for providing QoS, but fully satisfactory results are not yet available in this area and there is space for further investigation and research[TF03, AP06, LDP01].

2.1.1 Technical Approach

This chapter investigates the use of the Cognitive Packet Network (CPN) routing protocol for real-time traffic. The real-time application we study is Voice-over-IP(VOIP) because of the dominant role of the voice communication To address the needs of real-time traffic, this chapter introduces a variant of CPN with a “goal” function that addresses real-time needs with regard to ”Jitter” according to RFC3393 [DC02], and implements new functions in CPN to support multiple QoS classes for multiple traffic flows. Furthermore, since adaptive routing takes place when CPN is used in order to meet some of the requirements of this QoS goal, path switching can occur and we evaluate the correlation of end-to-end packet loss with path switching and with the delay that occurs at the end-user's re-sequencing buffer that is needed to insure that packets are received in time-stamp order. The main experimental result of this chapter is that using Jitter Minimisation as the QoS goal for routing all of the traffic flows in the network, we can minimize jitter but also minimize delay and loss for voice traffic. However we also see that adaptive schemes that switch paths to minimize delay, loss or jitter, may also cause buffer overflow and hence losses in the output re-sequencing buffers that deliver packets in time-stamp order for voice traffic.

2.1.2 CPN for Real-Time Traffic

Previous research has suggested the ability of CPN to provide improved QoS for real-time traffic compared with the conventional Internet protocol such as OSPF [Gel07]. This chapter presents an experimental study specifically for voice service over CPN, the principal building block of which is shown in Figure 2.1.

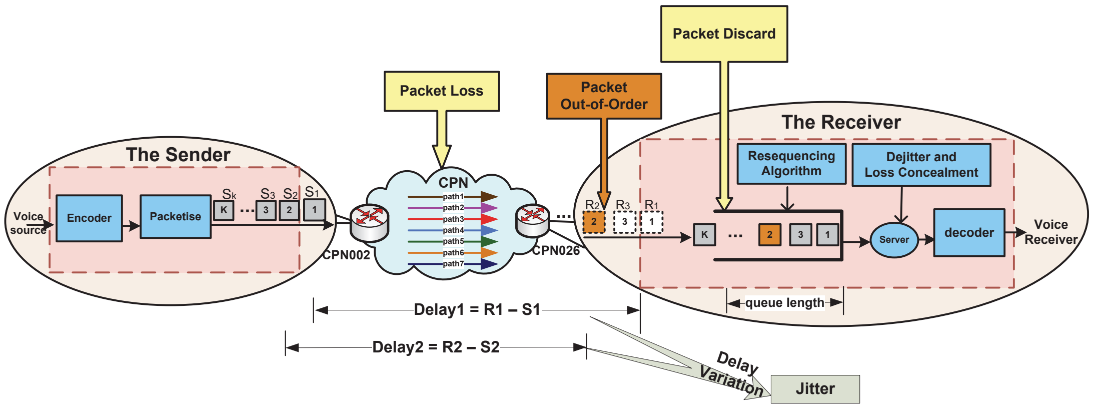


Figure 2.1: VOIP System Structure [BGP84]

In Real-Time services that are important in a variety of applications, including industrial control and media transmission, the traffic sent from a source must not only arrive with a small delay, but it must also arrive with small variance not to disrupt the needs of the control application or the media receiver.

However, very importantly, the end receiver must receive the sender's packets *in the order in which they are sent*.

Indeed in most control applications the order in which the control messages are received has great importance for industrial control [MQCdM08, BM13]. Similarly, any measurements that are sent to a controller from different parts of a distributed system, have to be received in time-stamp sending order. This is similar for media where video must arrive in the order that was prescribed by the sender, and for real-time conversations where the meaning of a conversation cannot be preserved if the packets arrive in some other than time-stamp order. Note that for data packets in file transfers, this aspect is of no great importance since the packets can be reassembled in a receiving file system. Because of the human delays at the terminal, email also does not have such stringent constraints on the order of packet arrival. Thus, at the sender which resides in a VOIP application installed at a CPN node, the original analog signals are sampled with a fixed frequency, which is commonly 8000Hz, and then each sample is encoded by using different standards (e.g. G.711, G.729, G.729a, G.723.1, G.726, G.722, G.728, for audio) which differ in the compression algorithms and the resulting bitstream bandwidth. The encoded bitstream is packetised into IP packets by adding RTP header, UDP header, and IP header.

These packets can then be transmitted across the IP network employing the CPN protocol, where IP-CPN conversion is performed at the source node by encapsulating IP packets into CPN packets which are routed based on their QoS requirements. Due to the shared nature of the IP network, voice traffic may undergo transmission impairments including delay, jitter, packet de-sequencing and packet loss. CPN can alleviate these impairments by smartly selecting the path that provides the best possible QoS required by the user (or an application) [GLMX02] and thus the packets in a voice traffic flow may traverse the CPN network successively along several different paths. At the receiver, packets are queued in a buffer called the re-sequencing buffer. The purpose of the buffer is to reorder packets and buffer them to reduce jitter. Packets that arrive later than the maximum time allowed for the voice signal recovery, or those that provoke buffer overflow, are discarded, contributing to the end-to-end packet loss. To achieve better speech quality, several packet loss concealment techniques are applied before the recovery of the original voice signals.

To provide a satisfactory level of QoS we have developed a ”goal” function which aims at ”Jitter” according to its definition in the real-time protocol RFC3393. We implemented new functions for CPN to support multiple QoS classes for multiple traffic flows, whereby different traffic flows (users) can declare their desired QoS goals before initiating a communication session at the same source node and then each flow is routed based on its own QoS criteria. Then, experiments were conducted to examine which QoS goal is better for RT packet delivery in a multiple traffic environment under varied traffic conditions via measurements of delay, delay variation (jitter) and loss.

Furthermore, we also consider packet end-to-end loss which consists of loss occurring within the network, plus the packet discards happening inside the VOIP receiver which are affected by path switching induced by the adaptive nature of CPN. The correlation of packet end-to-end loss and path switching were carefully studied by an off-line analysis of packet traces from the CPN test-bed network, together with a discrete event simulation of the behaviour of the re-sequencing buffer that resides in the VOIP receiver.

The main results of this chapter are based on the measurements we conduct on the use of CPN to support VOIP. In particular, our work results in interesting insights which are counterintuitive:

- Our measurements compare the use of Delay and Jitter Minimisation as a means to offer QoS to VOIP connections. These measurements clearly show that using Jitter Minimisation as the QoS goal for routing all of the traffic flows in the network, namely the voice traffic and the background (other) traffic, will not only minimise jitter but also delay and loss for voice traffic.

- Furthermore we see that packet loss in the network is clearly correlated with path switching that is induced by the adaptive scheme that is inherent to CPN. However we also see that when path switching does not occur sufficiently often, peaks in packet loss can occur because *all the traffic* including the background non-real-time traffic, will head for paths which are viewed to be good, resulting in unwanted congestion and hence loss, which in turn can only be mitigated by switching to less loaded paths and parts of the network.

- Finally, packet loss that can be provoked by path switching and congestion in a network, also results in further buffer overflows and hence further losses in the output re-sequencing buffers of the VOIP codecs which comes on top of the losses in the earlier stages of the network.

2.2 Real-Time Traffic over CPN for Multiple QoS Classes

As real-time needs are sensitive to the time-based QoS metrics “delay” on the one hand, and “delay variation” on the other, our experiments will consider both of these QoS classes. In fact we will allow the foreground or primary traffic class to have one or the other of these two QoS objectives, and similarly the background traffic will have one or the other of them as well. Thus our experiments on the CPN test-bed were conducted with voice traffic and one of the two QoS requirements between arbitrary source-destination pair within the CPN test-bed network, in the simultaneous presence of several background traffic flows with the same or the other QoS goal. The measurements we conducted were then used to see which of the QoS goals in effect provided better QoS for the voice traffic and under which conditions of background traffic this was actually happening. Thus this section presents the implementation of the goal function for the QoS criterion of ”Jitter”, as well as the implementation of CPN that supports multiple QoS classes for multiple traffic flows simultaneously. We note that in previous experiments reported with CPN, all flows used the same QoS goal so that this change has required some significant modifications to the CPN software that is installed at each node.

2.2.1 Real-Time Packets over CPN with QoS Class Jitter

In RFC3393 and RFC5481 , ”Packet Delay Variation” is used to refer to ”Jitter”. One of the specific formulations of delay variation implemented in the industry is called Instantaneous packet delay variation (IPDV) which refers to the difference in packet delay between successive packets, where the reference is the previous packet in the stream’s sending sequence so that the reference changes for each packet in the stream.

The measurement of IPDV for packets consecutively numbered $i = 1, 2, 3, ...$ is as follows. If S_i denotes the departure time of the ith packet from the source node, and R_i denotes the arrival time of the i-th packet at the destination node, then the one-way delay of the i-th packet $D_i = R_i - S_i$, and

IPDV is

$$IPDV_i = |D_i - D_{i-1}| = |(R_i - S_i) - (R_{i-1} - S_{i-1})| \tag{2.1}$$

To fulfill the QoS goal of minimising jitter, online measurement collects the jitter experienced by each DP. Since in CPN each DP carries the time stamp of its arrival instant at each node along its path, so when a DP say DP_i arrives at the destination, an ACK is generated with the arrival time-stamp provided by the DP, and as ACK_i heads back along the inverse path of the DP, and at each node the forward delay $Delay_i$ is estimated from this node to the destination by taking the difference between the current arrival time at the node and the time at which the DP_i reached the same node [GLMX02], divided by two. This quantity is deposited in the mailbox at the node. The instantaneous packet delay variation is computed as the difference between the value of $Delay_i$ and $Delay_{i-1}$ of the previous packet in the same traffic flow as in (2.1, and jitter is approximated by the smoothed exponential average of IPDV with factor a smoothing factor 0.5:

$$\overline{J_i} = \frac{J_{i-1}}{2} + \frac{J_i}{2} \tag{2.2}$$

Then, the corresponding value of jitter in the node's mailbox is also updated. When a subsequent SP for the QoS class of Jitter and the same destination enters the node, it uses data from the mailbox to compute the reward $Reward_i$ and then trigger the execution of the RNN which corresponds to the QoS class of Jitter and the destination to decide the outgoing link [GLMX02].

$$Reward_i = \frac{1}{\overline{J_i} + \epsilon} \tag{2.3}$$

where ϵ is used to ensure the denominator is non-zero.

2.2.2 Implementation of CPN Supporting multiple QoS Classes

We enable CPN to support multiple QoS classes simultaneously, for multiple flows that originate at any node and each flow is routed based on its specific QoS criteria, we use the following components:

- The Traffic Differentiation can rely on source MAC (or IP), destination MAC (or IP) and the TCP/UDP port of the applications. For instance, the VOIP application "Linphone" has its dedicated SIP port (5060) and audio port (7080), which resides in the fields of the UDP header so that voice traffic can be differentiated.

- The QoS Class Assignment can be defined according to the QoS requirements of different users or applications, and is stored in a configuration file which is loaded into the memory while CPN is being initiated.

- We can then treat each traffic flow according to its QoS requirement using multiple RNNs at each node, where each RNN corresponds to a QoS class and a source-destination pair. For instance, a real-time traffic flow with a specified QoS class (eg. forwarding delay or jitter) traveling across CPN corresponds to an RNN at each node along its selected path. ACKs coming back from the DPs of the real-time flow are used to collect the measurements of the specific QoS metric of the real-time flow itself and deposit the measurement results in the corresponding mailbox at each node on their way back to the source. SPs decide the outgoing link at each node they arrive by selecting the most excited neuron in the RNN based on the measurement result in the mailbox.

2.3 Measurement Methodology for Real-Time Packet Path Switching, Reordering and End-to-End Loss

CPN adaptively selects the path that provides best possible QoS requested for traffic transmission, leading to possible path switches so that traffic may suffer packet de-sequencing and loss. Various techniques for mitigating this effect have been demonstrated, such as the use of a switching probability that avoids simultaneous conflicting path switches among distinct flows that share some common routers, or the use of a minimum "improvement threshold" so that path switches occur only if the expected improvement in QoS is sufficiently large [Gel07]. Accordingly, we are interested in examining the correlation between undesirable effects such as packet de-sequencing and end-to-end loss, and path switching. In the following sections, we described methods to carry out measurements and statistics for the three metrics. The measurements based on the off-line analysis of packet data which were captured by running "TCPDUMP" at the source and destination node so as to obtain the most detailed information of each packet.

2.3.1 Packet Path Switching

For a given flow, the path traversed by packets may change from time to time so as to satisfy the specific QoS requirement. This routing information provided by the corresponding SP is encapsulated into the cognitive map field of the DP while as it originates from the source node. Thus, the path used by each DP can be detected by extracting its routing information field after being captured at the source node. The metric we are interested in is the path switching ratio, which is defined as:

$$Ratio_{path} = \frac{Q_{path}}{N} \tag{2.4}$$

where Q_{path} is the number of path switches in a given flow during the time interval being considered, and N is the total number of packets forwarded in that time interval. We can also define the path switching rate as:

$$Rate_{path} = \frac{Q_{path}}{T} \tag{2.5}$$

where T is the length of the time interval.

2.3.2 Packet Reordering

Packet reordering or re-sequencing is an important metric for real-time because packets have to be forwarded to the end user sequentially at the receiver in the same order that they have been sent, and those that arrive later than the required maximum will have to be discarded. When packets travel over multiple paths, packets sent later may actually arrive at the receiver before their predecessors, so that to obtain a fully correct playback, packets have to be stored until all their predecessors have arrived. However the reordering buffer at the receiver is necessarily of finite length, so that packets arriving to a buffer that is full will be discarded, and packets will have to be forwarded after a given time-out even when their predecessors have not arrived in order to avoid excessive time gaps with their predecessors that have already been played back. Packet reordering is done according to the recommendation from [AP06], which is based on the monotonic ordering of sequence numbers. Specifically, we add a 4-byte field in the CPN header to store the unique sequence identifier which is incremented by 1 each time a new packet is sent into a traffic flow. In addition, the sequence number residing in the Real-Time Protocol (RTP) header can also be used as the identifier, which is strictly monotonically increasing with increments of ”320”. To detect packet reordering, at the receiver we reproduce the sender’s identifier function. The Next Expected Identifier is determined by the most recently received in-order packet plus the increment (denoted by Seq_{inc}) and denoted by $NextExp$. Thus for CPN the increment is ”1”, while for the RTP sequence number its ”320”. If S is the identifier of the currently arrived packet at the receiver, if $S < NextExp$, the packet is reordered and the number of reordered packet $Q_{reordered}$ is incremented by 1, else the packet is in-order and $NextExp$ is updated to $S + Seq_{inc}$. For example, if the packets arrive with the identifiers $1, 2, 5, 3, 4, 6$, $packet$ 3 and $packet$ 4 will be reordered.

To quantify the degree of de-sequencing, we also defined the “Packet Reordering Ratio/Rate” similar to (2.4) and (2.5), and the “Packet Reordering Density” denoted by $Density_r$, so that we may differentiate between isolated and bursty packet reordering as well as to measure the degree of burstiness of packet reordering, which may affect the packet drop rate of the re-sequencing buffer at the receiver. $Density_r$ is calculated as:

$$Density_r + = \begin{cases} Cout_r^2 & for\ bursty\ packet\ reordering \\ Cout_r & for\ isolated\ packet\ reordering. \end{cases} \tag{2.6}$$

where $Cout_r$ is the number of successively reordered packets; it resets to zero when the in-order packets arrive and is incremented when reordering occurs.

2.3.3 Packet End-to-End Loss

The recommendation in [TF03] states that packet loss should be reported "separately on packets lost in a network, and those that have been received but then discarded by the jitter buffer" at the receiver for voice packet delivery, because both have an equal effect on the quality of voice services. The combination of packet loss and packet discard is usually called packet end-to-end loss. In this section, we present the approaches we use to study these two metrics.

Here, packet loss will only refer to the packets lost in the network, as detected for a packet that is sent out but not received by its destination node. Since the sending packets are consecutively numbered by monotonically increasing identifiers at the source while entering CPN 2.3.2, for sent packets falling within the test interval, each packet received at the destination node will be matched with the combination of packet identifier, the source and destination IP address. Non-matched sent packets are identified as the lost packets, so that the timestamp of each lost packet and the number of lost packets are obtained. The packet loss Ratio/Rate is a commonly-used metric to quantify the degree of loss as in (2.4) and (2.5).

At the receiver in the VOIP application at CPN node, the received packets are stored in the "jitter buffer" which reorders packets and buffers them to reduce jitter. Packets that arrive later than the expected time which is required for the voice signal recovery, and those that cause buffer overflow, will be discarded, contributing to the end-to-end loss. To achieve better speech quality, VOIP applications provide several packet loss concealment approaches to compensate packet loss before voice playback. Thus, packet discard cannot be measured inside a VOIP application, and we cannot directly access the run-time version of the VOIP application. Accordingly, we have had to simulate the operation of a jitter buffer which employs resequecing so as to study packet discards and the buffer queue length, and their correlation with packet reordering and packet loss.

The discrete event simulation approach we use simulates the Arrival Events which correspond to the arrival of packets at the receiver, the Departure Events which indicate the departure of packets after being processed by the server, and the Wait Event. Due to the real-time demand of voice traffic, the waiting time of packets that are delayed for re-sequencing in the buffer will have an upper bound. Each of this events is stored in the Future Event List (FEL) and is executed in time sequence. The queue used in the simulation consists of a waiting queue $LQ_{reordered}(t)$ for packets waiting for re-sequencing, and a processing queue $LQ(t)$, for packets waiting to be delivered to the end user. The event scheduling process which is integrated with the Re-sequencing Algorithm is as follows.

To capture a packet trace, we collect the packets in a voice stream received at the CPN destination node during a test interval. The arrival of each packet corresponds to an arrival event scheduled in the FEL with its arrival time. It is assumed that the service time of the server follows an exponential distribution with average value 1ms. We define $WT_{threshold}$ as the maximum wait time for a packet and assign it the value of 200ms which is substantially larger than the typical inter-arrival time of voice packets, which is 20ms. As has been mentioned in Section 2.3.2, $NextExp$, a key variable to identify the next expected packet consistent with the sending sequence for reordering detection, is initialised as the identifier of the first received in-order packet and updated by reproducing the sender's identifier function.

For each arrival event, If $Current\_Event\_ID > NextExp$, the packet should be delayed in the waiting queue (increament $LQ_{reordered}(t)$ by 1) to wait for the expected in-order packets. Moreover, if the early arrived packet waits for more than one packet, the sequence discontinuity size (indicated by d_s) is required to indicate the difference between the identifier of the current arrived packet and the $NextExp$. Therefore, the waiting time is calculated as:

$$tw = d_s * WT_{threshold} \tag{2.7}$$

This produces a wait event which is scheduled to be executed at the time $t + tw$, where t is the arrival time of the current packet. The wait event executes when the waiting time of the packet in the waiting queue expires. The packet is then ordered at the head of the queue and the $NextExp$ is assigned the identifier of this packet and incremented by Seq_inc which is a constant equal to the increment between the identifiers of the two successive packets at the sender. Subsequently, the checking loop starts in the waiting queue. Each remaining packet in the waiting queue $LQ_{reordered}(t)$ will be checked to find the next expected packet. If the next expected packet is found, it is ordered and moved in the processing queue ($LQ = LQ+1$) and $NextExp$ is udated as described above again. In addition, each time the expected packet arrives or is found in the waiting queue, the waiting time of each waiting packet is reduced by the length of the waiting threshold $tw \leftarrow tw - WT_{threshold}$. The checking runs until there none of the next expected packets are in the waiting queue.

If $Current\_Event\_ID == NextExp$, the packet will be buffered in the processing queue ($LQ \leftarrow LQ + 1$) if the server is busy; otherwise, the packet will be processed by the server, which produces the departure event with the occurrence time at $t + ts$ (ts is the simulated service time of the server). $NextExp$ is incremented by Seq_{inc}. Subsequently, the same checking loop executes in the waiting queue as described above to reorder the packets. On the other hand if $Current\_Event\_ID < NextExp$, the packet will be discarded because it arrives too late for voice playback, and we increment $Q_{discarded}$ by 1.

During the simulation, we count the number of discarded packets $Q_{discarded}$ due to excessively late arrivals, as well as the time of the occurrence of the discard. The summation of $LQ(t)$ and $LQ_{reordered}(t)$ accounts for the total queue length in the re-sequencing buffer observed at time t and packet end-to-end loss is measured. To record the difference between isolated and bursty packet loss, we have defined the Packet End-to-End Loss Density and measure it similarly to $PacketReorderingDensity$.

2.4 Experimental Results

Our experiments were carried out on a wired test-bed network consisting of 8 nodes with the topology shown in Figure 2.2, whereby multiple paths are available for packet delivery between source-destination pairs. There are seven available paths between the node CPN002 and the node CPN026, which provides sufficient network complexity so that we are able to study the ability of CPN to adaptively route the packets in favor of the specified QoS goals. Moreover, the small scale test-bed allows us to easily load and saturate the system and evaluate the algorithms in both high, medium and low traffic load conditions. CPN was installed as a loadable kernel module [Gel09] running under linux 2.6.32 at each node. Adjacent nodes are connected with 100Mbps Ethernet links. The purpose of experiments is to examine two issues:

1. Network performance for real-time traffic with respect to the packet delay, jitter and packet loss, under varied background and obstructing traffic loads that use different QoS goals in their routing algorithm.
2. The correlation of packet end-to-end loss and path switching.

Note that we use 20% of SPs in the total traffic which has been shown to be able to achieve the major gain in performance improvement[GLN04]. The voice traffic rate is at 50pps.

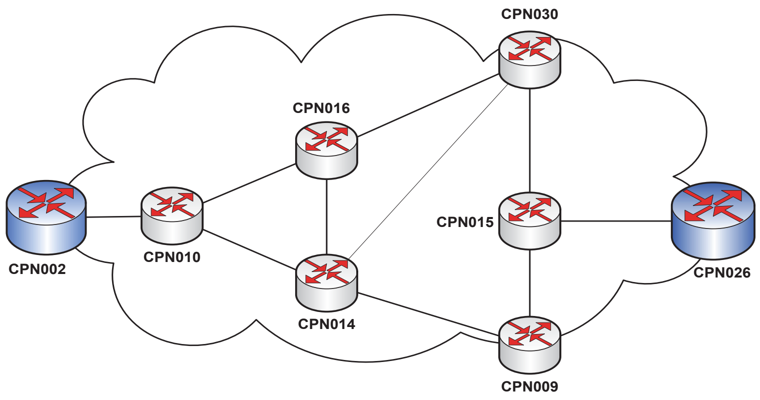


Figure 2.2: CPN network testbed topology used in the experiments.

2.4.1 Performance of real-time Traffic in CPN using multiple QoS Classes

To generate actual real-time traffic, ”Linphone”, a VOIP phone, was installed at each node in the network test-bed, whereby voice traffic can be originated everywhere within the network. Its SIP port is ”5060” and audio port is ”0780”. Thus, voice traffic can be differentiated according to the UDP port number exclusively used by Linphone. We used three flows of voice traffic as shown in Table 2.1.

Source Node	Destination Node	Traffic Rate
cpn002	cpn026	50pps
cpn010	cpn015	50pps
cpn016	cpn009	50pps

Table 2.1: Voice Traffic Distribution used in the experiments.

Due to the constant rate of the voice traffic produced by ”Linphone”, background traffic flows with a range of data rates and constant packet sizes of 1024 bytes were introduced into CPN and allowed to travel between any source-destination pair to provide varied traffic conditions. For the sake of simplicity, we used six UDP traffic flows as the background traffic, which were distributed as shown in the Table 2.2. We repeated each experiment with data rates of 1M, 2M, 3.2M, 6.4M, 10M, 15M, 20M, 25M, and 30M bps for each traffic flow in order to evaluate performance under traffic conditions that vary from light to heavy load, including saturation.

Furthermore, CPN routing was implemented both for the voice and the background traffic, and we set a distinct QoS goal setting for the voice traffic and the background UDP traffic. To this effect, we used three scenarios as shown in Table 2.3. Thus in one case we allowed both the voice and the background traffic to use Delay Minimisation as their QoS goal, in two other cases we used Jitter and Delay Minimisation (and vice-versa) for the two types of traffic, and finally we used Jitter Minimisation for both the voice and the background traffic, and we report measurements for each of these cases.

Source Node	Destination Node
cpn002	cpn026
cpn010	cpn015
cpn016	cpn009
cpn030	cpn010
cpn014	cpn002
cpn015	cpn016

Table 2.2: Background Traffic Distribution used in the experiments.

voice Traffic	Background Traffic
QoS:Delay	QoS:Delay
QoS:Jitter	QoS:Delay
QoS:Jitter	QoS:Jitter

Table 2.3: Distinct QoS goal combinations used in our measurements for the voice and the background UDP traffic flows.

For each test, three flows of voice packets and six flows of background packets with a specified rate were originated using one of the three QoS goal setting scenarios for a duration of ten minutes. The voice traffic flow from CPN002 to CPN026 was selected as the object of our measurement, since it had the greatest number of intermediate nodes between this source and destination pair in the test-bed. The average delay and delay variation, and the packet loss ratio for this flow were measured so as to examine which QoS goal is better for voice packets delivery in the multiple QoS goal environment we considered under widely varying traffic conditions.

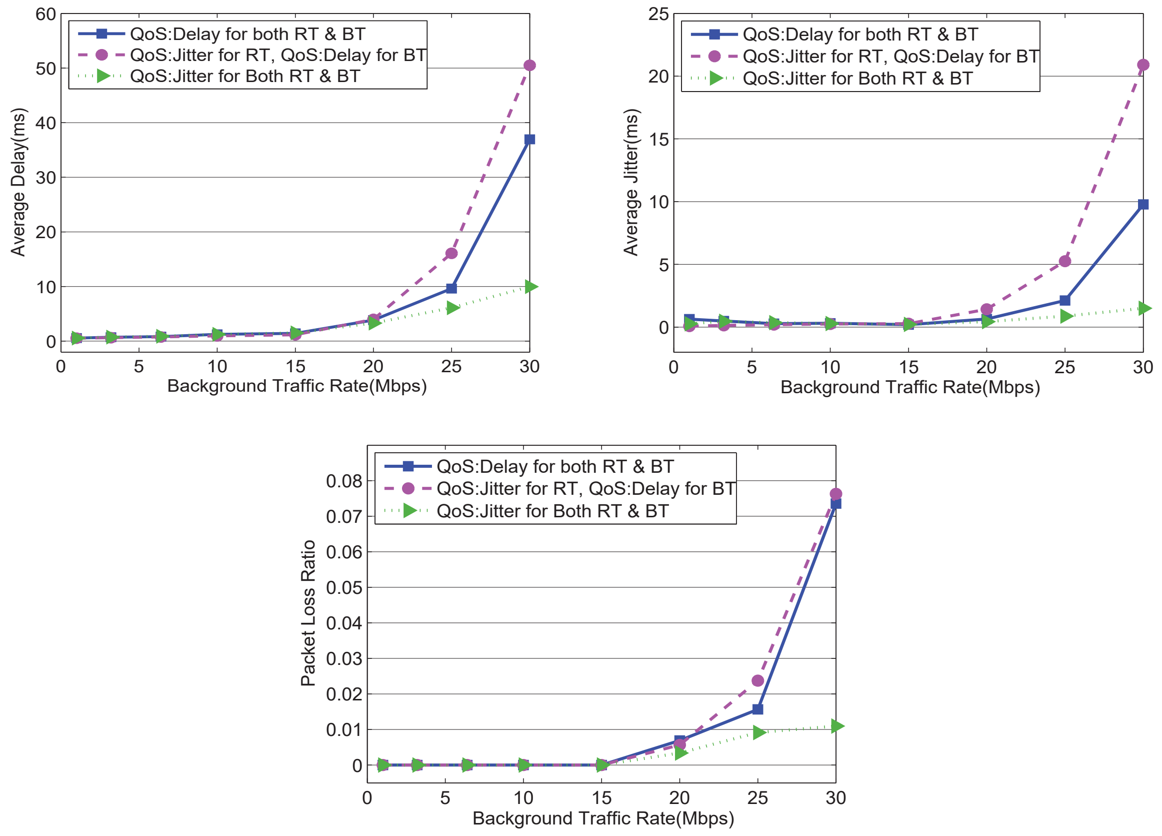


Figure 2.3: The performance for voice Traffic under varied background traffic conditions. We show the average delay (top), the delay variation or jitter (middle), and the packet loss ratio (bottom) for the voice flow from CPN002 to CPN026 for different values of the background traffic load.

From the results shown in the Figure 2.3, we can find the same trend for the three performance metrics in the three QoS goal setting scenarios under varying traffic conditions. With low background traffic load, the two different QoS goals (Delay and Jitter) that are used for either the voice or the background traffic have little effect, as may be expected, since overall delay, jitter and loss are very low.

We see quite clearly, that using Jitter as the QoS goal for *both* the voice itself and the background traffic (BT) provides the *best results* for the voice traffic at medium to high loads. Surprisingly enough, using Jitter for voice and Delay for the background traffic (BT) yields the worst results. While, quite surprisingly, if Delay is used *both* for voice and the background traffic, then the results in all three metrics (delay, jitter and loss) are not as good as when Jitter is used for both, but provides an intermediate result.

To explain this results, we have to refer to the definition of $Jitter$ which means delay variation. Adaptive routing will necessarily cause some path switching, as well as some resulting end-to-end packet loss, and switching causes significant delay variation. Thus the QoS requirement of Jitter Minimisation applied to both voice and the background traffic is likely to result in the least amount of path switching, even though at heavy traffic some route oscillations will still occur and will potentially degrade network performance. Therefore overall, we see that routing based on the QoS goal of $Jitter$ is effective in alleviating route oscillations and losses, although the fact that it seems to reduce end-to-end delay itself (top figure) was definitely an unexpected result of these experiments.

2.5 Correlation between real-time Packet Path Switching, Reordering and End-to-End Loss

While the previous result show that Jitter Minimisation is overall useful in the network to better satisfy the QoS needs of voice traffic, not just when it is applied directly to voice traffic but also when it is used for the background traffic, we were also interested in better understanding the detailed behaviour of the voice traffic during these experiments. Thus we looked more carefully at the data collected in the experiments of Section 2.4.1 and measured the voice traffic flow between CPN002 and CPN026 during the test interval being considered.

The results showed in the below figures are the most representative ones among the results of the repeated experiments, which give the best contract of the variables being studied in this section.

As summarised in Figure 2.3, there is negligible packet loss under low traffic load. As we increase the background traffic rate, Figure 2.4 shows us that as the six background traffic flows reach 20Mbps, looking at the timestamp at which packets are send (the $x - axis$), the path switching rate of voice traffic is around 10 packets per second. Though most of the time hardly any packets are lost, *we were indeed surprised to observe that in several time intervals there was indeed a burst of packet loss*. During three time intervals (800s-900s, 900s-1000s, 1300s-1400s), *packet loss occurred in bursts while the path switching rates became very low*. This is the contrary of what we would have expected. However we feel that this does have an intuitive explanation.

The explanation is that when a given path for voice traffic satisfies the Jitter Minimisation QoS criterion for a relatively long time, and hence the path switching rate is close to ”0”, *this path becomes attractive for background traffic and then becomes saturated*, resulting in bursty packet loss with the loss rate increasing sharply and the loss ratio reaching ”1”. However, shortly after that, CPN reacts as it should to this QoS degradation: the SPs detect the performance degradation and another path is selected. Subsequently, loss rate decreases to ”0” and the path switching rate increases.

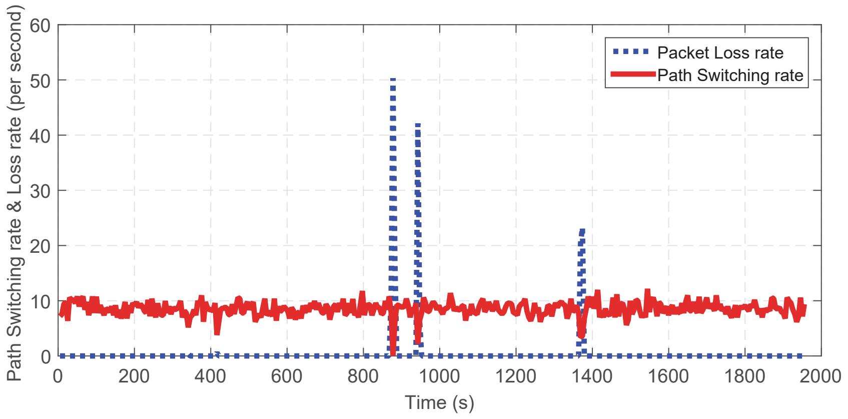


Figure 2.4: The correlation of Packet Loss and Path Switching under medium traffic conditions: an apparently good path attracts more background traffic (BT), resulting in spurious bursty performance degradation and packet loss, followed by switching to better paths thanks to CPN’s ability to adapt.

As the rate of the six background traffic flows increased to 30Mbps respectively, loss occurred more frequently and a large amount of packet de-sequencing was observed at the destination node. Figure 2.5 describes the correlation of Packet End-to-End Loss, Reordering and Path Switching under heavy traffic conditions in a test run of duration of 700 seconds.

We can see that packet reordering rate (caused by de-sequencing) varies in proportion to path switching. This strong linear correlation between the two metrics is confirmed by regression analysis, showing that packet reordering in CPN is mainly due to path switching. Furthermore, it was found that

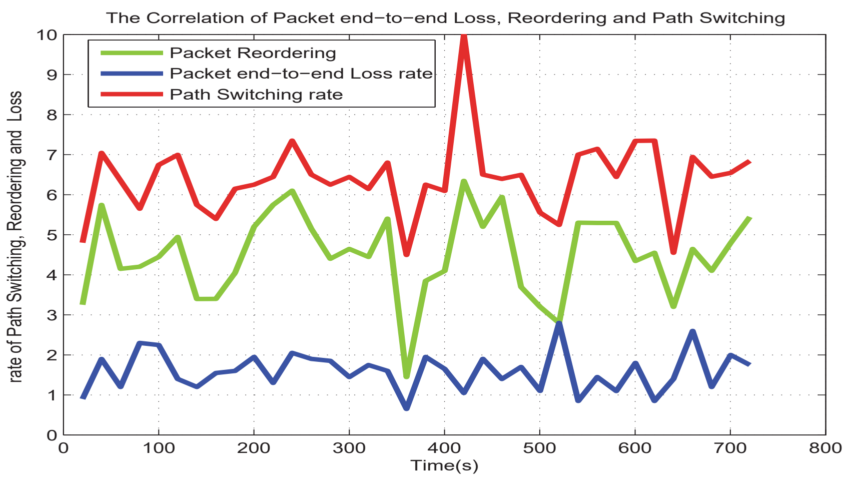


Figure 2.5: The correlation of Packet end-to-en Loss, reordering and Path Switching under heavy traffic conditions

packets were lost more frequently when the loss rates/ratio were not high.

It is not easy to observe the correlation of packet path switching and packet loss from the figure. By applying regression analysis, we also obtained a very weak correlation of the two metrics. It is possibly because under heavy traffic conditions, packet loss is not only due to link saturated, route oscillation induced by heavy traffic loads also leads to the occurrence of loss. We can found that the proper path switching rate (or ratio) is beneficial to loss reduction. If path switching rate is increased excessively, it is converted to route oscillation which also lead to packet loss.

We have observed that packet discards in the re-sequencing buffer contributes to packet end-to-end loss for voice traffic, and heavy traffic load does provoke packet reordering and loss. To illustrate this, we used the voice packets traveling between CPN002 at CPN026 in an experiment where six background traffic flows were simultaneously forwarded at a rate of 30Mbps, under the same conditions as those discussed in the simulation of Section 2.3.3. It was found that the successive occurrence of bursty packet loss and reordering, are together the main reason for the significant increase in queue length at the re-sequencing buffer, causing buffer overflow. Note that packet loss in the network will cause packets to wait for their predecessors (who do not arrive) in the re-sequencing buffer; thus paradoxically buffer length increases when packets are lost. The higher the degree of burstiness of packet reordering and loss, the longer the waiting queue in the re-sequencing buffer and hence the higher the probability of buffer overflow.

Thus, the reordering density and loss density defined in Section 2.3.2 and 2.3.3, together with the queue length translated into the length of time spent in the buffer, are shown in Figure 2.6. We can see that during the measurements queue length was relatively small when the reordering and loss density were low. At the time of around 520 seconds, many packets were lost or reordered successively, and subsequently the queue length in the buffer increased sharply. This data confirms our previous statement that packet reordering provokes large buffer queue lengths and actually accentuates (or creates a positive correlation) to the overall end-to-end packet loss, which results from the initial packet losses plus the buffer overflows.

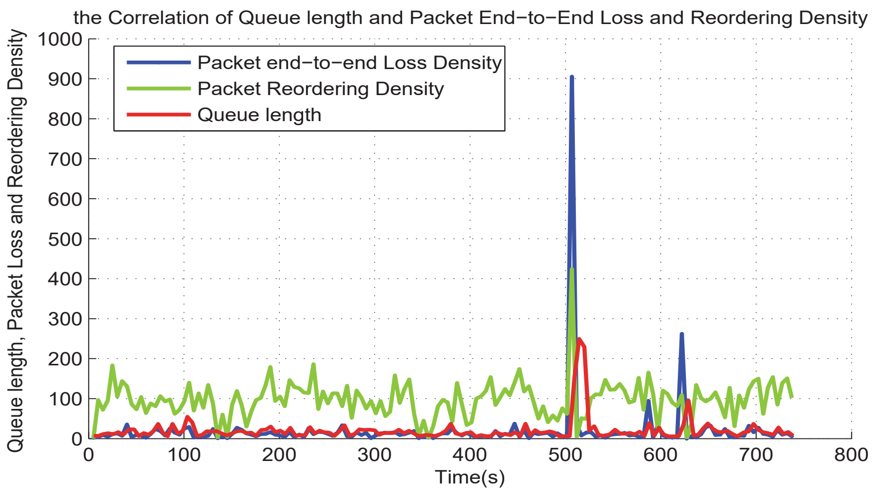


Figure 2.6: The Correlation of the Queue Length in the re-sequencing Buffer, with Packet Loss and Reordering

2.6 Summary

This chapter has presented an extension of the CPN routing algorithm so that multiple QoS classes can inhabit all routers of the network. Then these scheme has been applied to networks which carry voice as well as other background traffic. Detailed experiments on CPN were conducted for voice traffic, as well as other background traffic, with two distinct QoS requirements: $Delay$ and $Jitter$ Minimisation for any source-destination pair within the CPN test-bed.

Surprisingly enough, measurement results for voice traffic regarding average delay, jitter and loss have shown that when ”Jitter” is specified as the QoS goal for both voice and background traffic, better performance is achieved for all QoS metrics and all network load conditions. This seems to result from the useful effect of ”Jitter” as a means to reduce route oscillations, also generally giving rise to less packet loss. However, we notice that long periods of path stability can also result in congestion with all traffic tending to use the same paths, when these paths do not oscillate and hence have low jitter: at that point, bursty effects of loss occur and CPN can mitigate for them by switching paths. Thus novel sensible routing schemes which distribute traffic over many paths in the network [Gel03] may be useful even though they may result in the other disadvantages of multi-path packet forwarding. Furthermore we observe that packet loss in some parts of the network will provoke delays for other packets in the re-sequencing output VOIP codec buffers, which in turn can provoke buffer overflow and further losses. Thus, through a detailed and careful measurement study in a well instrumented network test-bed, this chapter presents a complex series of cause and effects that allow us to better understand and better design networks that need to support real-time traffic. Future work in this area is planned to consider a specific application in distributed industrial control of manufacturing systems, so as to apply this research in a specific practical context.

Chapter 3

A Learning Based Adaptive Multi-Hop Overlay

We present a learning based network overlay whose purpose is to improve the QoS experienced by long-haul intercontinental packet communications. In our system the links between neighbouring overlay nodes use the conventional Internet protocol (IP), while multi-hop links between overlay nodes are dynamically updated based on Reinforcement Learning using Random Neural Networks. Internet-scale experiments with an intercontinental overlay exhibit systematic improvements in end-to-end delay and available throughput with respect to IP.

This chapter is based on the following paper that we have published:[BWG16].

3.1 Introduction

Autonomic communications [DDF+06] were introduced as a means to set-up and adaptively manage large scale networks based on user needs, without direct human intervention. Although the field emerged from active networks [GPS+00], it is making its mark in software overlay networks [MSG+07] and distributed system design [AS15]. In the future, autonomic communications may further increase their hold through the flexibility offered by Software Defined Networks (SDN) [KC14].

Well known network measurements have shown that IP (Internet Protocol) routing often results in paths that are sub-optimal with respect to a number of metrics [SCH$^+$99, Pax96]. Besides, measurements have also established that the routing scalability of the Internet comes at the expense of reduced fault-tolerance of end-to-end communications between Internet hosts [LMJ98, LABJ00, DCGN01, HJ04, Pax96]. Current routing protocols may work reasonably well when only "best effort" delivery is required, but the requirements for modern distributed services are typically far more stringent, demanding greater performance and availability of end-to-end routes than these protocols can deliver. The ideal solution would be a complete rethink of the Internet routing infrastructure, doing away with the existing architecture and redesigning it with the benefit of hind-sight about its deficiencies. Unfortunately, the Internet has become resistant to major changes, preventing even necessary changes to take place.

On the other hand, routing overlays have been proposed as a method for improving performance, without the need to re-engineer the underlying network [PST04, TWEF03, FBR$^+$04, BMP03]. The basic idea is to move some of the control over routing into the hands of end-systems. As illustrated in Figure 3.1, a routing overlay is formed by software routers, which are deployed in different spots over the Internet. The overlay nodes monitor the quality of the Internet routes between themselves and cooperate with each other to share data. By adding intermediate routing hops into the path taken by streams of packets, they influence the overall path taken by the packets, without modifying the underlying IP mechanism for computing routes. In a routing overlay, the endpoints of the information exchange are unchanged from what they would have been in the absence of the overlay, but the route through the network that the packets traverse may be quite different.

Routing overlays can be used to quickly recover from path outages, and also improve the QoS of data flows. Indeed, the overlay nodes constantly monitor the IP routes between themselves so that failed parts of the Internet can be avoided when a node detects that the primary Internet path is subject to anomalies. Similarly, this approach makes it possible to override the routes determined by Internet protocols and to route traffic based on metrics directly related to the performance needs of the application.

The Resilient Overlay Network (RON) [ABKM01] was the first routing overlay to be implemented in

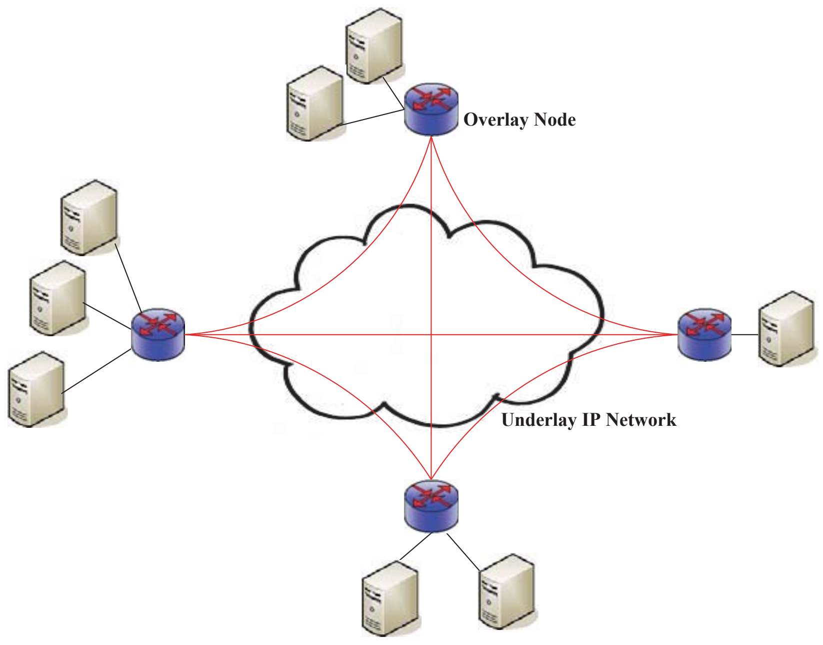


Figure 3.1: Schematic description of the structure of a routing overlay, where the Overlay Nodes exchange packets with each other with packets that tunnel through the IP connections, while paths between Overlay Nodes may transit through intermediate Overlay Nodes.

a wide-area network, demonstrating that adding an extra (overlay) routing hop could benefit an application in terms of improved delay and reachability. To find and use alternate paths, RON monitors the health of the underlying Internet paths between overlay nodes, dynamically selecting paths that avoid faulty areas. The main drawback of RON is that it does not scale very well: as the number of participating routers N increases, the $O(N^2)$ probing overhead becomes a limiting factor, because RON uses all-pairs probing. The downside is that a reasonable RON overlay can support only about 50 routers before the probing overhead becomes overwhelming. However RON has inspired many other approaches [GMG$^+$04, HL04, NPB06], but no existing work has tackled the problem of building an overlay that can be widely and efficiently deployed over a sizable population of routers.

3.1.1 Technical Approach

In this chapter we detail the design of SMART (Self-MAnaging Routing overlay) that is a self-healing, self-optimizing and highly scalable routing overlay that accepts customized routing policies formulated by distributed applications according to their own needs. The overlay network is formed by software routers deployed over the Internet, and it operates by monitoring the quality of Internet paths (latency, bandwidth, loss rate) between overlay nodes and re-routing packets along an alternate path when the primary path becomes unavailable or suffers from congestion. In particular, we investigate the use of the *Cognitive Packet Network* (CPN) [Gel09, Sak10] to design and evaluate the *scalable routing overlay*.

In previous work, CPN has been shown to be effective for a variety of uses, including QoS optimisation, network security enhancement and energy savings [GM11a], and adaptive routing of evacuees in emergency situations [GW12]. The overlay we design is based on a set of proxies installed at different Cloud servers, or they may be in other servers across the network, so that the overlay itself operates over these proxies from some source to destination in source routed manner, while the flow of packets between proxies travels in conventional IP mode. The routing between proxies provides QoS-driven source routing, and performs self-improvement in a distributed manner by learning from QoS measurements gathered on-line, and discovering new routes [Gel09].

This data driven intercontinental packet routing scheme constantly, say every two minutes, collects round-trip delay data at the overlay nodes; it then makes scalable distributed decisions using a machine learning approach from this massive amount of data. In view of the N overlay nodes used in our experiments, every two minutes the system may collect up to N^2 data points. Thus over 24 hours with 20 overlay nodes, each checking connectivity and round-trip-delays (RTT) with 19 other nodes, the network can collect up to some 2.7×10^5 data points per day. However, our work shows that most of the benefit of the technique is achieved when only a small number of alternate paths (eg. two paths) are tested, so that there can be considerable reduction in complexity of data processing and decision making. Furthermore, it is also possible to use the full CPN scheme for the overlay, which means that only the best one-overlay-hop connections are probed, as when CPN seeks the paths with the best QoS across a large network.

The routing decisions are made on-line at each proxy of the overlay network based on adaptive learning techniques using the random neural network (RNN) [GF99]. Each overlay node uses a RNN which is trained using Reinforcement Learning with the data collected at the node itself, while intermediate IP routers proceed using standard Internet routing.

3.2 SMART self-healing and self-optimizing routing overlay

SMART is a self-healing, self-optimizing and highly scalable routing overlay that accepts customized routing policies formulated by distributed applications according to their own needs. As shown in Figure 3.2, the SMART overlay network is formed by software agents that are deployed at Virtual Machines (VM) in different sites.

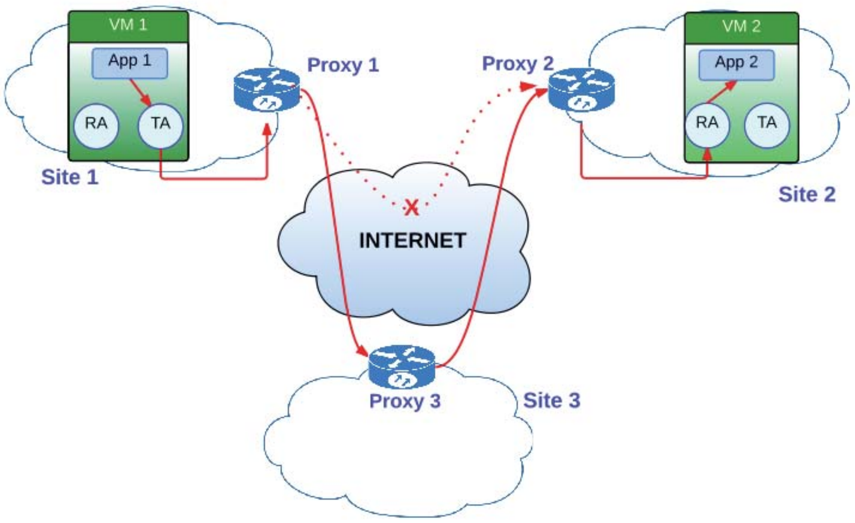


Figure 3.2: SMART's Overlay Nodes exchange packet streams via the Internet using the Internet Protocol (IP) either directly, or via intermediate Overlay Nodes.

In each site, there is a single software router, which is called Proxy. The proxy is in fact constituted of three software agents. The **Monitoring Agent** monitors the quality of the Internet paths between the local site and the other sites in terms of latency, bandwidth, and loss rate. It can be queried by the routing agent in order to discover the quality of a path according to a given metric. The **Routing Agent** drives the monitoring agent and uses the data it collects to discover an optimal path (e.g., low-latency, high-throughput, etc.) with minimum monitoring effort, and writes the optimal path for a given destination into the routing table of the forwarding agent. The **Forwarding Agent** forwards each incoming packet to its destination on the path it was instructed to use by the Routing Agent. We use source routing, that is, the routing table of the source proxy describes the complete path *between overlay nodes proxies* to be followed by a packet to reach its destination, while the path between proxies is determined by the conventional IP protocol.

On each VM, a **Transmission Agent** (TA) and a **Reception Agent** (RA) run together with various Applications or tasks. These agents represent the ingress and egress points of data traffic in the overlay network, respectively. Indeed, as illustrated in Figure 3.3, when a packet is sent by a source task to a destination task located in a different site, it is first intercepted using a filtering mechanism of the Linux kernel known as NetFilter NFQUEUE, and then made available to the TA. The TA uses IP-in-IP encapsulation to forward an altered packet to the Proxy. The payload of the altered packet is the original packet, plus an additional SMART header. Upon reception of the SMART packet, the routing agent of the Proxy looks-up its routing table in order to determine the path to the destination. The sequence of intermediate proxies is written in the SMART header, and then the SMART packet is forwarded to the first one of these proxies. Each intermediate proxy then forwards the packet to the next hop on the path, until the final proxy is reached. When this happens, the packet is forwarded to the RA of the destination VM. The RA de-encapsulates the SMART packet and forwards the original IP packet to the destination task using a raw socket.

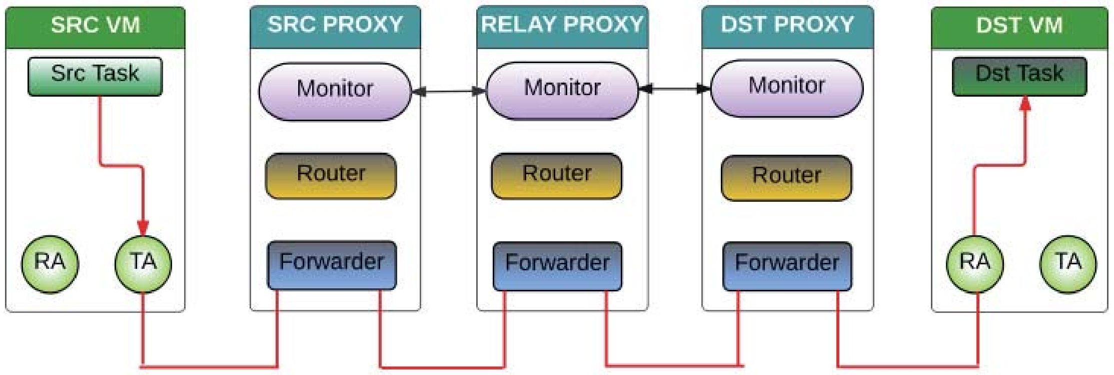


Figure 3.3: Details of the SMART packet forwarding process.

The optimal path to a destination site is sought using active monitoring. For the latency for instance, the monitoring agent of a proxy regularly measures the delay to the destination site by sending probe packets along paths to that destination. Each probe packet is time-stamped by each intermediate node on its way forward and on its way back, so that the source proxy can easily deduce the latency of each segment of the path. Note that for the first experiment described in Section 3.5, the size of the overlay network we are using, and the measurements at 2 minute intervals imply that 2.7×10^5 data points are being collected per day.

3.3 Learning with Random Neural Networks

Since we build a routing overlay that can be widely deployed over a sizable population of routers, instead of requiring an optimal path to be found at each measurement epoch, we look for an on-line decision algorithm that uses a limited monitoring effort but achieves close to the same average end-to-end performance as the best path. The idea is to exploit past observations and quickly learn path performance, to efficiently select the optimal path. At each measurement epoch, the algorithm chooses a subset of paths to probe, and measures their QoS,e.g., the round-trip delay, or the inverse of the available throughput. The algorithm then sends its packet over the minimum cost path among those it has probed. Costs may change from one time slot to the next one, and those paths are probed whose QoS is not significantly worse than that of the best route from source to destination: probing does not cover *all* possible paths but those paths which have been observed in recent probing steps to provide low overall forwarding delay or high throughput. CPN routing differs from this approach in that [Gel09] uses Reinforcement Learning [Hal00] with Random Neural Network (RNN) based critics to select the *next hops* that require further probing at each node [GK14].

To describe the algorithm implemented by the source Proxy for learning an optimal route to the destination, Proxy is implemented by the Routing Agent of the Proxy and is based on a RNN. As input, we are given a set $\mathcal{P}$ of N possible paths to the destination Proxy. This set always includes the direct IP route. Each neuron of the RNN is associated to one of these paths. At regular time intervals, the Routing Agent uses the RNN to select K paths to the destination, monitors the quality of these paths, and then chooses the path with the best performance as described in Algorithm 1.

Let $\mathcal{P}(t) \subset \mathcal{P}$ be the set of paths selected at time t by the algorithm. Each of them is represented by one of the neurons with the highest probabilities q_i at time t (line 1). The algorithm monitors the quality $G_j(t_i)$ of these paths at instants t_l for successive integers l, from which it deduces the reward $R_j(t_l)$ for each path $j \in \mathcal{P}(t_l)$ (line 2). On the other hand, the CPN algorithm was designed for large networks, and therefore only chooses the next hop (rather than the full path) using “smart packets” (SPs) that seek out paths in a hop-by-hop basis [Gel09] to a given destination. Since we deal with relatively small overlay networks with a small number of alternate paths for a given source to destination pair, each of the monitored full paths is considered. $R_j(t_l)$ is used to adjust the values of

Algorithm 1 Learning optimal paths with a RNN and Reinforcement Learning.

```
for $t = 1,\ 2,\ ..$ do
    $\mathcal{P}(t_l)$ is the set of $K$ neurons with highest probabilities $q_i$ at time $t_l$.
    $R_j(t_l) \leftarrow$ reward obtained with path $j$.
    $p^* \leftarrow \arg\max_{j \in \mathcal{P}(t_l)} R_j(t_l)$
    for $j \in \mathcal{P}(t_l)$ do
        $\nu_j \leftarrow R_j(t_l)/T(t_l)$.
        if $R_j(t_l) \geq T(t_l)$ then
            for $i = 1,\ ...,\ N$ do
                $\Delta_i \leftarrow (\nu_j - 1)\ w^+_{i,j}$.
                $w^+_{i,j} \leftarrow w^+_{i,j} + \Delta_i$.
                $w^-_{i,k} \leftarrow w^-_{i,k} + \frac{\Delta_i}{N-2}, \forall k \neq j$.
            end for
        else
            for $i = 1,\ ...\ , N$ do
                $\Delta_i \leftarrow (1 - \nu_j)\ w^-_{i,j}$.
                $w^-_{i,j} \leftarrow w^-_{i,j} + \Delta_i$.
                $w^+_{i,k} \leftarrow w^+_{i,k} + \frac{\Delta_i}{N-2}, \forall k \neq j$.
            end for
        end if
        for $i = 1,\ ..\ , N$ do
            $r^*_i = \sum_{k=1}^{n} w^+_{i,k} + w^-_{i,k}$.
            $w^+_{i,j} \leftarrow w^+_{i,j}\ \frac{r_i}{r^*_i}$.
            $w^-_{i,j} \leftarrow w^-_{i,j}\ \frac{r_i}{r^*_i}$.
        end for
        Solve the non-linear system
        $T(t_{l+1}) \leftarrow \beta\, T(t_l) + (1 - \beta)\, R_j(t_l)$.
    end for
end for
```

the two matrices W^+ and W^- based on Reinforcement Learning with the decision threshold T_{t_l}:

$$T(t_l) = \beta\, T(t_{i-l}) + (1-\beta)\, R_j(t_l),$$

where $\beta \in (0,1)$ is a number that is used to introduce forgetfulness: a larger β will give more importance to recent events. When round-trip delay is the goal function, $T(t_i)$ is the "exponential average" up to time t_i of the round-trip delay for a packet in that flow. To start the learning, the algorithm first determines whether the most recent value of the reward $R(t_l)$ is larger than the threshold $T(t_l)$. If that is the case, then it increases significantly the excitatory weights going into neuron j (line 10), that was the previous winner, rewarding it for its previous success, and increases the inhibitory weights leading to other neurons, but in a much smaller proportion (line 11). Otherwise, if the new reward is smaller than the previously observed threshold, it increases significantly the inhibitory weights leading to the previous winning neuron (line 16), punishing it for not being successful last time, and increases in a much smaller proportion all excitatory weights leading to other neurons (line 17) to give other decisions a greater chance of being selected. Then it normalizes them in lines 20 to 23 of the algorithm:

$$r_i^* = \sum_{k=1}^{n} w_{i,k}^+ + w_{i,k}^-,$$
$$w_{i,j}^+ \leftarrow w_{i,j}^+ \frac{r_i}{r_i^*} \quad w_{i,j}^- \leftarrow w_{i,j}^- \frac{r_i}{r_i^*}.$$

It then computes the stationary probability q_i that neuron i is excited (line 25) by solving the system of nonlinear simultaneous equations (labeleq:num), and then updates the decision threshold $T(t_l)$ (line 26).

3.4 The Analysis of the Overhead of CPN

We consider a network that has N nodes, and K links per node. Each link carries an average of T packets/second user traffic. These links are bilateral so that a node receives $2KT$ packets/sec. Assume that a node has an average of A connections, each generating an average of r packets/second.

Let $logN$ denote the logarithm to the base K of N, so that assuming that the IP protocol is used the paths in this network are at most of length $logN$.

If the network operates with the IP protocol. The total traffic per link is then:

$T = \frac{rA.logN}{2NK}$,

and the total traffic per node is:

$t = \frac{rA.logN}{N}$

We then consider the network operates with CPN. Each connection also generates $f.r$ $(0 < f < 1)$ smart packets (SP). In addition, we assume that each node of the network also generates smart packets at a smaller rate $v.r$ $(v < f)$ for additional monitoring.

Now if we also assume that the path lengths may be on average longer, let us call this quantity $L = logN + d$, we see that the total traffic per link is then:

$T^{*} = \frac{(r+2fr+2vr)A(logN+d)}{2NK}$,

and the total traffic per node is:

$t^{*} = \frac{(r+2fr+2vr)A(logN+d)}{N}$

So that the link or node traffic overhead (TO), related to packets per second, of using CPN becomes

$TO = \frac{T^{*}}{T} = \frac{t^{*}}{t} = [1 + 2(f + v)][1 + \frac{d}{logN}]$

However, because in CPN the SPs are shorter than payload packets, especially when the latter are full Ethernet packets, we can write the overhead regarding the bit rates as:

$TO = \frac{T^{*}}{T} = \frac{t*}{t} = [1 + 2F(f + v)][1 + \frac{d}{logN}]$

when a SP (or ACK) packet contains, on average, a fraction $0 < F < 1$ of bits, as compared to a payload packet.

The node computational cost (C^{CPN}) on the other hand is obtained as follows. Let C be the amount of computation for routing decisions for payload packets, when CPN is not used. If CPN is used, payload packets and ACK packets are source routed with a computational cost of $C^* < C$. However SPs have require a computation cost c for routing. Thus the node computational overhead becomes:

$$C^{CPN} = (rC + frc + frC^* + vrc + vrC^*)A.(logN + d) \tag{3.1}$$

In practice, we have noticed that d is around 1 or 2, while $logN$ is typically as large as 7 or 8[Gel].

3.5 Experimental Results

We now describe the results that we obtained with the proposed algorithm during an Internet-scale experiment,, where we used 20 *Overlay Nodes* of the $NLNOG\ RING$ (see Figure 3.4) which are composed of the servers located across the world. We measured the latency and loss rates between all pairs of ONs every two minutes, communicating through the Internet, for a period of one week using the ICMP-based ping utility. When five consecutive packets were lost between a specific pair of nodes, we considered that the particular source was disconnected from that destination. The path latency was measured as the round-trip time (RTD), i.e. *the length of time it takes for a packet to be sent to its destination, plus the length of time it takes for the corresponding acknowledgment (ACK) packet to be received at the source*.

The main observations from this Internet-scale experiment based on some 2.7×10^5 measurements per day, carried out over a whole week (i.e. a total of roughly 1.7×10^6 measurements), were the following:

- There was a path outage across the Internet at least once in the week for 65% of origin-destination pairs, and 21% of these path outages lasted more than 4 minutes. In fact, 11% of the outages lasted more than 14 minutes.
- We observed that the RTD of purely IP routes (without overlays) exhibits strong and unpredictable variations: see Figure 3.6, and these variations can be as large as 500%.

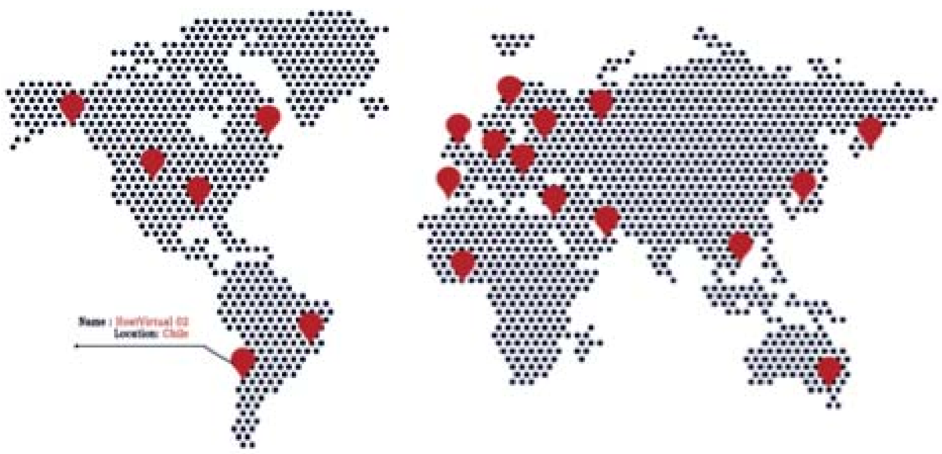


Figure 3.4: Geographical location of the 20 Source and Destination Nodes used in our experiments within the NLNog ring. These same nodes are also used for our direct IP based routing measurements, and also to evaluate the SMART Overlay Node based routing scheme.

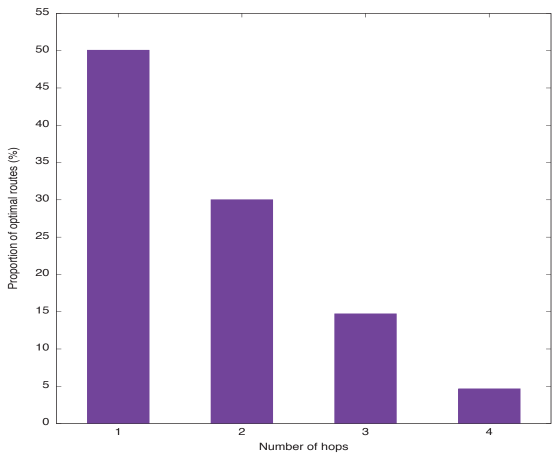


Figure 3.5: Percentage of instances when the overlay path that minimises RTD, which uses IP paths between Overlay Nodes, is observed to include 1, 2, 3 or 4 Overlay Hops. We see that at most two overlay hops cover most of the optimal cases.

- Throughout our experiments, the IP route was *very clearly* not the minimum latency path in 50% of the cases, as shown in Figure 3.5.

- There was always at least one origin to destination pair whose latency could be reduced by more than 76% by selecting an alternate path to the IP route, by using one or more overlay nodes.

- Surprisingly enough, as shown in Figure 3.5, for 30% of the cases the optimal path had only 2 Overlay-Hops. This shows that a limited deviation from IP can actually produce much better QoS than IP itself.

- Then, a natural question is whether it is enough to consider routes of at most two Overlay-Hops. Interestingly, in 20% of the cases the optimal path was a 3 or 4 Overlay-Hop path.

- However, on average, the relative difference in measured delay between the optimum path and the best 2 Overlay-Hop path was only 3.38%. However, as shown in Figure 3.7, for some origin to destination pairs, there was a significant benefit in using a path which had no more than two overlay hops.

- Similarly, more than 11% of the IP routes were observed to have a loss rate greater than 1%. An in particular, by selecting paths via the overlay that *had a different path than those proposed by IP*, it was possible to have no packet loss at all.

The measurements also show that path outages are routine events in the Internet. Furthermore, the paths selected by IP routing are strongly suboptimal.

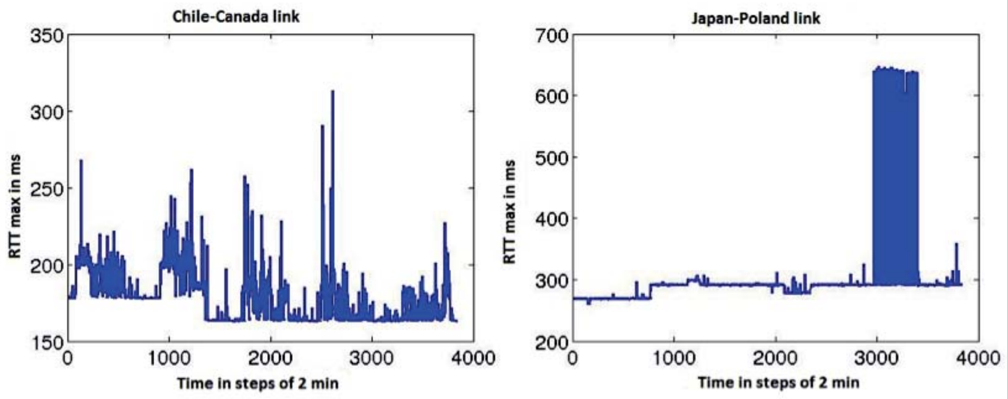


Figure 3.6: The RTD in milliseconds using the IP protocol, given in averages over successive 2 minute intervals over an observation period; each unit along the x-axis corresponds to successive 2 minute intervals over the long time period being shown. The connections measured are between Chile-Canada (left) and Japan-Poland (right), and we observe the large variation in average RTD depending on the time period being considered.

As we will now see, the CPN or RNN based algorithm that exploits ONs allows a significant decrease in round-trip delay, with a very modest monitoring and computational effort. In the experiment described below, we simplify the routing scheme so that we only consider the direct IP route and 2 Overlay-Hop routes. Therefore the number of possible overlay paths between an Overlay Source and

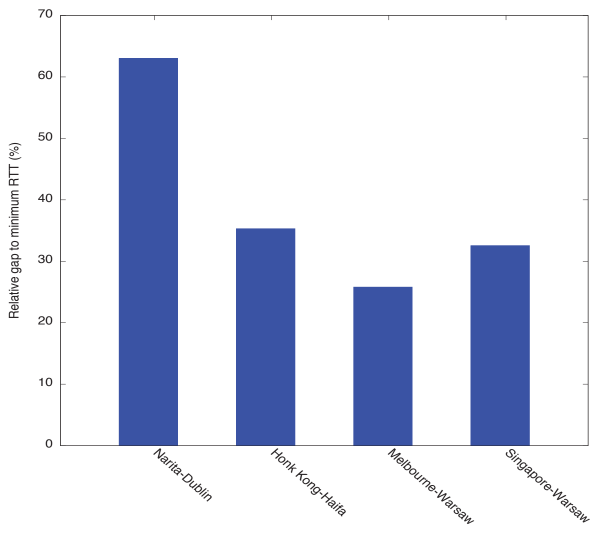


Figure 3.7: Percentage of average RTD, for the best 2-hop path relative to the minimum RTD path, for certain origin to destination pairs, where RTD is averaged over the successive 2 minute measurement intervals.

	Direct	2-hop Overlays
Percentage of non-optimal instances	50.08%	16.20%
Average % difference above min. latency	11.1%	4.24%

Table 3.1: Non-Optimal RTDs for all the measurements: IP vs. SMART routing with 2-hop overlays. 16% of 2-hop overlays and 50% of IP paths are non-optimal. RTDs for IP can substantially exceed the minima and averages over all measurements.

Overlay Destination is $N = 19$. Furthermore for a given connection, the algorithm chooses two alternate overlay paths to monitor in addition to the direct IP route, and selects the one with the minimum latency. Thus from a given source node, the algorithm will measure at most 4 links per measurement round. We see that SMART uses the minimum-latency path in 84% of the cases, whereas the IP route is optimal only in 50% of the cases. The average percentage difference above minimum latency is only 4.24% with SMART, whereas it is 11.1% for native IP routing. The resulting observed average delays are summarized in Table 3.1.

However, these average values do not truly measure the gains obtained in the pathological routing situations we seek to improve. Thus, in Table 3.2 we present some examples for which the system provides significant gains, despite the partial coverage of the overlay topology. The column OPT gives the minimum latency that can be achieved with two-hop routing. On the other hand, Figure 3.8 shows the RTD between the nodes in Japan and Chile over 5 successive days. The RTD of the direct IP route is about 400 ms, whereas the RTD of the minimum latency path is about 250 ms. As can

	IP route	SMART	OPT
Melbourne/Gibraltar	390	274.6	273.5
Narita/Santiago	406.7	253.8	253.0
Moscow/Dublin	179.9	81.7	80.8
Honk Kong/Calgary	267.1	130.7	130.0
Singapore/Paris	322.3	154.1	153.2
Tokyo/Haifa	322.6	180.8	180.1

Table 3.2: Average RTD (ms) for some pathological OD pairs.

be seen, the RNN-based algorithm learns very quickly which is the minimum latency path and tracks this path until the end of the 5 days. Figure 3.9 shows that it takes only 12 minutes for our algorithm to learn the optimal route.

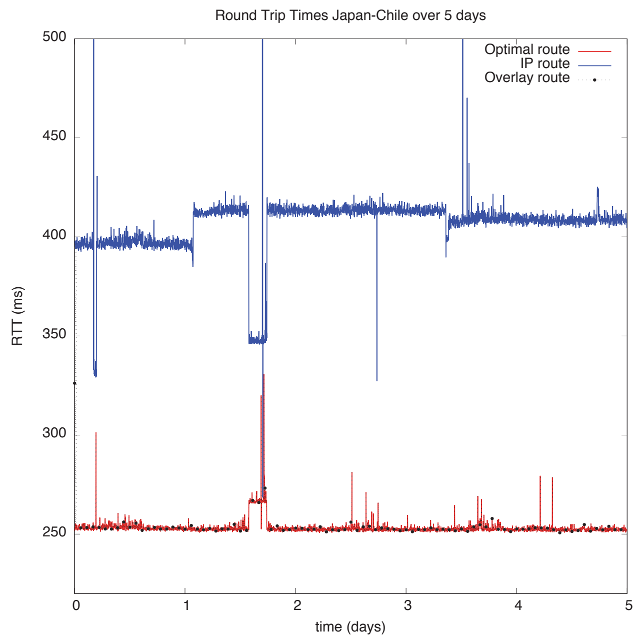


Figure 3.8: RTD in milllliseconds measured for the Narita (Japan)/Santiago (Chile) connection in an experiment lasting 5 successive days. We see that measured RTD for the SMART routing policy (in black dots) follows the values of the optimal (i.e. minimum) RTD very closely.

Figures 3.10 and 3.11 provide a similar comparison for the RTD between Norway and Singapore. The RTD of the direct IP route is about 340 ms, whereas the RTD of the minimum latency path is about 270 ms. Here again, the RNN-based algorithm learns the minimum latency path very quickly. However, it does not track the minimum latency path as well as in the previous case, and we can notice some discrepancies between the minimum RTD and the RTD of the overlay during the first day, between hours 3 and 5. Figure 3.11 shows that perturbations can last for a some tens of minutes.

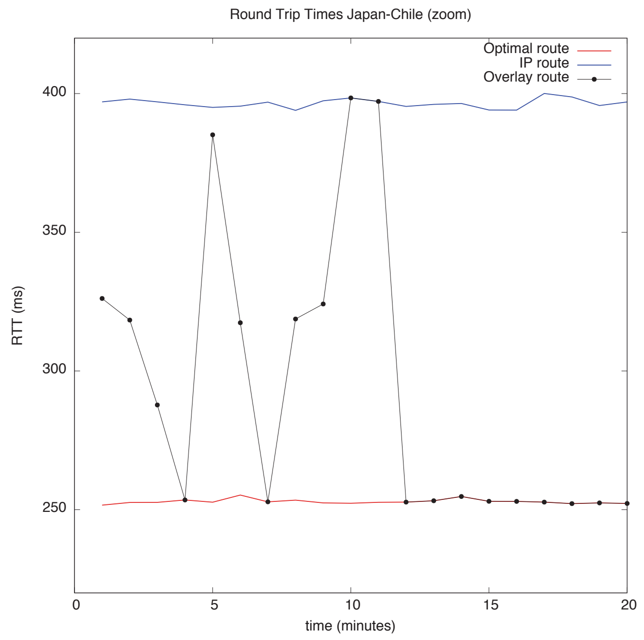


Figure 3.9: RTD in milliseconds for the Narita (Japan)/Santiago (Chile) connection for 20 successive 2-minute measurements, over the first 20 minutes of the experiment reported in Figure 3.8. We see that the SMART policy is learning right at the beginning, oscillating between the RTD measurements for IP (in blue) and the optimum in red. However, after a relatively short time interval, SMART settles to closely following the optimum.

Nevertheless, the overlay always provides better performance than what is offered by the IP routing protocols.

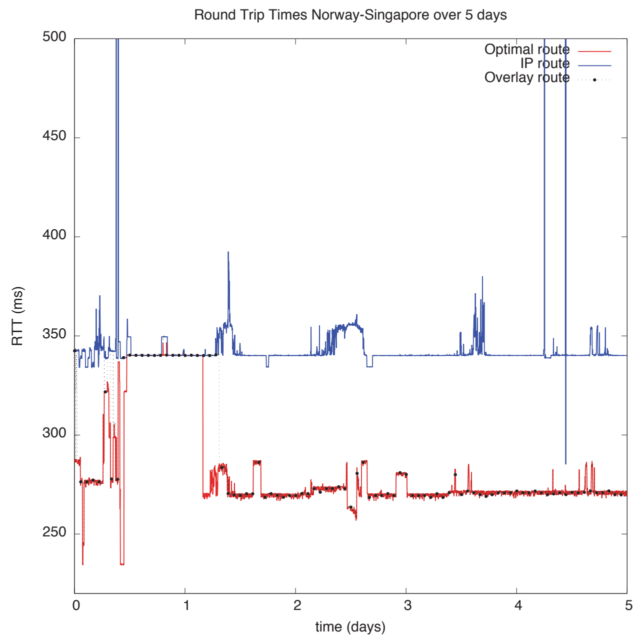


Figure 3.10: Measured RTD in milliseconds for the Norway-Singapore connection over 5 successive days. Again we observe that the RTD for the SMART scheme (dotted line) closely tracks the measurements for the optimal paths (red), even when they provide the same results as IP routing (blue).

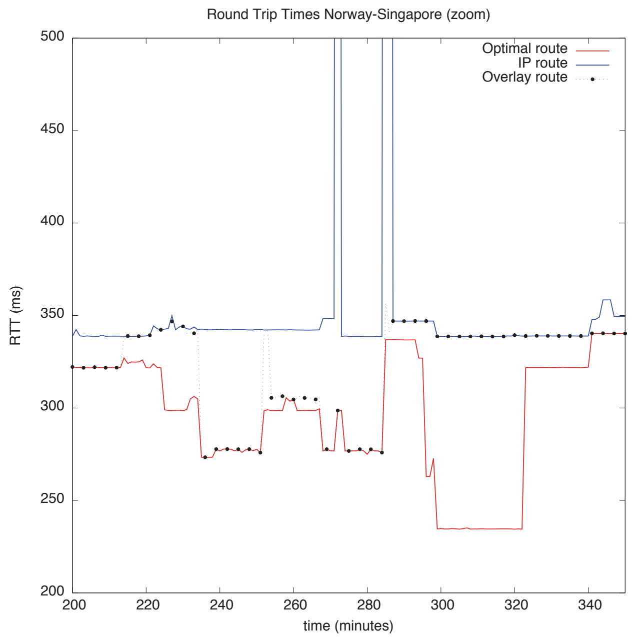


Figure 3.11: RTD in milliseconds for the Norway-Singapore connection between hours 3 and 5 of the experiment of Figure 3.10. We notice that in this early stage of the experiment, SMART is unable to achieve the optimum (red) RTD, while from the previous Figure 3.10, we know that as we enter the later stages of the experiment the optimum is definitely attained thanks to the use of the learning offered by the RNN based Reinforcement Learning algorithm.

We now describe the results obtained in an experiment involving 9 AWS (Amazon Web Services) data centres located as shown in Figure 3.12. We measured the available throughput between all pairs of data centres every five minutes, communicating through the Internet, for a period of four days. We thus collected some 8.3×10^4 measurement data over the 4 days period. Assuming that the available throughput over a path is the minimum of the throughputs of its constituent links, the analysis of these data revealed that the IP route is the maximum throughput route only in 23% of the cases, and that most of the time, the maximum throughput overlay route passes through 1 or 2 intermediate nodes (in 32% and 27% of the cases, respectively). We only consider the 8 overlay paths of at most two hops and assume that at each measurement epoch the algorithm probes only 3 paths (including the direct IP route). Thus, the algorithm will measure 5 links per measurement round. As for the RTD, we observe a clear improvement over native IP routing: the average relative gap between the throughput of SMART (resp. IP routing) and the maximum throughput that can be achieved with two hop routing is 2.7% (resp. 31.3%), and SMART (resp. IP routing) uses the maximum throughput two-hop path in 84.1% (resp. 26.2%) of the cases. Here again, we present in Table 3.3 the spectacular results obtained for some pathological OD pairs, for which the available throughput is at least doubled.

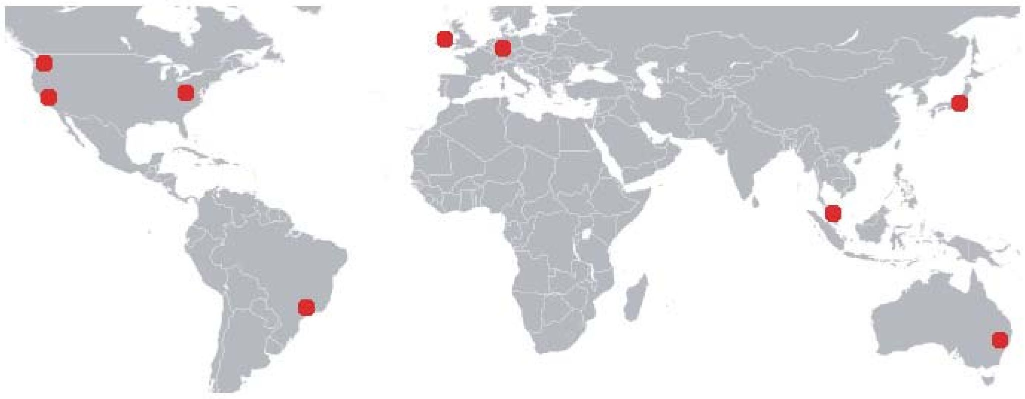


Figure 3.12: Geographical location of the 9 AWS data centres.

	IP route	SMART	OPT 2-hops
Dublin/Sydney	11.5	37.5	40.5
Singapore/Sao Paulo	12.8	42.0	43.6
Sydney/Virginia	8.5	52.3	55.3
Virginia/Singapore	7.4	33.8	36.1
Virginia/Sydney	6.9	35.0	36.7
Virginia/Tokyo	10.3	39.7	43.4

Table 3.3: Average throughputs (Mbps) for some pathological OD pairs.

Figure 3.13 shows the measured throughput between Virginia (USA) and Sydney (Australia) over the 4 successive days. The average throughput of the direct IP route is 8.5 Mbps, whereas the average throughput of the optimal path is 55.3 Mbps. Figure 3.14 shows the same results over the first 3 hours and we notice that it takes only 30 minutes for SMART to discover an optimal route.

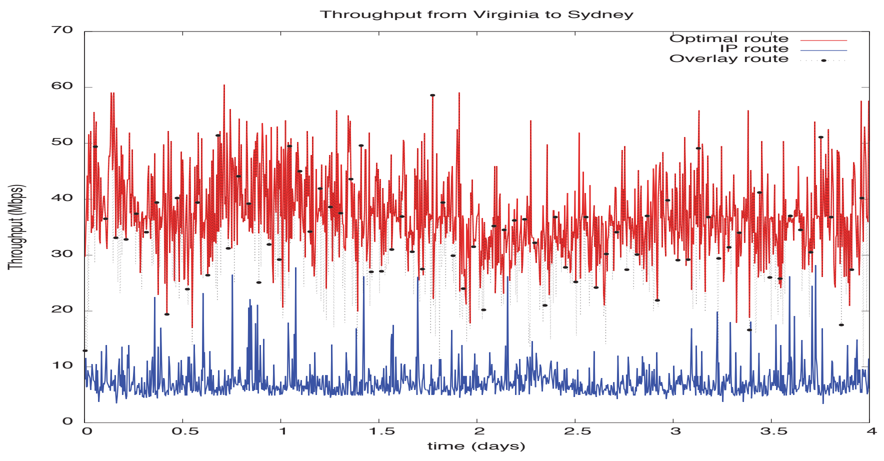


Figure 3.13: Throughput (Mbps) measured from Virginia (USA) to Sydney (Australia) over 4 consecutive days.

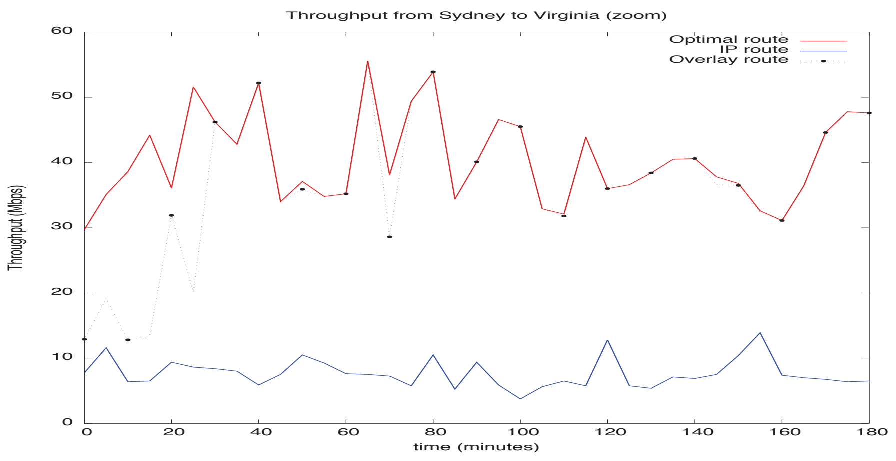


Figure 3.14: Throughput (Mbps) measured from Virginia (USA) to Sydney (Australia) over the first 3 hours of the experiment reported in Figure 3.13.

3.6 Summary

Since intercontinental IP routes do not achieve optimal performance, we propose and implement an intercontinental overlay network that adaptively selects paths to establish source-destination connections for packet traffic. Our experimental results show that even with a very small monitoring effort, significant QoS improvements can be achieved over the conventional Internet protocol. The results we have obtained have essentially considered paths of at most two overlay hops which may not be sufficient for some source to destination pairs. Thus we may consider in the future more extensive probing schemes, that use much more data collected at one or two hour intervals with probing of all overlay node pairs, in order to identify the best paths.

Chapter 4

Adaptive Dispatching of Tasks in the Cloud

The increasingly wide application of Cloud Computing enables the consolidation of tens of thousands of applications in shared infrastructures. Thus, meeting the QoS requirements of so many diverse applications in such shared resource environments has become a real challenge, especially since the characteristics and workload of applications differ widely and may change over time. This chapter presents an experimental system that can exploit a variety of online QoS aware adaptive task allocation schemes, and three such schemes are designed and compared. These are a measurement driven algorithm that uses reinforcement learning, secondly a "sensible" allocation algorithm that assigns tasks to sub-systems that are observed to provide a lower response time, and then an algorithm that splits the task arrival stream into sub-streams at rates computed from the hosts' processing capabilities. All of these schemes are compared via measurements among themselves and with a simple round-robin scheduler, on two experimental test-beds with homogenous and heterogenous hosts having different processing capacities.

This chapter is based on the following papers that we have published:[WG18, GW15, WG15a].

4.1 Introduction

Cloud computing enables elasticity and scalability of computing resources such as networks, servers, storage, applications, and services, which constitute a shared pool, providing on-demand services at the level of infrastructure, platform and software [MG09]. This makes it realistic to deliver computing services in a manner similar to utilities such as water and electricity where service providers take the responsibility of constructing IT infrastructure and end-users make use of the services through the Internet in a pay-as-you-go manner. This convenient and cost-effective way of access to services boosts the application of Cloud computing, which spans many domains including scientific, health care, government, banking, social networks, and commerce [Buy13].

An increasing number of applications from the general public or enterprise users are running in the Cloud, generating a diverse set of workloads in terms of resource demands, performance requirements and task execution [DK13]. For example, multi-tier web applications composed of several components which are commonly deployed on different nodes [PSZ$^+$07], impose varied stress on the respective node, and create interactions across components, tasks being executed in a Cloud environment may be of very different types, such as Web requests that demand fast response and produce loads that vary significantly over time [ZWL$^+$13], and scientific applications that are computation intensive, undergoing several phases with varied workload profiles [IOY$^+$11], or MapReduce tasks composed of different tasks of various sizes and resource requirements [ZWL$^+$13]. Furthermore, Cloud computing enables highly heterogeneous workloads to be served on a shared IT infrastructure leading to inevitable interference between co-located workloads [ZBF10], while end users require assurance of the quality and reliability of the execution of the tasks that they submit. Therefore, the Cloud service provider must dispatch incoming tasks to servers with consideration for the quality of service (QoS) and cost within a diverse and complex workload environment. Also, energy consumption remains a major issue that can be mitigated through judicious energy-aware scheduling [GL13].

Thus the present chapter focuses primarily on designing and evaluating adaptive schemes that exploit on-line measurement and take decisions with low computational overhead for fast on-line decision making. This work can be relevant to Cloud service providers that use the SaaS, model where customers pay for the services, while the service provider sets up the VMs where the required software components are installed to deal with the service requests from the customer.

Our experimental evaluations are conducted on a multiple host test-bed, running with low to high loads that are achieved by varying the types and arrival rates of tasks. To conduct these experiments with greater ease, we have also designed and implemented a portable software module, the Task Allocation Platform (TAP), that is Linux based and easily installed on a Linux based machine. TAP will dynamically allocate user tasks to the available machines, with or without making use of on-line measurements of the resulting performance, and adapt to changes in workload and on-going performance of the Cloud environment, while optimising goals such as cloud provider's profit while maintaining service level agreements (SLAs). TAP is flexible in that it can easily support distinct static or dynamic allocation schemes. It collects measurements on the test-bed, both to report on performance evaluation and also (for certain allocation algorithms) to exploit measurements for adaptive decisions.

Thus in this chapter we will report on the performance observed with two well known static allocation algorithms (Round-Robin and a probabilistic "equal loading" scheme), and three dynamic algorithms that are described in Section 4.3.1.

4.2 Prior Work

Extensive research in this challenging area includes work on static algorithms [TT85, KK92, KLKZ11] which are simple without excessive overhead; but they are only suitable for stable environments, and cannot easily adapt to dynamic changes in the Cloud. Dynamic algorithms [Wol01, RCL09, ZZ10, TZZ$^+$11] take into consideration different application characteristics and workload profiles both prior to, and during, run-time; however heir complexity can result in computational overhead that may cause performance degradation when implemented in a real system. Thus, many dynamic and adaptive schemes have only been evaluated through simulations [ZQQ11] rather than in practical experiments, while few have been tested in real environments but with low task arrival rates [DK13].

Much work on task assignment in the Cloud is based on a detailed representation of tasks to be executed with a rather simplistic representation of the hosts or processing sub-systems, leading to an evaluation based on simulation experiments rather than measurements on a real system. In [THyW02] an application composed of many tasks is represented by a directed acyclic graph (DAG) where tasks, inter-task dependency, computation cost, and inter-task communication cost are represented; two performance-effective and low-complexity algorithms rank the tasks to assign them to a processor in a heterogeneous environment. Related work is presented in [SL93, KA96], while optimisation algorithms based on genetic algorithms [HAR94], ant colony optimisation (ACO) [CZ09], Particle Swarm optimisation [PLGB10], Random Neural Network optimisation [GF99], and auction-based mechanisms [ZG11] have also been studied in this context, with potential applications to workload scheduling in the Cloud [LL11]. In [MGTX14], workload models which reflect the diversity of users and tasks in a Cloud production environment are obtained from a large number of tasks and users over a one month period, and exploited for evaluation in a simulated CloudSim framework.

Other work has used experiments on real test-beds rather than simulations [ZWL+13] where the characteristics of the typical heterogeneous workloads: parallel batch tasks, web servers, search engines, and MapReduce tasks, result in resource provisioning in a manner that reduces costs for the Cloud itself. Another cost-effective resource provisioning system dedicated to MapReduce tasks [PSL15] uses global resource optimisation. Hardware platform heterogeneity and co-scheduled workload interference are highlighted in [DK13], where robust analytical methods and collaborative filtering techniques are use to classify incoming workloads in terms of heterogeneity and interference before being greedily scheduled in a manner that achieves interference minimisation and server utilization maximization. The system is evaluated with a wide range of workload scenarios on both a small scale computer cluster and a large-scale Cloud environment applying Amazon EC2 to show its scalability and low computation overhead. However, the arrival rate of incoming workload is low and thus the system performance under saturation state is not examined. Furthermore, the contention for processor cache, memory controller and memory bus incurred by collocated workloads are studied in [ZBF10].

Early research that considers the important role of servers in delivering QoS in the Internet can be found in [BF99], where an architecture is proposed which provides web request classification, admission control, and scheduling with several priority policies to support distinct QoS requirements for different classes of users for multi-tier web applications. However, the scheduling approach is static and in [PSZ+07], an adaptive feed-back driven resource control system is developed to dynamically provision resource sharing for multi-tier applications in order to achieve both high resource utilization and application-level QoS. A two-tiered on-demand resource allocation mechanism is presented in [SSS13] with local allocation within a server and global allocation based on each local one, so as to achieve better resource utilization and dynamically adjust according to time-varying capacity demands. Energy consumption in computation, data storage and communications is also a challenge in the Cloud. A model for server performance and power consumption is derived in [GLD12] with the potential to predict power usage in terms of workload intensity. In [GL13], the authors examine the selection of system load that provides the best trade-off between energy consumption and QoS. A heterogeneity-aware dynamic capacity provisioning scheme for Cloud data centers is proposed in [ZZBH14], which classifies workloads based on the heterogeneity of both workload and machine hardware and dynamically adjusts the number of machines so as to optimise overall energy consumption and scheduling delay.

4.3 Overview of this chapter

The present chapter uses experiments to investigate adaptive dynamic allocation algorithms that take decisions based on on-line and up-to-date measurements, and make fast online decisions to achieve desirable QoS levels [DWC10]. The TAP that we have designed to this effect is a practical system implemented as a Linux kernel module which can be easily installed and loaded on any PC with the Linux OS.

TAP runs on a given host, and embeds measurement agents into each host in a Cloud to observe the system's state. These observations are then collected by "smart packets" (SPs) that TAP sends at regular intervals into the system in a manner which favours the search of those sub-systems which are of the greatest interest because they may be used more frequently or because they could provide better performance. The remainder of the chapter is organized as follows.

The task allocation algorithms, including three novel approaches, are discussed in Section 4.3.1. TAP, the task allocation platform that we have designed, is discussed in Section 4.4, where the dynamic algorithms are introduced. Section 4.4.1 discusses all the three measurement based allocation schemes, including a mathematical model based scheme presented in Section 4.4.1, the Sensible Algorithm in

Section 4.4.1, and the scheme that uses the RNN with reinforcement learning in Section 4.5.

The experimental results are introduced in Section 4.6, and first comparison of the different allocation schemes is presented in Section 4.7. In Section 4.7.2 we present further experimental results when the hosts being used have distinctly different processing speeds.

In Section 4.8 we introduce a "contradictory" performance metric based on the economic cost, as perceived by the Cloud platform, of executing tasks: this cost includes the *penalty* that the Cloud would have to pay to the end user when a SLA (service level agreement) is violated, as well as the intrinsic economic cost of using faster or slower hosts. This same cost function is used for task allocation in view of minimising the overall cost to the Cloud service, and it is then measured and reported for both the Sensible and the RNN based algorithms.

Finally, Section 4.9 draws our main conclusions and discusses directions for future research.

4.3.1 The Task Allocation Algorithms that are Investigated

In this chapter we design, implement in TAP and then experiment with several allocation algorithms:

- (a) round robin allocation of incoming tasks to distinct hosts,
- (b) a scheme that simply dispatches tasks with equal probability among hosts,
- (c) an allocation scheme that uses measurements of the execution times of tasks at hosts to allocate tasks probabilistically, where the probabilities are chosen via a mathematical model prediction so as to *minimise the average response time* for all tasks,
- (d) a Random Neural Network (RNN) [Gel00, GT08] based scheme that uses reinforcement learning with a numerically defined goal function that is updated with measurements brought back to TAP by SPs, and
- (e) an on-line greedy adaptive algorithm we call "sensible routing" [Gel03] that selects probabilistically the host whose measured QoS is the best.

To the best of our knowledge, the approaches (d) and (e) have not been used before for task allocation in Cloud or other multi-server environments, though related ideas were suggested for selecting packet routes in multi-hop packet networks [Gel09]. On the other hand, (a) and (b) are well known algorithms that are useful as benchmarks, and a scheme similar to (c) has been proposed in [GL13] for implementing trade-offs between energy consumption and quality-of-service in multiple-server computer systems.

We evaluate these schemes under varied task arrival rate via experiments on two test-beds: a cluster composed of hosts with similar processing speeds, and another one with where the hosts have significantly distinct processing capacities. The experimental results are then analysed and reported.

4.4 Task Allocation Platform and Test-Bed

TAP carries out online monitoring and measurement constantly in order to keep track of the state of the Cloud system, including resource utilisation (CPU, memory, and I/O), system load, application-level QoS requirements, such as task response time and bandwidth, as well as energy consumption, and possibly also (in future versions of TAP) system security and economic cost. With knowledge learned from these observations, the system can employ the QoS driven task allocation algorithms that we have designed, to make online decisions to achieve the best possible QoS as specified by the tasks' owners, while adapting to conditions that vary over time.

Figure 4.1 shows TAP's building blocks. The controller, which is the intellectual center of the system, accommodates the online task allocation algorithms, which work alongside the learning algorithm, with the potential to adaptively optimise the use of the Cloud infrastructure. TAP penetrates into the Cloud infrastructure by deploying measurement agents to conduct online observations that are relevant to the QoS requirements of end users, and send back the measurements to the controller. Three types of packets are used [Gel09] for communications between the components of the system: smart packets (SPs) for discovery and measurement, dumb packets (DPs) for carrying task requests or tasks, and acknowledgement packets (ACKs) that carry back the information that has been discovered by SPs. In this section, we present in detail the mechanisms that are implemented in the platform and the algorithms that are used.

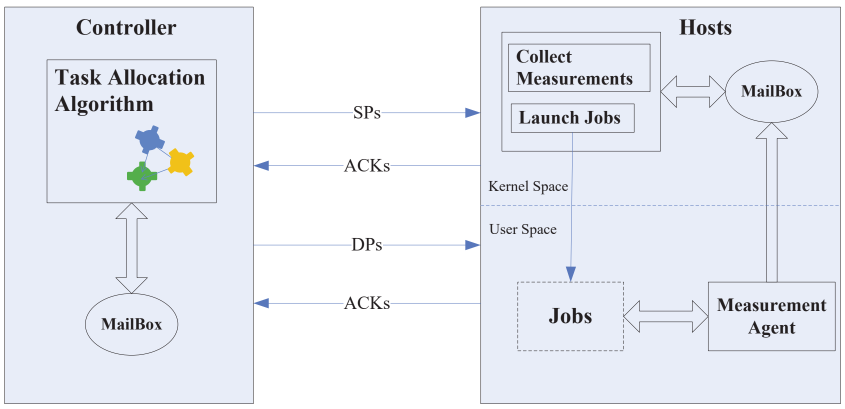


Figure 4.1: System Architecture showing the Task Allocation Platform (TAP), which is hosted by a specific computer that receives and dispatches jobs, and which interacts with the measurement system (at the right) which is installed on each host machine that executes jobs. The TAP communicates with each of the measurement systems at the hosts using SPs ("smart packets"), DPs ("dumb packets") and ACKs ("Acknowledgement Packets") as indicated in the text of the chapter.

SPs are first sent at random to the various hosts in order to obtain some initial information and inform the measurement agents in the hosts to activate the requested measurement. The task allocation algorithm in TAP learns from the information carried back by the ACKs and makes adaptively optimised decisions which are used to direct the subsequent SPs. Thus, the SPs collect online measurements in an efficient manner and pay more attention to the part of the Cloud where better QoS can be offered, visiting the worse performing parts less frequently.

The incoming tasks or task requests are encapsulated into the DPs, and exploit the decisions explored by SPs to select the host/Cloud sub-system that will to execute the task. Once a task (request) arrives at a host in the Cloud, its monitoring is started by the measurement agent which records the trace of the task execution until it is completed and deposits the records into a mailbox which is located in the kernel memory of the host. When an SP arrives at this host, it collects the measurements in the mailbox and generates an ACK which carries the measurements, and travels back to the controller where the measurement data is extracted and used for subsequent decisions of the task allocation algorithm. As soon as a task completes its execution, the agent also produces an ACK heading back to the controller with all the recorded data, such as the task arrival time at the Cloud, the time at which the task started running and the time at which the task execution completed. When the ACK of the DP reaches the controller, the task response time at the controller is estimated by taking the difference between the current arrival time at the node and the time at which the corresponding task arrives at the controller which is used by the algorithm when the task response time is required to be minimised.

4.4.1 Probabilistic Task Allocation Schemes

The schemes (b), (c), and (e) described in Section 4.3.1 are examples of probabilistic task allocation schemes. In these schemes, when a task arrives at from some user or source outside the Cloud system, TAP decides to allocate it to some host i among N with probability p_i so that *at decision time when the task must be allocated*:

- TAP first calculates p_i for each of the hosts i,
- Then TAP uses these probabilities to actually select the host that will receive the task.

In the case of (b) we obviously have $p_i = 1/N$.

Probabilistic schemes have the advantage that a host which is being preferred because, say it is providing better service, is *not* systematically overloaded by repeated allocation since the QoS it offers is only used probabilistically to make a task allocation. In other words, the chance that a given server receives two successive tasks is very small as compared to the case where successive tasks are allocated to distinct servers.

In addition to (b), we experiment with two distinct schemes to calculate p_i, Model Based Allocation (c) and Sensible Routing (e).

Model Based Task Allocation

Model Based Allocation (c) uses a mathematical model to predict the estimated performance at a host in order to make a randomised task allocation. This has been used in earlier work concerning task allocation schemes that help reduce the overall energy consumed in a system [GL13]. In this approach, if $W_i(\lambda, p_i)$ is the relevant QoS metric obtained for host i by allocating a randomised fraction p_i of tasks to host i when the overall arrival rate of tasks to TAP is λ, then the allocation probabilities $p_1, \ldots, p_N$ are chosen so as to minimise the overall average QoS metric:

$$W = \sum_{i=1}^{N} p_i W_i(\lambda, p_i). \tag{4.1}$$

At first glance, since each host i is a multiple-core machine with C_i cores, a simple mathematical model that can be used to compute, say the QoS metric "response time" $W_i(\lambda, p_i)$ that host i provides, assuming that there are no main memory limitations and no interference among processors (for instance for memory or disk access), is the $M/M/C_i$ queueing model [GM10], i.e. with Poisson arrivals, exponential service times, and C_i servers. Of course, both the Poisson arrival and the exponential service time assumptions are simplifications of reality, and more detailed and precise models are also possible for instance using diffusion approximations [Gel79] but would require greater computational effort and more measurement data.

However, a set of simple experiments we have conducted show that the $M/M/K$ model for each host would not correspond to reality. Indeed, in Figure 4.2 we report the *measured* completion rate of tasks on a host (y-axis) relative to the execution time for a single task running by itself, as a function of the number of simultaneously running tasks (x-axis). These measurements were conducted on a single host (Host 1), and for a single task running on the system, the average task processing time was 64.1ms.

If this were a perfectly running ideal parallel processing system, we could observe something close to a linear increase in the completion rate of tasks (red dots) when the number of simultaneously running tasks increases, until the number of cores in the machine C_1 have been reached. However the measurements shown in Figure 4.2 indicate (blue dots) a significant increase in completion rate as the number of tasks goes from 1 to 2, but then the rate remains constant, which reveals that there may be significant interference between tasks due to competition for resources. Indeed, if we call $\gamma(l)$ the average completion rate per task, we observed the following values for $\gamma_i(l)/\gamma_i(1)$ for $l = 2, \ldots, 10$ computed to two decimal digits: $0.67, 0.48, 0.34, 0.29, 0.23, 0.20, 0.17, 0.15, 0.13$. From this data, a linear regression estimate was then computed for the average execution time $\mu(i)^{-1}$ when there are

l tasks running simultaneously, as shown on Figure 4.3, yielding a quasi-linear increase. As a result we can quite accurately use the estimate $l.\gamma(l)/\gamma(1) \approx 1.386$. Based on this measured data, we

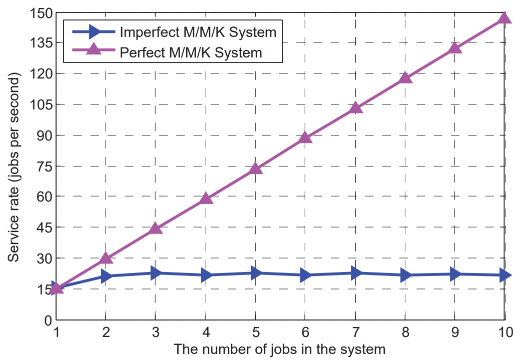


Figure 4.2: The ideal service rate provided by the perfect multiple core system (red), compared to the measured task completion rate on Host 1 (blue), plotted against the number of tasks running simultaneously on the host (x-axis).

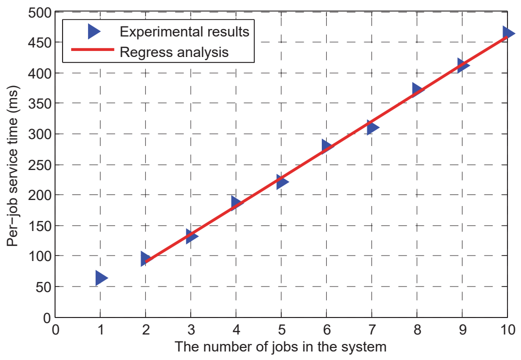


Figure 4.3: Measurement of the effective task execution time per task on Host 1, versus the number of simultaneously running tasks on the host (x-axis).

model the distribution of the number of tasks in a host server i as a random walk on the non-negative integers, where:

- $l = 0$ represents the empty host (i.e. with zero tasks at the host),
- The transition rate from any state $l \geq 0$ to state $l + 1$ is the arrival rate of tasks to the host λ_i,
- The transition rate from state 1 to state 0 is the $\mu_i(1) = T_i^{-1}$ where T_i is the average execution time of a task (by itself) on the host,

- The transition rate from state $l+1$ to state l if $l \geq 1$ is quasi constant given by $\mu_{i0} \equiv (l.\gamma(l)/\gamma(1))\mu_i(1)$,

- The arrival rate of tasks to Host i is $\lambda_i = p_i^m \lambda$ where p_i^m is the probability with which TAP using the model based algorithm assigns tasks to Host i, and λ is the overall arrival rate of tasks to TAP.

The probability that there are l tasks at Host i in steady-state is then:

$$p_i(1) = p_i(0)\frac{\lambda_i}{\mu_i(1)},\ p_i(l) = (\frac{\lambda_i}{\mu_{i0}})^{l-1} p_i(1),\ l > 1,$$
$$p_i(0) = \frac{1 - \frac{\lambda_i}{\mu_{i0}}}{1 + \lambda_i \frac{\mu_{i0} - \mu_i(1)}{\mu_{i0}\mu_i(1)}}.$$

so that using Little's formula [GM10] the overall average response time that we wish to minimise, by choosing the p_i^m for a given λ is:

$$W^m = \sum_{i=1}^{N} \frac{p_i}{\mu_i(1)} \frac{p_i(0)}{(1 - \frac{\lambda_i}{\mu_{i0}})^2}. \tag{4.2}$$

The appropriate values of the p_i^m with which the overall average response time 4.2 is minimized for a given system and a given arrival rate λ can be then obtained numerically.

Sensible Routing

The Sensible Decision Algorithm (e) uses a weighted average of G_i of the goal function that we wish to *minimise*, which is estimated from on-going measurements at each host i, and updated each time t that TAP receives a measurement that can be used to update the goal function. Specifically, when the goal function is the response time, its most recently measured value at time t, G_i^t, is received at TAP for host i, and the n-th update of G_i is computed:

$$G_i \leftarrow (1-\alpha)G_i + \alpha G_i^t, \tag{4.3}$$

where the parameter $0 \leq \alpha \leq 1$ is used to vary the weight given to the most recent measurement as compared to past values. Based on updating this value for each host i, the probability p_i^s that will be used to allocate a task to a host will be:

$$p_i = \frac{\frac{1}{G_i}}{\sum_{j=1}^{N} \frac{1}{G_j}}, \ 1 \leq i \leq N. \tag{4.4}$$

When TAP needs to allocate a task using (e), it will use the most recent value p_i which is available. Note that all of the G_i values for different i will not be as "fresh" as each other, though the probing via SPs from TAP to the hosts proceeds at the same rate for all hosts.

4.5 Random Neural Network Task Allocation with Reinforcement Learning

The Random Neural Network (RNN) has been used in static resource allocation as a "smart oracle" for allocating several resources to a set of tasks so as to minimise task execution times [GT08]. This earlier approach was based on first computing algorithmically a large set of optimum resource-to-task allocations, and then storing them in the RNN weights through a gradient descent learning algorithm. In order to select the best allocation, the trained RNN is then given an input which represents the set of available tasks, and it outputs the best known allocation.

This earlier work differs completely from the approach used in this chapter which is based on on-line search, similar to the search by autonomous robots [Gel10, GHK05, DFG10] with reinforcement learning [SB98] with real-time measurements. The RNN has also been used for packet routing [GLN04]; in that work, an RNN placed at each router to select the next hop for probe (or smart) packets which explore routes and collect quality of service information. Thus the probes are routed to explore the better paths in the network, and bring back the data they collect to each source router. End users then examine the data available at the source nodes, and select the best current paths from the data collected by the by the probes. This approach, where the RNNs serve to route the probes (but not the user traffic) also differs from the approach in this chapter, where an RNN is used to decide, for a given task, which server should be used.

In the present work, a RNN is used to select between N hosts to which a task will be allocated, using its N neurons in a fully connected form [GF99]. In the present case, a distinct RNN is set up within TAP to cover each distinct goal function G. However, these different RNNs need not be created in advance and stored at TAP indefinitely, but instead created when they are actually needed. Thus we will have a distinct RNN that is used to decide about allocations made on the basis of minimising economic cost (as when the end users pay a monetary price for the work they receive), or minimising task response time, or minimising task execution time, and so on.

A given RNN is initialised by setting $w^+(i,j) = w^-(i,j) = 1/2(N-1)$, so that $r(i) = 1$ for all i, and $\Lambda(i) = 0.25 + 0.5\lambda(i)$. In particular we can choose $\lambda(i) = 0$ so that all $\Lambda(i) = 0.25$. This of course results in $q_i = 0.5$ for all i.

TAP will then use the $q_i,\ i = 1,\ \ldots, N$ to make allocations so that a task is assigned to the host i that corresponds to the highest value of q_i. Initially, any one of the hosts will be chosen with equal probability. However with successive updates of the weights, this will change so that TAP selects the "better" hosts which provide a smaller value of G.

When TAP receives a value G_i^t of the goal function that was measured at time t at host i, and $\frac{1}{G_i^t}$ is the "reward", so that the RNN weights are updated as follows:

- We first update a decision threshold T_l as

$$T \leftarrow \alpha T + (1-\alpha)\frac{1}{G_i^t} \tag{4.5}$$

  where $0 < \alpha < 1$ is a parameter used to vary the relative importance of "past history".

- Then, if $G_i^t < T$, it is considered that the advice provided by the RNN in the past was successful and TAP updates the weights as follows:

$$\begin{aligned} w^+(j,i) &\leftarrow w^+(j,i) + \frac{1}{G_i^t} \\ w^-(j,k) &\leftarrow w^-(j,k) + \frac{1}{G_i^t(N-2)},\ if\ k \neq i \end{aligned}$$

- $else\, if\; G_i^t > T$

$$\begin{aligned} w^+(j,k) &\leftarrow w^+(j,k) + \frac{1}{G_i^t(N-2)},\; if\; k \neq i \\ w^-(j,i) &\leftarrow w^-(i,j) + \frac{1}{G_i^t}, \end{aligned}$$

- We compute $r^*(i) = \sum_{k=1}^{N}[w^+(i,k) + w^-(i,k)]$ for all i and renormalise all weights so that their values do not grow indefinitely:

$$\begin{aligned} w^+(i,k) &\leftarrow \frac{r(i)}{r^*(i)} w^+(i,k), \\ w^-(i,k) &\leftarrow \frac{r(i)}{r^*(i)} w^-(i,k). \end{aligned} \tag{4.6}$$

After the weights are updated, the q_i are computed with the new weights. Since this algorithm will tend to increase the probability q_i of those neurons which correspond to hosts that yield a smaller value of G_i, each time TAP assigns a task to a host, it uses the host i that corresponds to the largest q_i.

In order to make sure that TAP tries out other alternates and does not miss out on better options, a fraction f of the decisions are made in round robin fashion: thus we are sure that all hosts will be tried out in succession for $f \times 100\%$ of the decisions, and the resulting goal function values will also be collected and updated. In the experiments that we describe below, f was taken to be 0.1, i.e. 10%. We have actually evaluated this percentage experimentally and found 10% to provide the best value in the setting of our experiments, but depending on the size of the system this percentage may vary.

Note also that this algorithm can be modified to a probabilistic“sensible” version [Gel03] with:

$$p_i^{RNN-S} = \frac{q_i}{\sum_{j=1}^{N} q_j}. \tag{4.7}$$

4.6 Experiments

We conduct our experiments on a hardware test-bed composed of four nodes that each offer computation, storage and I/O. One node is dedicated to supporting the decision algorithms implemented in TAP, and the other three nodes are used as hosts running task, as shown in Figure 4.4, with each having a different processing power so that we may observe significant execution time differences for a given task. Since TAP takes decisions based on online measurements, even when there are no incoming tasks, the system maintains awareness of the state of the Cloud by sending SPs periodically. End users are allowed to declare the QoS requirements related to the tasks they submit, which is then translated into one or more QoS metrics which constitute a function called the "goal function" in our system. In this way, the QoS requirements are transformed into a goal function to be minimised, e.g. the minimisation of the task response time. The goal function determines which system parameters need to be measured and how task allocation will be optimised. TAP is implemented as a Linux kernel module which can be easily installed and loaded on any PC with Linux OS. The three hosts (with 2.8GHz, 2.4GHz, and 3.0GHz, respectively, dual-core CPU respectively) are used for task execution, while a separate host (2.8GHz dual-core CPU) supports the controller.

In these experiments we use a small scale test-bed so that we may easily load, and saturate, the system and evaluate the algorithms in both high, medium and low load conditions. However, TAP is scalable because most SPs are sent to to those hosts which are providing better performance, so that there is no "flooding" of SPs across the system.

A synthetic benchmark is generated with task profiles indicated by using the fields $\{task\ ID, QoS\ requirement, t$
which are packetised into an IP packet and sent to the controller. The task request generator uses this information to send a task requests. In order to vary the load, in addition to using tasks with distinct CPU and I/O needs, the average time between successive task initialisations is varied, and these times are either of fixed duration (denoted by CR in the figures), or follow a Poisson process denoted by EXP.

The first set of experiments we report were run with tasks that were defined as a "prime number generator with an upper bound B on the prime number being generated". Thus the choice of B allowed us to vary both the execution time and the memory requirements of the task. We did not actually "transfer" the tasks from the task controller to the host, but rather installed the task in advance on the host, and the allocation decision by TAP just resulted in arrival of a message from TAP to activate the task with specific value of B on that particular host. The measurement agent resident on that host then monitored the task execution and recorded its measurements into the mailbox. Both the tasks and the measurement agent run in the user's memory space, while the module that receives the SPs and task requests carried by DPs, collects measurements from the mailbox, and generates ACKs with the collected measurements runs in the kernel space of memory as shown in Figure 4.4, so that interference between the user program and the system aspects are avoided at least within the memory.

The two QoS goals that were considered were (i) the minimisation of either the execution time (denoted by ET in the figures) on the host, and (ii) the minimisation of the response time (denoted by RT in the figures) at TAP, where RT includes the message sent to activate the task at a host and the time it takes for an ACK to provide information back to TAP, where both the ET and the RT are provided to TAP from the host to the controller.

We first used TAP with the RNN algorithm with Reinforcement Learning (RL) as described above, and TAP with the sensible decision algorithm, and compared their performance. The RNN based TAP was experimented with both (i) and (ii), whereas the sensible decision based TAP only used (ii) the task response time at the controller.

In addition, according to the analytical model based approach was with (ii) task response time computed in terms of the task arrival rate and the system service rate, and then used to determine the optimum values of λ_1, λ_2, λ_3 corresponding to the three hosts subject to $\lambda = \lambda_1 + \lambda_2 + \lambda_3$, with an aim to minimise the overall task response time of the system as in (4.2), and then conducted experiments with task allocation probabilities to the three hosts selected so as to result in the arrival streams to the three hosts having the rates recommended by the analytical solution.

We also compared two static allocation schemes: Round Robin where successive tasks are sent to each host of the cluster in turn, and an equally probable allocation where a task is dispatched to each host with equal probability 0.33.

All these experiments were repeated for a range of average task arrival rates λ equal to 1, 2, 4, 8, 12, 16, 20, 25, 30, 40 tasks/sec, in order to evaluate performance under load conditions that vary from light to heavy load, including saturation. Each experiment lasted 5 mins so as to achieve a stable state.

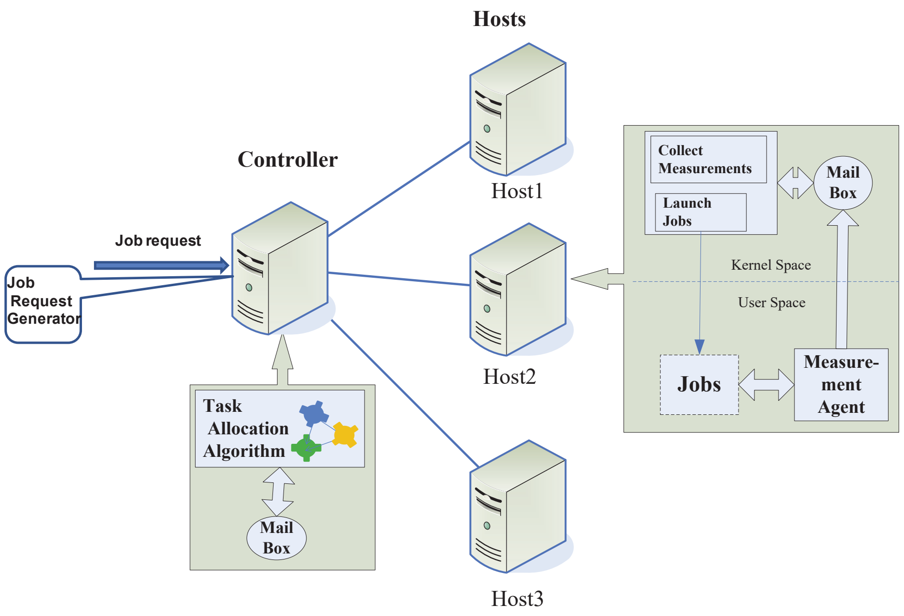


Figure 4.4: Schematic description of the task allocation test-bed. Jobs arrive at the controller machine for dispatching to the hosts. The TAP (Task Allocation Platform) software is installed at the controller and takes the dispatching decisions. TAP takes decisions based on data it gathers from each of the measurement systems (at the right) which are installed on each host machine that executes jobs.

4.7 Comparison of the Different Algorithms

We first compared the two approaches, the RNN and the Sensible Algorithm, based on the measured average task response time observed at the controller, the average task response time at the host and the average task execution time. We see that the three metrics exhibit the same trend as shown in Figure 4.5.

At low task arrival rates less than 8/sec, the RNN with RL performs better as shown in Figure 4.5(d), and it is even clearer with constant task arrival rates. However, as the average task arrival rates grows, the sensible decision algorithm outperforms the RNN, as in Figure 4.5(c). Also the RNN algorithm with online measurement of the task execution time always performs better than the RNN with the metric of task response time. However, the sensible decision is always best under high task arrival rates, as shown in Figure 4.5(c) .

To explain these experimental results, we note that in these we use CPU intensive tasks, and *each of them* experience longer execution time than when they are executed separately due to the competition for the same physical resource, the CPU. Indeed, the hosts are multi-core machines running Linux

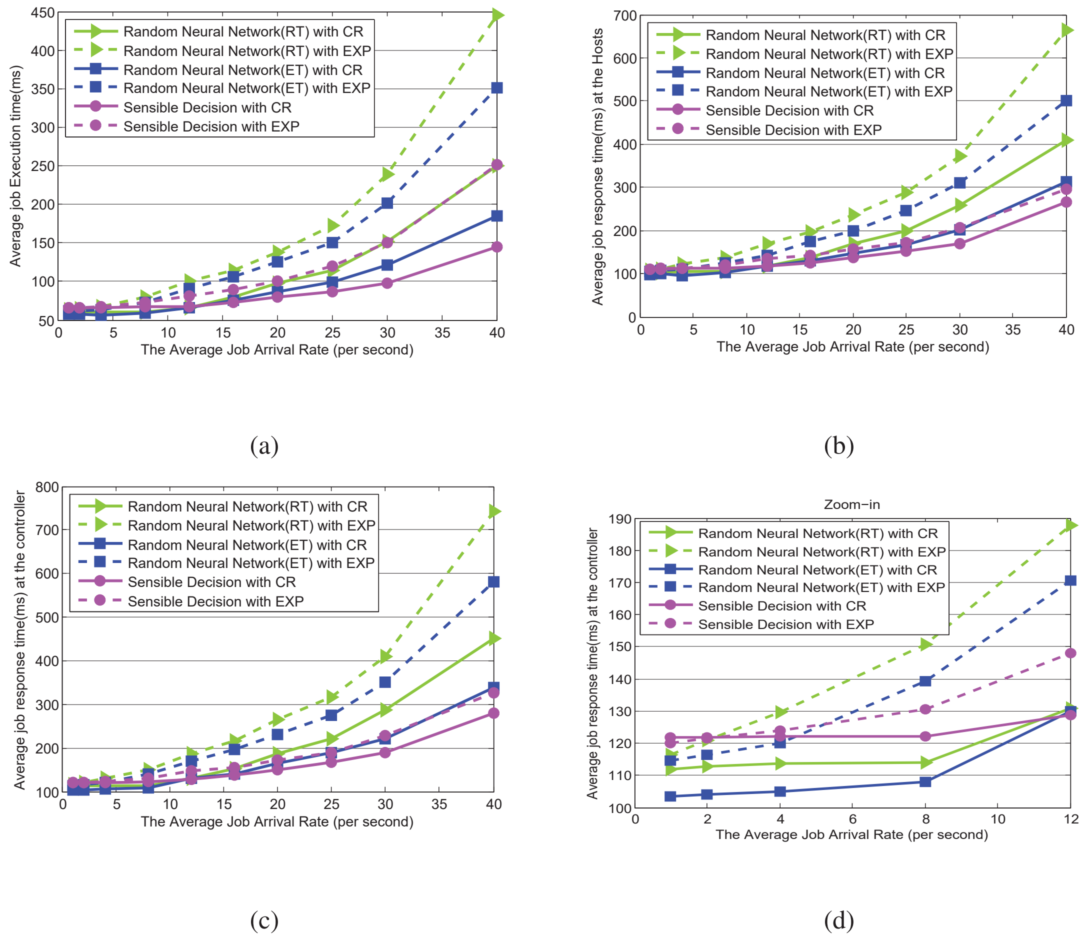


Figure 4.5: Comparison of TAP operating with either the “RNN with reinforcement learning (RL)”, or the “Sensible Algorithm” task allocation schemes. The metrics used for comparison are the resulting job execution time (ET) and job response time (RT) which are shown in the y-axis. Note that the Goal Function being optimised for each of the schemes, as shown in the legend for each curve, is the Response Time (RT) or the execution time (ET). We vary the rate at which jobs arrive (x-axis). Results are shown both for constant job inter-arrival times (CR), and for Poisson arrivals (EXP).

with a multitasking capability so that multiple tasks will run together and interfere with each other as shown in Figure 4.3. It can be found that, for example, if four tasks running in parallel, the average execution/response time per task increases two times. That is to say, the fluctuation of the execution time that the tasks experienced under varied number of tasks in the system is quite significant. Since the RNN with RL will send the tasks to the best performing hosts, it will tend to overload them, contrary to the Sensible Algorithm which dispatches tasks probabilistically and therefore tends to spread the load in a better manner.

When RNN used the task execution time as the QoS criterion, Figure 4.6(a) shows that it dispatched the majority of tasks correctly to Host 3 which provided the shortest service time. The other two hosts accommodated some tasks because the RNN algorithm was programmed to make 10% of its decisions at random with equal probability. Here, the sensible decision algorithm performed worse because it makes task allocation decision with a probability that is inversely proportional to the task response time/execution time, instead of exactly following the best QoS as the RNN. As shown in Figure 4.6(b), the proportion of the tasks allocated with the sensible decision algorithm coincides with the proportion of the respective speeds of the three hosts.

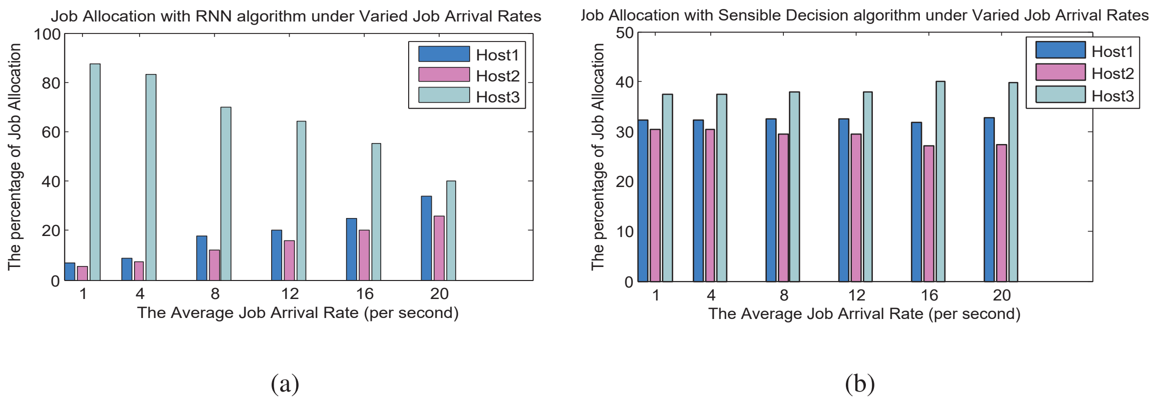
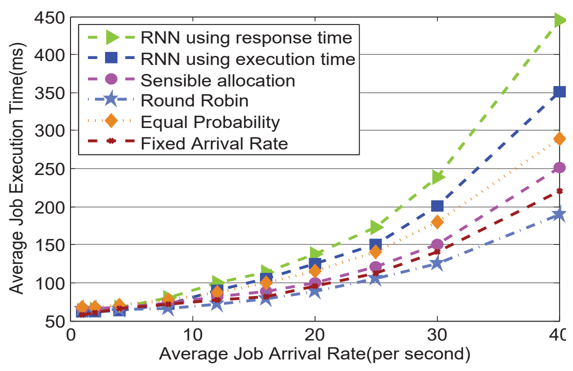


Figure 4.6: The Proportion of task allocations to the three hosts with the RNN and the Sensible Algorithm for different task arrival rates.

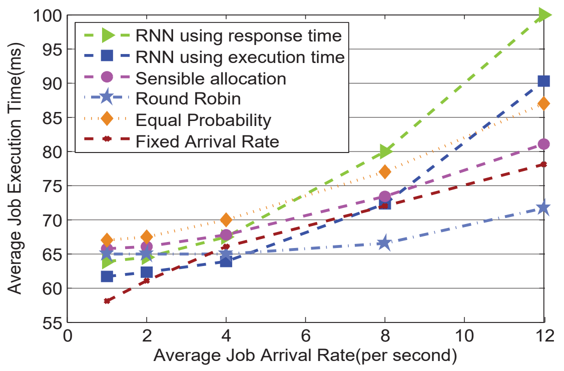
On the other hand, the Sensible Algorithm benefits from the fact that it does not overload the “best” hosts as shown in Figure 4.5(c) where the tasks may sometimes arrive to a host at rate that is higher than the host’s average processing rate, leading to overload or saturation of the host. In Figure 4.5 we also see that the RNN based algorithm, that uses the task execution time measured at the hosts as the QoS goal, outperforms the RNN with online measurement of the task response time, because the execution time can be a more accurate predictor of overall performance when the communication times between the hosts and TAP fluctuate significantly. However at high task arrival rates, the Sensible Algorithm again performed better.

4.7.1 Comparison with the Model Based and Static Allocation Schemes

Figure 4.7 shows the average task execution time for the RNN and the Sensible Algorithm, in comparison with the model based scheme, as well as the Round Robin and Equally Probable allocation. The model based scheme performed better than the RNN when the task arrival rate was low, and better



(a)

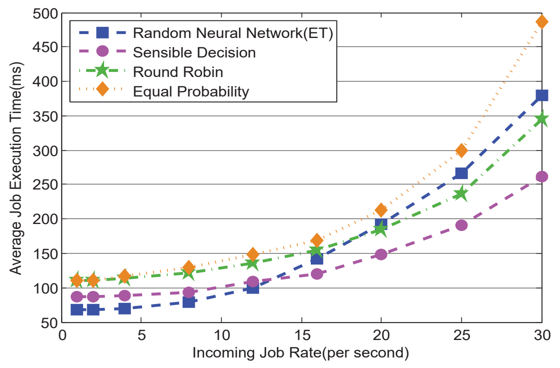


(b)

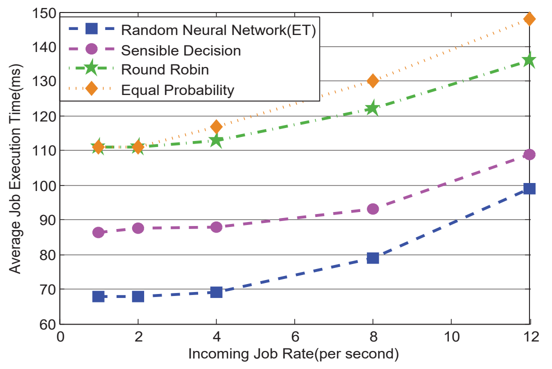
Figure 4.7: The average task execution time experienced under varied task arrival rates and different task allocation schemes when the three hosts have similar performance.

than the Sensible Algorithm at high arrival rates. However, the model based scheme can be viewed as an "ideal benchmark" since it relies on full information: it assumes knowledge of the arrival rate, it supposes that arrivals are Poisson, and it assumes knowledge of the task service rates at each host, while the RNN based scheme just observes the most recent measurement of the goal function.

As expected the equally probable allocation scheme performed worse. In this case where all servers are roughly equivalent in speed, Round Robin always outperformed the Sensible Algorithm, because it distributes work in a manner that does not overload any of the servers. These results are summarised in Figure 4.7(a). However the observed results change when the hosts have distinct performance characteristics as shown below.



(a)



(b)

Figure 4.8: Average execution time experienced in a cluster composed of hosts with non-uniform processing capacities. 4.8(b) is zoomed in of 4.8(a) when the incoming job arrival rates are blow 12 per second, which generates the light to medium load conditions

4.7.2 Performance Measurements when Hosts have Distinct Processing Rates

As a last step, we evaluate the algorithms that we have considered, in a situation where each hosts provides significantly different performance. To strongly differentiate the hosts, we introduce a *background load* on each host which runs constantly and independently of the tasks that TAP allocates to the hosts. This is in fact a realistic situation since in a Cloud, multiple sources of tasks may share the same set of hosts without knowing what their precise workload may be, except for external observations of their performance.

Thus we were emulate three hosts $i = 1,\ 2,\ 3$ with relative processing speeds of $2:\ 4:\ 1$, respectively. The results of these experiments are summarised in Figure 4.8. We see that TAP with both the RNN and the Sensible Algorithm benefits from the ability of these two schemes to measure the performance differences between the hosts, and dispatch tasks to the hosts which offer a better performance, whereas the two static allocation schemes (Round Robin and the allocation of tasks with equal probabilities) lead to worse performance as a whole.

The performance of the RNN-based scheme clearly stands out among the others under light to medium load conditions as shown in Figure 4.8(b), confirming that the RNN with RL algorithm is a very useful fine-grained QoS-aware task allocation algorithm which can make more accurate decisions provided that the online measurements are updated on time.

4.7.3 Multiple QoS Classes

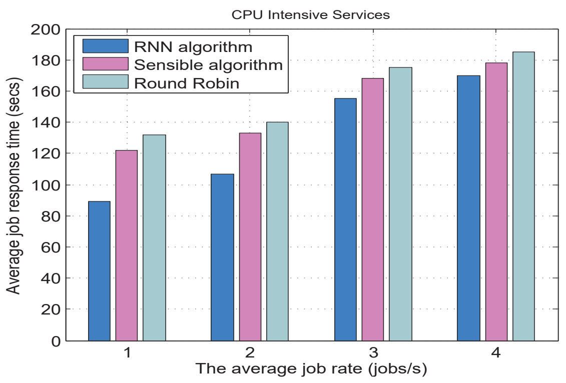
In this section, we will study the effectiveness of TAP when there is greater diversity both in the types of tasks, and in the type of QoS criteria and the SLA that they request. To evaluate the allocation algorithms with two different classes of tasks, we used we a web browsing workload generated with HTTPerf which is a well-known web server performance tool.

The first class corresponds to HTTP requests retrieve files from a web server, such as the Apache 2 HTTP server, whereby I/O bound workload is generated on the web server with very little CPU consumption, and the load on the I/O subsystem can be varied with the size of the retrieved files. In our TAP test-bed, the Apache server is deployed on each host in the cluster. HTTPerf generates HTTP requests at a rate that can be specified, while TAP receives the requests and dispatches them to the web servers.

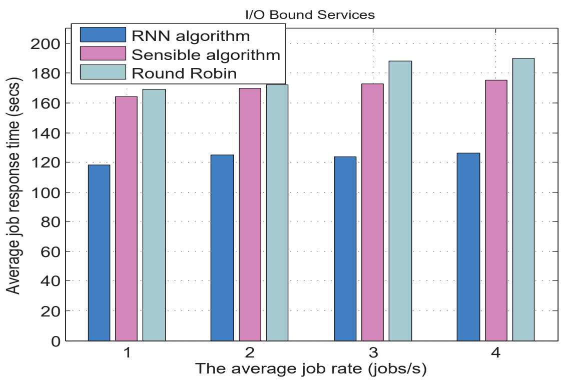
On the other hand, the web services which require a large amount of computation, mainly generate CPU load, are represented by CPU intensive tasks generated by the prime number generator.

In this case we compare the RNN based algorithms with the Sensible Algorithm, both using the Goal of minimising the response time. We also compare them to Round-Robin scheduling. The hosts themselves are stressed differently in terms of CPU and I/O in the cluster to provide different heterogeneous environments. The workload is generated so as to arrive at TAP following a Poisson process with different average rates of $1, 2, 3, 4$ tasks/sec.

The different performance level offered by the hosts is implemented by introducing a background load which stresses I/O differently on each host, resulting in relative processing speeds of $6 : 2 : 1$ for Hosts $1,\ 2,\ 3$ with regard to I/O bound services, while a background load which stresses CPU distinctly on each host, resulting in the relative processing speed of $2 : 3 : 6$ (corresponding to Hosts $1, 2, 3$) is used for the CPU bound case.



(a)



(b)

Figure 4.9: Average response time experienced by CPU intensive services and I/O bound services in a heterogeneous cluster. We compare Round-Robin with RNN based Reinforcement Learning and the Sensible Algorith.

The results in Figure 4.9 show that the RNN based algorithm performs better; the reason may be that it is able to detect the best possible host for the task based on its QoS requirement by effective learning from its historical performance experience and make more accurate decisions (compared with Sensible) which dispatch I/O bound tasks to the hosts where I/O is less stressed and dispatch CPU intensive tasks to the hosts which provide better CPU capacity. During the experiments, we reduced the background load in terms of both CPU and I/O stress on the $Host\ 2$ to the lowest level as compared with the $Hosts\ 1,\ 3$. It was found that the RNN based algorithm was able to detect the changes and dispatch the majority of subsequent tasks of both types to $Host2$, which also shows the host where CPU is heavily stressed to still provide good performance to I/O bound tasks. More generally, we also observe that Round-Robin performs worse than the two other algorithms.

4.8 Contradictory QoS Requirements

Workloads are often characterised differently in terms of their resource and performance requirements. For example, for an online financial management and accounting software providing SaaS services, the web browsing workloads which retrieve large files from web servers generate I/O bound workload, while accounting and financial reporting applications may need a large amount of computation and require high CPU capacity. Thus, in this section we discuss how we may support multiple tasks with multiple QoS requirements for different QoS classes.

However, distinct QoS classes may also generate different levels of income for the Cloud service, and will also have different running costs on different hosts. In addition, they will have distinct service level agreements (SLAs), and the violation of the SLAs will often have different financial consequences.

Thus we can consider a QoS Goal which includes two contradictory economic requirements: if we allocate a task to a fast processor, the cost will be higher since a more expensive resource is being used, however the resulting response time will be better resulting in fewer SLA violations and hence a lower penalty (and hence cost to the user). Obviously we have the opposite effect when we allocate a task to a slower machine, resulting in a lower economic cost for hosting the task, but in a higher resulting cost in terms of penalties for the Cloud due to SLA violations.

Formalising these two contradictory aspects, consider a set of host servers, and different classes of tasks, so that:

- Let the amount paid by the user of a class j task to the Cloud service be I_j when the SLA is respected.

- Also, let us assume that if a task of QoS class j is allocated to a host m which is of type M_i, where the type of host includes aspects such as its memory capacity, its I/O devices, and speed, and the services offered by its software, will result in a cost to the Cloud service of C_{ij}.

- However, if the SLA is not respected there will *also* be some penalty to be paid by the Cloud service to the user, and this penalty must then be deducted from the income that the Cloud service was expecting to receive. For instance, the penalty will be zero if the response time T of the task is below the SLA upper limit $T_{j,1} > 0$ for class j tasks. More generally, the penalty is c_{jl} if $T_{j,l-1} \leq T < kT_{j,l}$, where $T_{j0} = 0$ and $c_{j0} = 0$ (no penalty).

Using standard notation, let $1_{[X]}$ is the function that takes the value 1 if X is true, and 0 otherwise. Then the *net income* obtained by the Cloud service for running a task of type j, after deducing the host operating cost and the eventual penalty for SLA violations, can be written as:

$$I_j^* = I_j - C_{ij}1_{[m=M_i]} - \\ \sum_{l=1}^{n}\{c_{jl}1_{[T_{j,l} \leq T < T_{j,l+1}]}\} + c_{j,n+1}1_{[T \geq kT_{j,n+1}]}. \quad (4.8)$$

Obviously, the cloud server would like to *maximise* I_j^*, while I_j is fixed in advance as part of the service agreement.

Thus in this section we consider task allocation based on a Goal function that will allocate a machine M_i to a task of class j so as to minimise the *net cost* function:

$$C_j = C_{ij}1_{[m=M_i]} + \sum_{l=1}^{n}\{c_{jl}1_{[T_{j,l} \leq T < T_{j,l+1}}\} \\ + c_{j,n+1}1_{[T \geq kT_{j,n+1}]}. \quad (4.9)$$

Figure 4.10 shows an example of the penalty function, which is the second term in (4.10), for two distinct classes of tasks, where the x-axis is the value of the response time T.

4.8.1 Experiments with Conflicting QoS Objectives

We illustrate the preceding discussion with experiments involving tasks of two classes with distinct QoS requirements, and we emulate a heterogeneous host environment where there is a fast host, a slow host and a medium speed host. This is done by stressing the CPU of each host differently, resulting in the speed-up factor of $1 : 2 : 4$ for Host $1, 2, 3$.

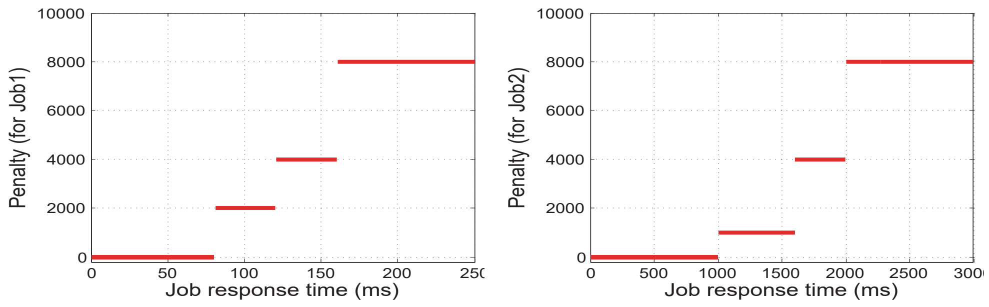
We conducted experiments using (4.10) as the Goal, with two classes of CPU intensive tasks and the two different penalty functions of Figure 4.10 regarding response time. The RNN based algorithm is compared with the Sensible Algorithm.

With regard to the terms in (4.10), we have $M_1 = 1000$, $M_2 = 2000$ and $M_3 = 4000$ coinciding with our assumption that the faster machines cost more, and the tasks either are “short” with an execution time of 56ms, or “long” with an execution time of 190ms as measured on the fastest host. Tasks were generated following independent and exponentially distributed inter-task intervals (Poisson arrivals), and four experiments with distinct task arrival rates of $1,\ 2,\ 3,\ 4$ tasks/sec were run separately for each class of tasks.

Figure 4.11 (a) shows that the RNN algorithm is able to do better in reducing the average response time in the case of the shorter tasks, while the Sensible Algorithm is more effective with regard to average response time for the longer tasks as seen in Figure 4.11 (b), though the RNN based algorithm manages to remain, at least *on average* below the 1000 penalty threshold . However we also see that the RNN does better in reducing the overall cost plus SLA violation penalty (indicated as the Total Penalty in the y-axis) as shown in Figure 4.11 (c).

We also conduct a separate set of measurements in order to estimate the standard deviation of the response times, and the maximum observed response times as shown in Figure 4.12. In these cases, each measurement point that we report is based on five experimental runs with Poisson arrivals, for each of the two classes of tasks, with each experiment lasting 20 minutes in order to generate ample statistical data. In this setting, we again compare the RNN based algorithm with Reinforcement Learning to the Sensible Algorithm, still using the Goal function with the cost defined in (4.10).

In most of the observed data, we see that the RNN-based algorithm can achieve better performance in



(a) (b)

Figure 4.10: The penalty function for tasks belonging to Class 1 (left) and Class 2 (right), versus the task response time on the x axis.

terms of reducing the standard deviation of the response times leading to more dependable results, and also to maximum response times that lead to a smaller penalty, as expected from the previous data. The exception to this general observation is when tasks arrive at the highest rate of 4 tasks/sec. In this case, it appears that the RNN based algorithm is not receiving timely data via the ACKs, leading to a level of performance that is worse than what is achieved by the Sensible Algorithm.

4.9 Summary

In this chapter we have first reviewed the area of task allocation to Cloud servers, and then presented TAP, an experimental task allocation platform which can incorporate a variety of different algorithms to dispatch tasks to hosts in the Cloud operating in SaaS mode, before reporting on numerous experiments that have used TAP to compare a variety of task allocation algorithms under different operating conditions and with different optimisation criteria.

We consider simple static allocations schemes, such as Round Robin, and a probabilistic allocation which distributes load evenly. We also study a model driven algorithm which uses model based estimates of response time to select distinct allocation rates to different hosts. Two measurement driven adaptive on-line algorithms are also considered : the RNN based algorithm with Reinforcement Leaning, and the Sensible Algorithm that bring intelligence to bear from observations and make judicious allocation decisions.

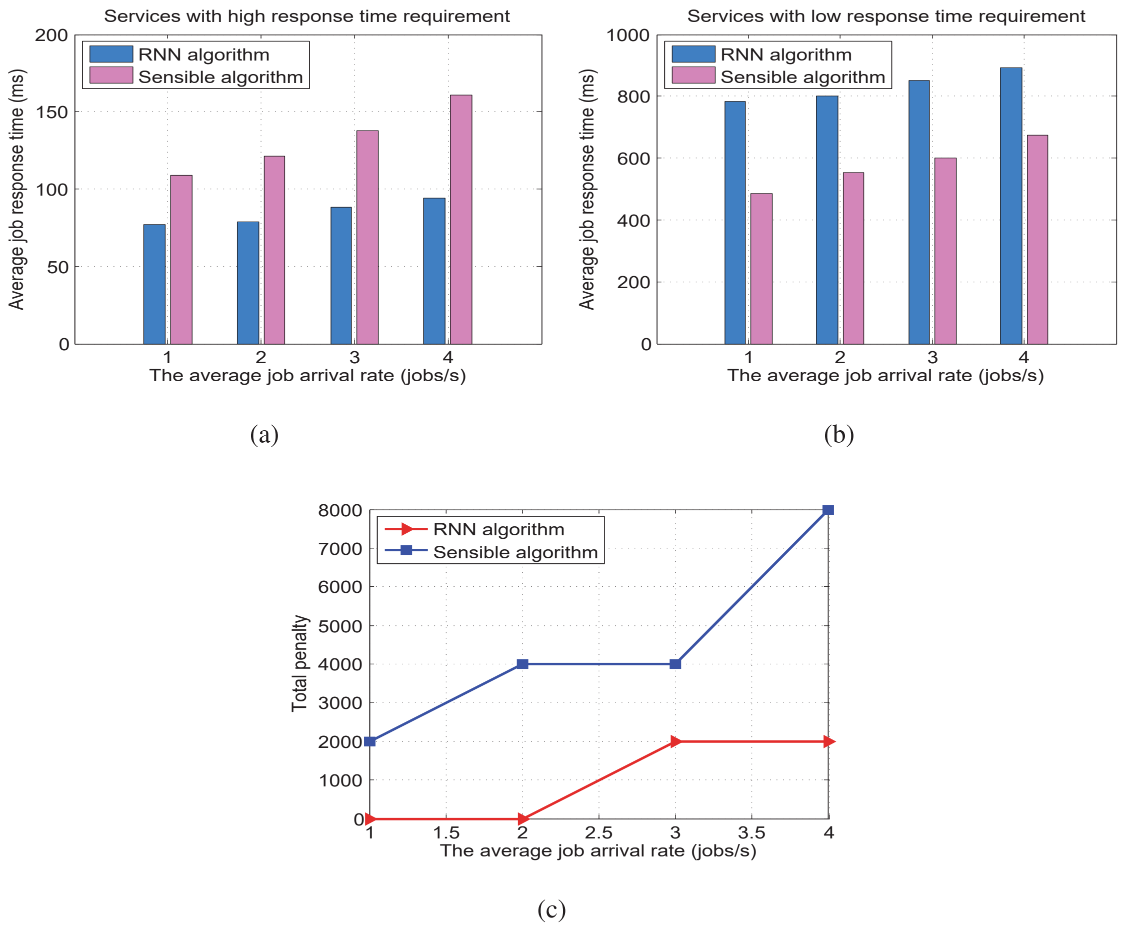
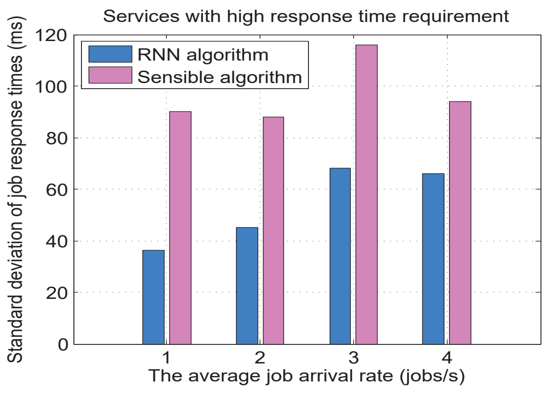


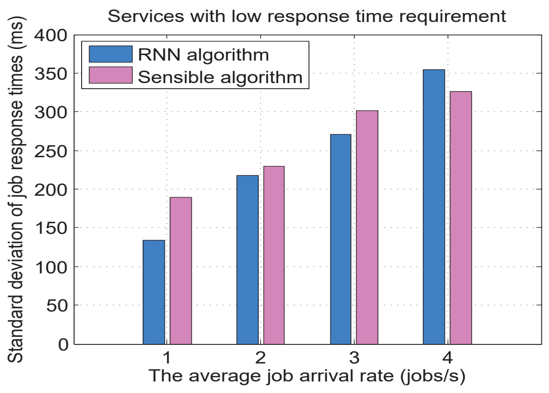
Figure 4.11: The average value of the measured total cost for the two classes of tasks, when allocation is based on the Goal function that includes both the economic cost and the penalty as in (4.10).

Numerous experiments with different task profiles, and optimisation objectives were considered, with two different both different sets of host machines: one composed of hosts with similar processing speeds, and another one with hosts having different speeds due to distinct background loads at each host.

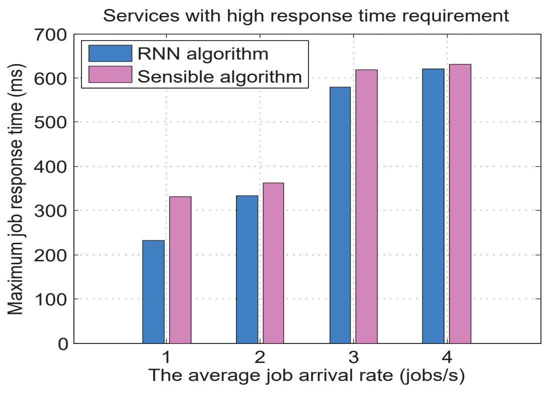
Experiments showed Round Robin is effective when the processing rates and loads at each of the hosts are very similar. However when the hosts are quite distinct, the RNN based algorithm with Reinforcement-Learning offered fine-grained QoS-aware task allocation algorithm for accurate decisions, provided that online measurements are frequently updated. We found that the Sensible Algorithm offers a robust QoS-aware scheme with the potential to perform better under heavier load. The fixed arrival rate scheme, with full information of arrival rates and service rates, outperformed both the RNN and "sensible" approach due to the fact that it employs the solution of an analytical model to minimise task response time under known mathematical assumptions. However such as-



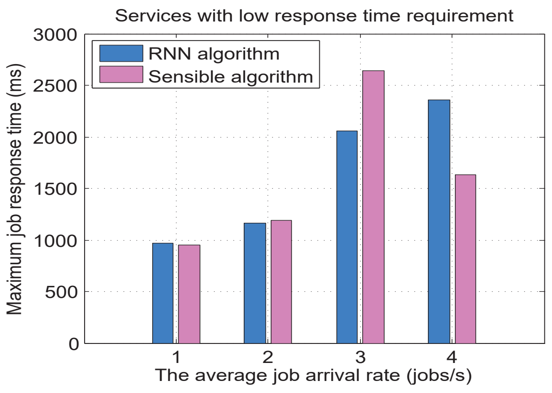
(a)



(b)



(c)



(d)

Figure 4.12: The standard deviation of the measured response time, and its maximum value as a function of the task arrival rate (x-axis), for the two classes of tasks when task allocation uses the Goal function including both the economic cost and the penalty (4.10). Note that the performance of the RNN and Reinforcement Learning based algorithm, and the Sensible Algorithm are being compared.

sumptions will not usually be known or valid in practice; thus it is useful as a benchmark but cannot be recommended in practical situations.

Chapter 5

Adaptive Workload Distribution for Local and Remote Clouds

Cloud systems include both locally based servers at user premises and remote servers and multiple Clouds that can be reached over the Internet. This chapter describes a smart distributed system that combines local and remote Cloud facilities. It operates with a task allocation system that takes decisions to allocate tasks dynamically to the service that offers the best overall Quality of Service and a routing overlay which optimizes network delay for data transfer between clouds. Internet-scale experiments exhibit the effectiveness of our approach in adaptively distributing workload across multiple clouds.

This chapter is based on the following papers that we have published:[WBG16].

5.1 Introduction

The wide-spread usage of IP networks allows large-scale Cloud providers such as Amazon EC2[ama], Microsoft Windows Azure[mic] and Google Compute Engine[goo] to build Cloud infrastructures at a global scale which facilitates the deployment of applications spanning multiple regions around the world for improving reliability and provisioning services with lower latency and a better experience for the end users. This enables service providers to distribute workloads across multiple Clouds for balancing load and offering better Quality of Service(QoS) to all the requests it receives. The selection of a Cloud for an end user also relies on other considerations such as security, cost, and energy consumption. For example, cloud bursting refers to “an application deployment model in which an application runs in a private cloud (on local servers) and bursts into a public cloud when the demand for capacity spikes”[clo].

Using the Internet as the network medium between Cloud regions is the default choice offered by many Cloud providers. It is known that the routes set up by IP do not necessarily result in the best performance [Pax96], and that IP connections may have lower reliability than other paths [LMJ98, LABJ00, DCGN01]. The Cloud processing latency is affected by workload distribution inside the Cloud; this not always easy to assess due to the heterogeneity in both workload and machine hardware, and the dynamic changes of load conditions over time. Much research on the solutions to these issues require sophisticated optimisation algorithms with relatively high computational complexity [THyW02, PLGB10, ZG11]. Some approaches are also based on substantial performance measurements resulting in traffic overhead [ABKM01, PST04, TWEF03, FBR^{+}04] and potential overhead in the Cloud. Such approaches can be difficult to put into practice due to scalability problems for large systems and the need for computational efficiency when one must make real-time and on-line decisions.

5.1.1 Technical Approach

In recent work [WG18] we have examined how a task allocation platform (TAP) that is *internal* to a Cloud can be used to dynamically make task allocation decisions to optimise QoS, and have suggested both model based and learning based approaches. In other recent work [BWG16] we have shown that a limited amount of re-routing over an overlay network using a measurement and big data approach can substantially average end-to-end delay for internet traffic and also reduce the perceived packet loss.

Here we combine these two ideas. In this chapter we focus on the QoS that tasks receive, in particular the overall response time which is determined by the network latency to access, forward data and programs, retrieve results and data, and which includes the local or remote Cloud processing delay. In particular, we propose a practical system for adaptive workload distribution across multiple Clouds over wide area networks. The system includes a TAP deployed in each Cloud that optimises user perceived QoS and exploits the routing overlay $SMART$ over Clouds [BWG16] for improving network delay incurred by data transfer.

In this approach, requests of users are routed to a designated local Cloud, which may well be the geographically closest one, provided that it has enough available capacity to handle the request. When the workload at the local Cloud increases, the TAP at the local Cloud can decide to forward requests to remote Clouds. In the process, TAP will consider the effect of both the data transfer delay and Cloud processing delay, each being weighted for their relative importance. The estimate of data transfer delay used in our system also takes into account the measured packet loss which will result in extra network delay for applications that use TCP for data transmission.

In order to optimise Cloud delay and network latency, our approach is meant to be easily deployable over a large population of machines, and it should be able to make fast on-line decisions resulting in good quality of service (QoS) with low computational overhead. Although it requires measurement and monitoring both of network characteristics and of local and remote Cloud delays, we limit the frequency and overhead related to the monitoring effort, and also limit the computational complexity of decision making using reinforcement learning [SB98].

We therefore propose an approach, applied both in TAP and to the routing overlay, which uses Reinforcement Learning [SB98] with the random neural network [GT08, Gel90, Gel89b, GF99] to make fast, judicious and efficient decisions based on the knowledge learned from the past observations, while adapting to changes in workload and on-going performance of the Cloud environment. Our approach benefits from limited measurement overhead as it probes the performance of sub-systems which provide better QoS, while still exploring less frequently a wider range of alternative systems that can in the future prove to provide improved QoS if the current set of frequently used subsystems result in poor QoS.

Experiments were conducted on a real large scale system operating on the Internet at a global scale and we empirically evaluate the potential of our proposed algorithms for adaptively distributing workload across multiple clouds. The experimental results that we obtain, validate the adaptiveness and effectiveness of our proposed system for dynamic environments.

5.2 Task Allocation Platform for multiple clouds

In recent work [WG18] we proposed a practical task allocation platform (TAP) that is *internal* to a Cloud. It includes a centralized controller for distributing incoming workloads and a measurement agent on each machine for observing service performance and server health state related to user-defined QoS goals. The controller receives the requests from end users and makes fast and judicious allocation decisions for optimising the specified QoS requirements based on the knowledge leaned from the observations using learning-based algorithms. We have validated that TAP is able to adaptively distribute workloads across available servers within a cloud in response to user required QoS when there is a great diversity in the types of jobs, the class of QoS goals and the resources which are required by workloads and which are possessed by servers.

TAP is scalable as it only needs limited measurement overhead for monitoring the performance of subsystems which provide better QoS, while still exploring less frequently a wider range of alternative systems that can in the future prove to provide improved QoS if the current set of frequently used subsystems result in poor QoS. We implemented TAP as a Linux based portable software module so that it can be easily installed on a machine with Linux OS.

In this section, we present an extension of the TAP designed for workload distribution across multiple clouds over wide area networks. As shown in Figure 5.1, user requests are routed to the local, which is perhaps the geographically closest cloud, if the cloud has enough capacity. The dispatcher at the local cloud receives the incoming workload and adaptively selects the best possible server based on the user-defined QoS (e.g. the response time).

If the workload in the local cloud increases, the dispatcher could decide whether to forward the subsequent requests to the remote clouds in order to balance the load and offer better QoS to all the requests it receives. The decision would rely on a variety of considerations, such as security, cost, QoS and energy consumption.

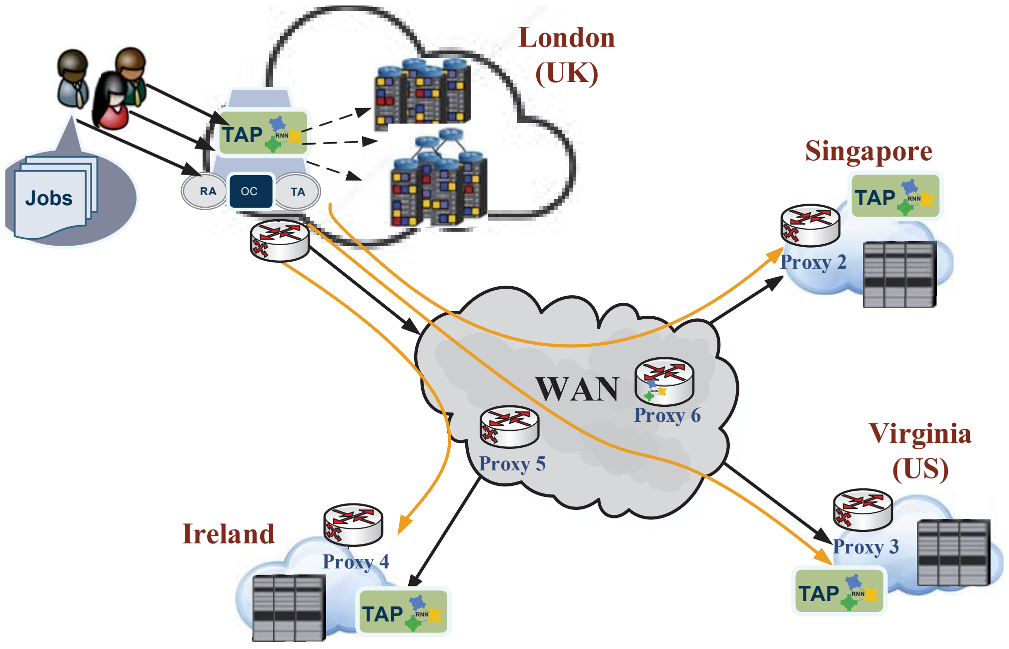


Figure 5.1: The overview of the TAP for workload distribution across multiple clouds over wide area networks

We currently only concern ourselves with the QoS that tasks receive, in particular the response time observed by tasks. Obviously this response time will be determined by the network delay incurred by the access to local or remote clouds, which includes the network delay to process the request, the network delay to forward the task with its possible code and data, plus the time it takes to return the results to the sender after execution, plus the waiting time and service time inside the clouds. That is to say, the dispatcher would select a remote cloud (say the j-th cloud) to share the workload of the local cloud by considering the response time (denoted by D_{job}^{j}) which consists of the data transfer latency (denoted by D_{n}^{j}) which depends on the traffic conditions on the connections to the remote clouds and the proximity of the remote cloud to the local cloud as well as the response time within the cloud (denoted by D_{e}^{j}) which is related to the viability or health state of the cloud. Hence, a goal function for the decision can be presented as,

$$D_{job}^{j} = aD_{n}^{j} + bD_{e}^{j} \tag{5.1}$$

where a and b are the relative importance being placed on the data transfer latency on network connections and the processing delay inside the cloud. It should be noted that the data transfer latency for the local cloud, though not zero, may be negligible. Each cloud can report its health state via TAP regularly, including the response time within the cloud. In the following section, we present the approaches we used to predict the data transfer latency via measuring the network delay on the connections to all the external clouds. This of course is easier said than done because it depends on the nature of the transfers and on the other traffic in the connections.

5.2.1 Data transfer delay estimation

To simplify matters, we measure the round-trip delay (denoted by $T_p^j(t)$ for the j-th cloud at time t) and the packet loss (denoted by $L^j(t)$) on each connection via pinging every a certain time interval (e.g. 1 second) thereby obtaining the network delay. A weighted average of the measurements for the round-trip delay is used as

$$T_p^j = \alpha T_p^j + (1 - \alpha) T_p^j(t) \tag{5.2}$$

where $0 < \alpha < 1$ is a parameter which reflects the relative importance between the past values and the most recent measurement.

The packet loss needs to be considered because some applications (e.g. HTTP requests) utilize TCP to transfer the data and require retransmissions for the lost packets, which results in extra delay. The loss is measured as follows:

$$L^j(t+1) = \frac{\alpha t L^j(t) + (1-\alpha) t 1_{[loss\ on\ the\ t-th\ ping]}}{t+1} \tag{5.3}$$

It should be noted that there is a loss on the connection to the j-th cloud if the pinging delay at the j-th cloud T_p^j is larger than some fixed value $T_{timeout}$.

Therefore, the network delay on the connection to the j-th cloud considering packet loss can be expressed as

$$D_p^j = L^j(D_p^j + T_p^j) + (1 - L^j)T_p^j \tag{5.4}$$

where D_p^j appears on both sides of eq. (5.4) because the retransmission of a lost packet might on the average surfer the same network delay D_p^j .

Therefore, we finally get

$$D_p^j = \frac{T_p^j}{1 - L^j} \tag{5.5}$$

The data transfer time of a request to a remote cloud and its response travelling back to the local cloud can be approximated using the corresponding network delay which is obtained from the ongoing measurements on the same connection:

$$D_n^j = (\frac{S}{M})D_p^j = \frac{ST_p^j}{M(1 - L^j)} \tag{5.6}$$

where M is the maximum packet size (bytes), and S is the total data size (bytes).

Upon the arrival of each job at TAP, the allocation decision is made using the simple greedy algorithm which selects the cloud offering the minimal response time.

$$\arg\min_j(a\frac{ST_p^j}{M(1 - L^j)} + bD_e^j) \tag{5.7}$$

5.2.2 Routing Overlay for Improving Data Transfer Performance

To improve data transfer performance on the connections between a local cloud and remote clouds, we use the routing overlay, “SMART”, which has been proposed in chapter3. It is a self-healing, self-optimizing and highly scalable routing overlay that accepts customized routing policies formulated by distributed applications according to their own needs.

The SMART overlay network is formed by software agents (denoted by “Proxy” in Figure 5.1) that are deployed at Virtual Machines (VM) in different sites. In our system the links between neighbouring overlay nodes use the conventional Internet protocol (IP), while multi-hop links between overlay nodes are dynamically updated based on Reinforcement Learning using Random Neural Networks. This approach only needs a limited monitoring effort by probing a few paths which have been observed to offer low packet transmission latency and yet achieves significant improvement in the end-to-end latency and asymptotically the same performance as the best path. Therefore, it can be widely deployed over a sizable population of routers.

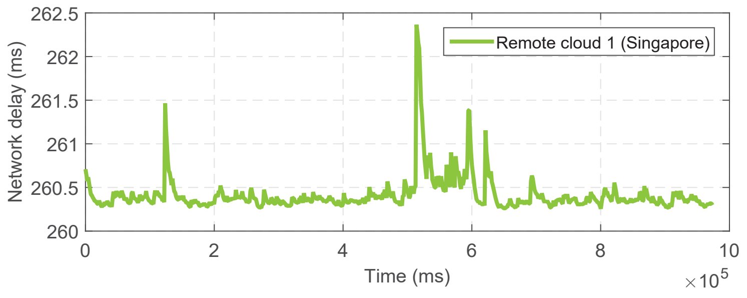
5.2.3 Experimental results

To validate our proposed system, we built an experimental system at the global intercontinental level which includes the local cloud in the test-bed of Imperial College London and the three remote clouds located in Ireland, Virginia and Singapore provided by Amazon AWS.

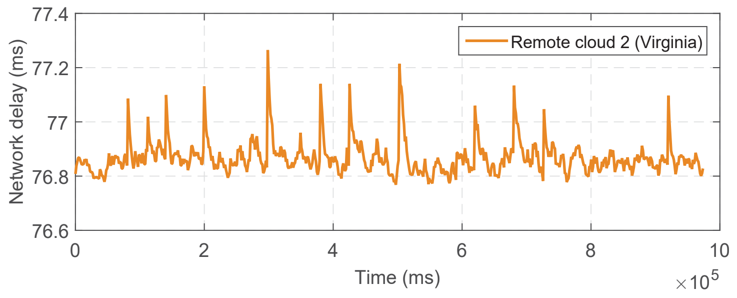
The web requests are originated using Httperf for retrieving a file of size 128K from the web servers, which generates I/O bound workload on the web servers. In the local cloud, the TAP is deployed for optimizing the web request allocation across the three web servers with distinct I/O capacity. In the remote clouds, there are other web servers deployed for load balancing. Between the local and remote clouds, the routing overlay–SMART–is used for routing the web requests and responses with the optimized network delay.

The measurement of the network delay on the connections to all the remote clouds is carried out every one second and the average response time for the Httperf requests inside each cloud is also reported every one second. As shown in Figure 5.2, the network connection to the cloud located in Ireland has the lowest network delay because this cloud is closer to the local cloud in London than the others, and the traffic on the connection appears to be low.

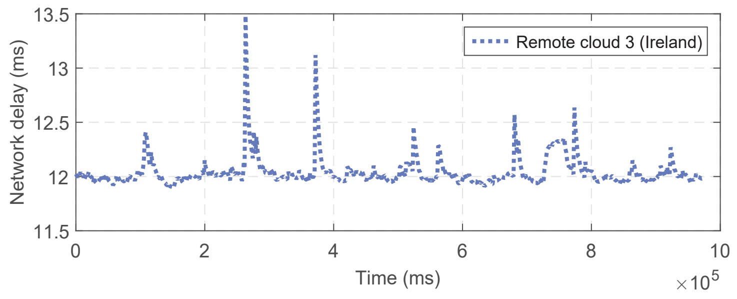
The web browser requests were originated from a client inside the local cloud at the rate of 0.5 requests per second. We varied the background workloads, which consume I/O capacity on the web servers in the local cloud over time in order to observe whether our TAP is able to adapt to the changing load conditions. Therefore, the local cloud is loaded lightly, modestly, heavily and finally lightly during 200 seconds for each of these successive conditions.



(a)



(b)



(c)

Figure 5.2: The weighted average network delay over time on the connections to the three remote clouds; it is derived from the measurement of the round-trip delay and loss via pinging.

As shown in Figure 5.3, the incoming web requests were dispatched to the local cloud when it was under light and modest load conditions. Our autonomic TAP learned the optimal request allocation based on the online measurement and directed the requests to the web server which provided the fastest response. As the workload in the local cloud increased to a certain high level which resulted in a response time that was significantly greater than that of one of the remote clouds (including network delays), the TAP selected the remote cloud for the subsequent web requests until it detected that the response time offered by the local cloud time had dropped significantly due to the offloading of workload from background tasks.

The improvement of network performance achieved using our overlay network is shown in Figures 5.4 and 5.5. We deployed our overlay network across Europe, Asia, North and South America, and

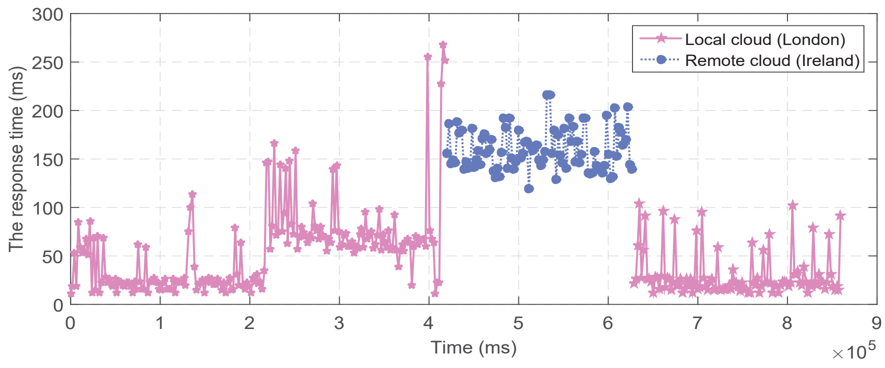


Figure 5.3: The measured response time from our experiment, measured for each web request as time elapses; the different colours represent the different clouds where the requests are allocated and processed.

Australia using 19 overlay nodes. The RTT observed between Moscow (Russia) and Dublin (Ireland) using the direct IP route was roughly 175 ms, whereas the RTT of the minimum latency path obtained with our overlay based adaptive routing was approximately 81.7 ms.

The results show that our proposed overlay routing algorithm learns the optimal path in terms of the latency very fast, and tracks it throughout the 5-day experiment. Similarly, the available throughput between Virginia (USA) and Tokyo (Japan) using the direct IP route was 10.85 Mbps on the average, whereas the average throughput of the optimal path turned out to be 40.2 Mbps.

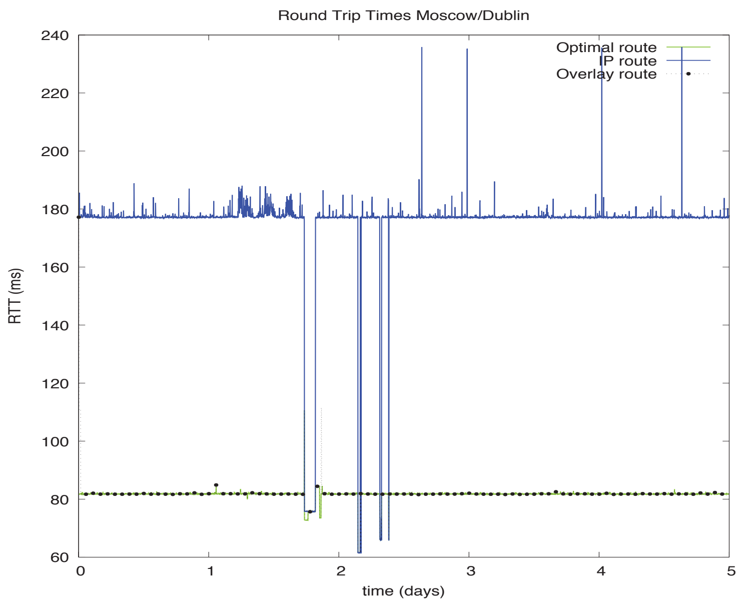


Figure 5.4: RTD in milliseconds measured for the Japan-Chile connection in an experiment lasting 5 successive days.

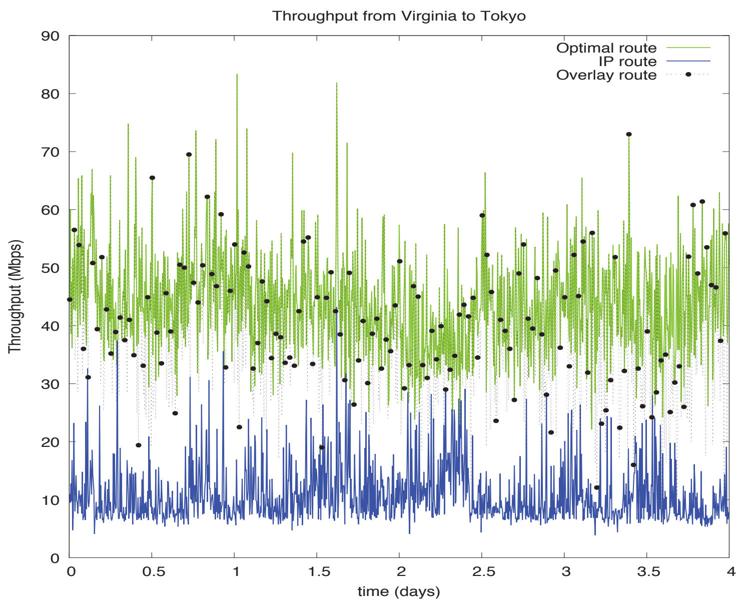


Figure 5.5: Throughput (Mbps) measured from Virginia (USA) to Sydney (Australia) over 4 consecutive days.

5.3 Summary

This chapter proposes a practical system for adaptive workload distribution across multiple clouds over wide area networks. It uses a Random Neural Network (RNN) based adaptive learning algorithm to optimize both the cloud delay within a cloud and data transfer latency between local and remote clouds in order to improve the responsiveness to user requests.

Internet-scale experiments conducted on a real large scale system at a global scale demonstrate the adaptiveness and effectiveness of our proposed system in dynamic environments. In future work, we will also take into account the energy consumption of jobs to take decisions for load distribution among local and remote Clouds.

Chapter 6

Conclusion and Future Work

In this Thesis we have addressed several related problems in Internet Routing and Task Allocation for Cloud Computing. Since the Cloud is accessed via the Internet, the performance issues of both systems are closely connected.

6.1 Conclusions

After the introduction of Chapter 1, **Chapter 2** addresses the QoS requirements of real-time traffic in networks which carry real-time as well as other background traffic. Real-time traffic arises through the needs of Voice as well as Multimedia, and we may expect that some of the Internet of Things (IoT) traffic will also have real-time constraints for sensing and industrial control applications. Thus we have presented an extension of the CPN routing protocol so that multiple QoS classes – including real-time – can inhabit all routers of the network. Detailed experiments were conducted for real-time traffic, as well as other background traffic, with two distinct QoS requirements: $Delay$ and $Jitter$ Minimisation for any source-destination pair within the CPN test-bed.

Surprisingly enough, measurement results for real-time traffic regarding average delay, jitter and loss have shown that when "Jitter" is specified as the QoS goal for both real-time and background traffic, better performance is achieved for all QoS metrics and all network load conditions. This seems to result from the useful effect of “Jitter” as a means to reduce route oscillations, also generally giving rise to less packet loss. However, we notice that long periods of path stability can also result in congestion with all traffic tending to use the same paths, when these paths do not oscillate and hence have low jitter: at that point, bursty effects of loss occur and CPN can mitigate for them by switching paths. Thus novel sensible routing schemes which distribute traffic over many paths in the network [Gel03] may be useful even though they may result in the other disadvantages of multi-path packet forwarding. Furthermore we observe that packet loss in some parts of the network will provoke delays for other packets in the re-sequencing output RTIP codec buffers, which in turn can provoke buffer overflow and further losses. Thus, through a detailed and careful measurement study in a well instrumented network test-bed, we presents a complex series of cause and effects that allow us to better understand and better design networks that need to support real-time traffic.

We pursue our work on routing in **Chapter 3** where we first observe experimentally that intercontinental IP routes are far from optimal with respect to QoS metrics such as delay and packet loss, and that they may also be subject to outages. We therefore develop a *Big Data and Machine Learning* approach to routing called SMART, to improve the overall QoS of Internet connections, while still exploiting the existing IP protocol with path variations offered by an overlay network. SMART uses an adaptive overlay system that creates a limited modification of IP routes resulting in much lower packet forwarding delays and loss rates.

The overlays we consider include few overlay hops, because the entrance and exit of traffic into overlay nodes can introduce further delay, and IP is seamlessly used for transporting packets between the overlay nodes. The overlay routing strategy is inspired from Cognitive Packet Network (CPN) routing, where paths are dynamically selected using a Random Neural Network (RNN) based adaptive learning algorithm that exploits smart search and probing in order to select the best paths. In this particular case, each connection explores a small number of alternate paths that are offered by the overlay network. However, all of this requires constant measurements between the overall nodes resulting in hundreds of thousands of data points being collected in a single day, as well as a fast (RNN based) machine learning algorithm.

The proposed system has been implemented and tested in an intercontinental overlay network that includes Europe, Asia, North and South America, and Australia, composed of 20 overlay nodes, and we have observed that with at no more than two overlay nodes in each connection, round trip packet delays are generally very close within a few percent, to the round trip delays observed with three or four overlay nodes per connection. We further observe that significant improvements can also be obtained when the RNN based adaptation uses no more than two alternate paths which emerge as the best, as a result of a wider search.

In **Chapter 4** we turn to Cloud computing which enables the accommodation of an increasing number of applications in shared infrastructures. Routing for the incoming jobs to different servers in the cloud has become a real challenge due to the heterogeneity in both workload and machine hardware and the changes of load conditions over time. Thus we design and investigate the adaptive dynamic allocation algorithms that take decisions based on on-line and up-to-date measurements, and make fast online decisions to achieve both desirable QoS levels and high resource utilization. The algorithms are incorporated into a Task Allocation Platform (TAP) that we implement as a practical system to accommodate the allocation algorithms and perform online measurement.

We consider both simple static allocations schemes, such as Round Robin, and a probabilistic allocation which distributes load evenly. We also study a model driven algorithm which uses queueing model based estimates of response time to select distinct allocation rates to different hosts. Two measurement driven adaptive on-line algorithms are also considered: an RNN based algorithm with Reinforcement Leaning, and the Sensible Algorithm that bring intelligence to bear from observations and make judicious allocation decisions.

We observe that the RNN-based approach limits the measurement overhead as it focuses on the performance of sub-systems which can offer better performance to the specified QoS request and explore, with a limited probability, the performance of the parts of the system that may not have been explored recently so that parts which may be able to offer better performance are not ignored. Other recent work [BWG16] uses the similar approach to constructing a routing overlay which achieved an substantial reduce in the end-to-end delay and packet loss for internet traffic.

Numerous experiments with different task profiles, and optimisation objectives were considered, with two different sets of host machines: one composed of hosts with similar processing speeds, and another one with hosts having different speeds due to distinct background loads at each host. We have observed [WG18] that the RNN-based algorithm is a fine-grained QoS-aware online task allocation algorithm. It benefits from its potential to detect the performance differences between the servers by leaning from the measurements, and directs tasks to the hosts which provide a better performance, outperforming sensible algorithm, Round Robin and equal probability task allocation which are commonly-practiced schemes. In the circumstances where there is greater diversity both in the types of jobs, the class of QoS criteria and the resources they request, the RNN-based algorithm outperforms the other schemes for multi-class task allocation due to its ability to be aware of the heterogeneity in both the workload and the machine hardware and adapt to changes in load conditions in the cloud over time.

In **Chapter 5**, we propose a practical approach for adaptive workload distribution across multiple clouds over wide area networks. It uses a Random Neural Network (RNN) based adaptive learning algorithm to optimize both the cloud delay within a cloud and data transfer latency between local and remote clouds in order to improve the responsiveness to user requests. Internet-scale experiments conducted at a global scale demonstrate the adaptiveness and effectiveness of our proposed system in dynamic environments.

6.2 Future Work

The research on overlay networks can be extended in several directions. An issue to be considered is that of reducing the amount of data that is stored, especially when measurements may have to be stored and exploited concurrently at multiple locations in the network because paths will typically share overlay nodes and IP network segments. One approach for consideration in future work is to resequence the data [BGP84] so as to drop data items before they enter into the learning algorithm if they have been superseded by fresher and more relevant items. Another important direction is to consider bandwidth optimisation or bandwidth guarantees, in addition to delay minimisation, as is done in recent work with CPN where applications require asymmetric QoS (delay for upstream and bandwidth for downstream) [GK14], as well as energy consumption aspects in the network as a whole [GM11b]. Another question of interest would be to consider the time and energy costs [Gel10] of finding users or other resources such as virtual machines or files, in a large overlay network, prior to setting up connections to the appropriate overlay node. The results we have obtained have essentially considered paths of at most two overlay hops. This may not be sufficient for some source to destination pairs, especially with complicated intercontinental patterns of connectivity: for instance Africa can be reached in several different ways, over the Mediterranean Sea, through the Middle East, or along its Atlantic Coast. Thus we may need to consider more extensive probing schemes, that use much more data collected at (say) one or two hour intervals, which may include probing of all overlay node pairs. Using such data we may be able to determine the shortest-delay paths between each source to destination pair, as a means to improve the effectiveness of the adaptive schemes that we propose in this chapter. Indeed, we believe that data driven observation and adaptive control of the large scale Internet is an important area of future work both for researchers and for network operators.

In future work we suggest the use of more sophisticated mathematical models such as diffusion approximations [Gel79] to build an on-line measurement and model driven allocation algorithm that exploiting measurements of the arrival and service statistics at each of the hosts in order to estimate the task allocation probabilities. Although we expect that such an approach will have its limits due to the increase of the data that it will need, it may offer a better predictor for more accurate task allocations, and especially it could be used to benchmark other approaches. We would also like to study the Cloud system we have described when a given set of hosts is used by multiple task allocation systems operating with heterogenous input streams (such as Web services, mobile services and compute intensive applications), to see which schemes are the most robust and resilient. One aspect we have not discussed is the order in which tasks are executed, for instance time-stamp order [BGP84]: although this may not be important in some applications, in others which are updated some global system state (such as a bank account), the order in which operations are carried out is critical, and tasks which are related by time-stamp order would have to be carried out in that order to avoid having to reprocess them if that order is violated. Another direction we wish to undertake is the study of the robustness of allocation schemes for Cloud services in the presence of network and service attacks [GL07] that are designed to disrupt normal operations. Another interesting direction for research will be to study how techniques that are similar to the ones we have developed in this chapter, may be exploited in the context of Grid Computing [BdF10].

Energy efficiency is an increasingly significant issue in ICT both due to the CO2 impact of electrical energy, and especially due to the economic cost of electricity in data centres, servers and network routers [GC15]. Thus reducing the energy consumption of servers while maintaining the best possible QoS for services is of crucial interest for cloud providers. Thus we suggest that future work should address the design of task allocation strategies that optimise the two contradictory criteria of reducing the energy consumption per job while maintaining the best possible job response time.

Servers in data centres have a load dependent power and hence energy consumption characteristic [GL12], and while the simplest energy saving strategy is to concentrate computation on a certain amount of compute resources so that the under-utilized machines can be hibernated or turned off, such a simplistic approach can lead to high response times and "service level agreement violations". Furthermore the power consumption of highly loaded servers is also high. Thus more sophisticated techniques should be used, for instance by dynamically optimising composite goal functions (6.1) such as:

$$C_{job} = aW_{job} + bJ_{job}, \tag{6.1}$$

which include both job response time W_{job} and energy consumed per job J_{job}. Here where a and b are the relative importance being placed on the job response time and energy consumption per job, respectively. This can allow an extension to algorithms for adaptively distributing workloads across available servers in order to achieve a better trade-off between the two requirement. Such approaches have been considered in [GL12, GL13], not for task allocation as we suggest, but as means of determining optimum load level in a system. Thus significant work remains to be done to move such considerations into the increasingly important area of task allocation in the Cloud and through the Internet.

Bibliography

[ABKM01] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP '01, pages 131–145, New York, NY, USA, 2001. ACM.

[ama] Amazon ec2. `http://aws.amazon.com/ec2/`.

[AP06] G. Ramachandran A.Morton, L.Ciavattone and J. Perser. Rfc4737: Packet reordering metrics. November 2006.

[AS15] Yousef Abuseta and Khaled Swesi. Design patterns for self-adaptive systems engineering. *International Journal of Software Engineering & Applications (IJSEA)*, 6(4):11–26, July 2015.

[BdF10] D.M. Batista and N.L.S. da Fonseca. A survey of self-adaptive grids. *Communications Magazine, IEEE*, 48(7):94–100, July 2010.

[BF99] N. Bhatti and R. Friedrich. Web server support for tiered services. *Network, IEEE*, 13(5):64–71, Sep 1999.

[BGP84] Franois Baccelli, Erol Gelenbe, and Brigitte Plateau. An end-to-end approach to the resequencing problem. *J. ACM*, 31(3):474–485, June 1984.

[BKC13] Leszek Borzemski and Anna Kaminska-Chuchmala. Distributed web systems performance forecasting using turning bands method. *IEEE Transactions on Industrial Informatics*, 9(1):254–261, 2013.

[BM13] M. Baldi and G. Marchetto. Time-driven priority router implementation: Analysis and experiments. *Computers, IEEE Transactions on*, 62(5):1017–1030, May 2013.

[BMMV08] M. Baldi, J.C. De Martin, E. Masala, and A. Vesco. Quality-oriented video transmission with pipeline forwarding. *Broadcasting, IEEE Transactions on*, 54(3):542–556, Sept 2008.

[BMP03] M. Beck, T. Moore, and J.S. Plank. An end-to-end approach to globally scalable programmable networking. In ACM Press, editor, *in Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, 2003.

[Buy13] R. Buyya. Introduction to the ieee transactions on cloud computing. *Cloud Computing, IEEE Transactions on*, 1(1):3–21, Jan 2013.

[BWG16] Olivier Brun, Lan Wang, and Erol Gelenbe. Big data for autonomic intercontinental overlays. *IEEE Journal on Selected Areas in Communications*, 34(3):575–583, March 2016.

[CC10] Sergio R. M. Canovas and Carlos E. Cugnasca. Implementation of a control loop experiment in a network-based control system with lonworks technology and ip networks. *IEEE Transactions on Industrial Electronics*, 57(11):3857–3867, 2010.

[cis] Quality of service for voice over ip. `https://www.cisco.com/c/en/us/td/docs/ios/solutions_docs/qos_solutions/QoSVoIP/QoSVoIP.pdf`.

[clo] cloud bursting. `http://searchcloudcomputing.techtarget.com/definition/cloud-bursting`.

[CMA$^+$09] Tommaso Cucinotta, Antonio Mancina, Gaetano Anastasi, Giuseppe Lipari, Leonardo Mangeruca, Roberto Checcozzo, and Fulvio Rusina. A real-time service-oriented architecture for industrial automation. *IEEE Transactions on Industrial Informatics*, 5(3):267–277, 2009.

[CZ09] WeiNeng Chen and Jun Zhang. An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 39(1):29–43, Jan 2009.

[DC02] C. Demichelis and P. Chimento. Ip packet delay variation metric for ip performance metrics (ippm). In *https://www.ietf.org/rfc/rfc3393.txt*. The Internet Society, November 2002.

[DCGN01] Mike Dahlin, Bharat Chandra, Let Gao, and Amol Nayate. End-to-end wan service availability. In *In Proc. 3rd USITS*, pages 97–108, 2001.

[DDF$^+$06] Simon Dobson, Spyros G. Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt, and Franco Zambonelli. A survey of autonomic communications. *TAAS*, 1(2):223–259, 2006.

[DFG10] N. Dimakis, A. Filippoupolitis, and E Gelenbe. Distributed building evacuation simulator for smart emergency management. *The Computer Journal*, 53(9):1384–1400, 2010.

[DH11] Hai Dong and Farookh Khadeer Hussain. Focused crawling for automatic service discovery, annotation, and classification in industrial digital ecosystems. *IEEE Transactions on Industrial Electronics*, 58(6):2106–2116, 2011.

[DHC11] Hai Dong, Farookh Khadeer Hussain, and Elizabeth Chang. A service search engine for the industrial digital ecosystems. *IEEE Transactions on Industrial Electronics*, 58(6):2183–2196, 2011.

[DK13] Christina Delimitrou and Christos Kozyrakis. Qos-aware scheduling in heterogeneous datacenters with paragon. *ACM Trans. Comput. Syst.*, 31(4):1–34, December 2013.

[DWC10] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: issues and challenges. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 27–33. Ieee, 2010.

[FBR$^+$04] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der Merwe. The case for separating routing from routers. In ACM Press, editor, *Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, 2004.

[FCC13] FCC. 2013 measuring broadband america. In *Office of Engineering and Technology and Consumer and Governmental Affairs Bureau*, Washington, DC, USA, February 2013. Federal Communications Commission.

[FJG13] Max Felser, Jürgen Jasperneite, and Piotr Gaj. Guest editorial special section on distributed computer systems in industry. *IEEE Transactions Industrial Informatics*, 9(1):181, 2013.

[GC15] Erol Gelenbe and Yves Caseau. The impact of information technology on energy consumption and carbon emissions. *Ubiquity*, 2015:1–15, June 2015.

[Gel] Erol Gelenbe. Overhead related to CPN routing in packet networks. *to appear*.

[Gel79] Erol Gelenbe. Probabilistic models of computer systems. *Acta Informatica*, 12:285–303, 1979.

[Gel89a] E. Gelenbe. Réseaux stochastiques ouverts avec clients négatifs et positifs et réseaux neuronaux. *Comptes-Rendus Acad. Sci. Paris*, 309, Série II:979–982, 1989.

[Gel89b] Erol Gelenbe. Random neural networks with negative and positive signals and product form solution. *Neural Computation*, 1(4):502–510, Dec 1989.

[Gel90] Erol Gelenbe. Stability of the random neural network model. *Neural Computation*, 2(2):239–247, 1990.

[Gel93] Erol Gelenbe. Learning in the recurrent random neural network. *Neural Computation*, 5:154–164, 1993.

[Gel00] E. Gelenbe. The first decade of g-networks. *European Journal of Operational Research*, 126(2):231–232, 2000.

[Gel03] Erol Gelenbe. Sensible decisions based on qos. *Computational management science*, 1(1):1–14, 2003.

[Gel04] E. Gelenbe. Cognitive packet network. In *U.S. Patent 6,804,201*, October 2004.

[Gel07] Michael Gellman. *QoS Routing for Real-time Traffic*. PhD thesis, Imperial College London, 2007.

[Gel09] Erol Gelenbe. Steps toward self-aware networks. *Commun. ACM*, 52(7):66–75, 2009.

[Gel10] Erol Gelenbe. Search in unknown random environments. *Physical Review E*, 82(6):061112, 2010.

[GF99] E. Gelenbe and J.M. Fourneau. Random neural networks with multiple classes of signals. *Neural Computation*, 11(4):953–963, 1999.

[GG07] Erol Gelenbe and Michael Gellman. Can routing oscillations be good? the benefits of route-switching in self-aware networks. In *15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2007), October 24-26, 2007, Istanbul, Turkey*, pages 343–352, 2007.

[GGL$^+$04] E. Gelenbe, M. Gellman, R. Lent, P. Liu, and Pu Su. Autonomous smart routing for network QoS. In *Proceedings of the First International Conference on Autonomic Computing*, pages 232–239, New York, NY, May 2004.

[GHK05] Erol Gelenbe, Khaled Hussain, and Varol Kaptan. Simulating autonomous agents in augmented reality. *Journal of Systems and Software*, 74(3):255–268, 2005.

[GJF13] Piotr Gaj, Jürgen Jasperneite, and Max Felser. Computer communication within industrial distributed environment - a survey. *IEEE Trans. Industrial Informatics*, 9(1):182–189, 2013.

[GK14] Erol Gelenbe and Zarina Kazhmaganbetova. Cognitive packet network for bilateral asymmetric connections. *IEEE Trans. Industrial Informatics*, 10(3):1717–1725, 2014.

[GL07] Erol Gelenbe and George Loukas. A self-aware approach to denial of service defence. *Computer Networks*, 51(5):1299–1314, April 2007.

[GL12] Erol Gelenbe and Ricardo Lent. Trade-offs between energy and quality of service. In *Sustainable Internet and ICT for Sustainability (SustainIT), 2012*, pages 1–5. IEEE, 2012.

[GL13] Erol Gelenbe and Ricardo Lent. Optimising server energy consumption and response time. *Theoretical and Applied Informatics*, (4):257–270, Jan 2013.

[GLD12] Erol Gelenbe, Ricardo Lent, and Markos Douratsos. Choosing a local or remote cloud. In *Second Symposium on Network Cloud Computing and Applications, NCCA 2012, London, United Kingdom, December 3-4, 2012*, pages 25–30. IEEE Computer Society, 2012.

[GLL06] Erol Gelenbe, Peixiang Liu, and Jeremy Laine. Genetic algorithms for route discovery. *IEEE Transactions on Systems, Man and Cybernetics B*, 36(6):1247–1254, Dec. 2006.

[GLMX00] Erol Gelenbe, R. Lent, A. Montuori, and Z. Xu. Towards networks with cognitive packets. In *Proc. 8th Int. Symp. Modeling, Analysis and Simulation of Computer and Telecommunication Systems (IEEE MASCOTS), San Francisco, CA, USA*, pages pp 3–12, August 29-September 1 2000.

[GLMX02] E. Gelenbe, R. Lent, A. Montuori, and Z. Xu. Cognitive packet networks: QoS and performance. In *Proceedings of the IEEE MASCOTS Conference*, pages 3–12, Ft. Worth, TX, October 2002. Opening Keynote Paper.

[GLN04] E. Gelenbe, R. Lent, and A. Nunez. Self-aware networks and QoS. *Proceedings of the IEEE*, 92(9):1478–1489, September 2004.

[GLX01] E. Gelenbe, Ricardo Lent, and Zhiguang Xu. Design and performance of cognitive packet networks. *Performance Evaluation*, 46(2,3):155–176, 10 2001.

[GM76] Erol Gelenbe and Richard R. Muntz. Probabilistic models of computer systems: Part i (exact results). *Acta Informatica*, 7(1):35–60, 1976.

[GM10] Erol Gelenbe and Isi Mitrani. *Analysis and Synthesis of Computer Systems*. World Scientific, Imperial College Press, 2010.

[GM11a] Erol Gelenbe and Toktam Mahmoodi. Energy-aware routing in the cognitive packet network. *ENERGY*, pages 7–12, 2011.

[GM11b] Erol Gelenbe and Christina Morfopoulou. A framework for energy-aware routing in packet networks. *The Computer Journal*, 54(6):850–859, 2011.

[GMG$^+$04] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall. Improving the reliability of internet paths with one-hop source routing. In *In Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, 2004.

[goo] Google compute engine. `https://cloud.google.com/compute/`.

[GPS$^+$00] Alex Galis, Bernhard Plattner, Jonathan M. Smith, Spyros G. Denazis, Eckhard Moeller, Hui Guo, Cornel Klein, Joan Serrat, Jan Laarhuis, George T. Karetsos, and Chris Todd. A flexible IP active networks architecture. In Hiroshi Yasuda, editor, *Active Networks, Second International Working Conference, IWAN 2000, Tokyo, Japan, October 16-18, 2000, Proceedings*, volume 1942 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2000.

[GT08] Erol Gelenbe and Stelios Timotheou. Random neural networks with synchronized interactions. *Neural Computation*, 20(9):2308–2324, 2008.

[GW12] Erol Gelenbe and Fang-Jing Wu. Large scale simulation for human evacuation and rescue. *Comput. Math. Appl.*, 64(12):3869–3880, December 2012.

[GW15] Erol Gelenbe and Lan Wang. Tap: A task allocation platform for the EU FP7 PANACEA project. In *The proceedings of the EU projects track*, September 2015.

[Hal00] Uğur Halıcı. Reinforcement learning with internal expectation for the random neural network. *European Journal of Operational Research*, 126(2):288–307, 2000.

[HAR94] E.S.H Hou, N. Ansari, and Hong Ren. A genetic algorithm for multiprocessor scheduling. *Parallel and Distributed Systems, IEEE Transactions on*, 5(2):113–120, Feb 1994.

[HJ04] J. Han and F. Jahanian. Impact of path diversity on multi-homed and overlay networks. In *In Proceedings of IEEE International Conference on Dependable Systems and Networks*, 2004.

[HL04] S.-Y. Hu and G.-M. Liao. Scalable peer-to-peer networked virtual environment. In *In NetGames'04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 129–133, New York, NY, USA, 2004. ACM Press.

[IOY$^+$11] A. Iosup, S. Ostermann, M.N. Yigitbasi, R. Prodan, T. Fahringer, and D. H. J. Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *Parallel and Distributed Systems, IEEE Transactions on*, 22(6):931–945, June 2011.

[JVK05] J.Soldatos, E. Vayias, and G. Kormentzas. On the building blocks of quality of service in heterogeneous ip networks. *Commun. Surveys Tuts.*, 7(1):69–88, January 2005.

[KA96] Yu-Kwong Kwok and I. Ahmad. Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. *Parallel and Distributed Systems, IEEE Transactions on*, 7(5):506–521, May 1996.

[KC14] Slawomir Kuklinski and Prosper Chemouil. Network management challenges in software-defined networks. *IEICE Transactions*, 97-B(1):2–9, 2014.

[KK92] Chonggun Kim and Hisao Kameda. An algorithm for optimal static load balancing in distributed computer systems. *IEEE Transactions on Computers*, 41(3):381–384, 1992.

[KLKZ11] Hisao Kameda, Jie Li, Chonggun Kim, and Yongbing Zhang. *Optimal Load Balancing in Distributed Computer Systems*. Springer, 2011.

[LABJ00] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. Delayed internet routing convergence. *SIGCOMM Comput. Commun. Rev.*, 30(4):175–187, August 2000.

[LDP01] Lars-Ake Larzon, Mikael Degermark, and Stephen Pink. Requirements on the tcp/ip protocol stack for real-time communication in wireless environments. In *Quality of Service in Multiservice IP Networks, Lecture Notes in Computer Science Volume 1989*, pages 273–283. Springer, 2001.

[LL11] Cui Lin and Shiyong Lu. Scheduling scientific workflows elastically for cloud computing. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 746–747, July 2011.

[LMJ98] C. Labovitz, R. Malan, and F. Jahanian. Internet routing instability. *IEEE/ACM Transactions on Networking*, 6(5):515–526, 1998.

[MG09] P. Mell and T. Grance. The nist definition of cloud computing. *NIST Special Publication 800-145*, Aug 2009.

[MGTX14] I. Solis Moreno, P. Garraghan, P. Townend, and J. Xu. Analysis, modeling and simulation of workload patterns in a large-scale utility cloud. *Cloud Computing, IEEE Transactions on*, PP(99):1–1, 2014.

[mic] Windows azure. `http://www.windowsazure.com/`.

[MQCdM08] E. Masala, D. Quaglia, and J. Carlos de Martin. Variable time scale multimedia streaming over ip networks. *Multimedia, IEEE Transactions on*, 10(8):1657–1670, Dec 2008.

[MSG$^+$07] Bertrand Mathieu, Meng Song, Alex Galis, Lawrence Cheng, Kerry Jean, Roel Ocampo, Marcus Brunner, Martin Stiemerling, and Marco Cassini. Self-management of context-aware overlay ambient networks. In *Integrated Network*

Management, IM 2007. 10th IFIP/IEEE International Symposium on Integrated Network Management, Munich, Germany, 21-25 May 2007, pages 749–752, 2007.

[NPB06] A. Nakao, L. Peterson, and A. Bavier. Scalable routing overlay networks. *SIGOPS Oper. Syst. Rev.*, 40(1):49–61, 2006.

[Pax96] V. Paxson. End-to-end routing behavior in the internet. In *in Proc. ACM SIGCOMM'96*, pages 25–38, Stanford, CA, USA, August 1996.

[PLGB10] S. Pandey, Wu Linlin, S.M. Guru, and R. Buyya. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 400–407, April 2010.

[PSL15] B. Palanisamy, A. Singh, and L. Liu. Cost-effective resource provisioning for mapreduce in a cloud. *IEEE Transactions on Parallel and Distributed Systems*, 26(5):1265–1279, May 2015.

[PST04] L. Peterson, S. Shenker, and J. Turner. Overcoming the internet impasse through virtualization. In *in Proceedings of the 3rd ACM Workshop on Hot Topics in Networks (HotNets-III)*, November 2004.

[PSZ$^+$07] Padala Pradeep, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, and Kenneth Salem. Adaptive control of virtualized resources in utility computing environments. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 289–302, New York, NY, USA, 2007. ACM.

[RAS05] Lopamudra Roychoudhuri and Ehab S. Al-Shaer. Real-time packet loss prediction based on end-to-end delay variation. *Network and Service Management, IEEE Transactions on*, 2(1):29–38, Nov 2005.

[RCL09] Bhaskar Prasad Rimal, Eunmi Choi, and Ian Lumb. A taxonomy and survey of cloud computing systems. In *INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on*, pages 44–51. IEEE, 2009.

[Sak10] Georgia Sakellari. The cognitive packet network: A survey. *The Computer Journal*, 53(3):268–279, 2010.

[SB98] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[SBAMP11] Javier Silvestre-Blanes, Luís Almeida, Ricardo Marau, and Paulo Pedreiras. Online qos management for multimedia real-time transmission in industrial networks. *IEEE Transactions on Industrial Electronics*, 58(3):1061–1071, 2011.

[SCH$^+$99] Stefan Savage, Andy Collins, Eric Hoffman, John Snell, and Thomas Anderson. The end-to-end effects of internet path selection. *SIGCOMM Comput. Commun. Rev.*, 29(4):289–299, August 1999.

[SG10] Georgia Sakellari and Erol Gelenbe. Demonstrating cognitive packet network resilience to worm attacks. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 636–638, 2010.

[SL93] G.C. Sih and E.A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *Parallel and Distributed Systems, IEEE Transactions on*, 4(2):175–187, Feb 1993.

[SPA12] Rui Santos, Paulo Pedreiras, and Luís Almeida. Demonstrating an enhanced ethernet switch supporting video sensing with dynamic qos. In *DCOSS*, pages 293–294, 2012.

[SS04] Stefan Soucek and Thilo Sauter. Quality of service concerns in ip-based control systems. *IEEE Transactions on Industrial Electronics*, 51(6):1249–1258, 2004.

[SSS13] Ying Song, YuZhong Sun, and Weisong Shi. A two-tiered on-demand resource allocation mechanism for vm-based data centers. *Services Computing, IEEE Transactions on*, 6(1):116–129, First 2013.

[TCAXEVTR11] H. Toral-Cruz, J. Argaez-Xool, L. Estrada-Vargas, and D. Torres-Roman. An introduction to voip: End-to-end elements and qos parameters. *InTech*, pages 79–94, 2011.

[TF03] A. Clark T. Friedman, R. Caceres. Rfc3611: Rtp control protocol extended reports (rtcp xr). November 2003.

[THyW02] Haluk Topcuouglu, Salim Hariri, and Min you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.*, 13(3):260–274, March 2002.

[TT85] Asser N Tantawi and Don Towsley. Optimal static load balancing in distributed computer systems. *Journal of the ACM (JACM)*, 32(2):445–465, 1985.

[TWEF03] J. Touch, Y. Wang, L. Eggert, and G. Finn. A virtual internet architecture. Technical Report ISI-TR-2003-570, ISI, March 2003.

[TZZ$^+$11] Wenhong Tian, Yong Zhao, Yuanliang Zhong, Minxian Xu, and Chen Jing. A dynamic and integrated load-balancing scheduling algorithm for cloud datacenters. In *Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on*, pages 311–315. IEEE, 2011.

[WBG16] Lan Wang, Olivier Brun, and Erol Gelenbe. Adaptive workload distribution for local and remote clouds. In *Proceedings of IEEE International Conference on SYSTEMS, MAN, AND CYBERNETICS (SMC2016)*. Budapest, Hungary, October 2016.

[WG15a] L. Wang and E. Gelenbe. Experiments with smart workload allocation to cloud servers. In *2015 IEEE Fourth Symposium on Network Cloud Computing and Applications (NCCA)*, pages 31–35, June 2015.

[WG15b] Lan Wang and Erol Gelenbe. Demonstrating voice over an autonomic network. In *2015 IEEE International Conference on Autonomic Computing, Grenoble, France, July 7-10, 2015*, pages 139–140, 2015.

[WG16] Lan Wang and Erol Gelenbe. Real-time traffic over the cognitive packet network. In *Proceedings of 23rd International Science Conference on Computer Networks (CN2016). "Official Distinction Award" at the 23th Computer Network Conference.*, Poland, June 2016.

[WG18] Lan Wang and Erol Gelenbe. Adaptive dispatching of tasks in the cloud. *IEEE Transactions on Cloud Computing*, 6(1):33–45, Jan 2018.

[Wol01] James J. Wolff. Dynamic load balancing of a network of client and server computers, February 6, 2001. U. S. Patent 6,185,601.

[ZBF10] Sergey Zhuravlev, Sergey Blagodurov, and Alexandra Fedorova. Addressing shared resource contention in multicore processors via scheduling. *SIGPLAN Not.*, 45(3):129–142, March 2010.

[ZG11] S. Zaman and D. Grosu. A combinatorial auction-based dynamic vm provisioning and allocation in clouds. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 107–114, Nov 2011.

[ZQQ11] Xiaomin Zhu, Xiao Qin, and Meikang Qiu. Qos-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters. *Computers, IEEE Transactions on*, 60(6):800–812, June 2011.

[ZWL$^+$13] Jianfeng Zhan, Lei Wang, Xiaona Li, Weisong Shi, Chuliang Weng, Wenyao Zhang, and Xiutao Zang. Cost-aware cooperative resource provisioning for heterogeneous workloads in data centers. *Computers, IEEE Transactions on*, 62(11):2155–2168, Nov 2013.

[ZZ10] Zehua Zhang and Xuejie Zhang. A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation. In *Industrial Mechatronics and Automation (ICIMA), 2010 2nd International Conference on*, volume 2, pages 240–243. IEEE, 2010.

[ZZBH14] Q. Zhang, M.F. Zhani, R. Boutaba, and J.L. Hellerstein. Dynamic heterogeneity-aware resource provisioning in the cloud. *Cloud Computing, IEEE Transactions on*, 2(1):14–28, March 2014.