INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI[®]

NOTE TO USERS

The original manuscript received by UMI contains pages with poor print. Pages were microfilmed as received.

108

This reproduction is the best copy available.

UMI

The Power 2 Ring Loading Problem

by
Ronald F. Arnold

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Computer Science

Ottawa-Carleton Institute for Computer Science

School of Computer Science
Carleton University
Ottawa, Ontario
2001

© Copyright
2001. Ronald F. Arnold

**National Library
of Canada**

**Acquisitions and
Bibliographic Services**

**395 Wellington Street
Ottawa ON K1A 0N4
Canada**

**Bibliothèque nationale
du Canada**

**Acquisitions et
services bibliographiques**

**395, rue Wellington
Ottawa ON K1A 0N4
Canada**

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-72078-0

Canada

The undersigned hereby recommend to
the Faculty of
Graduate Studies and Research
acceptance of the thesis

The Power 2 Ring Loading Problem

submitted by

Ronald F. Arnold

(Thesis Supervisor)

(Thesis Co-supervisor)

(Director, School of Computer Science)

Carleton University
October 2001

Abstract

Optical fiber has the potential for vast data transmission rates in the order of terabits per second. However, the cost and technological development of the optical components that connect optical fiber into data networks limit the availability of optical capacity. This limitation requires the ability to route optical connections to maximize cost effective utilization of available optical bandwidth.

The optical ring is currently the preferred optical network configuration. The Synchronous Optical NETWORK (SONET) is the technological standard for optical networks in the United States. The congestion of an optical network is the optical load on the maximum load link. For the cost-effective design of SONET ring networks, a significant issue is to route a set of requests for optical connections to minimize the optical congestion. This problem is referred to as the Ring Loading Problem (RLP). Optical ring network designs are selected based on their ability to minimize network congestion.

RLP is NP-complete. Moreover, network-planning software requires large numbers of iterations of ring loading software modules to determine effective network designs. Therefore, a requirement for routing algorithms is that they be computationally fast. There is a tradeoff between algorithmic precision and computational effectiveness. Therefore, this thesis defines the Power 2 Ring Loading Problem (RLP2) as a subproblem of RLP. RLP2 limits all connection request demands to integer powers of 2. The research contained herein studies RLP2 by finding solutions to two RLP2 subproblems, which limit the shortest path for connection requests to a maximum of 1 and 2 optical links respectively. The solutions to these subproblems are intended to provide insight into possible solutions or approximations for the general form of RLP2.

Acknowledgements

To my supervisors. Professors Evangelos Kranakis and Danny Krizanc. Thank you for your guidance, support, and for helping me to achieve my goal.

To my family, for all their love and understanding. They are:

my Mother, Bernadette Arnold
my Father, Frank Arnold
and my sisters, Paula and Sheila Arnold

And, to my friends who gave me their support.

Ronald F. Arnold

Carleton University
Ottawa, Ontario, Canada
October 2001

Contents

Abstract	iii
Acknowledgements	iv
Table of Contents	v
List of Figures	viii
List of Tables	ix
List of Abbreviations	x
1 Introduction	1
1.1 Motivation	2
1.2 The Problem	3
1.3 Background	4
1.4 The Power 2 Demand Ring Loading Problem	9
1.5 Summary of Results	11
1.6 Overview	12
1.7 Outline of Thesis	16
2 Preliminaries	18
2.1 Introduction	18
2.2 Routings of Q on $\mathcal{R}$	19
2.3 Independent and Interdependent Link Pairs	22
2.4 Routing zones of $\mathcal{R}$	24
2.5 Independent and Interdependent Routing Paths	26
2.6 Maximal Shortest Path Interval	28
2.7 Conclusion	31

3	RLP2[k] Problem Solving Principles	32
3.1	Introduction	32
3.2	The RLP2[k] Bin Packing Problem	34
3.3	Routing Set Selection	37
3.4	Minimum Independent Congestion	40
3.5	Alpha Routing Set Bandwidth Limits	43
3.6	Alternate Routing Sets	44
3.7	Conclusion	52
4	RLP2[1]	53
4.1	Introduction	53
4.2	The RLP2[1] Bin Packing Problem	53
4.3	RLOAD2[1] Overview	55
4.4	Proof of Correctness	56
4.5	Computational Complexity	57
4.6	RLOAD2[1]	58
4.7	Conclusion	66
5	Interfered Requests	67
5.1	Introduction	67
5.2	The Interdependent Zone Z_F	69
5.3	Interfered Requests	70
5.4	Transfers	71
5.5	MSI Capacity Transfers	72
5.6	PI Capacity Transfers	75
5.7	Repartition of the Eligible Set	80
5.8	Conclusion	85
6	RLP2[2]	86
6.1	Introduction	86
6.2	The Eligible Set - Terminal Sets	87
6.3	The Feasible Shortest Path Interval Constraint	89
6.4	PI Transfers	90
6.5	MSI Transfers	92
6.6	Beta Transfers	92
6.7	Additional RLP2[2] Preliminaries	95
6.8	The RLP2[2] Partition Algorithm. RLOAD2-PART[2]	98
6.9	RLOAD2-BPACK[2]	103

6.10 RLOAD2[2]	117
6.11 RLP2[2] Computational Complexity	137
6.12 Conclusion	140
7 Conclusion and Open Problems	141
A Notation	144
Index	147
Bibliography	151

List of Figures

1.1	Routing Zones	13
2.1	Link Pair $\{x, y\}$ Congestion	24
2.2	Path Independent Routing Sets	28
2.3	Path Interdependent Routing Sets	29
2.4	Maximal Shortest Path Intervals	31
3.1	Minimum Independent Congestion	41
3.2	Non-Minimum Independent Congestion	41
3.3	Independent Bandwidth Routing Bounds	44
3.4	Reduction Set $\tilde{R}$	46
3.5	Alternate Routing Set $\tilde{R}_y$	47
3.6	Routing $\tilde{R}'$	49
3.7	Routing $\tilde{R}'_y$	49
5.1	MSI Transfer - Routing $\tilde{R}'$	74
5.2	MSI Transfer - Routing $\tilde{R}'_y$	74
5.3	PI Transfer - Routing $\tilde{R}'$	76
5.4	PI Transfer - Routing $\tilde{R}'_y$	76
5.5	PI Transfer - Common Routing Set	78
5.6	PI Transfer - Interference on Common Request	80
5.7	RLP2[2]-BPP - Initial Routing	81
5.8	RLP2[2]-BPP - Partition 1	81
5.9	RLP2[2]-BPP - Partition 2	82
5.10	Beta Transfer	84
6.1	Terminal Set Zones	88
6.2	RLOAD2-BPACK[2] Beta Tree	104

List of Tables

3.1	Definitions - Alternate Routing Sets	45
4.1	RLOAD2[1] Input/Output Parameters	58
4.2	RLOAD2[1] Global Data	58
4.3	RLOAD2[1] Alpha Tree Node	59
6.1	RLOAD2[2] Beta Tree Node	107
6.2	RLOAD2[2] Beta Tree Node Continued	108
6.3	RLOAD2[2] Input/Output Parameters	118
6.4	RLOAD2[2] Global Data	119
6.5	RLOAD2[2] Alpha Tree Node	120

List of Abbreviations

ADM	add drop multiplexor
DCS	digital cross connect system
DRLP	Directed Ring Loading problem
EDP	Edge-Disjoint Paths problem
ILP	Integer Linear program
IMF	Integer Multicommodity Flow problem
IMF1	Integer Multicommodity Flow problem, maximum demand = 1
LMF	Linear Multicommodity Flow problem
LP	linear program
MPCP	Minimum Path Colouring problem
MSI	maximal shortest path interval
PI	path interdependent
RLP	Ring Loading problem
RLP2	Power 2 Demand Ring Loading problem
RLP2[k]	Power 2 Demand k Shortest Path Ring Loading problem
RLP2[k]-BPP	RLP2[k] Bin Packing problem
RWA	routing and wavelength assignment
SHD	synchronous digital hierarchy
SONET	synchronous optical network
SSP	subset sum problem
WDM	wavelength division multiplexing

Chapter 1

Introduction

Optical fiber networks using Wave Division Multiplexed (WDM) techniques are capable of transmission of vast bitrates of the order of terabits per second. On a WDM network each connection must be assigned a path and a wavelength with the restriction that the paths for any two connections cannot share an optical link on the same wavelength. The path and wavelength assignment is the Routing and Wavelength Assignment (RWA) function.

Without wavelength converters, an optical signal must pass from its source to its destination on the same wavelength. This restriction is the *wavelength continuity* constraint. With wavelength converters the optical signal may be converted to a different wavelength at the optical nodes.

Essentially, RWA can be decomposed into two subproblems, a connection routing or path assignment problem, and a wavelength assignment problem[2]. Without wavelength converters these RWA subproblems can be modeled as a path assignment and colouring problem, where no two paths sharing a common link are assigned the same colour. Rabani[19] names this problem the Minimum Path Colouring problem (MPCP). For point-to-point requests, an instance of MPCP specifies a graph G and a set of m terminal pairs $(i_1, j_1), (i_2, j_2), \dots, (i_m, j_m)$. A solution to this instance gives n paths with path x connecting (i_x, j_x) and a colour for each path. The wavelength assignment subproblem of MPCP reduces to the node colouring problem on a graph which is NP-complete[10],[2]. The objective function in the optimization version is to minimize the number of colours used. Rabani has found that, using a reduction from the Disjoint Paths problem[10], MPCP is NP-hard. With wavelength converters RWA is reduced to the assignment of a set of paths to the connections, so that the maximum number of paths assigned to

any one optical link is minimized. This latter value is the *congestion* of the network.

The Synchronous Optical NETWORK (SONET) is the dominant technological standard in the US[24]. These networks carry signals at the rate of OC-1 (51.48 Mbps) to OC-48 (2.48832 Gbps). As of 1999, SONET rings of size OC-192 were anticipated[24]. In Europe, a similar standard to SONET, known as Synchronous Digital Hierarchy (SDH), is in effect. SONET networks consist of Add Drop Multiplexors (ADMs) or Digital Cross Connect Systems (DCSs), which are interconnected by point-to-point optical links. The ADMs are connected to electronic equipment located at the optical nodes. The role of the ADM is to multiplex, switch and demultiplex the optical signals to which it interfaces. DCSs perform a similar function to the ADMs.

The symmetry of the optical ring architecture simplifies the RWA problem. Optical node symmetry allows each node to play an identical role within the network with the same optical equipment. SONET rings offer enhanced network protection and management capabilities[22]. The network protection attributes provide the option to route an optical connection onto either of two diverse optical paths in opposite directions around the ring. The SONET self-healing ring (SHR) provides protection optical capacity onto which disrupted connections are rerouted, in the opposite direction around the network, after a link or node failure. All disrupted traffic is restored after a link failure, while part of the disrupted traffic is rerouted after a node failure. SONET SHRs have a typical service restoration time of less than 50ms[22].

1.1 Motivation

The emerging paradigm for designing large-scale networks based on fiber-optics technology involves connecting clusters of nodes through a SONET ring and building a network over these rings[15]. SONET rings are also becoming increasingly deployed by major telecommunications carriers[25]. On such large networks a single optical node or link failure may cause major disruption of service. Network survivability and reliability issues are crucial. The self-healing properties and cost effectiveness of SONET SHRs make them attractive for high capacity large networks.

Mihail *et al.*[17] state that current technology makes 30–40 optical wave-

lengths available to research laboratories. Only about half this number is available for practical use. Gerstel[12] places the number of available wavelengths at 32. Moreover, this number is not expected to increase significantly in the foreseeable future. This number of wavelengths is a small percentage of the available optical fiber waveband, which provides virtually unlimited optical capacity. Rather, it is the capacity of the ADMs that limits bandwidth capacity on a SONET ring. Therefore, network costs increase with the bandwidth level supported by the network[15].

On SONET rings, the path assignment subproblem of RWA is the Ring Loading problem (RLP). Due to the prevalence of SONET rings, RLP has been studied in depth. The cost of a SONET ring is dependent on its capacity or bitrate. Network capacity is required both to route connections, and to provide additional protection capacity, onto which disrupted connections can be rerouted in the event of a failure. Therefore, the objective of network planning is to minimize the bandwidth required to support network traffic demands[15]. Cosares and Sanjeev[5] refer to the problem of the selection and configuration of groups of optical nodes into potential SONET rings for cost effective routing of traffic demands. This process involves routing of an enormous number of sets of connection requests on proposed ring configurations to determine those which best minimize ring congestion. The Bellcore SONET Toolkit[4] provides software modules to provide ring-loading solutions to the traffic demands on proposed ring configurations. This tool has enabled planners to design complex ring networks that were not possible to analyze previously. The relatively short computation times required by the ring loading modules in the planning tool allow rapid evaluation of large numbers of network design alternatives. After the routing of demands on a proposed network has been determined, the cost of the fiber and equipment required to satisfy the capacity requirements may be calculated. RLP is NP-complete. Therefore, Schrijver *et al.*[24] identify the requirement for an approximation scheme which is both computationally fast, and which provides a solution to RLP which exceeds the optimum by no more than 5% in most cases.

1.2 The Problem

This thesis studies the Power 2 Demand Ring Loading Problem RLP2, a subproblem of RLP. RLP2 is similar to RLP except that request demands

are limited to integral powers of 2. With this limitation, bandwidth under utilization is not expected to be significant especially for the lower bandwidth denominations.

$2^0, 2^1, \dots, 2^x$, integer x , is a superincreasing sequence such that $2^{x+1} = 2^x + 2^{x-1} + \dots + 2^0 + 2^0$. Therefore, an optimal packing of 2^x bin by $2^x, 2^{x-1}, \dots, 2^0$ values is achievable by greedy techniques in a single pass. This simplification of the bin packing process facilitates the definition and proof of bounds that limit the demands of feasible routing combinations. Non-feasible routing combinations are more easily and rapidly identifiable. The routing of large numbers of requests is simplified.

To date, RLP algorithms are either approximations or allow some form of demand splitting. Therefore, it is reasonable to allow some demand splitting by the end user. A request for 3 bandwidth units may be specified as a combination of a 2 unit request and a 1 unit request. In this way, the precision of the algorithm is enhanced.

This thesis does not attempt to solve the general form of RLP2. Instead, it explores possible solutions by solving two simplified versions, RLP2[1] and RLP2[2]. These versions restrict the shortest path between the terminal pairs to a maximum of 1 and 2 optical links respectively. These are instances of the Power 2 Demand k Shortest Path Ring Loading problem (RLP2[k]) which restrict the maximum shortest path length for a request to a maximum distance of k optical links.

The study of RLP2[1] and RLP2[2] is intended to provide insight into possible solutions to the general form of RLP2. Because RLP is NP-complete, these solutions are most likely to be approximations. The solution of RLP2 is expected to vary according to the application.

The units of demand need not be restricted to optical wavelengths. The principles, discussed by this thesis, are equally applicable to other units of bandwidth on optical rings. In turn, they may be applied to other routing problems and optical architectures that are not rings.

1.3 Background

Banerjee and Mukherjee[2] decompose the RWA problem for networks without wavelength converters into separate path selection and path colouring phases:

Path selection: The RWA problem without the wavelength continuity constraint is formulated as an integer multicommodity flow problem, which is reformulated as an integer linear program (ILP) with the objective function of minimizing the flow in each link (congestion). In their formulation of the integer multicommodity flow, each commodity is a path from a source to a destination; the capacity is the number of wavelengths supported on each path link, and the objective function is to minimize the number of wavelengths needed to support all requests. It is assumed that the existence of all requests is known beforehand.

The authors approximate this NP-complete problem by a method known as randomized rounding[20]. The first step of the randomized rounding procedure is to relax the integrality flow constraint of the original problem allowing fractional flows in the $[0,1]$ range. This can be solved by linear programming (LP) techniques in polynomial time. The solution of the LP is then used to give a set τ_i of possible paths for each commodity i . The final step is to select a path for each commodity using randomization methods.

As stated above, this procedure solves the RWA problem without the continuity constraint. This is equivalent to the routing problem on networks with wavelength converters.

Path colouring: Once a path set has been determined, wavelengths must be assigned to the paths in networks with the wavelength continuity constraint. This problem can be reformulated as the NP-complete graph colouring problem by representing each path as a node $v \in V$ in a path graph $P(V, E)$ with an edge $e \in E$ connecting a pair of nodes, if the two paths share a common optical link. The minimum number of colours needed to colour P is the *chromatic number* of P denoted by $\chi(P)$.

Aumann and Rabani[1] use a similar approach to determine an upper bound on the number of required wavelengths to route a permutation in a bounded degree network. A set of requests is termed a *permutation* if each node can be a source to, at most, 1 request and a sink to, at most, 1 request. An integer multicommodity flow is formulated[16]. Theorem 2 to determine a path set with *congestion* c and *dilation* d (maximum path length) both at $O(\log n/3)$ where n is the number of nodes and 3 is the edge expansion of the network. The path graph for this path set has maximum degree cd and can be coloured in greedy fashion using $O(\log^2 n/3^2)$ colours.

Ramaswami and Sivarajan[21] determine an upper bound on the number

of routed requests (lower bound on the blocking probability for a request) given a permutation (set of requests) R and a fixed number of wavelengths Λ for arbitrary networks with and without wavelength conversion. They formulate the RWA problem as a multicommodity flow problem according to [13], ch. 6, sect. 2 and, using that formulation, determine the required bounds.

The *Ring Loading problem* (RLP) is a subproblem of the RWA problem on Optical Rings. An informal definition of RLP is:

Definition 1.3.1 (Ring Loading problem) *Given an n node optical ring $\mathcal{R}$ and a set of m demands between pairs of nodes, find a direction for each of the demands so that the maximum load on the links of $\mathcal{R}$ is minimized.*

RLP solves the RWA path assignment subproblem for rings. If wavelength converters are not present, the wavelength assignment phase must assign wavelengths to the selected paths so as to minimize the number of required wavelengths. With wavelength converters the number of required wavelengths is the maximum load on any edge of the ring, which is the congestion [2]. Therefore, the wavelength assignment phase is not necessary. This thesis addresses the RLP subproblem of the RWA problem on rings.

There are two versions of RLP. The undirected version is represented by an undirected cyclic graph and assigns an undirected path to each (i, j) pair. The directed version (DRLP) is represented by a directed cyclic graph in which a directed path is assigned to each (i, j) pair. The directed version can be thought of as two oppositely directed rings on the same set of nodes with the loads on each link, in each direction, being independent. For the directed version, the objective is to minimize the load on any one link in any one direction. This thesis assumes the undirected version of RLP.

For undirected rings, Schrijver *et al.* [24] state that RLP is an Integer Multicommodity Flow problem (IMF), the general form of which is NP-complete [14]. The authors prove that, based upon the supposed input size, that their version of RLP is NP-complete by a reduction from the Partition problem. They provide an approximation algorithm that achieves a ring load which is at most $3/2$ times the maximum demand D . However, with all demands equal to 1, RLP is solvable in polynomial time as the IMF1 version of IMF. The proof is established by Frank [7], [8] and can be summarized as follows:

Frank [7] defines the Edge-Disjoint Paths problem (EDP). Given an undirected graph $G = (V, E)$, find m edge disjoint paths in G connecting m

specified (not necessarily distinct) pairs of nodes of G . IMF is a capacitated version of EDP, where each edge e has an integral capacity indicating how many paths are allowed to pass through e . IMF is NP-complete[14]. Frank proves (and derives an algorithm) that EDP is solvable in polynomial time with complexity $O(|V|^3 \log |V|)$ when:

- (i) G is planar, that is G can be inscribed onto the plane with no two edges crossing over each other.
- (ii) The terminals are all positioned on the outer face or external boundary of G .
- (iii) Each node not on the outer face has even degree.

The ring (cycle) fits this description. In[8], Theorems 2.1, 2.2, Frank *et al.* solve EDP with non-negative integer capacities $c(e)$ on the edges of G for cycle networks. The problem is to find paths in G connecting the pairs of terminals such that the number of paths using an edge is, at most, its capacity $c(e)$. This problem is the IMF1 problem. Using the Okamura-Seymour theorem[18];[8], Theorem 2.1, they claim an $O(n^2)$ algorithm for a n node cycle, but they do not explain how they derive this algorithm[25]. They also claim an $O(n + m)$ solution using the variable-priority queue data structure[23], but closer examination of that paper reveals that this method solves IMF1 only for even planar flow cycle networks and, of course, not all IMF1 problems on ring networks fit into this category.

Vachani, Shulman, Kubat and Ward[25] give algorithms for both the Linear Multicommodity Flow problem (LMF) and a definition of IMF on ring networks. This definition, which allows demand splitting at integral (integer values), is equivalent to the IMF1 definition given above. The time complexity of both algorithms is $O(n^3)$. They state that they do not focus on developing efficient algorithms because ring networks are not likely to involve more than 20 nodes.

Cosares and Sanjeev[5] also prove that the integral version of RLP, where the maximum demand D is > 1 , is NP-complete by reduction from the Partition problem. To clarify, this is the version, defined by Schrijver *et al.*, where specific demands cannot be split between the two path options. The authors provide three heuristic solutions to this problem. Two are greedy approaches, with time complexities of $O(m(\log m + n))$ and $O(m^2 n)$ where

m is the number of demands. The third is a dual ascent weight-based approach, which approximates the optimal ring load to within a factor of 2. The time complexity of this latter approach is $O(cm)$ where c is a predetermined number of algorithmic iterations.

S. Khanna[15] develops an algorithm that, for any deviation $\varepsilon > 0$ from the optimum, gives a polynomial time solution that approximates the optimum ring load to within a $(1 + \varepsilon)$ factor for the integral RLP with the maximum demand $D > 1$ (demand splitting not allowed). The running time for the algorithm is $O((1/\varepsilon) * n^{5+z})$ where $z = 3(1 + 1/n)/\varepsilon$.

Mihail *et al.* give an upper bound on the number of wavelengths required to route a set of requests on a ring. Let L_{wopt} be the maximum load per link for the routing that results in the optimal colouring with W_{opt} the optimal number of wavelengths. To derive this bound:

- (i) For the optimal routing break any single edge so that the remainder of the graph forms a line. All requests routed through the broken link must be rerouted through their alternate path option. The maximum load on the new graph is $L_{max} \leq 2L_{wopt}$ as, at most, L_{wopt} requests must be rerouted.
- (ii) $L_{max} \leq 2L_{wopt}$ holds for all edges on the ring. As the resulting graph is a line, the required number of colours is $L_{max} \leq 2L_{wopt} \leq 2W_{opt}$.

For directed rings, Wilfong and Winkler[26] study the Directed Ring load-ing problem (DRLP). DRLP is represented by a directed cyclic graph on which the objective is to assign directed paths to m sets of ordered pairs of terminal nodes (i, j) so that the load on any one link, in any one direction, is minimized. DRLP is an IMF which is NP-complete in its general form[9] (individual demand splitting is not permitted between the two routing options). The authors prove:

- (i) That DRLP is solvable in polynomial time by a partial linear relaxation and rounding technique in which demands may be split [Theorem 1]. The time complexity is not given.
- (ii) That the equivalent of DRLP without wavelength translators is NP-complete. This problem is reduced to the arc colouring problem on circular graphs, which is NP-hard[11]. Erlebach and Jansen[6] have independently proved this result.

- (iii) That only one wavelength translator is required on a ring network to achieve the optimal load for DRLP in polynomial time.

1.4 The Power 2 Demand Ring Loading Problem

Let $\mathcal{R}$ be a set of n optical nodes that form an optical ring. Number the nodes from 1 to n in the clockwise direction. With the 1-to-1 correspondence between optical nodes and optical links, assign to each optical link ℓ the node id i of its anti clockwise adjacent optical node. $\mathcal{R}$ therefore may denote either the complete set of optical nodes or optical links on the ring. As such, it is also convenient for $\mathcal{R}$ to refer to the ring itself.

Let q be a request for an optical connection of demand $d(q)$ between optical nodes i and j on $\mathcal{R}$. There are two possible assignments for the routing path $P(q)$ of q .

Definition 1.4.1 (shortest path) *The subset of optical links on $\mathcal{R}$ comprising the shortest path between i and j .*

Definition 1.4.2 (alternate path) *The complement of the shortest path between i and j on $\mathcal{R}$.*

Next, clarify the concept of *load* or *congestion*.

Definition 1.4.3 (link load) *For optical link ℓ , the link load L_ℓ is the sum of all the demands of all requests whose optical path includes ℓ . The link load is also the congestion on ℓ .*

Definition 1.4.4 (ring load) *For $\mathcal{R}$, the ring load $L_{\mathcal{R}}$ is the maximum link load for all links ℓ on $\mathcal{R}$. The ring load is also the congestion on $\mathcal{R}$.*

Let $\mathcal{Q}$ be the set of m connection requests q , $1 \leq q \leq m$.

Definition 1.4.5 (routing of $\mathcal{Q}$) *A routing r of $\mathcal{Q}$ assigns a path to each $q \in \mathcal{Q}$ such that:*

$r_q = 0$ *assigns q the shortest path.*

$r_q = 1$ *assigns q the alternate path.*

Definition 1.4.6 (optimal routing of $\mathcal{Q}$) A routing which minimizes the congestion on $\mathcal{R}$ is called an optimal routing. r_{opt} denotes an optimal routing. L_{opt} refers to the optimal attainable congestion. that is. the congestion of any optimal routing.

RLP is defined by Schrijver, Seymour and Winkler[24] as:

Instance: A n node ring $\mathcal{R}$. m requests $q \in \mathcal{Q}$ for an optical connection of demand $d(q)$ between node pairs (i_q, j_q) are to be routed onto either the shortest path or the alternate path. Multiple requests between identical node pairs are allowed.

Question: What routing ϕ such that:

$\phi_q = 0$ assigns q the shortest path.

$\phi_q = 1$ assigns q the alternate path.

minimizes the congestion on $\mathcal{R}$?

The fact that $\phi_q = 0$ or 1 only states that the total demand $d(q)$ cannot be split between the two routing path options. This is the *integrality constraint*.

This is the general form of RLP, which is NP-complete. If the bandwidth demand denominations are restricted to powers of 2. it is estimated that bandwidth under utilization may not be significant. and end user inconvenience would be minimal. The Power 2 Demand Ring Loading problem (RLP2) is defined as:

Instance: Same as RLP except that for request q . $d(q) = 2^x$. $0 \leq x \leq w$. non-negative integer x . w where $d_{max} = 2^w$ is the maximum demand denomination.

Question: Same as RLP.

Since multiple requests between terminal pairs are allowed. a form of demand splitting may be introduced by permitting a user to specify multiple requests.

1.5 Summary of Results

RLOAD2[K] is an algorithm, which solves RLP2[k]. It is important that RLOAD2[K] be fast. Therefore, an algorithm which finds a 'nearly optimal' congestion 'most of the time' in a short running time is probably more useful than an otherwise exact algorithm. A second point is that the research into RLP2[1] and RLP2[2] illustrate concepts applicable to the general RLP2 problem. Many of the concepts and their proofs are formulated to apply to RLP2. As such, possible solutions to RLP2 and related problems are suggested. From the NP-completeness of the general form of RLP and the relative complexity of RLP2[2], it is anticipated an exact solution to RLP2 is not feasible. Therefore, the concepts developed by this thesis are intended to be useful for the development of heuristic solutions to RLP2 type problems.

RLOAD2[1] implements an exact solution to RLP2[1] in $O(m \log_2 m + n)$ time. RLOAD2[2] implements an approximate solution to RLP2[2] in $O(2^w(n + m) + m \log_2 m)$ time. Both these values include a $O(m \log_2 m)$ request sorting factor. Sorting of requests may be accomplished in $O(m \log_2 m)$ worst case time by the merge sort and heapsort algorithms[3]. The quicksort algorithm has $O(m \log_2 m)$ average running time and $O(m^2)$ worst case run time[3]. Despite, the $O(m^2)$ worst case time, quicksort often performs better than the other two algorithms. Due to time constraints, an exact solution has not been developed for RLP2[2]. Also, from the discussion of RLP2[2], it is suggested that the usefulness of an exact solution may not justify its added complexity. It is estimated that an exact solution to RLP2[2] may be implemented in the same running time. A rationale for this estimate is provided.

The 2^w running time factor denotes an exponential RLOAD2[2] running time. As the number of available optical wavelengths is probably less than 100, this value may represent a constant. For $w = 12$ or 4096 available wavelengths this argument may remain valid. However, it remains desirable to significantly reduce this factor. The algorithms below are designed to quickly eliminate many nonfeasible routing possibilities so that in practice the 2^w may often be reduced to a polynomial or even a linear factor.

1.6 Overview

RLOAD2[K] approaches RLP2[k] by initializing all requests $q \in Q$ to their shortest path. r_{init} denotes this *initial routing*. The object of RLOAD2[K] is to select a subset of the requests in Q to be *reassigned* or *rerouted* to their alternate path.

Requests to be routed to their alternate path are selected from the *eligible request set on $\mathcal{R}$* $E_{\mathcal{R}}$ also referred to as *eligible set on $\mathcal{R}$* . This implies that not all of Q is always eligible for rerouting. The strategy of RLOAD2[K] is to successively remove requests from $E_{\mathcal{R}}$ so as to reduce the search space for r_{opt} . Removal of requests from $E_{\mathcal{R}}$ is irreversible. Therefore, $E_{\mathcal{R}}$ is steadily reduced in size. All requests routed to their alternate path are removed from $E_{\mathcal{R}}$. Therefore, once a request is routed to its alternate path, it is never reassigned to its shortest path.

RLOAD2[K] investigates, beginning with r_{init} , successive routings of Q for *feasibility*, the possibility that the current set of rerouted requests forms a subset of the rerouted requests in some optimal routing. Let r be the current routing. $E_{\mathcal{R},r}$ is the eligible set on $\mathcal{R}$ for r . Let R be the subset of Q already assigned to their alternate path. r can be feasible only if there exists some subset R' of $E_{\mathcal{R},r}$ such that reassignment of $(R + R') \subseteq Q$ to their alternate path represents an optimal routing.

Initially the current routing r is r_{init} . $E_{\mathcal{R},r}$ is initialized to Q . For r , the *maximum load link* is the link a with the maximum link load on $\mathcal{R}$. For r , RLOAD2[K] selects from $E_{\mathcal{R}}$ one or more routing sets $R_{r \rightarrow s}$, which map r to a new routing s . For this purpose, the *eligible request set on ℓ* or *eligible set on ℓ* , $E_{\ell,r}$ is the set of eligible requests on $\mathcal{R}$ for which the assigned path includes ℓ . The restrictions on $E_{\mathcal{R},r}$ ensure that all requests in $E_{\ell,r}$ are assigned to their shortest path.

A greedy bin packing algorithm RLOAD2-PART[K] selects requests from $E_{a,r}$ for possible reassignment to their alternate path. For request q , $d(q)$ is the value function for the bin packing process. Each link ℓ on $\mathcal{R}$ except a represents a bin constraint on the reassigned demand. A bin is referred to as *bin ℓ* when the maximum load link is known. The goal is to maximize the demand reassigned from a without overflowing any of the bins. This defines a Subset Sum problem (SSP). For the RLP2[k] application this bin packing problem is referred to as the RLP2[k] Bin Packing Problem (RLP2[k]-BPP). RLOAD2-BPACK[K] is an algorithm, which solves RLP2[k]-BPP.

RLOAD2-BPACK[K] partitions $\mathcal{R}$ into three disjunctive *routing zones* for

Figure 1.1: Routing Zones

routing r .

Definition 1.6.1 (maximum load link zone, Z_A) Z_A consists of the maximum load link a . Z_A is referred to as the A-zone of $\mathcal{R}$.

Definition 1.6.2 (interdependent zone, Z_F) Z_F is the set of links excluding a which are included on the shortest path of some request in $E_{a,r}$. Z_F is referred to as the F-zone of $\mathcal{R}$.

Definition 1.6.3 (independent zone, Z_B) Z_B is the set of links which are not included on the shortest path of some request in $E_{a,r}$. Z_B is referred to as the B-zone of $\mathcal{R}$.

For r , the independent zone maximum load link is the link b with the maximum link load on Z_B .

Figure 1.1 illustrates the routing zones on an 8 node optical ring $\mathcal{R}$. For ease of illustration, a linear profile view of $\mathcal{R}$ is given. All requests have equal bandwidth demand. All requests are assigned to their shortest path. Therefore, link 4 is the maximum load link a and $E_{a,r}$ consists of $q1$, $q2$, $q3$ and $q5$. The assigned paths of $q1$ and $q2$ intersect link 3, while the assigned

paths of $q3$ and $q5$ intersect link 5, so that the F-zone consists of links 3 and 5. The B-zone consists of the remaining links. Therefore, link 7 is the B-zone maximum load link b .

The RLP2[k] Eligible Set Partition Algorithm RLOAD2-PART[K] selects requests from $E_{a,r}$ in greedy fashion within the F-zone and B-zone bin constraints. RLOAD2-PART[K] produces two output lists, the *selection set* S which is the set of the requests selected from $E_{a,r}$ and the *rejection set* X the list of non-selected requests. As such RLOAD2-PART[K] partitions $E_{a,r}$ into S and X , therefore, its name. $\mathcal{P}_r$ denotes an arbitrary partition of $E_{a,r}$ for routing r . Because only requests in $E_{a,r}$ are being selected, $E_{a,r}$ is referred to as the *eligible request set* or the *eligible set*. A request in the eligible request set is referred to as an *eligible request for selection* or an *eligible request*.

The significance of Z_B is that all bandwidth load rerouted from a is routed onto all $\ell \in Z_b$. As b is the maximum load link in Z_B , bin b is the only independent zone bin constraint on RLOAD2-BPACK[K] for r . It is verifiable that $\lceil \frac{L_a + L_b}{2} \rceil$ is a lower bound on the ringload given routing r .

An eligible request qe may include, in addition to a , one or more F-zone links on its shortest path. Selection of qe , therefore, may not pack all of the F-zone bins. Also two different eligible requests may pack different subsets of the F-zone bins. At the same time, the F-zone imposes variable bin constraints and offers multiple eligible request selection options, which are, do not exist in the B-zone. The existence of this variability of constraints and options in the F-zone may account for the complexity of RLP2.

Insufficient capacity in one or more F-zone bins f may prevent the selection of an eligible request qf which would otherwise reduce the congestion on $\mathcal{R}$. For this reason, qf is *interfered* on bin f . Link f is referred to as an *interference link*. On detection of an interfered request, the F-zone request selection options must be explored to *resolve* the interference on bin f by increasing its bin capacity so that qf may be routed.

If a request qf is interfered on some F-zone bin f on $\mathcal{P}_r$, then RLOAD2-PART[K] may not have produced an optimal packing of $E_{a,r}$. In this case, a *repartition* of $E_{a,r}$ may resolve the interference. This thesis analyzes the repartition process for RLP2[2]. For RLP2[2], RLOAD2-BPACK[2] 'de-selects' a selected request $qb \in S$, which packs bin f and replaces qb on S by an equivalent $d(qb)$ bandwidth block X_1 from X . X_1 must include a minimum of 1 request which does not pack bin f . In this way, capacity is freed on bin f to select qf .

RLOAD2-BPACK[2] may require multiple repartitions to produce an optimal packing of $E_{a,r}$ within the bin constraints. The replacement requests qb are selected in greedy fashion of $d(qb)$. After qb is de-selected, RLOAD2-PART[2] repartitions $E_{a,r}$ from the point where qb was originally selected. A new partition of $E_{a,r}$ results. One qb is replaced per repartition. In its search for an optimal packing, RLOAD2-BPACK[K] uses a tree structure, referred to as the *beta tree*, on which each node represents a partition. When all repartition possibilities have been exhausted or an optimal packing has been identified, the optimal selection set is read from the beta tree.

On completion of the RLOAD2-BPACK[2] process, RLOAD2[2] selects from the beta tree one or more routing sets R to be investigated for possible feasibility. To be feasible, R must map r to a new feasible routing s . The algorithm must prove feasibility. Proving their *nonfeasibility* successively eliminates possible routing sets and routings. This elimination process finds an optimal routing.

There are two types of routing set:

non-terminal routing set $R_{r \rightarrow s}$. $R_{r \rightarrow s}$ maps to a new routing s . $R_{r \rightarrow s}$ is formed by selection of a request $qx \in X$ and all the requests in S of demand greater than or equal to $d(qx)$. The inclusion of qx in $R_{r \rightarrow s}$ overflows at least one of the bin capacities. Consequently, the maximum load link is shifted to a new $\ell \neq a$. After $R_{r \rightarrow s}$ is routed, s becomes the new current routing for which the bin packing and routing set selection process is reiterated if s is feasible. s is, therefore, a *non-terminal* routing or *intermediate* routing. More than one candidate non-terminal routing may set be selected from the beta tree.

terminal routing set $R_{r \rightarrow term}$. $R_{r \rightarrow term}$ is the routing set which achieves the optimum packing of $E_{a,r}$. Because $R_{r \rightarrow term}$ does not overflow any bin, it cannot relocate the maximum load link from link a . Therefore, after $R_{r \rightarrow term}$ is routed, there exist no new bin packing and routing set selection possibilities. As such, $R_{r \rightarrow term}$ maps to a *terminal* routing r_{term} . A necessary condition for r_{term} to be feasible is that its congestion be less than the previously achieved level.

The search for r_{opt} , therefore, defines a tree structure called the *alpha tree*. On the alpha tree, each node represents an intermediate routing r . Each edge represents a routing set $R_{r \rightarrow s}$ which maps r to routing s represented by a

descendant node. $R_{r \rightarrow term}$ is stored at its node representing r . The root node of the tree represents r_{init} . Each path from the root to a node defines a sequence of routings referred to as a *routing sequence*. Each alpha tree node defines a subtree for which the initial routing is r represented by the node. The goal of the algorithm is to find the minimum congestion for the subtree, given routing r and $E_{\mathcal{R}, r}$.

As RLOAD2[2] assesses routing sequences for feasibility, the subtrees for nonfeasible sequences are pruned. In this way the algorithm terminates with a linear tree representing the routing sequence that gives the best achievable congestion.

The key to achieving a fast algorithm is to minimize the number of non-terminal routing sets $R_{r \rightarrow s}$ to assess for feasibility. The purpose of many of these routing sets is to resolve interference on eligible request qf on some F-zone bin f . This is the *capacity transfer* process. A capacity transfer replaces a subset of S by a subset of $E_{\mathcal{R}}$ to increase the f bin capacity so that qf may be subsequently routed without interference. Each repartition of $E_{a, r}$ by RLOAD2-BPACK[2] is a capacity transfer where the replacement set is selected from $E_{a, r}$. A transfer is also accomplished by routing $R_{r \rightarrow t}$, which relocates the maximum load link to $y \neq a$ and then routing $R_{t \rightarrow r}$, which relocates the maximum load link back to a where qf may be selected without interference.

Even for RLOAD2[2] several non-terminal routing sets may be feasible for routing r . Many of these are feasible in theory but have a small probability of remaining feasible after further assessment. This is where heuristic solutions play a role: select only those routing sets that have a reasonable chance of achieving a minimum or optimum routing. In this way, a small tradeoff in congestion optimality for computation speed is accomplished.

1.7 Outline of Thesis

Chapter 2 formulates a preliminary set of definitions, observations, lemmas and proofs. Chapter 3 details a set of definitions and concepts applicable to all RLP2[k] problems and therefore, the general RLP2 problem. These principles are applied to the solutions of RLP2[1] and RLP2[2] in later chapters. Chapter 4 presents the RLOAD2[1] algorithm as a solution to RLP2[1]. Chapter 5 discusses the interdependent routing zone of $\mathcal{R}$ and interfered requests. The capacity transfer is introduced as a means to eliminate interfer-

ence on interfered requests. Chapter 6 applies the capacity transfer concepts to RLP2[2]. The RLOAD2-BPACK[2] and RLOAD2-PART[2] bin packing algorithms are developed. The RLOAD2[2] algorithm uses these algorithms to provide an approximate solution to RLP2[2].

Chapter 2

Preliminaries

2.1 Introduction

This chapter establishes general problem solving concepts upon which the remainder of the thesis is based. These concepts apply to general RLP2[k] solutions, in addition to RLP2[1] and RLP2[2].

Section 2.2 discusses routings of Q . Included are definitions relating to a routing, routing sets, link loads, congestion and eligible sets. Routing feasibility and routing sets is explained. Routing set minimality, which defines a partial order relationship on potential routing sets, is introduced as a routing set evaluation criterion.

Section 2.3 addresses congestion on a link pair $\{x, y\}$. The independent and common subsets of the eligible request sets E_x and E_y are defined. Then the independent and common congestion for links x and y is defined. These concepts are used to establish a lower bound on the overall $\{x, y\}$ congestion. Link pair capacity and link pair independence are then explained.

Section 2.4 expands upon the three routing zones of $\mathcal{R}$ for routing r . In particular, this section focuses on the significance of the independent zone (B-zone). The average of congestion on the maximum load link a and the B-zone maximum load link b is established as a lower bound on the ringload. The capacity of a relative to b is established as a single B-zone bin for the bin packing process.

In Section 2.5 path independence and interdependence of request sets eligible for rerouting is introduced. This is a key concept used to eliminate potential routing sets. Routing two equal demand requests sets for which

none of the assigned paths (always shortest path for eligible requests) in either set intersect with an assigned path from the other cannot reduce the load anywhere on $\mathcal{R}$. However, there is a possible reduction on a path intersection or overlap zone of the two sets. As will be seen, the latter condition offers more routing possibilities and increase the complexity of RLP2[k] problems.

Section 2.6 establishes the maximal shortest path subinterval constraint on RLOAD2-PART[K]. Essentially, this means that given the choice between reassigning equal demand requests q_1 and q_2 whose shortest path is a proper subset of the shortest path of q_1 , then q_1 is the superior choice, as the alternate path of q_1 is a proper subset of the alternate path of q_2 . This concept is generalized to comparing single requests versus a set of requests. It can be expanded to compare two sets of eligible requests, but that is not necessary here.

2.2 Routings of $\mathcal{Q}$ on $\mathcal{R}$

The objective of RLP2[k] is to find an optimal routing r_{opt} of a set of optical requests $\mathcal{Q}$ on an optical ring $\mathcal{R}$. There may be more than one possible optimal routing of $\mathcal{Q}$. However, since the objective is to minimize $L_{\mathcal{R}}$, it is sufficient to find only one optimal routing.

The definitions below are associated with routing r of $\mathcal{Q}$. For request q , $P(q)$ is assigned to either $P^-(q)$, the shortest path link set, or $P^+(q)$, the alternate path link set.

Definition 2.2.1 (link path set) *The link path set I_{ℓ} , $\ell \in \mathcal{R}$ is the set of requests q whose assigned optical path $P(q)$ includes link ℓ .*

$$I_{\ell} = \{q \in \mathcal{Q} : o_q = 0 \text{ and } \ell \in P^-(q)\} \cup \{q \in \mathcal{Q} : o_q = 1 \text{ and } \ell \in P^+(q)\}$$

For routing r the explicit form is $I_{\ell,r}$. I_{ℓ} is also referred to as the path set for link ℓ .

Definition 2.2.2 (maximum load link) *The maximum load link ℓ_1 logically defines the link $a \in \mathcal{R}$ with the maximum link load. For routing r , the value of ℓ_1 is referred to as link a , unless otherwise explicitly stated or contextually implied.*

Definition 2.2.3 (independent zone maximum load link) *The independent zone maximum load link ℓ_2 logically defines the link $b \in Z_B$ with the*

maximum link load. For routing r , the value of ℓ_2 is referred to as link b , unless otherwise explicitly stated or contextually implied.

If two or more candidates for either ℓ_1 or ℓ_2 have identical loads, then ℓ_1 or ℓ_2 is assigned arbitrarily.

Definition 2.2.4 (eligible request set on $\mathcal{R}$) The eligible request set on $\mathcal{R}$ $E_{\mathcal{R}}$ is the subset of $\mathcal{Q}$ eligible for reassignment to their alternate path. For routing r the explicit form of $E_{\mathcal{R}}$ is $E_{\mathcal{R},r}$.

Definition 2.2.5 (eligible request set on link ℓ) The link ℓ eligible request set E_{ℓ} is the subset of $E_{\mathcal{R}}$ which includes ℓ on its assigned path. For routing r , E_{ℓ} may be represented by $E_{\ell,r}$.

Definition 2.2.6 (eligible request set) The maximum load link eligible request set E_a where a is the maximum load link is referred to as the eligible request set or eligible set. For routing r , E_a may be represented by $E_{a,r}$. $E_{a,r}$ is the set of requests eligible for partitioning by $\text{RLOAD2-PART}[K]$. Within context, an eligible request set or eligible set may also refer to as any subset of $E_{\mathcal{R}}$.

Definition 2.2.7 (routing set) A routing set $R_{r \rightarrow s} \subseteq E_{\mathcal{R},r}$, selected for routing r , maps routing r to a new routing s . For each $q \in R_{r \rightarrow s}$, the value of mapping r_q is complemented to obtain a new mapping s .

Definition 2.2.8 (feasible routing) A routing r is feasible only if there exists some subset R' of $E_{\mathcal{R},r}$ such that reassignment of $(R + R') \subseteq \mathcal{Q}$ to their alternate path represents an optimal routing. If R' does not exist, then r is nonfeasible.

Definition 2.2.9 (feasible routing set) A routing set $R_{r \rightarrow s}$ is feasible for (feasible) routing r if $R_{r \rightarrow s}$ maps r to a feasible routing s . Otherwise, $R_{r \rightarrow s}$ is nonfeasible.

Definition 2.2.10 (minimum routing) A minimum routing $r_{\min,r}$ is an optimal routing for a given initial routing r . The ringload for $r_{\min,r}$ is denoted by $L_{\min,r}$. For r_{init} , $r_{\min,\text{init}} = r_{\text{opt}}$.

Definition 2.2.11 (minimum routing set) A routing set that maps r to $r_{\min,r}$ is a minimum routing set denoted by $R_{r \rightarrow \min}$. If $L_{\min,r} = L_{\text{opt}}$ then $R_{r \rightarrow \min}$ is an optimal routing set $R_{r \rightarrow \text{opt}}$.

The next set of definitions establish the concept of routing set minimality. Minimality defines a partial order relationship on routed loads when routing sets are selected from an arbitrary set E of requests eligible for rerouting to their alternate path. This relationship allows routing sets selected from E to be compared for their possible elimination as candidate routing sets by the routing algorithms.

Definition 2.2.12 (aggregate link load) *Let E be an eligible request set and R a subset of E . For optical link ℓ , the aggregate link load $\hat{L}_{E,R,\ell}$ is the total load assigned to ℓ by the assignment of all requests in R to their alternate path and all requests in $E - R$ to their shortest path. The aggregate load $\hat{L}_{E,R}$ is the array of aggregate link loads on $\mathcal{R}$ for E and R . If R consists of a single request q , then R may be replaced in the notation to give $\hat{L}_{E,q,\ell}$ and $\hat{L}_{E,q}$ respectively.*

Definition 2.2.13 (partial ordering of aggregate loads) *For eligible request set E and routing sets $R' \subseteq E$ and $R'' \subseteq E$:*

- (i) $\hat{L}_{E,R'} = \hat{L}_{E,R''}$ if $\hat{L}_{E,R',\ell} = \hat{L}_{E,R'',\ell}$ for all $\ell \in \mathcal{R}$.
- (ii) $\hat{L}_{E,R'} \leq \hat{L}_{E,R''}$ if $\hat{L}_{E,R',\ell} \leq \hat{L}_{E,R'',\ell}$ for all $\ell \in \mathcal{R}$.
- (iii) $\hat{L}_{E,R'} < \hat{L}_{E,R''}$ if $\hat{L}_{E,R',\ell} \leq \hat{L}_{E,R'',\ell}$ for all $\ell \in \mathcal{R}$ and $\hat{L}_{E,R',\ell} < \hat{L}_{E,R'',\ell}$ for a minimum of 1 $\ell \in \mathcal{R}$.

Definition 2.2.14 (minimal routing set) *For routing r , let E be an eligible request set. $R_{r \rightarrow s} \subseteq E$ is minimal if there exists no $R_{r \rightarrow t} \subseteq E$ such that $\hat{L}_{E,R_{r \rightarrow t}} < \hat{L}_{E,R_{r \rightarrow s}}$.*

Observation 2.2.1 establishes, for feasible routing r and feasible routing $R_{r \rightarrow s}$, a sufficient condition to assure the feasibility of $R_{r \rightarrow t}$.

Observation 2.2.1 *For feasible routing r , eligible set E , $R_{r \rightarrow s} \subseteq E$ is a feasible routing set. There exists $R_{r \rightarrow t} \subseteq E$ such that $\hat{L}_{E,R_{r \rightarrow t}} \leq \hat{L}_{E,R_{r \rightarrow s}}$. Then $R_{r \rightarrow t}$ is a feasible routing set.*

Proof: For $Q - E$ the path assignments are identical for s and t . If $\hat{L}_{E,R_{r \rightarrow t}} < \hat{L}_{E,R_{r \rightarrow s}}$, then for E all t link loads are less than or equal to their s values. Therefore, $R_{r \rightarrow t}$ can only decrease the overall congestion. $\square$

The concept of routing set minimality allows the disqualification of prospective routing sets from further assessment by routing algorithms. If R_{r-t} is minimal with aggregate load less than or equal to the aggregate load of R_{r-s} in the partial order according to Observation 2.2.1 then it is not possible that R_{r-s} yields a better congestion even when it is feasible. Therefore, R_{r-s} requires no further consideration during the search process.

It must be emphasized that a minimal routing set is not always a feasible routing set. Therefore, R_{r-t} may be minimal but not feasible. If the aggregate load of R_{r-t} is less than or equal to that of R_{r-s} , then R_{r-s} would also be nonfeasible.

2.3 Independent and Interdependent Link Pairs

Given link pair $\{x, y\}$ $x, y \in \mathcal{R}$. E_x may be partitioned into two subsets relative to y :

Interdependent subset $E_{xy} = E_x \cap E_y$. This subset is also referred to as the *common* subset.

Independent subset $\bar{E}_{xy} = E_x - E_{xy}$.

In Figure 2.1 $E_x = \{q1, q2, q3, q4\}$ and $E_y = \{q2, q4, q5\}$. $q2$ and $q4$ are common to both E_x and E_y so that $E_{xy} = \{q2, q4\}$. Therefore, $\bar{E}_{xy} = \{q1, q3\}$ and $\bar{E}_{yx} = \{q5\}$.

I_x is similarly partitioned relative to y :

Interdependent or common path set $I_{xy} = I_x \cap I_y$.

Independent path set $\bar{I}_{xy} = I_x - I_{xy}$.

L_x may now be partitioned relative to y :

Interdependent congestion or load $L_{xy} = \sum d(q) : q \in I_{xy}$.

Independent congestion or load $\bar{L}_{xy} = L_x - L_{xy}$.

$\bar{L}_{min,xy}$ denotes the minimum attainable $\{x, y\}$ independent congestion.

In the Figure 2.1 example, $I_x = E_x$ and $I_y = E_y$. Therefore, $L_x = 2^5 + 2^3 + 2^2$ and $L_y = 2^4 + 2^3 + 2^2$. $L_{xy} = 2^4 + 2^2$ is the congestion of E_{xy} . Therefore, $\bar{L}_{xy} = 2^4 + 2^3$ and $\bar{L}_{yx} = 2^3$.

By inspection, $\bar{L}_{xy}$ can be minimized at 2^4 so that $\bar{L}_{min,xy} = 2^4$. The minimization of $\bar{L}_{xy}$ is a recurring subproblem solved by the solutions proposed to RLP2[k] by this thesis. The following observations apply:

Observation 2.3.1 $\bar{L}_{min,xy} \geq \left\lceil \frac{\bar{L}_{xy} + \bar{L}_{yx}}{2} \right\rceil$, $x \neq y$.

Proof: The assigned paths for $q \in I_{xy}$ pass through both x and y . These paths may be complemented so that their load is removed from both links. The remainder of the proof consists of taking the average of $\bar{L}_{xy}$ and $\bar{L}_{yx}$. $\square$

Therefore, 2^4 is the lower bound on $\bar{L}_{min,xy}$ for the Figure 2.1 example.

Observation 2.3.2 Reassignment of $\left\lfloor \frac{\bar{L}_{xy} - \bar{L}_{yx}}{2} \right\rfloor$ bandwidth from x to y reduces $\bar{L}_{xy}$ to $\left\lceil \frac{\bar{L}_{xy} + \bar{L}_{yx}}{2} \right\rceil$, $x \neq y$. $L_x \geq L_y$.

Proof: Subtract $\left\lfloor \frac{\bar{L}_{xy} - \bar{L}_{yx}}{2} \right\rfloor$ from $\bar{L}_{xy}$. $\square$

Definition 2.3.1 (capacity) $C_{xy} = \left\lfloor \frac{\bar{L}_{xy} - \bar{L}_{yx}}{2} \right\rfloor$ is termed the capacity of link x with respect to link y .

Observation 2.3.1 shows that reassignment of requests in E_{xy} reduces both L_x and L_y simultaneously by the reassigned bandwidth. Therefore, the C_{xy} reassigned bandwidth must be selected from $\bar{E}_{xy}$. C_{xy} may be regarded as a bin into which bandwidth from $\bar{E}_{xy}$ is packed.

For the Figure 2.1 example, it is verifiable that $C_{xy} = 2^3$. Reassignment of q_1 to its alternate path packs exactly 2^3 load into this bin so that $\bar{L}_{xy}$ is reduced to the $\bar{L}_{min,xy} = 2^4$ lower bound.

Definition 2.3.2 (link pair independence) $\{x, y\}$ is an independent link pair if $E_{xy} = \emptyset$. Otherwise, $\{x, y\}$ is an interdependent link pair independent link pair.

Observation 2.3.3 limits the number of integral 2^i bandwidth blocks by which $\bar{L}_{xy}$ exceeds $\bar{L}_{yx}$ to a maximum of 1 when $\left\lfloor \frac{C_{xy}}{2^{i-1}} \right\rfloor = 0$. This observation is useful for later proofs.

Figure 2.1: Link Pair $\{x, y\}$ Congestion

Observation 2.3.3 Given $x, y \in \mathcal{R}$: $x \neq y$, $L_x \geq L_y$. Let 2^i be a power of 2, $0 \leq i \leq w$. If $\lfloor \frac{C_{xy}}{2^{i-1}} \rfloor = 0$ then $\lfloor \frac{\bar{L}_{xy}}{2^i} \rfloor - \lfloor \frac{\bar{L}_{yx}}{2^i} \rfloor \leq 1$.

Proof: If $\lfloor \frac{C_{xy}}{2^{i-1}} \rfloor = 0$ then $C_{xy} < 2^{i-1}$. Therefore, $\bar{L}_{xy} - \bar{L}_{yx} < 2^i$. Therefore, $\bar{L}_{xy}$ can have at most 1 more integral 2^i block of bandwidth than $\bar{L}_{yx}$. $\square$

2.4 Routing zones of $\mathcal{R}$

Given routing r , $\mathcal{R}$ may be subdivided into the three routing zones introduced in Chapter 1 Section 1.5. Each of these routing zones have attributes upon which path selection algorithms base their decisions during the search for r_{opt} . For r , let $\ell_1 = a \in \mathcal{R}$. To reduce $L_{\mathcal{R}}$ it is necessary to reduce L_a . For a minimum of 1 $q \in E_a$, $P(q)$ must be rerouted. A more formal definition of each routing zone is:

Definition 2.4.1 (A-zone, Z_A) Z_A consists of the maximum load link. $\ell_1 = a$.

Definition 2.4.2 (F-zone, Z_F) $Z_F =$

$$\{\ell \in \mathcal{R} : \ell \neq a, E_a \cap E_\ell \neq \emptyset\}$$

Definition 2.4.3 (B-zone, Z_B) $Z_B =$

$$\{\ell \in \mathcal{R} : E_a \cap E_\ell = \emptyset\}$$

Observation 2.4.1 assures that Z_B and therefore, ℓ_2 always exist.

Observation 2.4.1 $Z_B \neq \emptyset$

Proof: For RLP2[k], $k \leq \lfloor \frac{n}{2} \rfloor$. A path length above this value gives an alternate path length. If $k = \lfloor \frac{n}{2} \rfloor$ then, because all shortest paths must include ℓ_1 , the maximum length of Z_F and ℓ_1 is $n-1$. This leaves a minimum of 1 link for Z_B . $\square$

For all $a \neq \ell$, $C_{a\ell}$ represents a bin capacity constraint on the bandwidth which may be rerouted from a onto ℓ . One function of routing zones is to reduce the number of bins that must be considered. In particular, the independent zone is represented by a single bin C_{ab} .

Observation 2.4.2 establishes E_a as the independent eligible selection set for all $\ell \in Z_B$.

Observation 2.4.2 *Given routing r : $\ell_1 = a$. For all $\ell \in Z_B$, $\bar{E}_{a\ell} = E_a$, $E_{a\ell} = \emptyset$.*

Proof:

For all $\ell \in Z_B$, $E_{a\ell} = \emptyset$ by definition of independent link pairs and the independent zone. Because $E_a = \bar{E}_{a\ell} + E_{a\ell}$ and $E_{a\ell} = \emptyset$, $E_a = \bar{E}_{a\ell}$. $\square$

Lemma 2.4.1 illustrates the significance of link pair independence and the independent zone.

Lemma 2.4.1 *Given r : $\ell_1 = a$, $\ell_2 = b$:*

$$L_{min,r} \geq \left\lceil \frac{L_a + L_b}{2} \right\rceil$$

Proof: a and b are independent, therefore $L_{min,ab} = \left\lceil \frac{L_a + L_b}{2} \right\rceil$ is a lower bound on their congestion (Observation 2.3.1). $L_\ell \leq L_b$ for all $\ell \in Z_B$: $\ell \neq b$ (def. ℓ_2). Therefore, $L_{min,ab} \geq L_{min,a\ell}$ (Observation 2.3.1). $\square$

Definition 2.4.4 (independent zone capacity) $C_{ab} = \left\lfloor \frac{L_a - L_b}{2} \right\rfloor$ is referred to as the independent zone capacity for routing r .

Capacities represent bins to pack bandwidth from $\ell_1 = a$ onto some other link $\ell \in \mathcal{R}$ and are denoted by $C_{a\ell}$.

Corollary 2.4.1.1 For all $\ell \in Z_B$, $C_{a\ell} \geq C_{ab}$

Proof: Lemma 2.4.1 in conjunction with Observation 2.3.2 state that a maximum of C_{ab} bandwidth may be selected from E_a without the load on any $\ell \in Z_B$ being increased beyond L_a . $\square$

Corollary 2.4.1.2 For Z_B , C_{ab} is the only capacity constraint on selection of bandwidth from E_a .

Proof: For all $\ell \in Z_B$ Observation 2.4.2 establishes $\bar{E}_{a\ell} = E_a$. Thus selection of any request q from E_a adds $d(q)$ to all independent zone links. Lemma 2.4.1 Corollary 2.4.1.1 establishes C_{ab} as the minimum of the independent zone link capacities. $\square$

For each independent zone link ℓ , $C_{a\ell}$ is a bin to pack by selection of E_a requests. The above proofs show that all E_a request selections pack all of these bins equally and simultaneously. Because $C_{a\ell} \geq C_{ab}$, C_{ab} is the lower limit on the capacity of these bins. Therefore, only the C_{ab} bin needs to be considered when packing the independent zone. Therefore, C_{ab} is the only independent zone constraint on selection of E_a requests.

Returning to Figure 1.1, let the demand of q_6 and q_7 be 2^3 and the demand of all other requests be 2^4 . Lemma 2.4.1 establishes $2^5 + 2^4$ as the lower bound on the optimal congestion. $C_{ab} = 2^4$ is the B-zone bin capacity. Selection of any of q_1, q_2, q_3, q_5 packs all B-zone bins. Selection of one of these requests packs C_{ab} to achieve the optimal congestion.

2.5 Independent and Interdependent Routing Paths

Definition 2.5.1 (path independence of requests) Two requests $q_1, q_2 \in \mathcal{Q}$ are path independent if $P(q_1) \cap P(q_2) = \emptyset$. Otherwise, they are path interdependent. $P(q_1) \cap P(q_2)$ is the path overlap zone $P_{q_1 \cap q_2}$.

Definition 2.5.2 (aggregate path) *The aggregate path $P(S)$ of a set of requests S , $S \subseteq Q$ is the union of the assigned paths of each of the individual requests in S . $P(S) = \cup P(q): q \in S$.*

Definition 2.5.3 (request set path independence) *Two sets of requests S_1 and S_2 are path independent if the overlap zone of their aggregate paths $P_{S_1 \cap S_2} = \emptyset$. That is, no assigned path for $q_1 \in S_1$ shares a common link with an assigned path for $q_2 \in S_2$. Otherwise, they are path interdependent.*

Lemma 2.5.1 establishes the significance of the path overlap zone:

Lemma 2.5.1 *Rerouting of two equal demand d routing sets R_1 and R_2 :*

- (i) *cannot reduce the congestion on $\ell \notin P_{R_1 \cap R_2}$.*
- (ii) *may reduce the load on $\ell \in P_{R_1 \cap R_2}$.*

Proof:

- (i) $\ell \notin P_{R_1 \cap R_2}$ may lie on the aggregate path of either R_1 or R_2 but not both. As such, rerouting $R_1 + R_2$ always adds a minimum net load of 0 to ℓ .
- (ii) $\ell \in P_{R_1 \cap R_2}$ may lie on the aggregate path of both R_1 or R_2 . Therefore, rerouting $R_1 + R_2$ may result in a net reduction of L_ℓ .

□

Corollary 2.5.1.1 *Same as for Lemma 2.5.1 except $P_{R_1 \cap R_2} = \emptyset$. Then $R_1 + R_2$ is not a feasible routing set.*

Proof: With this added condition, Lemma 2.5.1 states that rerouting the $R_1 + R_2$ combination cannot reduce the load on any link of $\mathcal{R}$. Therefore, it is not feasible. This is also proven using Observation 2.2.1: When $R_1 + R_2$ are not rerouted, their aggregate link load is less than or equal to their rerouted aggregate link load; therefore, rerouting $R_1 + R_2$ is not feasible. □

In Figure 2.2 all requests have equal demand. $E_a = \{q_1, q_2, q_3\}$ with aggregate path on links 3-5. $E_b = \{q_4, q_5\}$ with aggregate path on links 6-7. Therefore E_a and E_b are path independent. Selection of equal demand

Figure 2.2: Path Independent Routing Sets

routing sets from each of E_a and E_b cannot reduce the load on any link and increases the load on some.

Figure 2.3 is identical to Figure 2.2 except that the path for q_4 includes link 5 instead of 7. Because link 5 lies on the E_a aggregate path, E_a and E_b are now path interdependent. Selection of the q_3 and q_4 combination reduces the load on link 5. Selection of these requests increases C_{ar} and C_{br} without increasing the congestion on links 4 and 6. For maximum load link a , a capacity transfer to link 5 occurs. This capacity transfer may allow a request interfered on link 5 to be subsequently routed.

2.6 Maximal Shortest Path Interval

Definition 2.6.1 (maximal shortest path interval) *Let S be a set of requests. The relation 'subset of' defines a partial order on the shortest path intervals of $q \in S$. Let $q' \in S$ be such that $P^-(q')$ is not a proper subset of any other $q \in S$. Then $P^-(q')$ has a maximal shortest path interval within S .*

During the greedy request selection process, $RLOAD2-PART[K]$ may be faced with the choice of one or more equal demand requests. This section shows that the feasible choice is a request with a maximal shortest path interval.

Figure 2.3: Path Interdependent Routing Sets

Observation 2.6.1 establishes an evaluation criterion for the feasibility of the selection of an eligible request whose shortest path is a subset of the shortest path of another request.

Observation 2.6.1 *Let $q' \in E$ and $q'' \in E$ be two equal demand d requests such that $P^-(q'')$ is a subset of $P^-(q')$. Then $\hat{L}_{E,q'} \leq \hat{L}_{E,q''}$.*

Proof: If $P^-(q'') \subseteq P^-(q')$ then selection of both q' and q'' result in the same added congestion on all $\mathcal{R} - (P^-(q') - P^-(q''))$. However, selection q'' adds to each $P^-(q') - P^-(q'')$ link d for the selection of q'' and d for the non-selection of q' while selection of q' avoids both these extra demands. If the shortest paths are equivalent then $P^-(q') - P^-(q'') = \emptyset$ and the aggregate loads are equivalent. $\square$

Therefore, given the choice of $R_{r \rightarrow s}$ and $R_{r \rightarrow t}$, whose only difference is that $R_{r \rightarrow s}$ reroutes q' while $R_{r \rightarrow t}$ reroutes q'' , $R_{r \rightarrow s}$ is minimal as its aggregate load is less than that for $R_{r \rightarrow t}$. Therefore, given the choice of equal demand requests, the selection of a maximal shortest path interval request is minimal.

Observation 2.6.2 generalizes Observation 2.6.1 for a subset R of E .

Observation 2.6.2 *Let $q' \in E$ and $q'' \in E$ be two equal demand d requests such that $P^-(q'')$ is a subset of $P^-(q')$.*

Let R be a subset of E such that $d(R) = d$. For all $q \in R$, $P^-(q) \subseteq P^-(q')$ and $P^-(q'') \subseteq P^-(q)$. Then:

$$(i) \hat{L}_{E,q'} \leq \hat{L}_{E,R}$$

$$(ii) \hat{L}_{E,R} \leq \hat{L}_{E,q''}$$

Proof: This observation is proven using the same principles as Observation 2.6.1 first, to compare $P^-(q')$ with the shortest paths for all $q \in R$ and second, to compare the shortest paths for all $q \in R$ versus $P^-(q'')$. $\square$

From this relation, the minimality of $R' = q'$ versus R and the minimality of R versus $R'' = q''$ is defined by the definition of a minimal routing set. All requests in the minimal choice have a maximal shortest path interval. Lemma 2.6.1 assures the feasibility of the minimal choice.

Lemma 2.6.1 *Same as for Observation 2.6.2. $R' = q'$ and $R'' = q''$ are routing sets selected from E .*

(i) *if R'' is a feasible then both R and R' are feasible.*

(ii) *if R is a feasible then R' are feasible.*

Proof: Proven using Observation 2.2.1. $\square$

Figure 2.4 illustrates Lemma 2.6.1. E consists of requests $q1 - q5$ with the choice of $q1$. $R = \{q2, q3, q4\}$ and $q5$ to route 2^5 bandwidth. All choices route 2^5 load to links 1-2 and 6-8. $q1$ is least desirable as it also routes 2^5 load to links 3 and 5. $q5$ routes the same load off these two links so that it is the most desirable choice. R achieves intermediate link 3 and 5 congestion levels which are inferior to the $q5$ values but superior to the $q1$ values.

Lemma 2.6.1 allows the following definition:

Definition 2.6.2 (maximal shortest path interval constraint) *Let R_1 be the minimal routing set of two or more routing set options as defined by Lemma 2.6.1. Therefore, all the requests R_1 have a maximal shortest path interval according to Observation 2.6.1. Therefore, all the requests R_1 satisfy the maximal shortest path interval constraint.*

The maximal shortest path interval constraint is an added constraint on the selection of requests for bin packing by $RLOAD2-PART[K]$. Therefore the bin packing constraints are the B-zone bin, the F-zone bins and the maximal shortest path interval constraint.

Figure 2.4: Maximal Shortest Path Intervals

2.7 Conclusion

This chapter has established the preliminary concepts and definitions necessary for the exploration of RLP2[k] solutions.

Chapter 3

RLP2[k] Problem Solving Principles

3.1 Introduction

This chapter introduces concepts applicable to the solution of RLP2[k] problems. As such, these principles may be used to derive solutions to the general form of RLP2.

Section 3.2 defines the RLP2[k] Bin Packing problem (RLP2[k]-BPP). RLP2[k]-BPP is a Subset Sum problem which packs request bandwidth from the maximum load link eligible set $E_{a,r}$ into the B-zone and F-Zone bins within the maximal subinterval shortest path constraint. For this purpose, the RLOAD2-BPACK[k] algorithm calls the RLP2[k] Partition Algorithm RLOAD2-PART[k] to perform the actual bin packing. RLOAD2-PART[k] partitions an eligible set E into a selection set S of requests to be rerouted and a rejection set X of requests not to be rerouted. RLOAD2-PART[k] does not generally achieve an optimal packing of E by a single partition. Therefore, RLOAD2-BPACK[k] repartitions $E_{a,r}$ by 'de-selecting' a single request from S to X . E is revised for the rejection of the de-selected request. RLOAD2-PART[k] is called to repartition $E_{a,r}$ by partitioning the updated E . The repartition process is studied more closely for RLP2[2].

RLOAD2-PART[k] packs $q \in E$ in greedy fashion of $d(q)$. Requests are first checked that they satisfy the maximum subinterval shortest path constraint on each of the bin constraints. Requests that are non-maximal on one or more bins are rejected as nonfeasible for the current partition. Qualify-

ing requests, which may be rerouted without overflowing a bin capacity, are selected. This partitioning process is repeated until all requests on E have either been selected or rejected.

This thesis does not solve the general case of RLP2[k]-BPP. Instead, it explores possible solutions by solving two simplified versions RLP2[1]-BPP and RLP2[2]-BPP.

Section 3.3 introduces preliminary definitions and concepts relating to the selection of routing sets from the set of partitions of $E_{a,r}$ produced by RLOAD2-BPACK[k]. These routing sets are subsets of or the 'segments' of routing sequences represented by the paths from the root to the terminal nodes of the alpha tree. Of particular importance is the alpha routing set $A_{r \rightarrow s} = A_t + \alpha_s$, which maps from routing r to s , and relocates the maximum load link. After A_t reduces the bin y capacity to less than $d(\alpha_s)$, routing of the *borrow request* α_s relocates the maximum load link from a to y .

Section 3.4 presents a criterion for the recognition of the minimum independent congestion $L_{min,ay}$ on link pair $\{a,y\}$ with a the maximum load link. This criterion is particularly useful when y is the B-zone maximum load link b . In this case, it identifies the minimum achievable congestion $L_{min,r}$ for given initial routing r . A second criterion allows identification of an exact packing of bin y . However, this latter criterion is less useful as a zero terminal capacity also denotes the same condition.

Section 3.5 sets exclusive upper bounds on the bandwidth selected from each of $E_{ay,r}$ and $E_{ya,s}$ after alpha routing set $A_{r \rightarrow s}$ routes from r to s . These lower bounds apply when the shortest path of the $A_{r \rightarrow s}$ borrow request α_s and the aggregate path of $E_{ya,s}$ are non-intersecting allowing the application of Lemma 2.5.1 Corollary 2.5.1.1. The value of upper bounds is established at $d(\alpha_s)$.

Section 3.6 defines an alternate routing set $\tilde{R}_y$ to reduction set $\tilde{R}$ for maximum load link a and bin y . Reduction set $\tilde{R} \subseteq E_{a,r}$ reduces the bin y capacity $C_{ay,r}$ to less than 2^x , non-negative integer x . $\tilde{R}_y$ achieves the same result by first routing $A_{r \rightarrow t}$ relocating the maximum load link from a to y then routing $A_{t \rightarrow r}$ relocating the maximum load link back from y to a . The demand of the $A_{t \rightarrow r}$ borrow request α_r is 2^x .

$\tilde{R}'_y$ and $\tilde{R}'$ are defined to be $\tilde{R}_y$ and $\tilde{R}$ with the $A_{r \rightarrow t}$ subset A_t , common to both, removed. Feasibility conditions are established for routing $\tilde{R}'_y$ instead of $\tilde{R}'$. To be feasible, $\tilde{R}'_y$ must satisfy a minimum of one of two feasibility requirements. These two feasibility requirements assure the possibility that

a lower congestion can be achieved on some $\ell \in P^-(\alpha_t)$ where α_t is the $A_{r \rightarrow t}$ borrow request which also belongs to $\tilde{R}_y'$. Satisfaction of either of these feasibility requirements establishes $A_{r \rightarrow t}$ as an alternate alpha routing set.

3.2 The RLP2[k] Bin Packing Problem

The RLP2[k] Bin Packing problem (RLP2[k]-BPP) is defined as:

Instance: A set $E = E_{a,r}$ of requests q of demand $d(q) = 2^i$, $0 \leq i \leq w$, non-negative integer i , w . An independent zone bin C_{ab} , non-negative integer C_{ab} . An array of interdependent zone bins $C_{a\ell}$: $\ell \in Z_F$, non-negative integer $C_{a\ell}$.

Question: What is a subset S of E such that:

- (i) $\sum d(q) : q \in S$ fills C_{ab} as closely as possible without exceeding C_{ab} .
- (ii) $\sum d(q) : q \in S \leq C_{a\ell}$ for all $\ell \in Z_F$: $\ell \in P^-(q)$?

The sequence 1, 2, 4... 2^w is a *super increasing sequence*. This means that each term is greater than the sum of all the preceding terms. For non-negative integer z , $2^z = \sum_{i=0}^{z-1} 2^i + 1$.

RLP2[k]-BPP is a Subset Sum problem. The problem may be thought of as an exercise to partition E into two disjoint subsets:

- (i) the selection set S of all requests packed into the bins.
- (ii) the rejection set X of non-selected requests.

RLOAD2-BPACK[k] employs the RLP2[k] Partition Algorithm RLOAD2-PART[k] to partition E . RLOAD2-PART[k] is a greedy algorithm that packs requests into the bins in decreasing order of bandwidth demand. For routing r , P_r denotes an arbitrary partition of E . The inputs into RLOAD2-BPACK[k] are the:

- (i) eligible set $E = E_{a,r}$. Because all eligible requests are assigned to their shortest path, k is the maximum length of the assigned path for $q \in E$.

- (ii) bin capacities $C_{ay,r}$. Link y identifies the bin and therefore is used as a *bin identifier*. $y = b$ for the independent zone bin. $y \in Z_F$ for an interdependent zone bin. C denotes the array of input bin capacities. For implied r , the representation may be reduced to C_{ay} .

The *current bin capacity* c_{ay} denotes the updated value of bin y capacity as bandwidth is packed into the bins. c denotes the array of all current bin capacities.

Key aspects of RLOAD2-PART[K] are:

- (i) Requests are selected within the feasible shortest path interval constraint. Let q'' be a request eligible to pack bin c_{ay} . Let E' be the remaining requests eligible to pack bin c_{ay} . q'' is not selected if:
 - (a) $d(E')$ is greater than or equal to c_{ay} and
 - (b) E' includes a bandwidth block B . $d(B) = d(q'')$ for which the aggregate load is less than or equal to $d(q'')$.
- (ii) For RLOAD2-PART[K] there may be multiple options for the selection of equal demand requests not eliminated by bin capacity or the feasible shortest path interval constraints. Excluding coincident shortest paths, the shortest paths for these requests partially overlap so requests pack overlapping intervals of the F-zone bin array. The result is the partition obtained by the initial greedy RLOAD2-PART[K] packing of E does not always yield the optimal packing. A *repartition* of E is required. RLOAD2-BPACK[K] re-executes RLOAD2-PART[K] once for each repartition of E . Chapter 5 addresses eligible set repartitioning.

Outputs from RLOAD2-PART[K] is set E partitioned into set S and X as listed above. Set X is further partitioned into two disjoint subsets that are:

- (i) The *non-maximal shortest path interval set* D . D is the set of requests with non-maximal shortest path intervals.
- (ii) The *borrow set* B . B is the set of requests rejected because their demand exceeded that of one or more bin capacities on their alternate path. Non-maximal shortest path interval requests which also qualify for B are placed onto D rather than B . A request $qx \in B$ is referred to as a *borrow request*.

Requests from $\bar{E}_{ay}$ are placed into one of:

- (i) $D_{ay} \subseteq D$. Requests which fail to comply with the maximal shortest path interval constraint on bin y .
- (ii) $B_{ay} \subseteq B$. Requests for which the demand exceeds c_{ay} but complies with the maximal shortest path constraint on all bins.
- (iii) $S_{ay} \subseteq S$. Requests whose bandwidth packs bin y and do not violate any other bin constraint.

There may exist $q' \in \bar{E}_{ay}$ which qualifies for S_{ay} but cannot be placed there as it is rejected on some other bin capacity. For example, q' may be placed onto D_{az} or B_{az} $z \neq y$ preventing it from been selected for bin y even though there is sufficient capacity there. Therefore q' may not appear in any of S_{ay} , D_{ay} or B_{ay} as the bin z rejection makes it ineligible to pack bin y , effectively removing it from $\bar{E}_{ay}$. Consequentially, the sum of these three sets does not always total $\bar{E}_{ay}$.

These definitions apply to borrow request qx :

Definition 3.2.1 (reduction set) *The reduction set $\tilde{S}(qx)$ is the set of $qs \in S$. $d(qs) \geq d(qx)$. The reduction capacity array $cr(qx)$ is the bin capacity array c after $\tilde{S}(qx)$ is selected during the partition process. $cr(qx, y)$ is the bin y reduction capacity for qx .*

Definition 3.2.2 (independent reduction set) *The independent reduction set $\tilde{S}(qx, y)$ is the set of $qs \in S_{ay}$, $d(qs) \geq d(qx)$.*

$\tilde{S}(qx, y)$ gives an exact packing of $\lfloor \frac{c_{ay}}{d(qx)} \rfloor$. Reduction capacities are calculated in the same manner for requests in D . For the terminal routing set:

Definition 3.2.3 (terminal reduction set) *The terminal reduction set $\tilde{T}$ is the set of all requests in S . The terminal reduction capacity array ct is the array of bin capacities c at the completion of the partition process. The independent terminal reduction set on bin y , $\tilde{T}(y)$, is the set of all selected requests in S_{ay} . $ct(y)$ is the bin y terminal capacity.*

Lemma 3.2.1 addresses the RLOAD2-PART[K] greedy packing method.

Lemma 3.2.1 *Let $\mathcal{P}$ be a partition of E_a . For $qx \in B_{ay}$ and bin y , the following are true:*

- (i) *An exact packing of $C_2 = \lfloor \frac{C_{ay}}{d(qx)} \rfloor$ has occurred.*
- (ii) *qx cannot be used to pack $C_1 = C_{ay} \bmod d(qx)$.*
- (iii) *When qx is placed onto B , all $q \in \bar{E}_{ay}$, $d(q) < d(qx)$ remain available to pack C_1 .*

Proof:

- (i) qx is placed onto B_{ay} because $d(qx) > c_{ay}$ where $c_{ay} = C_1$ at that point. Therefore the C_2 term of C_{ay} has been reduced to 0.
- (ii) Because $d(qx) > C_1$.
- (iii) Self evident from the greedy approach of RLOAD2-PART[K].

□

3.3 Routing Set Selection

Let $\mathcal{P}_r$ be a partition of $E_{a,r}$ for routing r . $\ell_1 = a$. y is some link on $\mathcal{R}$ other than a . This section addresses selection of routing sets from S_{ay} and B_{ay} . These definitions apply.

Definition 3.3.1 (alpha routing set) *Given routing r , partition $\mathcal{P}_r$, bin y . Let $A_{r \rightarrow s} = A_s + \alpha_s$, $\alpha_s \in B_{ay}$. $A_s = \tilde{S}(\alpha_s)$. be a non-terminal routing set that relocates the maximum load link ℓ_1 to y . $A_{r \rightarrow s}$ is a non-terminal routing set referred to as an alpha routing set. $A_{r \rightarrow s}$ maps from routing r to s .*

Lemma 3.3.1 lists properties of alpha routing sets:

Lemma 3.3.1 *$A_{r \rightarrow s}$ is selected according to Definition 3.3.1. Let $C_2 = \lfloor \frac{C_{ay}}{d(\alpha_s)} \rfloor$, $C_1 = C_{ay} \bmod d(\alpha_s)$. Then:*

- (i) *$\tilde{S}(\alpha_s, y)$ represents an exact packing of C_2 .*
- (ii) *After $A_{r \rightarrow s}$ is routed, $C_{ya,s} = d(\alpha_s) - C_1 < d(\alpha_s)$.*

- (iii) The selection of $q \in E_R$, $d(q) \geq d(\alpha_s)$ cannot reduce C_1 for r or C_{ya} for s .

Proof:

(i) Shown by Lemma 3.2.1(i).

(ii) After $A_{r \rightarrow s}$ is routed $C_{ya,s} =$

$$\left\lfloor \frac{(L_{y,r} + C_2 + d(\alpha_s)) - (L_{a,r} - C_2 - d(\alpha_s))}{2} \right\rfloor$$

$$= C_2 + d(\alpha_s) - C_{ay} = d(\alpha_s) - C_1$$

C_1 is by definition $< d(\alpha_s)$.

(iii) The resolution of $d(q) \geq d(\alpha_s)$ is too large to mathematically permit the reduction of these capacities.

□

Lemma 3.3.2 shows that a feasible alpha routing set includes the reduction set for its alpha request rather than only the independent reduction set.

Lemma 3.3.2 $A_{r \rightarrow s}$ is an alpha routing set selected according to Definition 3.3.1. Let S' be any subset of the reduction set common to a and y . $S' \subseteq \tilde{S}(\alpha_s) - \tilde{S}(\alpha_s, y)$.

Then $A_{r \rightarrow s'} = A_{r \rightarrow s} - S'$ is not a feasible routing set.

Proof: Compare routing $A_{r \rightarrow s}$ and $A_{r \rightarrow s'}$. Routing $A_{r \rightarrow s'}$ adds $xd(\alpha_s) = d(S')$, positive integer x , load to both $L_{a,s}$ and $L_{y,s}$ increasing both these loads to beyond their s values. For s' , S' must remain eligible to reach a feasible routing. In that case, RLOAD2-PART[K], being a greedy algorithm, would reselect all of S' before selecting any requests of demand less than $d(\alpha_s)$. □

These definitions apply to alpha routing set $A_{r \rightarrow s}$:

Definition 3.3.2 (reduction set A_s) The A_s term is the reduction set. Routing this set alone leaves ℓ_1 unchanged.

Definition 3.3.3 (borrow request α_s) The α_s term is the borrow request. α_s is the only request selected from B_{ay} . Its demand is less than or equal to that of each of the requests in A_s .

After A_s is routed c_{ay} is reduced to $C_1 < d(\alpha_s)$ (Lemma 3.3.1(ii)). Routing α_s represents a borrow of $d(\alpha_s)$ by y from a . The goal is to more closely approach $L_{min,ay}$ by replacement or repayment of a maximum of $d(\alpha_s) - C_1$ load back to a .

The non-terminal routing set A_{r-s} relocates the maximum load link ℓ_1 from a to y .

Either a non-terminal or terminal routing set may reduce $L_{ay,r}$:

non-terminal routing set Select alpha routing set A_{r-s} . For s , the RLOAD2-BPACK[K] packing process and the routing set selection options are repeatable for $\ell_1 = y$ defining a recursive process. Multiple alpha routing set candidates may exist.

terminal routing set R_{r-term} is assigned to the selection set which achieves the optimum packing of $E_{a,r}$. R_{r-term} maps to a terminal routing r_{term} . R_{r-term} does not relocate the maximum load link. Only one terminal reduction set is needed.

The selection of requests in D is nonfeasible during the greedy packing sequence (Lemma 2.6.1). Therefore, their inclusion in routing sets selected according to this packing sequence is also nonfeasible. Hence, requests in D are not considered by the routing set selection process.

The next definitions assign properties to alpha routing sets.

Definition 3.3.4 (minimum alpha routing set) Let $A_{r-s} = A_s + \alpha_s$ such that α_s is a minimum demand $q \in B_{ay}$. A_{r-s} is a minimum alpha routing set. Otherwise, A_{r-s} is a non-minimum alpha routing set.

Definition 3.3.5 (maximal shortest path interval alpha routing set) Let $A_{r-s} = A_s + \alpha_s$ such that there does not exist $q \in B_{ay}$ where $d(q) = d(\alpha_s)$ and $P^-(\alpha_s) \subset P^-(q)$. A_{r-s} is a feasible shortest path interval alpha routing set.

3.4 Minimum Independent Congestion

For routing r , partition $\mathcal{P}_r$ and bin y , this section presents criteria to identify the attainment of $\bar{L}_{min,ay}$. All requests in $\bar{E}_{ay}$ of demand less than the minimum demand $qx \in B_{ay}$ have been selected. Lemma 3.4.1 addresses the case where $qx \in B_{ay}$. This lemma establishes a criterion to identify nonfeasible minimum alpha routing sets. Lemma 3.4.1 Corollary 3.4.1.1 establishes a condition to identify the minimum congestion for r . Lemma 3.4.2 applies when $qx \in D_{ay}$.

Lemma 3.4.1 *Given routing r , partition $\mathcal{P}_r$ and bin y . $A_{r-s} = A_s + \alpha_s$ is a minimum alpha routing set.*

$ct(y)$ is greater than 0. Let E' be the set of all requests in $\bar{E}_{ay}$ of demand less than $d(\alpha_s)$. $E' \subseteq S_{ay}$. If:

- (i) $ct(y) \leq d(\alpha_s/2)$ then $\bar{L}_{min,ay} = \bar{L}_{ay}$.
- (ii) $ct(y) > d(\alpha_s/2)$ then routing A_{r-s} reduces $\bar{L}_{ay}$.

Proof: Let $r_a = (\bar{L}_{ay,r} - S_{ay}) \bmod d(\alpha_s)$. Let $r_y = (\bar{L}_{ya,r} + S_{ay}) \bmod d(\alpha_s)$. Because $E' \subseteq S_{ay}$, $r_a = 0$. Therefore, $\bar{L}_{ay,r} - S_{ay}$ is evenly divisible by $d(\alpha_s)$ in both cases. Figures 3.1 and 3.2 depict the link load differentials for both cases. Link y shows r_y . Link a shows the 1 or 2 matching $d(\alpha_s)$ blocks.

- (i) Then Observation 2.3.3 states that the number of integral $d(\alpha_s)$ blocks in $\bar{L}_{ay,r} - S_{ay}$ exceeds the number in $\bar{L}_{ya,r} + S_{ay}$ by at most 1. Because $ct(y) \neq 0$, this excess is always 1. α_s represents this odd block on a . α_s is matched by r_y on y . At best α_s and r_y can be exchanged between a and y . Hence the independent congestion cannot be reduced. Figure 3.1 shows this case. $z2^x$, non-negative z , load matches on a and y . The odd 2^x block on a must appear on either a or y without being split.
- (ii) Then it is verifiable that there are exactly 2 unmatched $d(\alpha_s)$ blocks on a versus r_y on y . Therefore, the α_s borrow results in an automatic reduction of the $\{a,y\}$ independent congestion. In Figure 3.2, moving the α_s block from a to y automatically reduces the a, y congestion to less than $(z + 2)2^x$.

□

Figure 3.1: Minimum Independent Congestion

Figure 3.2: Non-Minimum Independent Congestion

Corollary 3.4.1.1 establishes the existence of $L_{min,r}$ when the Lemma 3.4.1(i) condition applies to the independent zone maximum load link $\ell_2 = b$.

Corollary 3.4.1.1 *Same as for Lemma 3.4.1(i) except $y = b$. Then $L_{a,r} - d(S)$ is the minimum attainable congestion, $L_{min,r}$.*

Proof: In this case, $\bar{E}_{ay} = E_{a,r}$. By Lemma 3.3.1(i) an exact packing of $\lfloor \frac{C_{ab,r}}{d(\alpha_s)} \rfloor$ has occurred. All requests of demand less than $d(\alpha_s)$ have been selected. Therefore, no further reduction of the independent zone capacity is possible. This denotes the minimum attainable congestion for routing r , $L_{min,r}$. $\square$

Lemma 3.4.2 establishes a condition for an exact packing of C_{ay} when $qx \in D_{ay}$.

Lemma 3.4.2 *Given routing r , partition $\mathcal{P}_r$ and bin y . Let qx be a minimum demand request in D_{ay} . Let E' be the set of all requests in $\bar{E}_{ay}$ of demand less than $d(qx)$. $E' \subseteq S_{ay}$. Then:*

(i) $ct(y) = 0$. *An exact packing of bin y has occurred.*

(ii) $cr(qx, y) = d(E')$.

Proof: qx was placed onto D_{ay} because of shortest path interval non-feasibility. This means that there was sufficient bandwidth in E' to pack $cr(qx, y)$. Because $E' \subseteq S_{ay}$, $cr(qx, y)$ was reduced to 0. $\square$

Corollary 3.4.2.1 applies Lemma 3.4.2 to the independent zone maximum load link $\ell_2 = b$ to identify an exact packing of the independent zone capacity and therefore, the attainment of $L_{min,r}$.

Corollary 3.4.2.1 *Same as for Lemma 3.4.2 except $y = b$. Then an exact packing of bin $C_{ab,r}$ has occurred. $L_{a,r} - d(S)$ is the minimum attainable congestion $L_{min,r}$.*

Proof: Same as for Lemma 3.4.2 except $y = b$. Therefore, an exact packing of bin $C_{ab,r}$ has occurred. Then $L_{a,r} - d(S)$ is the minimum attainable congestion, $L_{min,r}$ (Lemma 2.4.1). $\square$

3.5 Alpha Routing Set Bandwidth Limits

In this section, upper bounds are placed on bandwidth routed subsequently to $A_{r \rightarrow s}$ in a feasible routing sequence. $\bar{E}_{ay,r}$ and $\bar{E}_{ya,r}$ are path independent. Lemma 3.5.1 establishes a $d(\alpha_s)$ exclusive upper bound on the bandwidth routed from each of $\bar{E}_{ay,r}$ and $\bar{E}_{ya,r}$ after $A_{r \rightarrow s}$ is routed. Therefore, $d(\alpha_s)$ is established as an exclusive upper bound on the demand of requests in $\bar{E}_{ya,s}$ and $\bar{E}_{ay,s}$.

Lemma 3.5.1 *Given routing r , partition $\mathcal{P}_r$ and bin y . $A_{r \rightarrow s} = A_s + \alpha_s$ is routed. For s , ℓ_1 relocates to y . $\bar{E}_{ay,r}$ and $\bar{E}_{ya,r}$ are path independent. Let R be a minimum routing set $R_{r \rightarrow \min}$ which includes $A_{r \rightarrow s}$ as a subset. For $R - A_{r \rightarrow s}$, let R' be the subset of requests which belong to $\bar{E}_{ay,r}$ and R'' be the subset of requests which belong to $\bar{E}_{ya,r}$. Then:*

$$(i) \ d(R') < d(\alpha_s)$$

$$(ii) \ d(R'') < d(\alpha_s)$$

Proof: Route $A_{r \rightarrow s}$. Because $\bar{E}_{ay,r}$ and $\bar{E}_{ya,r}$ are path independent, the routing of an equal demand bandwidth block from each of these eligible sets is nonfeasible by Lemma 2.5.1 Corollary 2.5.1.1. Therefore, $R' + \alpha_s$ and R'' cannot have any matching equal demand request subblocks.

For r , A_s reduces $C_{ay,r}$ to $C_{ay,r} \bmod d(\alpha_s)$ (Lemma 3.3.1(i)). For s , $C_{ya,s} < d(\alpha_s)$ (Lemma 3.3.1(ii)). These capacities assure that the number of integral $d(\alpha_s)$ blocks on a and y differs by at most 1 (observation 2.3.3). If one of R' or R'' has a bandwidth block that is an integral multiple of $d(\alpha_s)$, then the other must have a compensating block to restore this balance to L_a and L_y . Such a sequence cannot be feasible by Lemma 2.5.1 Corollary 2.5.1.1. Therefore, α_s is the only integral $d(\alpha_s)$ block in either $R' + \alpha_s$ or R'' . $\square$

Figure 3.3 illustrates Lemma 3.5.1. This figure shows the difference between the link a and y loads after A_s has been routed from a to y . By routing integral 2^x multiples only, the congestion cannot be reduced below the minimum of the load on a and the load on y after α_s is routed onto y . Therefore all other integral 2^x multiples routed from either link must be compensated by an equivalent routing from the other link. Because $\bar{E}_{ay,r}$ and $\bar{E}_{ya,r}$ are path independent, this additional routing cannot be feasible.

Figure 3.3: Independent Bandwidth Routing Bounds

3.6 Alternate Routing Sets

For partition $\mathcal{P}_r$, $A_{r \rightarrow t}$ is routed relocating ℓ_1 from link a to y . For routing t , partition $\mathcal{P}_t$, alpha routing set $A_{t \rightarrow r}$ is routed. $A_{t \rightarrow r}$ relocates ℓ_1 back to link a from y . It is shown that $\tilde{R}_y = A_{r \rightarrow t} + A_{t \rightarrow r}$ routes a net load from a to y equivalent to $\mathcal{P}_r$ reduction set $\tilde{R} \subseteq S$. $\tilde{R}_y$ is an *alternate routing set* to $\tilde{R}$.

Because bandwidth is being routed between a and y , only subsets of $\tilde{E}_{ay,r}$ and $\tilde{E}_{ya,t}$ need to be considered. Requests from E_{ay} set have the same routing effect on both L_a and L_y so that selection of $\tilde{R}$ and $\tilde{R}_y$ is independent of these requests. Therefore the requests in $\tilde{R}$ and $\tilde{R}_y$ are limited to $\tilde{E}_{ay,r}$ and $\tilde{E}_{ya,t}$ with the common set requests excluded. This approach simplifies the formulation of concepts and notation. The definitions and notation below apply to the discussion to follow on alternate routing sets.

Lemma 3.6.1 *Given routing r , partition $\mathcal{P}_r$ and bin y . Let $\tilde{R} \subseteq \tilde{E}_{ay,r}$ be a reduction set. Let $\tilde{R}_y$ be a subset of $E_{\mathcal{R}}$: $\tilde{R}_y$ is not a subset of $\tilde{E}_{ay,r}$. $\tilde{R}_y$ reduces $L_{a,r}$ and increases $L_{y,r}$ by $d(\tilde{R})$. All other definitions are according to Table 3.1. Then $\tilde{R}_y$ exists if and only if:*

- (i) *there exists $A_{r \rightarrow t}$: $A_t \subseteq \tilde{R}$ and*
- (ii) *For routing t , partition $\mathcal{P}_t$ of $\tilde{E}_{ya,t}$, there exists $A_{t \rightarrow r}$ such that $d(A_{t \rightarrow r}) = d(\alpha_t) - d(\tilde{R}')$.*

$\mathcal{R}_r$ routing r partition. $\ell_1 = a$. $E = E_{a,r}$.

$[\tilde{R}]$ $\mathcal{P}_r$ reduction set $\tilde{R} \subseteq S_{ay}$. $\tilde{R}$ reduces C_{ay} to $C_{ay} \bmod 2^x$, integer x .
 y $\mathcal{P}_r$ bin.

$A_{r \rightarrow t}$ $\mathcal{P}_r$ alpha routing set. $A_{r \rightarrow t} = A_t + \alpha_t$: $A_t = \tilde{S}(\alpha_t, y)$, $\alpha_t \in B_{ay}$.

$\mathcal{P}_t$ routing t partition. $\ell_1 = y$. $E = E_{y,t}$.

$A_{t \rightarrow r'}$ $\mathcal{P}_t$ alpha routing set. $A_{t \rightarrow r'} = A_{r'} + \alpha_{r'}$: $A_{r'} = \tilde{S}(\alpha_{r'}, a)$, $\alpha_{r'} \in B_{ya}$.

$\tilde{R}_y$ alternate routing set to $\tilde{R}$. $\tilde{R}_y = A_{r \rightarrow t} + A_{t \rightarrow r'}$.

$\tilde{R}'$ $\tilde{R}' = \tilde{R} - A_t$. $\tilde{R}$ without reduction set A_t common to $\tilde{R}_y$.

$\tilde{R}'_y$ $\tilde{R}'_y = \alpha_t + A_{t \rightarrow r'}$. $\tilde{R}_y$ without reduction set A_t common to $\tilde{R}$.

Table 3.1: Definitions - Alternate Routing Sets

Proof:

$A_{r \rightarrow t}$ and $A_{t \rightarrow r'}$ exist. The difference between routing $\tilde{R}$ and $A_{r \rightarrow t} + A_{t \rightarrow r'}$ is the difference between routing $\tilde{R}'$ versus $\alpha_t + A_{t \rightarrow r'}$. Routing the latter routing set reduces L_a by $d(\alpha_t) - (d(\alpha_t) - d(\tilde{R}')) = d(\tilde{R}')$ and adds the same amount to L_y . Therefore, $\tilde{R}_y$ exists. Therefore, $\tilde{R}_y = A_{r \rightarrow t} + A_{t \rightarrow r'}$.

$\tilde{R}_y$ exists. Then for $\mathcal{P}_r$ and bin y both $\tilde{R}$ and $\tilde{R}_y$ reduce L_a and increase L_y by $d(\tilde{R})$. Since $\tilde{R}_y$ is not a subset of $E_{ay,r}$, the only way to achieve this equivalent load transfer is by relocation of ℓ_1 to y and selection of requests from $\tilde{E}_{ya,t}$. Therefore two alpha routing sets are required: $A_{r \rightarrow t}$ to relocate ℓ_1 to y and $A_{t \rightarrow r'}$, selected from $\tilde{E}_{ya,t}$ to relocate ℓ_1 back to a where $L_{a,r'} = L_{a,r} - d(\tilde{R})$ and $L_{y,r'} = L_{y,r} + d(\tilde{R})$. Therefore, $\tilde{R}_y = A_{r \rightarrow t} + A_{t \rightarrow r'}$.

□

Figures 3.4 and 3.5 illustrate $\tilde{R}$ and $\tilde{R}_y$ respectively. The arrows denote the routing of one or more requests. The starting point of each arrow gives the current maximum load link. A vertical arrow denotes a reduction set: a

Figure 3.4: Reduction Set $\tilde{R}$

horizontal arrow denotes a borrow request. In Figure 3.4, routing $\tilde{R}$ reduces $C_{ay,r}$ to $C_{ay,r'}$, which is less than $d(\alpha_{r'})$. In Figure 3.5, routing $\tilde{R}_y$ has the same effect, with $\tilde{R}'_y$ replacing the $\tilde{R}'$ subset of $\tilde{R}$.

Definition 3.6.1 (alternate routing set) Let $\tilde{R}$ and $\tilde{R}_y$ be routing sets as defined by Lemma 3.6.1. $\tilde{R}_y$ is an alternate routing set to $\tilde{R}$.

Corollary 3.6.1.1 establishes $d(\alpha_t)$ as the sum of the $\tilde{R}'$ and $A_{t \rightarrow r'}$ demands. It also establishes $d(\alpha_{r'})$ as the upper bound on the minimum demand request in $\tilde{R}$.

Corollary 3.6.1.1 Same as for Lemma 3.6.1.

$$(i) \quad d(\tilde{R}') + d(A_{t \rightarrow r'}) = d(\alpha_t)$$

(ii) Let q' be the minimum demand request in $\tilde{R}$. Then $d(q') \leq d(\alpha_{r'})$.

Proof:

(i) From Lemma 3.6.1, $d(A_{t \rightarrow r'}) = d(\alpha_t) - d(\tilde{R}')$.

(ii) From Lemma 3.6.1, it is evident that $\tilde{R}$ must contain 1 block of bandwidth of demand $d(\alpha_{r'})$. This requires a minimum of 1 request whose demand is less than or equal to this value.

Figure 3.5: Alternate Routing Set $\tilde{R}_y$

□

$\tilde{R}$ and $\tilde{R}_y$ both reduce $C_{ay,r}$ to the same value which is less than $d(\alpha_{r'})$. With A_t common to both routing sets, only subsets $\tilde{R}'$ and $\tilde{R}'_y$ need to be considered. The following definition applies:

Definition 3.6.2 (alternate link reduction set) Let $\tilde{R}$ and $\tilde{R}_y$ be defined as for Lemma 3.6.1. $\tilde{R}'_y$ is an alternate link reduction set to $\tilde{R}'$. y is the alternate link for $\tilde{R}'_y$.

Routing A_t reduces $C_{ay,r}$ to a value less than $d(\alpha_t)$. $\tilde{R}'$ further reduces this value below $d(\alpha_{r'})$. $\tilde{R}'_y$ has the same effect as $\tilde{R}'$ except that it involves a borrow and repayment of bandwidth.

For $\tilde{R}'_y$ versus $\tilde{R}'$ Lemma 3.6.2 subdivides $\mathcal{R}$ into regions based upon α_t . The minimality of these two sets is compared in each region.

Lemma 3.6.2 Given routing r , partition $\mathcal{P}_r$ and bin y . Let $\tilde{R} \subseteq \tilde{E}_{ay,r}$ be a reduction set. $A_{r-t} = A_t + \alpha_t$ is an alpha routing set which relocates ℓ_1 to y . Let $\tilde{R}_y = A_{r-t} + A_{t-r'}$ be an alternate routing set to $\tilde{R}$ as defined for Lemma 3.6.1.

Compare routing $\tilde{R}'_y$ versus $\tilde{R}'$. Then:

- (i) $\tilde{R}'_y$ routes an equivalent or greater load than $\tilde{R}'$ to all $\ell \in P^+(\alpha_t)$.

(ii) Partition $P^-(\alpha_t)$ into two zones: $Z_{I,t}$, the region of overlap with the aggregate path of $\bar{E}_{ya,t}$, and the corresponding path independent region $\bar{Z}_{I,t} = P^-(\alpha_t) - Z_{I,t}$. Then:

- (a) $\bar{R}'_y$ routes exactly $d(\bar{R}')$ load from all $\ell \in \bar{Z}_{I,t}$.
- (b) $\bar{R}'_y$ routes a minimum of $d(\bar{R}')$ load from all $\ell \in Z_{I,t}$.

Proof:

- (i) Selection of $\bar{R}'_y$ adds a minimum of $d(\alpha_t) - d(\bar{R}'_y) = d(\bar{R}')$ to all $\ell \in P^+(\alpha_t)$. Selection of $\bar{R}'$ adds a maximum of this value to the same region.
- (ii) (a) Selection of $\bar{R}'_y$ removes exactly $d(\alpha_t) - d(\bar{R}'_y) = d(\bar{R}')$ from all $\ell \in \bar{Z}_{I,t}$.
- (b) Selection of $\bar{R}'_y$ removes an initial $d(\alpha_t) - d(\bar{R}'_y) = d(\bar{R}')$ from all $\ell \in Z_{I,t}$. For $\ell \in Z_{I,t}$ and each $q' \in A_{t-r'}$ for which $\ell \in P^-(q')$ an additional $d(q')$ load is removed from ℓ .

In comparison, selection of $\bar{R}'$ removes a maximum of $d(\bar{R}')$ from all of $P^-(\alpha_t)$.

The bounds on the congestion added to $P^+(\alpha_t)$ and on the possible reductions in the sub zones of $P^-(\alpha_t)$ are shown in Figure 3.6 for routing $\bar{R}$ and in Figure 3.7 for routing $\bar{R}'_y$. $\square$

Lemma 3.6.3 states a necessary condition for the feasibility of $\bar{R}'_y$ as an alternate to $\bar{R}'$. This necessary condition is that $\bar{R}'_y$ satisfy a minimum one of the following:

- (i) $Z_{I,t} \neq \emptyset$. That is α_t and $\bar{E}_{ya,t}$ are path interdependent.
- (ii) $\bar{R}'$ does not route $d(\bar{R}')$ load from all $\ell \in P^-(\alpha_t)$.

Absence of either of these conditions precludes the feasibility of $\bar{R}'_y$ as an alternate to $\bar{R}'$.

Figure 3.6: Routing $\tilde{R}'$

Figure 3.7: Routing $\tilde{R}_y$

Lemma 3.6.3 Let $\tilde{R}'_y$ be an alternate link reduction set to $\tilde{R}'$ on bin y as defined by Lemma 3.6.1. Define $Z_{l,t}$ and $\tilde{Z}_{l,t}$ as for Lemma 3.6.2. If:

- (i) $Z_{l,t} = \emptyset$ and
- (ii) If there does not exist $qr \in \tilde{R}'$ such that $P^-(qr) \cap P^-(\alpha_t)$ is nonempty

then $\tilde{R}'_y$ is not a feasible alternate to $\tilde{R}'$.

Proof:

- (i) Elimination of $Z_{l,t}$ precludes the path interdependent load reductions which occur on $\ell \in Z_{l,t}$ for each $q' \in A_{t \leftarrow r}$ where $\ell \in P^-(q')$ (Lemma 3.6.2(ii)). The possibility of these load reductions on $Z_{l,t}$ establish possible feasibility of $\tilde{R}'_y$.
- (ii) If $Z_{l,t} = \emptyset$ then $\tilde{Z}_{l,t} = P^-(\alpha_t)$. Therefore, selection of $\tilde{R}'_y$ removes exactly $d(\tilde{R}')$ from all $P^-(\alpha_t)$ (Lemma 3.6.2(ii)). With the added condition on $qr \in \tilde{R}'$, selection of $\tilde{R}'$ removes exactly this value from all $P^-(\alpha_t)$. Therefore, the advantage of $\tilde{R}'_y$ over $\tilde{R}'$ on $P^-(\alpha_t)$ is eliminated. Since $\tilde{R}'_y$ may route a greater load to some $\ell \in P^+(\alpha_t)$ than $\tilde{R}'$ (Lemma 3.6.2(i)), $\tilde{R}'_y$ cannot be feasible as an alternate to $\tilde{R}'$.

□

Lemma 3.6.3 is illustrated by Figures 3.6 and 3.7. (i) reduces the $Z_{l,t}$ region to $\emptyset$ in both figures eliminating its load reduction advantage to $\tilde{R}'_y$. (ii) revises the $\tilde{Z}_{l,t}$ congestion reduction to $d(\tilde{R}')$ from a maximum of $d(\tilde{R}')$ in Figure 3.6. This equals the $d(\tilde{R}')$ value for $\tilde{R}'_y$ shown in Figure 3.7. All possible advantage to routing $\tilde{R}'_y$ has now disappeared.

The requirement that $Z_{l,t} \neq \emptyset$ is the *path interdependent* or *PI* requirement for $\tilde{R}'_y$ feasibility. The Lemma 3.6.3(ii) proof can be seen in a different way. Routing $\tilde{R}'_y$ is equivalent to routing a $d(\tilde{R}')$ routing set R_1 from all of $P^-(\alpha_t)$. If there exists $qr \in \tilde{R}'$ such that $P^-(qr) \cap P^-(\alpha_t)$ is nonempty, then $\tilde{R}'$ has a non-maximal shortest path interval in the $P^-(\alpha_t)$ zone. Otherwise, both R_1 and $\tilde{R}'$ remove the same bandwidth from this zone. Therefore, the second requirement can be stated as the *maximal shortest path interval* or *MSI* requirement for $\tilde{R}'_y$ feasibility. Based upon Lemma 3.6.3, an *alternate alpha* routing set is defined:

Definition 3.6.3 (alternate alpha routing set) *Let $\tilde{R}_y$ be an alternate routing set to $\tilde{R}$. $A_{r \rightarrow t} = A_t + \alpha_t$ is an alpha routing set which relocates ℓ_1 from a to y . If $\tilde{R}'_y$ satisfies a minimum of one of the Lemma 3.6.3 feasibility requirements then $A_{r \rightarrow t}$ is an alternate alpha routing set. A PI alternate alpha set satisfies the PI feasibility requirement. A MSI alternate alpha set satisfies the MSI feasibility requirement.*

Both minimum and non-minimum alpha routing sets qualify as alternate alpha routing sets. Minimum alpha routing sets are routed most of the time. The Lemma 3.6.3 feasibility requirements are useful for the elimination of prospective non-minimum alpha routing sets.

Lemma 3.6.4 compares two alpha routing sets $A_{r \rightarrow s'}$ and $A_{r \rightarrow s}$, where $d(\alpha_{s'}) \geq d(\alpha_s)$. If $A_{r \rightarrow s} - A_{s'}$ forms a bandwidth block which:

- (i) has equal demand to $d(\alpha_{s'})$ and
- (ii) routes an aggregate load greater than or equal to $\alpha_{s'}$,

then $A_{r \rightarrow s}$ cannot be feasible. $A_{r \rightarrow s'}$ may be routed instead of $A_{r \rightarrow s}$.

Lemma 3.6.4 *Given routing r , partition $\mathcal{P}_r$ and bin y . $A_{r \rightarrow s} = A_s + \alpha_s$ is an alpha routing set. $A_{r \rightarrow s'} = A_{s'} + \alpha_{s'}$: $d(\alpha_{s'}) \geq d(\alpha_s)$ is a second alpha routing set. Let $S' = A_{r \rightarrow s} - A_{s'}$. If:*

- (i) $d(S') = d(\alpha_{s'})$
- (ii) For all $qs \in S'$, $P^-(qs) \subseteq P^-(\alpha_{s'})$

then $A_{r \rightarrow s}$ is not feasible.

Proof: Assume that both these conditions are true. The difference between routing $A_{r \rightarrow s}$ and $A_{r \rightarrow s'}$ is the difference between routing S' and $\alpha_{s'}$. Compare routing S' and $\alpha_{s'}$. Observation 2.6.2 states that S' always routes an aggregate load greater than or equal to that of $\alpha_{s'}$. Therefore S' is not feasible according to Lemma 2.6.1. $\square$

3.7 Conclusion

RLP2[k]-BPP has been defined. RLOAD2-BPACK[k] is the algorithm that solves RLP2[k]-BPP. RLOAD2-BPACK[k] calls the RLOAD2-PART[k] algorithm to select eligible bandwidth in greedy fashion within a set of bin constraints and the maximal shortest path interval constraint. RLOAD2-PART[k] effectively partitions an eligible set E into a selection set S and rejection set X . RLOAD2-BPACK[k] may require multiple partitions of $E_{a,r}$ to obtain an optimal packing within the specified constraints.

Non-terminal and terminal routing sets are selected from the selection and rejection sets of the partitions output by RLOAD2-BPACK[k]. Non-terminal routing sets are represented as alpha routing sets $A_{r \rightarrow s}$, which route from routing r to s . This mapping is represented by a mapping from node r to node s on the alpha tree where the RLOAD2-BPACK[k] process is repeated. Terminal routing sets represent the final segment on a routing sequence and, as such, do not map to a new alpha tree node.

Section 3.4 presents a criterion to recognize the minimum independent congestion on bin y , $L_{min,ay}$. This allows the termination of searches for $L_{min,ay}$. This is particularly useful for the recognition of $L_{min,r}$ when y is the maximum load link b . Because $L_{min,r}$ is the minimum congestion for the alpha subtree with root node r , the search of this subtree may be completed.

The upper bounds established in Section 3.5 on independent bandwidth selected after routing $A_{r \rightarrow s}$ restrict eligible request sets thereby reducing the search space for r_{opt} .

Section 3.6 shows that routing an alternate routing set $\tilde{R}_y$ may achieve link load reductions not possible by routing of a reduction set $\tilde{R}$. To be a feasible alternative to $\tilde{R}$, $\tilde{R}_y$ must satisfy the Path Interdependent Requirement or the Maximal Shortest Path Interval Requirement. Alternate routing sets or alternate alpha routing sets are an effective means of resolving interference on interfered requests. At the same time, the two feasibility requirements represent a means to limit the number of possible candidates.

Chapter 4

RLP2[1]

4.1 Introduction

This chapter proposes RLOAD2[1] as an $O(n + m \log_2 m)$ solution to RLP2[1].

Section 4.2 defines RLP2[1]-BPP. It shows that RLOAD2-PART[1] gives an optimal solution to RLP2[1]-BPP in $O(m)$ time without request sorting and in $O(m \log_2 m)$ time with request sorting.

Section 4.3 gives a overview of RLOAD2[1]. Section 4.4 proves the correctness of RLOAD2[1] as a solution to RLP2[1]. In section 4.5, the computational complexity of RLOAD2[1] is shown to be $O(n + m)$ without request sorting and $O(n + m \log_2 m)$ with request sorting. Section 4.6 is an algorithmic description of RLOAD2[1].

4.2 The RLP2[1] Bin Packing Problem

The RLP2[1] Bin Packing problem (RLP2[1]-BPP) is defined as:

Instance: A set $E = E_{a,r}$ of m requests q of demand $d(q) = 2^i$, $0 \leq i \leq w$, non-negative integer i , w . A bin C , non-negative integer C .

Question: What is a subset S of E such $\sum d(q) : q \in S$ fills C as closely as possible without exceeding C .

Because the shortest path length for $q \in E$ is always 1, the F-zone is nonexistent for RLP2[1]-BPP. Therefore C is the B-zone capacity $C_{a,b}$.

The RLP2[1] Eligible Set Partition Algorithm RLOAD2-PART[1] is:

```

RLOAD2-PART[1]( $E, m, C, S, X$ )
1 Sort  $E$  into descending order of  $d(q)$ ,  $q \in E$ ;
2  $c \leftarrow C$ ;
3 while  $E \neq \text{NIL}$ 
4     Remove the first  $q$  from the sorted  $E$ ;
5     if  $d(q) \leq c$ 
6         then place  $q$  onto  $S$ ;  $c \leftarrow c - d(q)$ ;
7         else place  $q$  onto  $X$ ;
Return:

```

Line 1 sorts E into descending order of $d(q)$. Line 2 initializes the current capacity c to C . Lines 3-7 pack bandwidth from E into c in greedy fashion. Line 4 removes the maximum demand request q from E . If $c \geq d(q)$ then line 6 places q on S and updates c ; else line 7 places q on X .

Lemma 4.2.1 states that RLOAD2-PART[1] gives an optimal packing of C :

Lemma 4.2.1 RLOAD2-PART[1] gives an optimal packing of bin capacity C .

Proof: Assume that the algorithm is optimal to the point at which $q \in E$ is scanned. When q is scanned, c is the remaining capacity. Either:

$c \geq d(q)$ Then $d(q)$ can be packed into c . q is placed onto S . The algorithm remains optimal. If the total demand of the remaining items on E is less than or equal to $c - d(q)$ then all these items are moved to S and the algorithm terminates with an optimal packing.

$c < d(q)$ Then $d(q)$ cannot be packed into c . The $\lfloor \frac{C}{d(q)} \rfloor$ component of C has been packed exactly (Lemma 3.2.1(i)) and q is added to the X list. All q , $d(q) < c$ remain on E , eligible for packing Lemma 3.2.1(iii). The algorithm remains optimal.

From the above proof, RLOAD2-PART[1] is optimal throughout the packing sequence. $\square$

Therefore, RLOAD2-PART[1] gives an optimal packing of $E_{a,r}$ into C_{ab} , the only RLP2[1]-BPP bin constraint. Therefore, RLOAD2-PART[1] solves RLP2[1]-BPP.

The computational complexity of sorting E is $O(m \log_2 m)$. During bin packing, each $q \in E$ is checked once and only once for $O(m)$ computational complexity. With sorting, this value increases to $O(m \log_2 m)$.

4.3 Rload2[1] Overview

An outline of RLOAD2[1] is:

- 1 - **initialization:** Assign all requests to their shortest path. This is the initial routing r_{init} . Initialize the current routing r to r_{init} . Let B_d be the exclusive upper bound on the demand of requests in the eligible list. Initialize B_d to infinity.
- 2 - **maximum load link:** Determine the maximum load link $\ell_1 = a$ on $\mathcal{R}$ for r . Choose arbitrarily from links with equal load.
- 3 - **independent zone maximum load link:** Determine the maximum load link $\ell_2 = b$ in the independent zone of a for r . Choose arbitrarily from links with equal load.
- 4 - **capacity:** Determine the independent zone capacity $C_{ab} = \lfloor \frac{L_a - L_b}{2} \rfloor$.
- 5 - **eligible set:** Assign to the eligible set for E_a the set of all requests $q \in \mathcal{Q}$ meeting the following criteria:
 1. $d(q) < B_d$
 2. $a \in P(q)$
- 6 - **Rload2-Part[1]:** Call RLOAD2-PART[1] to partition E_a into S and X by packing C_{ab} .
- 7 - **routing sets:** Assign the terminal routing set R_{r-term} to S . Find the minimum demand request α_s in X . Assign the non-terminal routing set R_{r-s} to the minimum alpha routing set $A_{r-s} = A_s + \alpha_s$, reduction set $A_s = \{q \in S : d(q) \geq d(\alpha_s)\}$.
- 8 - **termination:** If one of:
 1. $X = \emptyset$. that is no non-terminal routing set exists.
 2. $d(A_s) + d(\alpha_s) = B_d$.

3. $C_{ab} - d(S) \leq d(\alpha_s/2)$. $R_{r \rightarrow \text{term}}$ routes to r_{opt} .

then route the terminal routing set. For all $q \in R_{r \rightarrow \text{term}}$, assign $P(q)$ to $P^+(q)$. Accept $L_a - d(S)$ as L_{opt} . Terminate the algorithm.

9 - **reiteration:** Route to routing s . For all $q \in R_{r \rightarrow s}$, assign $P(q)$ to $P^+(q)$. Recalculate L_ℓ , $\ell \in \mathcal{R}$. Assign B_d to $d(\alpha_s)$. r now becomes s . Return to step 2.

Each non-terminal routing r is represented on the RLOAD2[1] alpha tree by $\text{anode}[r]$. $\text{anode}[r_{init}]$ is the root node. S is stored at $\text{anode}[r]$ as the terminal routing set. Each tree edge represents the minimum alpha routing set $A_{r \rightarrow s}$, selected at item 6. $A_{r \rightarrow s}$ maps from parent node $\text{anode}[r]$ to descendant node $\text{anode}[s]$. With only 1 possible minimum alpha routing set, there is only 1 possible non-terminal routing set and, therefore, only 1 possible descendant per node. Therefore, the alpha tree is a linear structure. If $A_{r \rightarrow s}$ does not exist, there is no descendant and the search terminates. After the search has been completed, r_{opt} is read from the alpha tree.

4.4 Proof of Correctness

Lemma 4.4.1 RLOAD2[1] finds L_{opt} for RLP2[1]

Proof: The proof is given by this sequence:

1 - **r = r_{init}:** The path length for all requests in E_a is 1. Therefore, the F-zone is $\emptyset$ and the B-zone is $\mathcal{R}$ -a. $L_{opt} \geq \left\lceil \frac{L_a + L_b}{2} \right\rceil$ (Lemma 2.4.1). $B_d \leftarrow \infty$ denotes no limit exists on rerouted bandwidth.

2 - **Rload2-Part[1]:** RLOAD2-PART[1] gives an optimal packing of $C_{ab} = \left\lfloor \frac{L_a - L_b}{2} \right\rfloor$ (Lemma 4.2.1).

3 - **selection of $A_{r \rightarrow s}$:** For all $q \in E_a$, $\delta(q) = 1$. Therefore, $A_{r \rightarrow s}$ is the only minimum non-terminal routing set. Because the F-zone is non-existent, the Lemma 3.6.3 PI and MSI feasibility requirements for a non-minimum alpha routing set cannot be satisfied. Therefore, for each r , the only feasible alpha routing set is $A_{r \rightarrow s}$.

4 - **termination:** For the termination conditions listed in item 7 of the algorithm:

1. $X = \emptyset$. Then no non-terminal routing set exists.
 2. $d(A_s) + d(\alpha_s) = B_d$. Then no minimal non-terminal routing set exists (Lemma 3.5.1(i)).
 3. $C_{ab} - d(S) \leq d(\alpha_s/2)$. Then $A_{r \rightarrow s}$ cannot reduce $\bar{L}_{ab}$ (Lemma 3.4.1(i)). Because C_{ab} is the B-zone congestion, the optimal congestion has been found (Lemma 3.4.1 Corollary 3.4.1.1).
- 5 - $r = s$: An exact packing of $\left\lfloor \frac{C_{ab,r}}{d(\alpha_s)} \right\rfloor$ has occurred (Lemma 3.3.1). Because $\left\lceil \frac{L_a + L_b}{2} \right\rceil$ is a lower bound on L_{opt} and no feasible non-minimum alpha routing sets exist, RLP2[1] reiterates itself for initial routing s .

For s , ℓ_1 relocates to b . No bandwidth block B , $d(B) \geq d(\alpha_s)$ can be routed back to a in a feasible routing sequence (Lemma 3.5.1(ii)). For s , this excludes all requests of demand greater than or equal to $d(\alpha_s)$ from the eligible set. For s , the path length for all requests in E_a is 1, therefore, the F-zone remains $\emptyset$ and the B-zone remains $\mathcal{R} - a$. $B_d \leftarrow d(\alpha_s)$ becomes the new limit on rerouted bandwidth. After the algorithm recalculates E_a and ℓ_2 , the proof repeats itself for s at item 2.

□

4.5 Computational Complexity

During initialization, RLOAD2[1] sorts $\mathcal{Q}$ by terminal node id and into descending order of $d(q)$ for each terminal node. For sorting m requests by standard methods, the overall computational complexity is $O(m \log_2 m)$.

A maximum of $O(w)$ iterations of RLOAD2[1] are required to find L_{opt} . On each iteration, the load of each $\ell \in \mathcal{R}$ is calculated. ℓ_1 and ℓ_2 are determined. A maximum of m requests are packed. If w is regarded as a constant, this gives $O(n + m)$ computational complexity. The $O(m \log_2 m)$ sorting complexity increases this value to $O(n + m \log_2 m)$.

4.6 Rload2[1]

Parameters

Table 4.1 describes the RLOAD2[1] input/output parameters.

Name	Type	Parameter Description
n	input	number of optical nodes (links) on $\mathcal{R}$
m	input	number of requests in $\mathcal{Q}$
i[m]	input	array of request terminal nodes
d[m]	input	array of request bandwidth demands
w	input	$d_{max} = 2^w$
phi[m]	output	path assignment for request q ; 0 = shortest path, 1 = alternate path

Table 4.1: RLOAD2[1] Input/Output Parameters

Global data

This data is applicable to routing r represented by the currently accessed alpha tree node, *anode*. It is updated each time a new alpha tree node (descendant or return to ancestor) is accessed. Therefore, only a single copy of the data is required. Table 4.2 describes the RLOAD2[1] global data.

Data Item Name	Data Item Description	Remarks
phi[m]	routing r for <i>anode</i> : $r = phi$	
L[n]	total load (congestion) on link ℓ	

Table 4.2: RLOAD2[1] Global Data

Alpha Tree

The alpha tree has one tree node *anode* for each alpha routing r . The root node denotes the initial routing r_{init} . Each edge represents an alpha

routing set $A_{p \rightarrow r}$ which maps to routing r from the routing p represented by its parent node. $A_{p \rightarrow r}$ is stored in *anode*. Table 4.3 lists the RLOAD2[1] alpha tree node data items.

Data Item Name	Data Item Description	Remarks
Rt.qa	$A_{p \rightarrow r}$ alpha request	not defined on root node
Rt .A_list	$A_{p \rightarrow r}$ reduction set	not defined on root node
Rt .C	$C_{ab,r}$	
c	current capacity for r	RLOAD2-PART[1] execution capacity
pnode	pointer to ancestor node	not defined on root node
dnode	pointer to descendant node	
lnk[a]	$\ell_1 = a$ for r	
lnk[b]	$\ell_2 = b$ for r	
E_list	RLOAD2-PART[1] set $E = E_{a,r}$	
S_list	RLOAD2-PART[1] set S for r	Gives the terminal routing set for routing r
X_list	RLOAD2-PART[1] set X for r	

Table 4.3: RLOAD2[1] Alpha Tree Node

Description:

RLOAD2[1] is now described in a modular format.

Rload2[1] Find RLP2[1] optimal routing.

Inputs RLOAD2[1]inputs $n. k. i. d. w$

Outputs ϕ_i RLP2[1] optimal routing

```

RLOAD2[1]( $n. k. i. d. w$ )
1 INIT- $\mathcal{R}(n. k. i. d. w)$ ;
2 ADD-ALPHA-ROOT( $roota$ );
3 ROUTE( $roota$ );
4 RLOAD2-PART[1]( $roota$ );

```

```

5 ADD-ALPHA-BRANCH( $roota$ );
6 READ-ROPT( $roota$ ):
7 Delete  $anode$  and all its subtree:
  Return:

```

Line 1 initializes the global data structures. Line 2 adds the alpha tree root node. Line 3 routes to r_{init} by initializing the root node data structures. Line 4 performs the RLOAD2-PART[1] bin packing for r_{init} . Line 5 determines if there is a feasible alpha routing set $R_{init \rightarrow s}$ and recursively repeats the RLOAD2-BPACK[1] process for s . Line 6 assigns the optimal routing to the final value of phi . Line 7 deletes the alpha tree.

Init- $\mathcal{R}$ Initialize global data

Inputs RLOAD2[1] inputs n, k, i, d, w

Outputs Global data initialized as below

```

INIT- $\mathcal{R}(n, k, i, d, w)$ 
1 Clear each  $phi[q]$  to 0:
2 Sort  $i[q]$  into ascending order of  $i[q]$ :
  For  $q1, q2, i[q1] = i[q2]$ , sort into descending order of  $d(q)$ :
3 From the  $i, d$  and  $phi$  vectors, calculate  $L[\ell]$  for all  $\ell \in \mathcal{R}$ :
  Return:

```

Line 1 assigns phi to the initial routing r_{init} . $i[q]$ gives the shortest path link for request q . Line 2 sorts the requests, first into ascending order of $i[q]$, then into descending order of $d[q]$. Request sorting facilitates the calculation of link loads and the determination of eligible sets. Line 3 calculates the link loads for r_{init} .

Read-ropt Read optimal routing

Inputs *anode* alpha tree root node.

Outputs *phi* r_{opt} flag settings

Remarks recursive subroutine

```
READ-ROPT(anode)
1 if anode has a descendant dnode
2   then anode  $\leftarrow$  dnode;
3     for  $q \in dnode.Rt.A\_list$ 
4        $phi[q] \leftarrow 1$ ;
5      $phi[dnode.Rt.qa] \leftarrow 1$ ;
6     READ-ROPT(anode);
7   else for:
8     Return;
```

This module traverses the optimal routing sequence. Lines 3–4 set the *phi* flags for non-terminal routing set requests. Line 5 recursively invokes the procedure for a non-terminal node. Line 7 sets the *phi* flags for terminal routing set requests.

Add-alpha-root Add root node to the alpha tree

Inputs none

Outputs *anode* alpha tree root node.

```
ADD-ALPHA-ROOT(anode)
1 ALLOCATE(anode):
2 anode.Rt.qa  $\leftarrow$  NIL;
3 anode.Rt.A_list  $\leftarrow$  NIL;
4 anode.S_list.anode.X_list  $\leftarrow$  NIL;
```

5 $anode.dnode.anode.pnode \leftarrow NIL$;
 Return:

This module allocates and initializes the alpha tree root node.

Route Route from routing p to routing r

Inputs $anode$

Outputs L with r link loads
 r eligible list, capacity, routing zones stored in $anode$

ROUTE($anode$)
 1 $R_list \leftarrow anode.Rt.A_list + anode.Rt.qa$;
 2 From R_list , calculate updated L array for r ;
 3 $anode.lnk[a] \leftarrow a: L[a] \geq L[\ell]: \ell \in \mathcal{R}$;
 4 $anode.E_list \leftarrow \{q \in \mathcal{Q}: anode.lnk[a] \in P^-(q) \text{ and } d(q) < d(anode.Rt.qa)\}$;
 5 $Zb \leftarrow \mathcal{R} - anode.lnk[a]$;
 6 $anode.lnk[b] \leftarrow b: b \in Zb \text{ and } L[b] \geq L[\ell]: \ell \in Zb$;
 7 $anode.Rt.C \leftarrow \lfloor \frac{L[a] - L[b]}{2} \rfloor$;
 Return:

This module routes from p to r using $A_{p \rightarrow r}$ stored in $anode$. Line 1 assigns $A_{p \rightarrow r}$ to R_list . Line 2 updates the L array for r . Line 3 assigns the maximum load link for r ; ties are broken arbitrarily. Line 4 reads the eligible request set E_list . Line 6 assigns the independent zone maximum load link; ties are broken arbitrarily. Line 7 assigns the independent zone capacity.

Add-alpha-branch Add alpha tree branch

Inputs *anode* ancestor of branch root node.

Outputs *anode* alpha tree with added branch (may be null).

Remarks recursive subroutine

```

ADD-ALPHA-BRANCH(anode)
1 READ-ALPHA-ROUTING-SET(anode, A_list, qa);
2 if qa ≠ NIL
3   then ADD-ALPHA-NODE(anode, dnode, A_list, qa);
4     ROUTE(dnode);
5     RLOAD2-PART[1](dnode);
6     ADD-ALPHA-BRANCH(dnode);
7     REVERSE-ROUTE(dnode);
Return:

```

Line 1 reads the minimal alpha routing set. Line 2 verifies that $A_{r \rightarrow s}$ exists: if not, the descendant node for routing s is not added and the alpha tree recursion terminates. Line 3 adds the descendant node. Line 4 routes from r to s . Line 5 performs the RLOAD2-PART[1] bin packing for s . Line 6 repeats the recursion for routing s . Line 7 returns to routing r from routing s .

Read-alpha-routing-set Read $A_{r \rightarrow s}$

Inputs *anode* represents routing r .

Outputs *qa* minimum alpha request for $A_{r \rightarrow s}$
 A_list $A_{r \rightarrow s}$ reduction set

```

READ-ALPHA-ROUTING-SET(anode.A_list, qa)
1 qa ← q:  $d(q) \leq d(q') : q' \in \text{anode.X\_list}$ :
2 if qa ≠ NIL
3   thenif  $\text{anode.Rt.C} - d(\text{S\_list}) > d(\text{qa})/2$ 
4     then A_list ← {q ∈ anode.S_list:  $d(q) \geq d(\text{qa})$ }:
5     if  $d(\text{A\_list}) + d(\text{qa}) \geq d(\text{anode.Rt.qa})$ 
6       then qa ← NIL:
7   else qa ← NIL:
  Return:

```

Line 1 assigns the minimum demand *X_list* request to α_s . Line 2 checks for the existence of α_s . Line 3 checks if *r* is optimal by Lemma 3.4.1 Corollary 3.4.1.1. If not, line 4 reads the reduction set *A_s* for α_s . Line 5 checks *A_{r→s}* for Lemma 3.5.1(ii) feasibility.

Add-alpha-node Add alpha tree node for routing *s*

Inputs *pnode* ancestor of new node - represents routing *r*
 A_list Reduction set for *A_{r→s}*,
 qa α_s for *A_{r→s}*,

Outputs *anode* new node - represents routing *s*

```

ADD-ALPHA-NODE(pnode.anode.A_list.qa)
1 ALLOCATE(anode):
2 anode.Rt.qa ← qa:
3 anode.Rt.A_list ← A_list:
4 anode.S_list.anode.X_list ← NIL:
5 anode.dnode ← NIL:
6 pnode.dnode ← anode:
7 anode.pnode ← pnode:
  Return:

```

This module adds *anode* to the alpha tree as the descendant of *pnode*. The edge joining the two nodes represents $A_{r \rightarrow s}$.

Line 1 allocates memory space for *anode*. Lines 2-3 assign *anode.Rt* to $A_{r \rightarrow s}$. The remainder of the module sets up node pointers.

Reverse-route Route from *r* to the parent routing *p*.

Inputs *anode* represents routing *r*.

Outputs *L* updated for ancestor of *anode*

Remarks Reverse routing is used only to traverse the alpha tree.
 Reverse routing does not occur as part of a feasible routing sequence.

REVERSE-ROUTE(*anode*)

1 $R_list \leftarrow anode.Rt.A_list + anode.Rt.qa$:

2 From *R_list*, calculate *L* array for *anode.pnode*:

Return:

Lines 1-2 restore the link loads to the *p* values.

Rload2-Part[1] Partition $E_{a,r}$ into *S* and *X*

Inputs *anode* represents routing *r*.
 anode.E_list $E_{a,r}$

Outputs *anode.S_list* RLOAD2-PART[1] set *S*
 anode.X_list RLOAD2-PART[1] set *X*


```

RLOAD2-PART[1](anode)
1 anode.c ← anode.Rt.C;
2 for d ←  $2^w$ : d > 1: d ← d/2
3   P_list ← {q ∈ anode.E_list : d(q) = d};
4   boundb ←  $\lfloor \frac{\text{anode.c}}{d} \rfloor$ ;
5   rd ← MIN(SIZE(P_list), boundb);
6   move rd q ∈ P_list from P_list to anode.S_list;
7   move remainder of P_list to anode.X_list;
8   subtract rd * d from anode.c;
Return:

```

Line 1 initializes *c* to the *r* independent zone capacity. Lines 2-8 pack bandwidth from *E_list* into *c* in greedy fashion. Line 3 moves all *E_list* requests of demand *d* from *E_list* to a holding list, *P_list*. Lines 4-5 determine, based upon *c* and number of available requests on *P_list*, the number of *d* requests to pack onto *S_list*. Line 6 moves these requests from *P_list* to *anode.S_list*. Line 7 moves the excess *d* requests to *anode.X_list*. Line 8 updates *c* for the packed bandwidth.

4.7 Conclusion

RLOAD2[1] is a an $O(n + m \log_2 m)$ optimal solution to RLP2[1]. For all iterations of RLOAD2[1], the F-zone of the maximum load link is nonexistent. All requests eligible for packing have a maximal shortest path interval of 1. Both these factors restrict the complexity of RLOAD2[1].

Chapter 5

Interfered Requests

5.1 Introduction

RLOAD2[1] illustrates principles applicable to solutions to RLP2[k]:

- (i) There is no interdependent zone Z_F of $\ell_1 = a$. For all $\ell \in \mathcal{R} - a$, the common eligible set $E_{a\ell} = \emptyset$. C_{ab} is the only bin packing constraint. The path selection options afforded by the common eligible set are nonexistent.
- (ii) $E_{a,r}$ and $E_{\mathcal{R}} - E_{a,r}$ are path independent. There are no path overlap zones in Z_F on which overlapping shortest paths reduce the link load on one or more optical links. No non-minimum alpha routing sets are feasible by Lemma 3.6.3(i) as path interdependent alternate alpha routing sets. Lemma 3.5.1 applies to all alpha routing sets.
- (iii) All shortest path lengths are 1. All requests have a feasible shortest path interval.

The independent zone represents a single bin C_{ab} into which bandwidth is packed from the maximum load link eligible set $E_{a,r}$. C_{ab} establishes a lower bound on the optimal packing of $E_{a,r}$ by RLOAD2-BPACK[k]. Bandwidth may be packed into C_{ab} as desired as long as there exists sufficient capacity in the F-zone bins that the selected requests pack. If there is insufficient capacity in a F-zone bin f to pack a request qf , which would otherwise pack bin b , then qf is said to be interfered on bin f . Therefore, the interdependent zone is sometimes referred to as the *interference* or F-zone.

The F-zone bin constraints are not restricted to the packing of $E_{a,r}$ by RLOAD2-BPACK[K]. An alpha routing set $A_{r \rightarrow s}$ is composed of a reduction set A_s and a borrow request α_s . The A_s portion of $A_{r \rightarrow s}$ packs within the bin constraints. However, α_s overflows one or more bins after A_s is routed. Since a sequence of alpha routing sets is usually necessary to achieve the optimal ring congestion, this overflow places an additional capacity requirement on the bins packed by α_s . This added capacity requirement may lead to interference on requests packed from $E_{a,r}$ on a subsequent routing t for which the maximum load link is a . The formal definition of an interfered request takes this added interference into account.

Section 5.2 defines concepts and terminology related to the interdependent zone. It uses the RLP2[2] interdependent zone to illustrate these concepts.

Section 5.3 formally defines an interfered request. A method of identifying possible interfered requests using the current partition $\mathcal{P}_r$ is provided.

Section 5.4 introduces the capacity transfer $\mathcal{T}_{af}$. For qf interfered on f , a capacity transfer increases the c_{af} capacity so that qf may subsequently be routed without interference. This section includes a formal definition and general concepts related to transfers. Categories and types of transfers are listed.

Sections 5.5 and 5.6 discuss the first category of transfer, that is, transfers that occur after an alternate alpha routing set is routed according to Lemma 3.6.3. This category includes two transfer types. Section 5.5 defines the MSI capacity transfer based upon Lemma 3.6.3(ii). A MSI transfer routes an alternate link reduction set to avoid selection of a request $qr \in E_{a,r}$ which would otherwise pack c_{af} . By 'de-selecting' qr c_{af} capacity is freed to allow qf to be routed.

Section 5.6 defines the PI capacity transfer based upon Lemma 3.6.3(i). For a PI capacity transfer, the alternate link reduction set includes at least one request qi for which f but not a is included in $P^-(qi)$. Therefore, routing qi increases c_{af} by decreasing c_{fa} .

The second category of transfer is a repartition of $E_{a,r}$. A repartition partitions $E_{a,r}$ into new sets S and X different from the sets for original partition. Section 5.7 covers repartitions. For qf interfered on f a single transfer request $qb \in S$: $f \notin P^-(qb)$ is replaced by a $d(qb)$ replacement block from X . The replacement block includes at least one common request qc : $f \in P^-(qc)$. $d(qc)$ is greater than $d(qf)$ so that sufficient c_{af} capacity is freed to select qf without interference. A repartition of $E_{a,r}$ constitutes the third

transfer type, which is referred to as a beta transfer β_{af} .

These principles apply to all RLP2[k]. In the next chapter they are used to develop a solution to RLP2[2]. This solution suggests possible solutions to RLP2[k] and the general RLP2.

Unless otherwise noted, assume routing r with maximum load link $\ell_1 = a$ and maximum independent zone load link $\ell_2 = b$.

5.2 The Interdependent Zone Z_F

For RLP2[1], C_{ab} is the only constraint on the selection of requests from the eligible set. RLOAD2-PART[1] is adequate as RLOAD2-BPACK[1] where the sole task is to pack C_{ab} as closely possible.

For $\ell \in Z_F$, $E_{a\ell} \neq \emptyset$ by definition. $C_{a\ell}$ is a constraint on the amount of bandwidth selected from $E_{a\ell}$. However, selection from $E_{a\ell}$ does not alter $C_{a\ell}$. Thus there exists the option of selecting $E_{a\ell}$ bandwidth without penalty on ℓ . Therefore, the interdependent zone must be treated differently from the independent zone where the selection of an eligible request increases all link loads.

Because the independent zone is always nonempty (Observation 2.4.1), its capacity C_{ab} remains a constraint for all RLP2[k]-BPP. ℓ_1 divides the interdependent zone into two subzones. The *left interdependent subzone* Z_L consists of all interdependent zone links clockwise from ℓ_1 to the independent zone. The *right interdependent subzone* Z_R consists of all interdependent zone links anti clockwise from ℓ_1 to the independent zone.

For the interdependent zone, the constraints of the link capacities in combination with the routing options for request selection from common eligible sets must be considered. These options increase the search space for the optimal ringload and hence, the computational complexity of routing algorithms. However, they must be considered as, in a general RLP2 problem, the interdependent zone may encompass most of the ring.

RLP2[2] illustrates these principles. The RLP2[2] shortest path length for a request is either 1 or 2. Therefore, there is a maximum of 1 left interdependent zone link $\ell_\ell = (a + 1 + n) \bmod n$ and 1 right interdependent zone link $\ell_r = (a - 1 + n) \bmod n$. Therefore, $Z_L = \ell_\ell$ and $Z_R = \ell_r$. ℓ_ℓ and ℓ_r are undefined if they have no common set with ℓ_1 . Their respective capacities are $C_{a\ell} = C_{a\ell} = \lfloor \frac{L_a - L_\ell}{2} \rfloor$ and $C_{ar} = C_{ar} = \lfloor \frac{L_a - L_r}{2} \rfloor$. The values of ℓ_ℓ and ℓ_r are, by convention, denoted by $l \in \mathcal{R}$ and $r \in \mathcal{R}$, unless otherwise noted.

Figure 1.1 is an example of an RLP2[2]-BPP. $\ell_r = 3$. $\ell_t = 5$. These individual links comprise Z_R and Z_L respectively. If $d(q2) = d(q3)$, then there exists a choice of selection of either $q2$ or $q3$. These two requests pack different F-zone bins. Therefore, both $q2$ and $q3$ may have to be considered to achieve the minimum congestion.

Therefore, for RLOAD2-BPACK[2], there are three packing constraints: C_{ab} , C_{at} and C_{ar} . Each of these capacity constraints represents a bin. Because the eligible set for C_{ab} is E_a (Observation 2.4.2), all selected bandwidth packs C_{ab} so that the goal of RLOAD2-BPACK[k] is to optimize the bandwidth load packed into the C_{ab} bin. This objective must be met within the interdependent zone bin capacity and the feasible shortest path interval constraints, while exploiting the advantages afforded by the interdependent zone common eligible set options.

As stated earlier, the independent zone may be referred to as the B-zone. This is linked to the fact that the maximum load link on $\mathcal{R}$. $\ell_1 = a$ is the A-zone, and that the independent zone maximum load link is referred to as link b . The congestion on ℓ_t and ℓ_r *interferes* with the packing of C_{ab} . The values of C_{at} and C_{ar} are a formal measure of this interference. Therefore, the interdependent zone may be referred to as the *interference* or F-zone.

For RLP2[2], ℓ_t is referred to as the *left interference link*. ℓ_r is referred to as the *right interference link*. C_{at} and C_{ar} are the *left* and *right* interference capacities. $f \in Z_F$ and $f' \in Z_F$ denote the interference links. When the value of f is assigned to an RLP2[2] interference link, the value of f' is automatically assigned to identifier of the alternate interference link and vice versa. If the alternate link is nonexistent then f' is undefined.

5.3 Interfered Requests

The main issue for the solution of RLP2[k], based on RLOAD2-BPACK[k], is the resolution of *interfered* requests. A formal definition is:

Definition 5.3.1 (interfered request) For routing r , $\ell_1 = a$. $\mathcal{P}_r$ is a partition of $E_{a,r}$. $\mathcal{P}_t$ is a partition for routing t . $\ell_1 = a$. t is either r itself or a subsequent routing to r .

Let $q' \in E_{a,r}$ be a request rejected by $\mathcal{P}_t$. q' belongs to a minimum routing $R_{r \rightarrow \min}$ from routing r . If there does not exist a minimum alpha routing set $A_{r \rightarrow s} \subset R_{r \rightarrow \min}$ selectable from $\mathcal{P}_r$, then q' is referred to as an *interfered* request.

This definition states that routing of a minimum alpha set from $\mathcal{P}_r$ cannot achieve $L_{min,r}$ because there is insufficient space in some F-zone bin f to route q' .

As defined, interfered requests are not always readily identified. It must suffice to identify the *possibility* that q' is interfered on $\mathcal{P}_t$. If q' is interfered on $\mathcal{P}_t$ then c_{af} is insufficient on $\mathcal{P}_r$ so there exists a rejected request qf that packs bin f . qf is one of:

- (i) $qf \in B_{af}$. qf is a borrow request on F-zone bin f .
- (ii) $qf \in D_{af}$. qf has a non-maximal shortest path interval for packing F-zone bin f .
- (iii) $qf \in B_{ab}$: $qf \notin B_{af}$ and $qf \notin D_{af}$. qf is a B-zone borrow. qf is not a borrow request on bin f . However, after qf is routed, q' may be interfered on bin f for some later routing t . For $\mathcal{P}_t$, the c_{af} capacity may be insufficient to pack q' into bin f . Therefore, qf is said to be interfered on $\mathcal{P}_r$ because of the interference deferred on q' to $\mathcal{P}_t$ by routing qf .

If qf does not exist on routing r then it is verifiable that the bin f capacity is sufficient to pack all required $\bar{E}_{af,r}$ bandwidth. Because the existence of qf indicates the possibility that q' is interfered on $\mathcal{P}_t$, the definition of an interfered request is revised to refer to qf rather than q' :

Definition 5.3.2 (interfered request) *An interfered request is redefined to be qf as defined above. This approach allows the possible existence of q' on $\mathcal{P}_t$ to be predicted on the current partition $\mathcal{P}_r$. The remainder of this thesis adheres to this latter definition.*

5.4 Transfers

When the possibility exists that request qf is interfered on F-zone link f , the request selection options afforded by the eligible set common to the maximum load link a and f may be explored in order to provide the additional capacity required on f to route qf without interference. The method described here is the *af capacity transfer* or *af transfer* $\mathcal{T}_{af}$. Within context, $\mathcal{T}_{af}$ may be referred to as a *transfer*. $\mathcal{T}_{af}$ consists of two types of request:

Definition 5.4.1 (common request) *The selection of one or more $qc \in B$: $qc \in E_{af}$ is required by a transfer to allow the selection or de-selection of one or more transfer requests to increase c_{af} . qc is referred to as an af common request or a common request.*

Definition 5.4.2 (transfer request) c_{af} is increased by:

- (i) *The de-selection of one or more $q \in S$: $q \in \bar{E}_{af}$.*
- (ii) *The selection of one or more $q \in \bar{E}_{fa}$.*

Each de-selected/selected request q is referred to as an f capacity transfer request or a transfer request.

Definition 5.4.3 (transfer gain) *The transfer gain Δ is the total increase of c_{af} effected by the transfer requests.*

There are two categories of transfer:

- (i) *alternate routing set capacity transfers.* These are capacity transfers effected by the routing of an alternate alpha routing set according to Lemma 3.6.3. There are two subcategories:
 - (a) Lemma 3.6.3(i) identifies the PI alternate alpha routing set. A minimum of one $qi \in \bar{E}_{fa}$ in $A_{t \rightarrow r'}$ increases c_{af} .
 - (b) Lemma 3.6.3(ii) identifies the MSI alternate alpha routing set. The de-selection of at least one $qr \in \bar{R}$: $f \notin P^-(qr)$, increases c_{af} .
- (ii) *A repartition of partition $\mathcal{P}_r$ of $E_{a,r}$. One qb : $f \notin P^-(qb)$ is replaced by an equivalent bandwidth block from the rejection set. The replacement block must include at least one af common request to increase c_{af} .*

5.5 MSI Capacity Transfers

Let $A_{r \rightarrow t} = A_t + \alpha_t$ be an MSI alternate alpha routing set defined according to Lemma 3.6.3(ii). $A_{r \rightarrow t}$ relocates ℓ_1 from a to y . If $f \in P^-(\alpha_t)$ then α_t qualifies as an af common request. The capacity transfer to c_{af} is achieved

by the replacement of reduction subset $\tilde{R}'$ by the alternate link reduction set $\tilde{R}'_y$. For this capacity transfer to occur, there must exist a minimum of one transfer request $qr \in \tilde{R}'$: $f \notin P^-(qr)$. $\tilde{R}'$ and $\tilde{R}'_y$ are defined according to Table 3.1 and Lemma 3.6.1. Therefore:

Definition 5.5.1 (MSI capacity transfer) *Let T_{af} be a capacity transfer as defined above. T_{af} is an MSI capacity transfer or MSI transfer. α_t is the common request. The $qr \in \tilde{R}'$: $f \notin P^-(qr)$ are the transfer requests.*

To illustrate a MSI transfer, Figures 5.1 and 5.2 compare routing $\tilde{R}'$ and $\tilde{R}'_y$. Requests appear from top to bottom in decreasing order of demand. Asterisks denote bins overflowed by borrow (rejected) requests from $E_{a,r}$. These conventions also apply to subsequent illustrations.

Figure 5.1 shows a partition of $E_{a,r}$. qr is a $\tilde{R}'$ request that packs the c_{af} bin. The figure shows the effect of routing $\tilde{R}$ which includes subset $\tilde{R}'$. qr is assigned to its alternate path. qf is interfered on bin f .

Figure 5.2 shows the effect of routing $A_{r \rightarrow t}$. First, ℓ_1 relocates to y . Next, $A_{t \rightarrow r}$ including α_r is routed to relocate ℓ_1 back to a . $\tilde{R}'_y$ is routed instead of $\tilde{R}'$. The values of L_a and L_y after routing $\tilde{R}'_y$ are equivalent to those for routing $\tilde{R}'$. Therefore, the need to route qr is eliminated. qr remains assigned to its shortest path. This frees up sufficient capacity to subsequently route qf . Note that $y \notin P^-(qr)$. Otherwise, $qr \in E_{ay}$ so that qr cannot belong to $\tilde{R}'$ as defined by Lemma 3.6.1.

Lemma 5.5.1 states bounds on the demands of the MSI transfer common and transfer requests.

Lemma 5.5.1 *Let T_{af} be a MSI capacity transfer. T_{af} eliminates the interference on request qf on F -zone link f . Then:*

$$(i) \ d(qr) > d(qf).$$

$$(ii) \ d(\alpha_t) > d(qr). \text{ Therefore, } d(\alpha_t) > 2d(qf).$$

Proof:

- (i) If $d(qr) = d(qf)$, then de-selection of qr to select qf may occur without routing $A_{r \rightarrow t}$. Therefore routing of $A_{r \rightarrow t}$ as a MSI alternate alpha routing set becomes nonfeasible by Lemma 3.6.3. Therefore, $d(qr) > d(qf)$.

Figure 5.1: MSI Transfer - Routing $\tilde{R}'$

Figure 5.2: MSI Transfer - Routing $\tilde{R}_y$

- (ii) Because $d(A_{t \rightarrow r'}) > 0$, the demand of all $qr \in \tilde{R}'$ is less than $d(\alpha_t)$ (Lemma 3.6.1 Corollary 3.6.1.1. Therefore, $d(\alpha_t) > d(qr)$.

From (i), it is evident that $d(\alpha_t) > 2d(qf)$.

□

Multiple interfered requests may exist on f . These interfered requests may require different alternate alpha routing sets to release the required capacity on f in accordance with Lemma 5.5.1. The limiting factor is the increased load routed to $P^+(\alpha_t)$ by $A_{r \rightarrow t}$ with α_t of increasing demand. Therefore, each candidate alternate alpha routing set must be routed to determine $L_{r, \min}$. This is one reason for the complexity of RLP2[k].

5.6 PI Capacity Transfers

Let $A_{r \rightarrow t} = A_t + \alpha_t$ be a PI alternate alpha routing set defined according to Lemma 3.6.3(i). $A_{r \rightarrow t}$ relocates ℓ_1 from a to y . If $f \in P^-(\alpha_t)$ then α_t qualifies as an af common request. The capacity transfer to c_{af} is achieved by the replacement of reduction subset $\tilde{R}'$ by the alternate link reduction set $\tilde{R}'_y$. For this capacity transfer to occur, there must exist a minimum of one transfer request $qi \in \tilde{R}'_y$: $f \in P^-(qi)$. $\tilde{R}'$ and $\tilde{R}'_y$ are defined according to Table 3.1 and Lemma 3.6.1. Therefore:

Definition 5.6.1 (PI capacity transfer) *Let T_{af} be a capacity transfer as defined above. T_{af} is an PI capacity transfer or PI transfer. α_t is the common request. The $qi \in \tilde{R}'_y$: $f \in P^-(qi)$ are the transfer requests.*

To illustrate a PI transfer, Figures 5.3 and 5.4 compare routing $\tilde{R}'$ and $\tilde{R}'_y$. Figure 5.3 shows a partition of $E_{a,r}$. For simplicity, no S requests are shown. If $\tilde{R}'$, which includes $\tilde{R}'$, is routed then qf is interfered on f .

Figure 5.4 shows the effect of routing $A_{r \rightarrow t}$. First, ℓ_1 relocates to y . Next, $A_{t \rightarrow r'}$ including qi and $\alpha_{r'}$ is routed to relocate ℓ_1 back to a . $\tilde{R}'_y$ is routed instead of $\tilde{R}'$. The values of L_a and L_y after routing $\tilde{R}'_y$ are equivalent to those for routing $\tilde{R}'$. Routing qi sufficiently increases c_{af} to subsequently route qf . The shortest path of qi must include f but not a so that c_{af} is increased. $\alpha_{r'}$ qualifies as qi if its shortest path includes f .

Lemma 5.6.1 states bounds on the demands of the PI transfer common and transfer requests.

Figure 5.3: PI Transfer - Routing $\tilde{R}'$

Figure 5.4: PI Transfer - Routing $\tilde{R}'_y$

Lemma 5.6.1 *Let $\mathcal{T}_{af}$ be a PI capacity transfer. $\mathcal{T}_{af}$ eliminates the interference on request qf on F-zone link f . Then:*

- (i) $d(qi) > d(qf)$.
- (ii) $d(\alpha_t) > d(qf)$.

Proof:

- (i) Because $d(\alpha_r) \geq d(qf)$ and $qi \in A_{t \rightarrow r'}$, it follows that $d(qi) \geq d(qf)$. If $d(qi) = d(qf)$ then routing these two requests cannot be feasible except on some F-zone link $\ell \neq f$: $\ell \in P^-(qi)$ and $\ell \in P^-(qf)$. This constitutes a af PI transfer $\mathcal{T}_{af}$ which, by itself, does not reduce L_a . Therefore, $d(qi) > d(qf)$.
- (ii) Because $d(qi) > d(qf)$ and $d(\alpha_t) \geq d(qi)$, it follows that $d(\alpha_t) > d(qf)$ (Lemma 3.6.1).

□

Because $\bar{E}_{ay,r}$ and $\bar{E}_{ya,r}$ are path interdependent, the Lemma 3.5.1 bounds on bandwidth load selected from these eligible sets do not apply. For $A_{t \rightarrow r'}$, selected in accordance with Lemma 3.6.1, this means that $d(A_{t \rightarrow r'})$ may equal $d(\alpha_t)$ if $\bar{R}' = \emptyset$. Therefore, $d(\alpha_r) \leq d(\alpha_t)$. However, there may still exist $qi \in \bar{E}_{ya}$: $f \in P^-(qi)$ where $d(qi) > d(\alpha_t)$. If qi does not belong to an alternate link reduction set as defined by Lemma 3.6.1 then it may be necessary to match $d(qi)$ on y with a common request set $A_{\alpha,t}$ formally defined as:

Definition 5.6.2 (common request set) *Let y be an alternate link bin. Let $qi \in \bar{E}_{ya}$: $f \in P^-(qi)$ be a PI transfer request. Let $A_{\alpha,t}$ be a subset of the rejection set X such that:*

- (i) $d(A_{\alpha,t}) = d(qi)$ and
- (ii) there exists at least one $qc \in A_{\alpha,t}$: $qc \in B_{ay}$ and $f \in P^-(qc)$ to qualify as an af common request (α_t) and
- (iii) the demand of all $q \in A_{\alpha,t}$ is less than $d(qi)$.

Figure 5.5: PI Transfer - Common Routing Set

Then $A_{\alpha,t}$ is a common request set. $A_{\alpha,t}$ routes $d(qi)$ load off L_a to balance the added $d(qi)$ load routed from y while including the common request qc necessary for a transfer.

Letting $A_{\alpha,t}$ take the place of a single common request α_t , $\tilde{R}'$ becomes $\emptyset$ and $\tilde{R}'_y = qi$. Because $d(A_{\alpha,t}) = d(\tilde{R}' + A_{t \rightarrow r'})$, Lemma 3.6.1 Corollary 3.6.1.1 is satisfied. $A_{t \rightarrow r'} = qi$ so that $\alpha_{r'} = qi$.

Figure 5.5 illustrates a common routing set. Selection of qi from $\tilde{E}_{ya,r}$ and qc and ql from $\tilde{E}_{ya,r}$ increases c_{af} by 2^7 while having no effect on L_a or L_y . qf may then be routed without interference afterwards.

The search for $A_{\alpha,t}$ as a subset of $\tilde{E}_{ay}$ is meant to bypass the requirement for searching $\tilde{E}_{ya}$ for alternate link transfer requests. Once $A_{\alpha,t}$ is found, it is routed to discover if qi exists. This increases the computational complexity of RLP2[k]. However, the combination of lemmas (defined below) and heuristics may minimize the search space.

To reduce the number of PI alternate alpha routing sets that may have to be routed, Lemmas 5.6.2 and 5.6.3 are useful:

Lemma 5.6.2 states PI alternate alpha routing set $A_{r \rightarrow t}$ is not feasible if α_t is also a borrow on some link $z \in P^+(\alpha_t)$ which does not lie on the

aggregate path of $\bar{E}_{ya,t}$:

Lemma 5.6.2 *For routing r , let qf be a request interfered on link f . Let $A_{r \rightarrow t}$ be an alternate alpha routing set such that α_t is an af common request. α_t relocates $\ell_1 = a$ to link y . Partition $P^+(\alpha_t)$ into two subzones:*

Z' *The set of links in the aggregate path of $\bar{E}_{ya,t}$.*

Z'' *The set of links not in the aggregate path of $\bar{E}_{ya,t}$. That is $Z'' = P^+(\alpha_t) - Z'$.*

If α_t is a borrow on bin z , $z \in Z''$ then $A_{r \rightarrow t}$ is not a feasible PI alternate alpha routing set.

Proof: Because $\alpha_t \in B_{az}$, $L_{z,t} > L_{a,t}$. That is, L_z is increased beyond L_a after $A_{r \rightarrow t}$ is routed. Since z lies in the alternate path of all $q \in \bar{E}_{ya,t}$, selection from $\bar{E}_{ya,t}$ can only further increase L_z relative to L_a . Therefore, for $qi \in \bar{E}_{ya,t}$: $f \in P^-(qi)$, L_z exceeds L_a by a minimum of $d(qi)$ after $A_{t \rightarrow r'}$ is routed. Because $d(qi) > d(qf)$ (Lemma 5.6.1), the added penalty on z cancels all possible gain on f . $\square$

The partition for Figure 5.6 is identical to that in Figure 5.1 except that α_t is also interfered on F-zone link f' . Assuming that f' lies outside of the aggregate path of $\bar{E}_{ya,t}$, $L_{f',r'}$ will exceed $L_{a,r'}$ by a minimum of $d(qi)$ after $A_{t \rightarrow r'}$ is routed.

Lemma 5.6.3 applies the Lemma 5.6.2 concept to common request sets:

Lemma 5.6.3 *For routing r , let qf be a request interfered on link f . Let $A_{\alpha,t}$ be an af common request set. Routing $A_{\alpha,t}$ relocates $\ell_1 = a$ to link y . Let P' be the complement of the aggregate path of $A_{\alpha,t}$ on $\mathcal{R}$, that is, $P' = \mathcal{R} - P(A_{\alpha,t})$. Partition P' into two subzones:*

Z' *The set of links in the aggregate path of $\bar{E}_{ya,t}$.*

Z'' *The set of links not in the aggregate path of $\bar{E}_{ya,t}$. That is $Z'' = P' - Z'$.*

If $A_{\alpha,t}$ is a net borrow on bin z , $z \in Z''$ then $A_{\alpha,t}$ is not feasible.

Proof: The proof is similar to that for Lemma 3.6.1. $\square$

Figure 5.6: PI Transfer - Interference on Common Request

5.7 Repartition of the Eligible Set

RLOAD2-PART[K] does not always provide an optimal packing of the independent zone bin capacity C_{ab} . Figure 5.7 shows a RLP2[2]-BPP for which an optimal routing of the eligible set of requests within the listed capacities is to be found. Link 4 is the maximum load link. Link 7 is the B-zone maximum load link. ℓ_l and ℓ_r denote the left and right F-zone links. E_a consists of requests $q_1 - q_5$. For simplicity, only requests in E_a are shown.

RLOAD2-PART[2] yields Partition 1, shown by Figure 5.8. Partition 2 yields the optimal packing shown by Figure 5.9. Final bin capacities are shown in these two figures.

From this example, the following are noted:

- (i) The packing in Figure 5.8 is optimal up to the point that request q_4 is rejected on account of insufficient c_{ar} capacity. As seen in Figure 5.9 it is necessary to select q_4 to achieve an optimal packing. Therefore, q_4 is an interfered request.
- (ii) q_1 , $d = 2^8$ in Figure 5.8 is replaced by an equivalent bandwidth block consisting of q_2 and q_3 in Figure 5.9.

Figure 5.7: RLP2[2]-BPP - Initial Routing

Figure 5.8: RLP2[2]-BPP - Partition 1

Figure 5.9: RLP2[2]-BPP - Partition 2

- (iii) A minimum of 1 request in the replacement block for q_1 does not add any load to the c_{ar} bin. The demand of this request q_3 is *greater* than that of q_4 .
- (iv) During the original packing, request q_3 is rejected because its demand overflowed c_{ab} not c_{al} .

The interfered request q_4 in Partition 1 implies a *repartition* of E_a into a new selection and rejection set.

Partition 2 yields minimum congestion for the above example. First, q_1 has been replaced by the repartition and is not considered. Lemma 3.4.1 states that selection of q_5 cannot reduce c_{al} , therefore, it does not qualify as a borrow request. All other requests are selected so that no further routing options remain. Lemma 5.7.1 formalizes the repartition process:

Lemma 5.7.1 *For routing r , $\mathcal{P}_r$ is a partition of eligible request set $E_{a,r}$ into selection set S and rejection set X . On $\mathcal{P}_r$, request q_f is interfered on link f . On partition $\mathcal{P}_{rx}$ of $E_{a,r}$, capacity transfer T_{af} occurs. T_{af} allows q_f to be routed without interference within $R_{r-\min}$. T_{af} occurs if and only if:*

- (i) $qb: f \notin P^-(qb)$ and $d(qb) > d(qf)$ is 'de-selected' from S .
- (ii) there exists a $d(qb)$ bandwidth replacement block on X .
- (iii) the replacement block includes $qc \in X: f \in P^-(qc)$.
- (iv) $d(qc) > d(qf)$.

Proof:

$\mathcal{T}_{af}$ occurs Then:

- (i) Sufficient c_{af} capacity must have been released to allow qf to be routed without interference within R_{r-min} . For the greedy RLOAD2-PART[K] selection process this is only possible if some $qb: f \notin P^-(qb)$ and $d(qb) \geq d(qf)$ is 'de-selected' from S .

If $d(qb) = d(qf)$ then RLOAD2-PART[K] would have selected qf instead of qb in a feasible selection process. The transfer gain is zero. Therefore, the replacement of qb by qf cannot be feasible and cannot reduce the congestion. Therefore, $d(qb) > d(qf)$.

- (ii) Assume that a $d(qb)$ bandwidth replacement block does not exist on X . Then the qb bandwidth is not fully replaced on S so that S can no longer produce an optimal packing of $E_{a,r}$.
 - (iii) Otherwise, the replacement block places $d(qb)$ load back onto f so that no transfer occurs.
 - (iv) Assume $d(qc) \leq d(qf)$. Then the replacement of qb is equivalent to the replacement of a $d(qf)$ transfer request. From (i) this is not feasible.
- (i) - (iv) are satisfied Then it is verifiable that $\mathcal{T}_{af}$ occurs. $d(qc) > d(qf)$ capacity is released on bin f so that qf may be routed without interference within R_{r-min} . qb is replaced on S by $d(qb)$ bandwidth from X .

□

Definition 5.7.1 (beta transfer) Let $\mathcal{T}_{af}$ be a feasible transfer defined according to Lemma 5.7.1. $\mathcal{T}_{af}$ is referred to as a beta transfer $\mathcal{J}_{af}$. This definition applies to $\mathcal{J}_{af}$:

Figure 5.10: Beta Transfer

Definition 5.7.2 (beta request) For beta transfer β_{af} , the transfer request qb is referred to as a beta request.

Figure 5.10 illustrates a beta transfer. Request qb is selected. Requests $q1$, $q2$ and qc are rejected. qf is interfered on bin f . Substitution of $q1$, $q2$ and qc for qb on the selection list maintains the 2^8 load reduction from a . However, c_{af} is increased by $d(qc) = 2^6$ providing sufficient capacity to route qf .

Lemma 5.7.1, therefore, shows that a beta transfer effects a feasible repartition of $\mathcal{P}_r$ and, conversely, a feasible repartition $\mathcal{P}_{rx}$ of $\mathcal{P}_r$ requires a beta transfer from $\mathcal{P}_r$. For each beta transfer, a single beta request qb is replaced on S by a replacement block from X . In Chapter 6, RLOAD2-BPACK[2] selects beta requests in greedy fashion to define the beta tree structure. From the beta tree, an optimal packing of $E_{a,r}$ within the given bin constraints is read.

The above RLP2[2]-BPP is solved with a 2-node beta tree. Let β_{ar} denote the transfer which maps from Partition 1 to Partition 2. For β_{ar} , $qb = q1$ and $qc = q3$. β_{ar} frees 2^7 c_{ar} capacity to allow $q4$ to be routed. If $q3$ is replaced in Figure 5.7 by a $2^7 - 2^4$ block coincident with $q2$ and a 2^4 $q3$ request then

q_3 would be selected on Partition 1 after q_1 . q_5 would be the final request selected. An optimal packing would occur.

5.8 Conclusion

This chapter discusses F-zone interference on request selection. F-zone interference arises from the limits that the F-zone bins place on the selection of request bandwidth to pack the B-zone bin.

The capacity transfer is a means to increase c_{af} to allow q_f , interfered on bin f , to be routed without interference. Lemma 3.6.3 identifies the two cases for which a feasible transfer $\mathcal{T}_{af}$ may be effected by routing eligible requests not belonging to $E_{a,r}$. These are the MSI capacity transfer and the PI capacity transfer. Sections 5.5 and 5.6 identify requirements for the selection of transfer and common requests for $\mathcal{T}_{af}$.

The second type of transfer is the beta transfer, β_{af} . A beta transfer is effected solely by repartitioning $E_{a,r}$. Section 5.7 states requirements for beta transfers.

Chapter 6

RLP2[2]

6.1 Introduction

This chapter proposes RLOAD2[2] as a solution to RLP2[2]. RLOAD2[2] employs the same alpha tree data structure as RLOAD2[1] to assess the non-terminal and terminal routing set options for all non-terminal routings. Unlike RLOAD2[1], there may be more than one feasible alpha routing set $A_{r \rightarrow s}$ for routing r so that each alpha tree node may have multiple descendants. After all feasible alpha routings have been assessed, r_{opt} is read from the alpha tree.

RLOAD2[2] uses RLOAD2-BPACK[2] as a bin packing subfunction. Let r be the routing represented by the alpha node and let $E = E_{a,r}$ be its eligible set. r may be referred to as the *alpha routing*. r is the initial *beta routing*. After the initial partition $\mathcal{P}_r$ of E , one or more beta transfers may be required to find an optimal packing within the bin capacity constraints. Each beta transfer represents a transition from beta routing rx to beta routing ry of $\mathcal{Q}$. For each beta routing, the eligible set is partitioned by RLOAD2-PART[2]. Multiple beta transfer candidates may exist for a beta routing. The result is a tree-like structure referred to as the *beta tree*.

Section 6.2 categorizes $E_{a,r}$ requests according to their terminal nodes. Each category forms a set of requests referred to as a *terminal set*. This classification of $E_{a,r}$ requests facilitates the formulation of related RLP2[2] concepts.

Section 6.3 clarifies the maximal shortest path interval constraint for RLP2[2]. Observation 6.3.1 restates Observation 2.6.2 for RLP2[2].

Sections 6.4, 6.5 and 6.6 discuss capacity transfers as applied to RLP2[2]. For each transfer type, these sections define RLP2[2] transfer feasibility requirements. In addition, the transfer gain is established as an upper bound on subsequent bandwidth routed into the bin to which the capacity transfer occurred.

Section 6.7 states criteria for the identification of an optimal packing of $E_{a,r}$ by RLOAD2-BPACK[2]. This section also identifies a criterion to eliminate nonfeasible alternate alpha routing sets after alpha routing set $A_{r,t}$ has been routed.

Section 6.8 describes the RLP2[2] Partition Algorithm RLOAD2-PART[2]. The computational complexity of RLOAD2-PART[2] is $O(m)$ without request sorting and $O(m \log_2 m)$ with sorting.

Section 6.9 describes the RLP2[2] Bin Packing Algorithm, RLOAD2-BPACK[2]. This recursive algorithm calls RLOAD2-PART[2] to find an optimum packing of the $E_{a,r}$ bandwidth within the B-zone and RLP2[2] F-zone bin limits. RLOAD2-BPACK[2] must also identify feasible alpha routing sets to subsequent routings when it cannot achieve $L_{min,r}$. Lemma 6.9.1 shows that this algorithm gives an optimal packing of $E_{a,r}$ and finds at least one feasible alpha routing set when required. The computational complexity of RLOAD2-BPACK[2] is $O(m)$ without request sorting and $O(m \log_2 m)$ with sorting.

Section 6.10 describes the RLOAD2[2] algorithm. RLOAD2[2] is expected to provide a 'good approximation' to the optimal RLP2[2] congestion most of the time. This algorithm does not route common request sets as described by Chapter 5 Section 5.6.

Section 6.11 estimates the RLOAD2[2] computational complexity. A good estimate is $O(2^w(m+n) + m \log_2 m)$. This value includes preliminary request sorting.

6.2 The Eligible Set - Terminal Sets

Definition 6.2.1 (terminal set) *A terminal set T is a set of requests q all of which have an identical terminal node pair id (i_q, j_q) .*

Recall that link ids are given by their anticlockwise optical node id. Let $\ell_l = l$, $\ell_r = r$ where $l, r \in \mathcal{R}$. The RLOAD2-PART[2] eligible set E_a is partitioned into three terminal sets:

Figure 6.1: Terminal Set Zones

T_l , the set of all requests q with terminal node pair (l, a) . For $q \in T_l$, $\delta(q) = 2$.

T_r , the set of all requests q with terminal node pair (a, r) . For $q \in T_r$, $\delta(q) = 2$.

T_t , the set of all requests q with terminal node pair (a, a) . For $q \in T_t$, $\delta(q) = 1$.

$\tau(q) \in \{l, r, t\}$ gives the name of the terminal set (left, right, tl) of request q .

Figure 5.10 illustrates the shortest path zones for requests in each of T_l , T_r and T_t .

For $ql \in T_l$, $qr \in T_r$, $qt \in T_t$, all demand d requests, rerouting:

1. ql , qr or qt reduces L_a by d and increases L_ℓ , $\ell \in Z_B$, by d . Therefore, c_{ab} decreases by d .
2. ql reduces L_l by d and increases L_r by d . Therefore, c_{ar} decreases by d but c_{al} remains the same.

3. qr reduces L_r by d and increases L_l by d . Therefore, c_{al} decreases by d but c_{ar} remains the same.
4. qt increases both L_l and L_r by d . Therefore, both c_{al} and c_{ar} decrease by d .

6.3 The Feasible Shortest Path Interval Constraint

For RLP2[2], the shortest path for a T_t request is a proper subset of the shortest path for either a T_l or T_r request. This means that, given the choice, an eligible $q2$, $\delta(q2) = 2$ should be selected rather than a $q1$, $\delta(q1) = 1$ of equal demand (Observation 2.6.1).

Observation 6.3.1 is Observation 2.6.2 specifically applied to RLP2[2].

Observation 6.3.1 *Let $q' \in E$. $\delta(q') = 2$ and $q'' \in E$. $\delta(q'') = 1$ be two equal demand d requests.*

Let R be a subset of E such that $d(R) = d$. For all $q \in R$, $P^-(q) \subseteq P^-(q')$ and $P^-(q'') \subseteq P^-(q)$. Then:

$$(i) \hat{L}_{E,q'} \leq \hat{L}_{E,R}$$

$$(ii) \hat{L}_{E,R} \leq \hat{L}_{E,q''}$$

Proof: Same as for Observation 2.6.2. □

For RLP2[2], let $qt \in \bar{E}_{ay}$: $qt \in T_t$ and $d(qt) = d$. If the total demand of requests of demand less than d remaining on $\bar{E}_{ay}$ is greater than or equal to c_{ay} , then selection of qt cannot be feasible within the maximal shortest path interval constraint. Since a T_t request packs c_{ab} , c_{al} and c_{ar} , this check is required for all three capacities. Feasibility of selection of qt occurs only if qt is feasible on all the individual bins it packs. This leads to the following definition:

Definition 6.3.1 (optional T_t selection) *Let qt be an eligible T_t request whose selection is feasible on all bins it packs. On bin y , let d_0 be the total demand of requests of demand less than d remaining on $\bar{E}_{ay}$. If $d \leq c_{ay}$, $d_0 < c_{ay}$ and $d + d_0 \geq c_{ay}$ then qt is an optional T_t selection on bin y . There exists the option of:*

- (i) Selection of qt and continuing the greedy packing process or
- (ii) Rejection of qt and accepting a terminal routing.

If qt is rejected, the shortest path interval constraint assures sufficient c_{ab} , c_{ad} and c_{db} space to pack all $q \in E_{a,r}$ of demand less than d . Therefore, a terminal routing occurs. This terminal routing is an alternative to the terminal reduction set. Therefore, this terminal routing is referred to as an alternate terminal routing.

6.4 PI Transfers

For RLP2[2], the $k = 2$ limit on $\delta(q)$ limits the number of PI alternate routing sets.

Lemma 6.4.1 establishes two restrictions on RLP2[2] PI alternate routing sets. First, the alternate link must be b and it must be adjacent to f . Second, $L_{f'}$ cannot increase beyond L_a after α_t , the PI transfer common request, is routed.

Lemma 6.4.1 *Given RLP2[2], routing r , partition $\mathcal{P}_r$. Request qf is interfered on F -zone link f . $A_{r \rightarrow t}$ is a alternate alpha routing set as defined by Lemma 3.6.1. $A_{r \rightarrow t}$ relocates ℓ_1 to y . To be feasible as a PI alpha alternate routing set (Lemma 3.6.3(i)), $A_{r \rightarrow t}$ must satisfy these restrictions:*

- (i) y must be the B -zone maximum load link b .
- (ii) α_t must not be a borrow request on F -zone link f' .

Proof: For $A_{r \rightarrow t}$ to be a PI alternate routing set:

- (i) $P^-(\alpha_t)$ must include f and a as a transfer common request.
- (ii) A transfer request qi must include f and y on $P^-(qi)$.

Because $k = 2$, the overlap zone of $\bar{E}_{ay}$ and $\bar{E}_{ya}$ can have at most 1 link. To effect a PI transfer, this overlap link must be f . Therefore:

- (i) y must always be in the independent zone adjacent to f . Therefore, to become the maximum load link, y must be b .

- (ii) Partition $P^+(\alpha_t)$ into two zones: Z' and Z'' as defined by Lemma 5.6.2. For RLP2[2], f' always belongs to Z'' . Therefore, if α_t is a borrow request on f' then $A_{r \rightarrow t}$ is not a feasible PI alternate alpha routing set (Lemma 5.6.2).

□

In Figure 5.6, let y be b for RLP2[2]. If α_t is also a borrow on f' then $L_{f',r}$ will exceed $L_{a,r}$ by a minimum of $d(qi)$ after $A_{t \rightarrow r}$ is routed.

After PI transfer T_{af} of gain Δ to bin f , Lemma 6.4.2 imposes an exclusive upper bound of Δ on the bandwidth subsequently routed from a to f . Lemma 6.4.2 is used to prove Lemma 6.7.1 which limits the total number of af transfers to a maximum of 1 during the execution of RLOAD2[2].

Lemma 6.4.2 *Given RLP2[2], routing r , partition $\mathcal{P}_r$. For $\mathcal{P}_r$, let T_{af} be a PI transfer. Let Δ be the T_{af} transfer gain. Route T_{af} to route from r to r' (Lemma 3.6.1).*

Then Δ is an exclusive upper bound on the $\bar{E}_{af,r'}$ bandwidth routed in a feasible routing sequence.

Proof: Because the maximum shortest path length is 2, E_{bf} and $\bar{E}_{af}$ are path independent. As such, Lemma 2.5.1 Corollary 2.5.1.1 imposes Δ as the exclusive upper bound on all $\bar{E}_{af}$ bandwidth routed after T_{af} . □

Refer back to Figure 5.3. For RLP2[2], routing qi followed by $d(qi)$ $\bar{E}_{af}$ bandwidth cannot reduce the load on any $\ell \in \mathcal{R}$.

On account of time constraints, the search for PI transfers is limited to transfers formed by alternate alpha routing sets as defined by Lemma 3.6.1. PI transfers formed by common request sets as defined in Chapter 5 Section 5.6 are not investigated by this thesis. Therefore, on occasion, the proposed RLOAD2[2] algorithm does not find an optimal solution to RLP2[2]. The tradeoff is the elimination of logic necessary to search for the common request set on link a to balance the demand of the single transfer request qi . Because of the Lemma 6.4.1 restrictions on RLP2[2] transfers, the loss in accuracy is expected to be small. In fact, this approach suggests heuristical solutions to RLP2 and more general RLP2[k] problems.

6.5 MSI Transfers

After MSI transfer $\mathcal{T}_{af}$ of gain Δ to bin f , Lemma 6.5.1 imposes an exclusive upper bound of Δ on the bandwidth subsequently routed from a to f . Lemma 6.5.1 is used to prove Lemma 6.7.1 which limits the total number of af transfers to a maximum of 1 during the execution of RLOAD2[2]. The Lemma 6.5.1 restriction on MSI transfers is similar to the Lemma 6.4.2 restriction on PI transfers.

Lemma 6.5.1 *Given RLP2[2], routing r , partition $\mathcal{P}_r$. For $\mathcal{P}_r$, let $\mathcal{T}_{af}$ be a MSI transfer. Let Δ be the $\mathcal{T}_{af}$ transfer gain. Route $\mathcal{T}_{af}$ to route from r to r' (Lemma 3.6.1).*

Then Δ is an exclusive upper bound on the $\bar{E}_{af,r'}$ bandwidth routed in a feasible routing sequence..

Proof: For $\mathcal{T}_{af}$, Δ is the sum of all $d(qr)$, where qr are the transfer requests de-selected by replacing R' by R_y according to Lemma 3.6.3(ii). Selection of Δ from $\bar{E}_{af,r'}$ is equivalent to replacing the R' transfer bandwidth thereby 'canceling' the transfer. This cannot be feasible. RLOAD2-PART[2] would select the $\Delta \bar{E}_{af,r'}$ bandwidth rather than the $\Delta R'$ transfer bandwidth if the former were feasible. Therefore, routing the $\Delta \bar{E}_{af,r'}$ block effectively replaces the feasible $\Delta R'$ block by an equivalent demand nonfeasible block. Hence the Δ upper bound on $\bar{E}_{af,r'}$ selections. □

Figures 5.1 and 5.2 illustrate Lemma 6.5.1. First, RLOAD2-PART[2] originally selected qr instead of a $d(qr) \bar{E}_{af,r'}$ bandwidth block because the former was the feasible option. Therefore, it remains nonfeasible to switch to the latter. On the diagram, qr is a T_r request. Therefore, for RLP2[2], routing qr is always the feasible choice. If qr is a T_t request, then the $\bar{E}_{af,r'}$ bandwidth block would be the feasible choice. However, the fact that RLOAD2-PART[2] has selected qr would deny the availability of the $\bar{E}_{af,r'}$ bandwidth block in this case.

6.6 Beta Transfers

Let $\mathcal{P}_r$ be a partition for routing r . For $\mathcal{P}_r$ and F-zone link f , there exists beta transfer β_{af} with beta request qb and one or more common requests qc . These definitions apply to a beta transfer.

Definition 6.6.1 (beta reduction set) For qb , the beta reduction set $\tilde{B}(qb)$ is the set of $qs \in S$. $d(qs) \geq d(qb)$, excluding qb itself.

Definition 6.6.2 (beta reduction capacity) For qb , the beta reduction capacity array $crb(qb)$ is the array of current bin capacities c for the selection of $\tilde{B}(qb)$.

Definition 6.6.3 (beta routing set) The beta routing set $B_{r \rightarrow rx} = \tilde{B}(qb)$ is the set of requests that RLOAD2-PART[2] selects from $E_{a,r}$ before qb is replaced on the selection set by its replacement block. $B_{r \rightarrow rx}$ maps from routing r to rx .

The eligible set for rx . $E_{a,rx}$, is a subset of $E_{a,r}$. $E_{a,rx}$ includes:

- (i) qc .
- (ii) all $q \in E_{a,r}$: $d(q) < d(qb)$.

Lemma 6.6.1 establishes restrictions on qb and qc .

Lemma 6.6.1 Given RLP2[2] and $\mathcal{J}_{af}$:

- (i) $f' \in P^-(qb)$
- (ii) $qc \in B_{ab}$ and $qc \notin B_{af'}$

Proof:

- (i) It has been previously shown how the optimal alternate terminal routing set is obtained by the de-selection of T_t requests from S . Therefore, T_t requests need not be considered. Furthermore, selection of a T_t request with an equivalent demand replacement block on the rejected set is not feasible (Lemma 2.6.1). Therefore, only de-selection of $T_{f'}$ requests can release the transfer capacity.
- (ii) (i) states that qb is a $T_{f'}$ request. Therefore, de-selection of qb cannot release any $c_{af'}$ capacity. Consequently, if $qc \in B_{af'}$ then qc would be a F-zone borrow on f' and would form an alpha routing set. The qb bandwidth would not be successfully replaced. The feasibility of alpha routing sets is being assessed elsewhere. Therefore, to appear on the rejection set, qb must be a B-zone borrow. Hence the (ii) restrictions.

□

Recall the example shown by Figures 5.7 to 5.9. On Partition 1, q_4 is interfered on $f = r$. q_1 is chosen as qb . Note that $P^-(q_1)$ includes $f' = l$. If q_1 was a T_l request then q_2 and q_3 would have been selected rather than q_1 on Partition 1 (Lemma 2.6.1). De-selection of q_1 releases 2^8 c_{ar} capacity. $qc = q_3$. If the c_{al} capacity is insufficient to pack q_3 then q_3 forms an alpha routing set which terminates the packing before the replacement of qb is completed. An optimal packing would not occur. Because q_3 is on the rejected list, then it must be a B-zone borrow.

Lemma 6.6.2 places an upper bound on the load routed into bin c_{af} after beta transfer β_{af} .

Lemma 6.6.2 *Given RLP2[2], routing r , partition $\mathcal{P}_r$ of $E_{a,r}$. Let β_{af} be a beta transfer for which Δ is the gain on bin c_{af} . If β_{af} is routed then is not feasible to subsequently pack Δ or more $\bar{E}_{af}$ bandwidth into c_{af} .*

Proof: Route β_{af} followed by a block of $\Delta \bar{E}_{af}$ bandwidth into c_{af} . Because the qb shortest path includes f' (Lemma 6.6.1), $\delta(qb) = 2$. Therefore, the $\Delta \bar{E}_{af}$ block cannot satisfy the shortest path interval constraint when compared with qb and, therefore, is not feasible (Lemma 2.6.1). Routing the $\Delta \bar{E}_{af}$ block negates the c_{af} capacity gain. □

Refer to the beta transfer β_{af} in Figure 5.10. β_{af} increases c_{af} by 2^6 . After β_{af} , routing an additional $2^6 \bar{E}_{af}$ bandwidth negates the effect of the transfer on c_{af} . Moreover, routing q_2 , q_1 and the final 2^6 from $\bar{E}_{af}$ cannot be feasible when compared to routing qb .

Lemma 6.6.3 states that if the minimum demand rejected request is interfered on link f then there exists no feasible af' beta transfer for requests interfered on f' . Therefore, for partition $\mathcal{P}_r$, a feasible beta transfer exists for a maximum of one F-zone link.

Lemma 6.6.3 *Given RLP2[2], partition $\mathcal{P}_r$. Let q' be the minimum demand $q \in X$. If q' is interfered on F-zone link f then there does not exist a feasible beta transfer $\beta_{af'}$.*

Proof: For $\beta_{af'}$, Lemma 6.6.1 requires that there exist common request $qc' \in B_{ab}$ such that $cr(qc', f) \geq d(qc')$. Assume these conditions to be true. Then:

- (i) There exists sufficient c_{af} capacity to assure that $q' \notin B_{af}$.
- (ii) If $q' \in D_{af}$ then $ct(f) = 0$ (Lemma 3.4.2). Because q' satisfies the Lemma 6.6.1 requirement, there exists sufficient c_{af} capacity to assure that this is impossible.
- (iii) If $q' \in B_{ab}$ then an exact packing of $\lfloor \frac{c_{ab}}{d(q')} \rfloor$ has occurred (Lemma 3.2.1). Therefore, the congestion cannot be reduced by routing q : $d(q) \geq d(q')$ (Lemma 3.2.1). Because all $qe \in E_{a,r}$: $d(qe) < d(q')$ have been selected, there is no need for additional $c_{af'}$ capacity. Therefore, $\beta_{af'}$ is not required.

Therefore, if $q' \in B_{af}$ or $q' \in D_{af}$ then q' cannot meet the requirements for a feasible $\beta_{af'}$ common request. If q' is a B-zone borrow then no $\beta_{af'}$ is required. Therefore, $\beta_{af'}$ is not feasible. $\square$

To identify the interference link f for a feasible beta transfer on partition $\mathcal{P}_r$ using Lemma 6.6.3:

- (i) Find the minimum demand $q' \in X$.
- (ii) Find the link f on which q' is interfered.

6.7 Additional RLP2[2] Preliminaries

This section defines concepts to facilitate the search for the optimal RLP2[2] congestion. The lemmas to follow provide criteria to eliminate interfered requests, and to reduce the number of transfers and alternate routing sets that must be routed.

Lemma 6.7.1 limits the number of feasible af transfers to a maximum of 1 for the execution of RLOAD2[2].

Lemma 6.7.1 *Given RLP2[2], routing r , partition $\mathcal{P}_r$. From $\mathcal{P}_r$ transfer T_{af} is routed. T_{af} is either an alternate link or beta transfer. Then after T_{af} is routed:*

- (i) *No subsequent feasible af transfer $T_{af,1}$ exists.*
- (ii) *The routing of $qe \in \bar{E}_{af}$ is feasible only if $c_{af} - d(qe) \geq d(qe)$.*

Proof:

- (i) Let Δ be the $\mathcal{T}_{af}$ gain. $\mathcal{T}_{af,1}$ is 'useful' as a transfer only if more than $\Delta \bar{E}_{af}$ bandwidth is routed (packed into c_{af}) after $\mathcal{T}_{af}$. Lemmas 6.4.2, 6.5.1 and 6.6.2 preclude the feasibility of packing Δ load into c_{af} after $\mathcal{T}_{af}$ is routed. Therefore, $\mathcal{T}_{af,1}$ is not feasible.
- (ii) Requests are packed into the bins in greedy fashion. To be a feasible selection, qe cannot pack c_{af} after $\mathcal{T}_{af}$ is routed. Therefore, a minimum of $d(qe)$ available capacity must remain in c_{af} .

□

Lemma 6.7.2 identifies the case where F-zone bin z current capacity c_{az} is greater than or equal to 2^{i+1} and the B-zone current capacity c_{ab} is less than 2^i . The nonexistence of interfered requests of demand less than 2^i on bin z is confirmed.

Lemma 6.7.2 *Given RLP2[2], partition $\mathcal{P}_r$, F-zone bin z . During the partition, let $\bar{E}'_{az}$ be the subset of $\bar{E}_{az}$ remaining to be partitioned. d' is the maximum demand of $q \in \bar{E}'_{az}$. $d' = 2^i$, integer i . The current independent zone capacity c_{ab} is less than 2^i . If the link z current capacity c_{az} is greater than or equal to 2^{i+1} then no request q' , selected from $\bar{E}'_{az}$, is interfered on z .*

Proof: First, consider the case where q' is the alpha request of an alternate alpha routing set $\mathcal{A}_{r \rightarrow t}$. Let z' be the other F-zone link. $\mathcal{A}_{r \rightarrow t}$ is routed to perform a az' transfer $\mathcal{T}_{az'}$. The $\mathcal{T}_{az'}$ alternate link is b (Lemma 6.7.1). For $\mathcal{T}_{az'}$:

- (i) No subsequent az' transfer exists (Lemma 6.7.1).
- (ii) After $\mathcal{T}_{az'}$ is routed, it is not feasible to route $d(\alpha_{r'})$ (ref. Lemma 3.6.1 definition) $\bar{E}_{az}$ bandwidth from a back to b thereby negating the effect of routing $\alpha_{r'}$ (Lemma 3.5.1).

Assume the worst case where $d(\alpha_t) = 2^i$ and $d(\alpha_{r'}) = 2^i$. This would be a PI transfer. α_t requires $2^i c_{az}$ capacity. After $\mathcal{T}_{az'}$ is routed, (i) precludes any subsequent az' transfer. (ii) restricts the bandwidth routed into c_{az} to less than 2^i . The total c_{az} requirement is less than 2^{i+1} .

Assume that $d(\alpha_t) < d(2^i)$. It is verifiable that the 2^{i+1} limit still holds.

□

Corollary 6.7.2.1 identifies a B-zone borrow request qx : $d(qx) \leq 2^t$, which packs bin z . A criterion is established for the nonexistence of interfered requests. Therefore, the routing of transfers and alternate routing sets is not necessary for $\mathcal{P}_r$.

Corollary 6.7.2.1 *Same as Lemma 6.7.2. qx is a B-zone borrow selected from $\bar{E}'_{az}$. Let z' be the other F-zone link. qx is not interfered on z' . Then there does not exist an interfered request qf : $d(qf) \geq d(qx)$.*

Proof: First, there are no requests interfered on bin z (Lemma 6.7.2). Therefore, no az transfers are necessary. Second, qx represents an exact packing of $\left\lfloor \frac{C_{ab,r}}{d(qx)} \right\rfloor$ (Lemma 3.2.1). Therefore, $cr(qx, b)$ is reduced only by selection of requests qy of demand less than $d(qx)$. Therefore, an az' transfer is not feasible for any request q : $d(q) \geq d(qx)$. $\square$

Lemma 6.7.3 establishes a similar finding for qx when $cr(qx, z)$ has sufficient space for qx and all requests of demand less than $d(qx)$ in $\bar{E}_{az}$.

Lemma 6.7.3 *Given RLP2[2], partition $\mathcal{P}_r$, F-zone bin z . $qx \in \bar{E}_{az}$ is a B-zone borrow. Let E' be the set of $q \in \bar{E}_{az}$ of demand less than $d(qx)$. If $cr(qx, z) \geq d(qx + E')$ then there exists no request q' , $d(q') \leq d(qx)$ interfered on z .*

Proof: First, an initial $d(qx)$ bin z capacity is required to route qx . Second, the bin z capacity can only be reduced by routing $q \in \bar{E}_{az,r}$. $\square$

Corollary 6.7.3.1 uses Lemma 6.7.3 to establish a condition similar to that of Lemma 6.7.2 Corollary 6.7.2.1 for the nonexistence of interfered requests on $\mathcal{P}_r$.

Corollary 6.7.3.1 *Same as Lemma 6.7.3. Let z' be the alternate F-zone link to z . qx is not interfered on z' . Then there does not exist an interfered request qf : $d(qf) \geq d(qx)$.*

Proof: Similar to Lemma 6.7.2 Corollary 6.7.2.1. $\square$

Lemma 6.7.4 establishes a criterion for the nonfeasibility of alternate alpha routing set $A_{r-t'}$. Routing A_{r-t} increases the minimum load on two independent links z and z' beyond the current minimum congestion. Therefore, routing $A_{r-t'}$, as described below, cannot be feasible:

Lemma 6.7.4 *Let $\mathcal{P}_r$ be an RLP2[2] partition for feasible routing r . For $\mathcal{P}_r$, $A_{r \rightarrow t}$ is an alpha routing set. After $A_{r \rightarrow t}$ is routed, there exist two independent links z and z' for which $\lceil \frac{L_z + L_{z'}}{2} \rceil$ exceeds the maximum feasible congestion. Therefore, $A_{r \rightarrow t}$ is not feasible (Observation 2.3.1 or Lemma 2.4.1).*

Let $A_{r \rightarrow t'}$ be an alpha routing set such that $d(\alpha_{t'}) > d(\alpha_t)$ and $z, z' \in P^+(\alpha_{t'})$. Then $A_{r \rightarrow t'}$ cannot form a feasible alternate alpha routing set.

Proof: $z, z' \in P^+(\alpha_t)$. Otherwise, r would not be feasible by the application of Observation 2.3.1 (or Lemma 2.4.1) to these two links.

Compare routing $A_{r \rightarrow t}$ and $A_{r \rightarrow t'}$. From reasoning similar to that for the Lemma 3.6.2 proof, $A_{r \rightarrow t'}$ cannot achieve a lesser congestion on $z, z' \in P^+(\alpha_{t'})$ than $A_{r \rightarrow t}$. Because z and z' are independent, and $z, z' \in P^+(\alpha_{t'})$ this cannot be a feasible congestion. $\square$

Lemma 6.7.4 establishes a criterion for the disqualification of alternate alpha routing set $A_{r \rightarrow t'}$ when $A_{r \rightarrow t}$ is nonfeasible as described above. Much of the time z is the t maximum load link. This lemma provides a quick means for the elimination of nonfeasible alternate alpha routing sets from the alpha tree. This lemma may be generalized for RLP2[k] application.

6.8 The RLP2[2] Partition Algorithm, Rload2-Part[2]

RLOAD2-PART[2] is constructed with the concepts discussed so far. In summary, the constraints on RLOAD2-PART[2] are:

- (i) the B-zone capacity C_{ab} .
- (ii) the left and right F-zone capacities C_{al} and C_{ar} .
- (iii) The feasible shortest path interval constraint.

In addition to these constraints, are the packing options afforded by the left and right interdependent zones. The strategy of RLOAD2-PART[2] is:

- (i) Pack E_a bandwidth within the C_{al} , C_{ar} and C_{ab} bin constraints in greedy fashion.

- (ii) When packing equal demand requests, always select a $\delta(q) = 2$ request rather than a $\delta(q) = 1$ request when given the choice.
- (iii) Do not pack a $\delta(q) = 1$ demand d request into bin y if the requests for less than demand d remaining on $\bar{E}_{ay}$ provide sufficient bandwidth to pack c_{ay} .
- (iv) Find the optimal alternate terminal routing. Associated with an alternate terminal routing is the demand d of the optional T_t request qt and the independent zone terminal capacity $ct(b)$. The alternate terminal routing with the minimum $ct(b)$ value is the optimal alternate terminal routing.

RLOAD2-PART[2] is presented as a generic algorithm for routing r . The inputs are the r eligible set $E = E_{a,r}$ and capacity array C . For simplicity, the division of the X list into the B and D sublists is not explicitly described. The RLOAD2-PART[2] outline is:

- 1 - **initialization:** Sort E if it is not already sorted. Initialize the current capacity to c to C . Initialize the alternate terminal routing capacity.
- 2 - **Repeat:** Find the maximum demand denomination di on E . Move all di requests on E to List 1.
- 3 - **Select requests:** Select the maximum possible number of $\delta = 2$ requests from List 1. Then select all feasible $\delta = 1$ requests on List 1. Reject the requests remaining on List 1. Update the alternate terminal routing information. After selection, save the current capacity array as a reduction capacity array.
- 4 - **reiteration:** Return to step 2 if E nonempty. Else, terminate.

Rload2-Part[2] RLP2[2] bin packing routine

Inputs E_list eligible request set.
 C capacity array
 $C[b]$ = B-zone capacity
 $C[l]$ = left F-zone link capacity
 $C[r]$ = right F-zone link capacity

Outputs S_list selection list
 X_list rejection list

```

RLOAD2-PART[2]( $E\_list, C, S\_list, X\_list$ )
1 Sort  $E\_list$  into descending order of  $d(q)$ ,  $q \in E\_list$ ;
  Sort equal demand requests by terminal set
2  $c \leftarrow C$ ;
3  $altterm.cb \leftarrow \infty$ ;
3 while  $E\_list \neq \text{NIL}$ 
4    $di \leftarrow \text{MAX}(d(q), q \in E\_list)$ ;
5    $list1 \leftarrow \{q \in E\_list, d(q) = di\}$ ;
6    $E\_list \leftarrow E\_list - list1$ ;
7    $list2 \leftarrow \{q \in E\_list, d(q) < di\}$ ;
8   SELECT-2S( $c, di, list1, S\_list$ );
9   SELECT-1S( $c, di, list1, list2, S\_list, X\_list, altterm$ );
10   $clist[\log_2 di] \leftarrow c$ 
  Return:

```

Line 1 sorts E_list into descending order. Line 2 initializes the current capacity array to C . Line 3 initializes the alternate terminal routing capacity to infinity. Lines 4-9 are repeated until E_list has been completely partitioned into S_list and X_list . Line 4 assigns di to the maximum of the demands on the sorted E_list . Lines 5-6 move all di requests from E_list to the $list1$ holding list. Line 7 moves the remainder of E_list to $list2$. In actual practice this move does not happen: $list2$ is used only to denote the remaining portion of E_list . Line 8 selects $\delta(q) = 2$ requests and if all $\delta(q) = 2$ requests are selected then line 9 selects any required $\delta(q) = 1$ request. Line 10 saves the current capacity array for demand di requests.

select-2s Pack $\delta(q)=2$ requests into the C_{ab} bin

Inputs c current capacity array
 di demand of requests being packed.
 $list1$ eligible request set of q , $d(q) = di$
 S_list selection list

Outputs c updated current capacity array
 $list1$ $list1$ - selected $\delta(q)=2$ requests
 S_list updated selection list

Remarks On input $list1$ contains all eligible q , $d(q) = di$ requests. The $\delta(q)=2$ requests are moved from $list1$ to S_list . No requests are placed onto X_list yet.

SELECT-2S($c, di, list1, S_list$)
1 $b[l] \leftarrow \text{MIN}(\text{SIZE}(list1[l]), \lfloor \frac{c[r]}{di} \rfloor)$;
 $b[r] \leftarrow \text{MIN}(\text{SIZE}(list1[r]), \lfloor \frac{c[l]}{di} \rfloor)$;
 $b[b] \leftarrow \text{MIN}(b[l] + b[r], \lfloor \frac{c[b]}{di} \rfloor)$;
2 select: any $n[l]$ $q \in list1[l]$, any $n[r]$ $q \in list1[r]$,
 $n[l] + n[r] = b[b]$, $n[l] \leq b[l]$, $n[r] \leq b[r]$;
3 move the selected pairs to S_list ;
4 $c[b] \leftarrow c[b] - b[b] * di$;
 $c[l] \leftarrow c[l] - n[r] * di$;
 $c[r] \leftarrow c[r] - n[l] * di$;

Line 1 establishes the left and right terminal set bounds based upon available bandwidth and capacity constraints. It then establishes the independent zone bound based upon the sum of the terminal set bounds and the independent zone capacity. Line 2 selects an arbitrary set of requests which pack c_{ab} within the established bounds. Line 3 moves the requests to S_list . Line 4 updates the current capacities.

select-1s Pack $\delta(q)=1$ requests into the C_{ab} bin

Inputs

c	current capacity array
di	demand of requests being packed.
$list1$	eligible request set of q , $d(q) = di$
$list2$	eligible request set of q , $d(q) < di$
S_list	selection list
X_list	rejection list
$altterm$	alternate terminal routing

Outputs

c	updated current capacity array
$list1$	empty
S_list	updated selection list
X_list	updated rejection list
$altterm$	updated alternate terminal routing

Remarks On input, $list1$ contains all remaining eligible q , $d(q) = di$. All $\delta(q) = 2$ requests on $list1$ should be selected before any $\delta(q) = 1$ requests. therefore, if $\delta(q)=2$ requests remain on $list1$ then no $\delta(q)=1$ requests are selectable. Thus assume when selecting requests that only $\delta(q)=1$ requests are available.

All $list1$ requests are moved to S_list and X_list .

```

SELECT-1s( $c, di, list1, list2, S\_list, X\_list, altterm$ )
1  $b[b] \leftarrow \text{MIN}(\text{SIZE}(list1[t]), \left\lfloor \frac{c[b]}{di} \right\rfloor, \left\lceil \frac{c[b]-d(list2)}{di} \right\rceil)$ ;
    $b[l] \leftarrow \text{MIN}(\text{SIZE}(list1[t]), \left\lfloor \frac{c[l]}{di} \right\rfloor, \left\lceil \frac{c[l]-d(list2[r]+list2[t])}{di} \right\rceil)$ ;
    $b[r] \leftarrow \text{MIN}(\text{SIZE}(list1[t]), \left\lfloor \frac{c[r]}{di} \right\rfloor, \left\lceil \frac{c[r]-d(list2[l]+list2[t])}{di} \right\rceil)$ ;
2  $b1 \leftarrow \text{MIN}(b[b], b[l], b[r])$ ;
3 if  $b1 > 0$ 
4   then move any  $b1$  requests from  $list1[t]$  to  $S\_list$ :
      move the remaining requests to  $X\_list$ :
5      $c[b] \leftarrow c[b] - b1 * di$ ;
       $c[l] \leftarrow c[l] - b1 * di$ ;
       $c[r] \leftarrow c[r] - b1 * di$ ;
6   if  $c[b] + di - d(list2) < altterm.cb$ 
7     then  $altterm.d \leftarrow di$ ;

```

8 $altterm.cb \leftarrow c[b] + di - d(list2);$
 Return:

Line 1 establishes the B-zone and F-zone lower bounds on selectable *list1* T_t requests based upon three factors. The first is available T_t bandwidth on *list1*. The second is capacity constraints: the selection of a *list1* T_t request reduces all capacities by di . The third is the restriction to select a di T_t request only if insufficient *list2* bandwidth is available to pack the capacity. This is because the *list2* bandwidth may contain $\delta(q) = 2$ requests which require capacity on only one F-zone link. Line 2 takes the minimum of these bounds as the T_t bandwidth requirement. Lines 3-4 move the required requests to *S_list* while the remainder of *list1* goes to *X_list*. Line 5 updates the current capacities. Line 6 checks if a di T_t request yields the optimal alternate terminal routing and, if yes, lines 7-8 update *altterm*.

E_list contains a maximum of m requests. The computational complexity of sorting m items is $O(m \log_2 m)$. The partition process checks each request once which requires $O(m)$ complexity. With sorting, the total value is $O(m \log_2 m)$.

6.9 Rload2-Bpack[2]

RLOAD2-BPACK[2] uses the beta tree data structure where each node represents beta routing rx of $\mathcal{Q}$, and each edge represents a beta routing set. The root node represents the initial beta routing which is the current alpha tree node routing r . Figure 6.2 depicts a RLOAD2-BPACK[2] beta tree.

For the RLOAD2-BPACK[2] description and proof of correctness keep in mind:

- (i) The common request qc for a beta transfer β_{af} is a B-zone borrow request (Lemma 6.6.1). Therefore, it is verifiable that, after β_{af} is routed, c_f exceeds c_b so that c_f is sufficient to pack all necessary bandwidth into c_b .
- (ii) Beta transfers are feasible for a maximum of 1 interference link $f \in Z_f$ per partition (Lemma 6.6.3).

Figure 6.2: RLOAD2-BPACK[2] Beta Tree

- (iii) Beta transfers are selected in greedy fashion. qb is the maximum demand request eligible for replacement according to Lemma 5.7.1.
- (iv) After $\mathcal{J}_{af}$ is routed, the transfer gain Δ is an upper bound on the bandwidth packed back into c_{af} (Lemma 6.6.2). Therefore, the gain Δ_1 on a subsequent af' beta transfer must be less than Δ .
- (v) After $\mathcal{J}_{af}$ is routed, no requests are interfered on f (Lemma 6.7.1).

$\mathcal{P}_{rx}$ is a partition for beta routing rx . $E_{a,rx}$ is the eligible request set for rx . For $\mathcal{P}_{rx}$ and F-zone link f , beta transfer $\mathcal{J}_{af}$ defines beta routing set $B_{rx \rightarrow ry}$ which maps from beta routing rx to beta routing ry . The eligible set for ry , $E_{a,ry}$, is a subset of $E_{a,rx}$. Section 6.6 describes the selection of $E_{a,ry}$. For $\mathcal{J}_{af}$, qb is the beta request and qc represents a common request.

An overview of RLOAD2-BPACK[2] is now given:

1 - Initialization: Assign E to $E_{a,r}$ and C to the r C array. Assign rx to r .

- 2 - **Partition:** Call RLOAD2-PART[2] to pack bandwidth from E into C for rx . Refer to this packing as Partition 1.
- 3 - **Reiteration and Termination:** On Partition 1 check for:
 - 3 - 1 $L_{min,r}$: Lemma 3.4.1 Corollary 3.4.1.1 is used to check for $L_{min,r}$. A B-zone 0 terminal capacity also denotes $L_{min,r}$. In either case, the algorithm terminates.
 - 3 - 2 **optimal packing:** Lemma 6.7.2 Corollary 6.7.2.1 and Lemma 6.7.3 Corollary 6.7.3.1 determine if an optimal packing has occurred. If yes, the algorithm terminates.
 - 3 - 3 **interfered request:** Let $qminf$ be the minimum demand interfered request. Let f denote the F-zone link on which $qminf$ is interfered. For $qminf$ find beta transfers in greedy fashion. For each beta transfer, determine B_{rx-ry} . Assign E to $E_{a,ry}$ and C to the B_{rx-ry} reduction capacity. Assign rx to ry . For the new rx , repeat steps 2 and 3 (recursive algorithm).

The initial RLOAD2-BPACK[2] partition $\mathcal{P}_r$ is stored at the root node. Each beta tree node stores a repartition of the partition stored at its ancestor. The beta tree is allocated only while r is the active alpha routing. This method saves memory required by alpha trees with a large number of nodes. Tables 6.1 and 6.2 list the RLOAD2[2] beta tree node data items.

The RLOAD2-BPACK[2] algorithm is now constructed:

Rload2-Bpack[2] RLP2[2] bin packing algorithm

Inputs *anode* routing r alpha tree node

Outputs *rootb* beta tree root node

```

RLOAD2-BPACK[2](anode.rootb)
1 ADD-BETA-ROOT(anode.bnode);
2 RLOAD2-PART[2](rootb);
3 READ-MIN-REJECTED-REQUESTS(rootb);
4 READ-PARTITION-STATUS(rootb);
5 ADD-BETA-BRANCHES(rootb);
Return:

```

Line 1 allocates and initializes the beta tree root node for the initial beta routing r . Line 2 partitions $E_{a,r}$. Line 3 reads the minimum rejected requests on $\mathcal{P}_r$. Line 4 reads the $\mathcal{P}_r$ partition status. Line 5 finds the necessary beta transfers for $\mathcal{P}_r$ and adds a branch to the beta tree for each repartition.

Add-beta-root Add root to beta tree

Inputs *anode* alpha tree node

Outputs *bnode* beta tree root node

```

ADD-BETA-ROOT(anode.bnode)
1 ALLOCATE(bnode);
2 bnode.pnode  $\leftarrow$  NIL;
3 bnode.Rt.qb  $\leftarrow$  NIL;
4 bnode.Rt.Alist  $\leftarrow$  NIL;

```

Data Item Name	Data Item Description	Remarks
Bt.qb	β_{af} beta request	not defined on root node
Bt.A_list	β_{af} reduction set $\tilde{B}(qb)$	not defined on root node; cumulative A_list from root node
Bt.C[l, r, b]	Bt.qb reduction capacities C_{al}, C_{ar}, C_{ab}	
pnode	pointer to ancestor node	not defined on root node
dcount	total number of descendant nodes	
dnode[]	pointers to descendant nodes	one for each descendant
c[b.l.r]	execution capacities	RLOAD2-PART[2] current capacities
clist[w+1][l.r.b]	clist[log ₂ (d)][l.r.b] = c[b.l.r] after all $q: d(q) \geq d$ have been selected by RLOAD2-PART[2]	effectively gives final capacities for $q \in E_list: d(q) \geq d$
E_list	rx eligible request set	
S_list	rx selection set	
X_list	rx rejection set	
qfmin[l.r]	minimum demand $q \in X_list: f \notin P^-(q)$. f is a F-zone link and q is interfered on link f	see Chapter 5 Section 5.3 for interference criteria
qbmin[l.r]	minimum demand $q \in X_list: f \notin P^-(q)$. f is a F-zone link and q is a B-zone borrow not interfered on link f	selected according to Lemmas 6.7.2 and 6.7.3
minr	flag to denote minimum congestion for alpha routing r	=1: minimum congestion for routing r obtained =0: final congestion may not be minimum for r
min	flag to denote optimal packing of Bt.C[b]	=1: optimum packing of C[b] obtained - may not be minimum congestion for r =0: final congestion may not be optimum

Table 6.1: RLOAD2[2] Beta Tree Node

NOTE TO USERS

The original manuscript received by UMI contains pages with poor print. Pages were microfilmed as received.

108

This reproduction is the best copy available.

UMI'

Data Item Name	Data Item Description	Remarks
qminb	minimum demand $q \in X_list$ when q is non-interfered B-zone borrow	min flag set if qminb exists
qminf	minimum demand $q \in X_list$ when q is interfered	min flag clear if qminf exists
tlimit[l, r]	flag to denote transfer limit on F-zone link f	=1 transfer limit = $c[f]$ on f (Lemma 6.6.2) =0 no transfer limit on f
altterm.qt	optional T_t request for minimum congestion alternate terminal routing set	de-selection of <i>altterm.qt</i> allows selection of the minimum congestion alt term routing set
altterm.cb	minimum alternate terminal routing independent zone capacity.	

Table 6.2: RLOAD2[2] Beta Tree Node Continued

```

5 bnode.Rt.C, bnode.c ← anode.Rt.C:
6 bnode.E_list ← anode.E_list:
7 bnode.S_list, bnode.X_list ← NIL:
8 bnode.altterm.cb ← ∞:
9 bnode.tlimit[l, r] ← 0:
  Return:

```

This module allocates and initializes the beta tree root node.

Add-beta-branches Add branches to beta tree

Inputs *bnode* routing *rx* beta tree node

Outputs *bnode* beta tree node with new descendant nodes

```

ADD-BETA-BRANCHES(bnode)
1 qf ← bnode.qminf; lf ← bnode.lnkf;
2 if qf exists
3   then Start finding-transfers:
4     dqcmay ← ∞;
5     while finding-transfers
6       FIND-TRANSFER(bnode.lf, d(qf).dqcmay.xfer):
7       if xfer.status TRUE
8         then add descendant node dnode to beta tree.
          save transfer E_list, xfer.C in dnode:
9         RLOAD2-PART[2](dnode);
10        READ-MIN-REJECTED-REQUESTS(dnode);
11        READ-PARTITION-STATUS(dnode);
12        ADD-BETA-BRANCHES(dnode);
13        if dnode.qminf = FALSE
14          then Stop finding-transfers:
15        if dnode.min TRUE
16          then bnode.min ← TRUE:
17        else Stop finding-transfers:
Return:

```

Line 1 reads the minimum demand rejected request qf and the F-zone link f on which qf is interfered. Only af beta transfers are feasible on this partition (Lemma 6.6.3). Lines 4-21 conduct a greedy search for as many transfers as are required for qf (max 2 for RLP2[2]).

Line 4 initializes the upper limit on beta transfer common requests. The greedy search is conducted in descending order of demand of the feasible common requests qc . For qc , a minimum demand transfer request qb is read. A check occurs for a feasible transfer for qb . The search is repeated until all required transfers have been found or no more qc candidates are left. $dqcmay$ is updated on each call to FIND-TRANSFER.

Line 6 calls FIND-TRANSFER to find a feasible transfer $\mathcal{J}_{af}$ for qf . If $\mathcal{J}_{af}$ does not exist, the search terminates. If $\mathcal{J}_{af}$ exists, let $B_{rx \rightarrow ry}$ be the beta routing set. Line 8 adds a descendent node to the beta tree for ry . The eligible request set $E_{a,ry}$ and capacity array for ry are read from the transfer data. Line 9 partitions $E_{a,ry}$. Line 10 reads the $\mathcal{P}_{ry}$ minimum demand rejected requests. Line 11 reads the ry partition status. Line 12 calls this

module recursively for ry .

Line 13 checks for an interfered request on $\mathcal{P}_{ry}$. If none exists, no additional repartitions of $\mathcal{P}_{rx}$ are necessary and the search terminates.

Add-beta-node Add node to beta tree for routing ry

Inputs $pnode$ ancestor of new node - represents routing rx
 $xfer$ transfer table
 contains qb , qc and ry capacity array

Outputs $bnode$ new node - represents routing ry

```

ADD-BETA-NODE( $pnode.bnode.xfer$ )
1 ALLOCATE( $bnode$ ):
2  $bnode.pnode \leftarrow pnode$ :
   $bnode.dcount \leftarrow 0$ :
3  $bnode.Rt.qb \leftarrow xfer.qb$ :
4  $bnode.Rt.Alist \leftarrow pnode.Alist +$ 
   $\{q \in pnode.Slist : d(q) \geq d(xfer.qb)\} - xfer.qb$ :
5  $bnode.Rt.C.bnode.c \leftarrow xfer.C$ :
6  $bnode.Elist \leftarrow \{q \in pnode.Elist : d(q) < d(xfer.qb)\} + xfer.qc$ :
7  $bnode.Slist.bnode.Xlist \leftarrow NIL$ :
8  $bnode.altterm.cb \leftarrow \infty$ :
9  $bnode.tlimit \leftarrow pnode.tlimit$ :
   $bnode.tlimit[xfer.lf] \leftarrow TRUE$ :
  Return:

```

Line 1 allocates the new beta node $bnode$. Line 2 assigns $bnode$ pointers. Lines 3–4 assign $bnode.Rt$ to B_{rx-ry} . The $Alist$ for the root node is stored. Line 5 assigns the beta routing capacity array to the $crb(qb)$ reduction capacity. Line 6 assigns the ry eligible request set. Line 7 initialize the selection and rejection sets. Line 8 initializes the alternate terminal routing capacity. Line 9 initializes the transfer limit flags to the $bnode$ settings. The flag for the interference link is then set.

Read-min-rejected-requests Read minimum demand rejected requests

Inputs $bnode$ routing rx beta tree node

Outputs $bnode.qfmin$, $bnode.qbmin$

```

READ-MIN-REJECTED-REQUESTS( $bnode$ )
1 for each F-zone link  $lf$ 
2  $bnode.qfmin[lf] \leftarrow NIL$ ;  $bnode.qbmin[lf] \leftarrow NIL$ ;
3  $X' = \{q' : q' \in bnode.X\_list \text{ and } lf \notin P^-(q')\}$ ;
4 Find  $qb \in X'$ :  $qb$  is a minimum demand B-zone borrow not interfered on  $lf$ ;
5 if  $qb$  exists
6   then  $bnode.qbmin[lf] \leftarrow qb$ ;
7   else Find  $qf \in X'$ :  $qf$  is a minimum demand request interfered on  $lf$ 
8       if  $qf$  exists
9         then  $bnode.qfmin[lf] \leftarrow qf$ ;
10      READ-MAXIMAL-BORROW-REQUEST( $bnode.bnode.qfmin[lf]$ ,  $lf$ ):
Return:

```

Line 1 repeats the process for each F-zone link f . Line 4 reads the minimum demand B-zone borrow request qb not interfered on f (Lemmas 6.7.2 and 6.7.3). If a tie occurs among equal demand candidates, the request with the maximal shortest path interval is chosen.

If qb exists then no transfers are required to f and no check for qf occurs. Otherwise, Line 7 reads the minimum demand request qf interfered on f . qf is read from the B_list and D_list subtables of X_list according to the interfered request criteria laid out by Chapter 5 Section 5.3. If a tie occurs among equal demand candidates, the request with the maximal shortest path interval is chosen. Line 10 checks that qf is a feasible alpha request according to Lemma 3.6.4.

Read-maximal-borrow-request Read maximal borrow request

Inputs *bnode* routing *rx* beta node
 qa borrow request
 f F-zone link

Outputs *qa* maximal borrow request

Remarks This module accepts a request *qa* whose shortest path does not include *f* and checks that *qa* forms a feasible alpha routing set according to Lemma 3.6.4. If not, the maximal borrow request is returned as *qa*.

READ-MAXIMAL-BORROW-REQUEST(*bnode*, *qa*, *lf*)

```

1 Start Reading requests:
2 while Reading requests
3   Read minimum demand q:  $q \in bnode.X\_list$  and  $d(q) > d(qa)$ ;
   and  $lf \in P^-(q)$ .
4   if q exists
5     thenif  $bnode.clst[\log_2 d(q)][lf] - bnode.clst[\log_2 d(qa)][lf] = d(qa)$ 
6       then qa ← q;
7       else Stop Reading requests:
8     else Stop Reading requests:
Return:
```

Line 2 repeats until the maximal *qa* is found. Line 3 reads the minimum demand rejected request *q* of demand greater than the current *qa*. To be maximal, its shortest path must include link *f*'. If *q* exists the reduction capacities of *q* and *qa* are checked to see which request is maximal according to Lemma 3.6.4.

read-partition-status read partition status

Inputs *bnode* routing *rx* beta tree node

Outputs *bnode* beta tree node with partition data


```

READ-PARTITION-STATUS(bnode)
1 bnode.qminf  $\leftarrow$  NIL; bnode.qminb  $\leftarrow$  NIL;
  bnode.lnkf  $\leftarrow$  NIL;
  bnode.minr  $\leftarrow$  FALSE; bnode.min  $\leftarrow$  FALSE;
2 if bnode.c[b]  $\neq$  0
3   then let qmin be a minimum demand rejected request on bnode:
      (for ties, chose a bnode.qbmin request if possible)
4     if qmin  $\neq$  NIL
5       thenif qmin denotes the minimum congestion for alpha routing r
          according to Lemma 3.4.1 Corollary 3.4.1.1
6         then bnode.minr  $\leftarrow$  TRUE; bnode.min  $\leftarrow$  TRUE;
7         elseif qmin satisfies either Lemma 6.7.2 Corollary 6.7.2.1 or
          Lemma 6.7.3 Corollary 6.7.3.1.
8         then bnode.qminb  $\leftarrow$  qmin; bnode.min  $\leftarrow$  TRUE;
9         else bnode.qminf  $\leftarrow$  qmin;
          Assign bnode.lnkf to the link on which qf is interfered:
10      else bnode.min  $\leftarrow$  TRUE;
11 else bnode.minr  $\leftarrow$  TRUE; bnode.min  $\leftarrow$  TRUE;
  Return:

```

Line 1 initializes the partition status data. Line 2 checks if the B-zone capacity is 0. This condition denotes $L_{min,r}$. Line 3 reads the minimum demand rejected request *qmin* from *bnode.qbmin* and *bnode.qfmin*. If no rejected request exists then an optimal packing of $E_{a,rz}$ has occurred. Line 5 checks for $L_{min,r}$. Line 7 checks if *qmin* represents an optimal packing of $E_{a,rz}$. If yes, line 8 saves *qmin* as *bnode.qminb*. Otherwise, line 8 saves *qmin* as *bnode.qminf* and saves the F-zone link *f* on which it is interfered.

Find-transfer find a beta transfer

Inputs *bnode* routing *rx* beta tree node
 lf *qf* interference link *f*
 dqf *d(qf)*
 dqcmaz upper limit on common request demand *d(qc)*

Outputs *dqcmaz* upper limit on *d(qc)* for next call
 xfer transfer data

FIND-TRANSFER(*bnode.lf, dqf, dqcmaz, xfer*)

```
1 Place into C_list one  $q \in bnode.B\_list$  of each bandwidth denomination such that:
  a.  $q$  is an independent zone borrow:
  b.  $q$  is not an interdependent zone borrow:
  c.  $lf \in P^-(q)$ :
  d.  $d(q) > dqf$ ;
  e.  $d(q) \leq dqcmaz$ :
2 Place into B_list one  $q \in bnode.S\_list$  of each bandwidth denomination such that:
  a.  $lf \in P^-(q)$ :
  b.  $d(q) > dqf$ :
3 xfer.status  $\leftarrow$  FALSE:
4 while C_list  $\neq$  NIL
5   qc  $\leftarrow$  maximum demand C_list request:
6   qb  $\leftarrow$  minimum demand B_list request of demand
   greater than or equal to d(qc)
7   if qb exists
8     thenif there exists d(qb) bandwidth which includes qc on bnode.X_list
9       then xfer.qb  $\leftarrow$  qb:
         xfer.qc  $\leftarrow$  qc:
         xfer.lf  $\leftarrow$  lf:
         Assign to xfer.C qb beta reduction capacity:
         bnode.Rt.C[lf]  $\leftarrow$  d(qb) - 1:
         xfer.status  $\leftarrow$  TRUE:
         dqcmaz  $\leftarrow$  d(qc)/2:
10        C_list  $\leftarrow$  NIL:
11      else Remove from C_list. B_list all  $q \geq d(qc)$ :
12    else Remove from C_list. B_list all  $q \geq d(qc)$ :
Return:
```

Line 1 lists common requests which are valid according to Lemmas 5.7.1 and 6.6.1 and are within the algorithmic demand limit $dqcm_{ax}$. Line 2 lists beta (transfer) requests, which are valid according to these lemmas. Lines 4-12 search for a feasible beta transfer in greedy fashion:

Line 5 reads the maximum demand qc from C_list . Line 6 reads the minimum demand qb for which a replacement block may include qc . Line 8 checks that the required replacement bandwidth resides on from $bnode.X_list$. If yes, the transfer data is placed into the $xfer$ parameter and $dqcm_{ax}$ is updated for the next call. If no feasible transfer is found, the lists are updated and the search is reiterated.

Lemma 6.9.1 *For alpha routing r , RLOAD2-BPACK[2]:*

- (i) *finds an optimal packing of $E_{a,r}$ within the $C_{ab,r}$, $C_{al,r}$ and $C_{ar,r}$ limits.*
- (ii) *finds a minimum of one feasible alpha routing set (if it exists).*
- (iii) *has a maximum of 4 nodes in its beta tree.*

Proof:

For this proof, refer back to the list of concepts at the beginning of the section. Figure 6.2 illustrates the beta tree used by this proof.

r is the beta tree root node routing. On $\mathcal{P}_r$, request qf is interfered on F-zone link $f = z$. Therefore, there does not exist a feasible az' beta transfer (Lemma 6.6.3). Let $\beta_{az,1}$ and $\beta_{az,2}$ be two az beta transfers such that:

- (i) $\beta_{az,1}$ maps from r to $r1$. The beta request is $qb1$. The routing set is $B_{r \rightarrow r1}$. The transfer gain is Δ_1 .
- (ii) $\beta_{az,2}$ maps from r to $r2$. The beta request is $qb2$. The routing set is $B_{r \rightarrow r2}$. The transfer gain is Δ_2 . $d(qb1) > d(qb2)$.

Route $B_{r \rightarrow r1}$. On $\mathcal{P}_{r1}$, Lemma 6.7.1 assures the elimination of bin z interference for all feasible request selections. Lemma 6.6.1 assures sufficient bin z capacity to pack bin b (see above). Let $qf1$ be a request interfered on F-zone link $f = z'$. Let $\beta_{az',3}$ be an az' beta transfer such that:

- (i) $\beta_{az',3}$ maps from $r1$ to $r3$. The beta request is $qb3$. The routing set is $B_{r1 \rightarrow r3}$. The transfer gain is Δ_3 .

Route $B_{r1 \rightarrow r3}$. On $\mathcal{P}_{r3}$, Lemma 6.7.1 assures the elimination of bin z' interference. Lemma 6.6.1 assures sufficient bin z' capacity to pack bin b (see above). Lemma 6.6.2 assures that $\Delta_3 < \Delta_1$.

Route $B_{r \rightarrow r2}$. It is verifiable that routing $\beta_{az,2}$ is equivalent to routing an az transfer followed by an az' transfer. Therefore, Lemma 6.7.1 assures the elimination of interference on bins z and z' on $\mathcal{P}_{r3}$. Lemma 6.6.1 assures sufficient capacity on each F-zone bin to pack bin b (see above).

Let β_{af} be an arbitrary beta transfer which maps routing $r1$ to $r2$. Because of the greedy nature of transfer and common request selection, β_{af} maximizes its capacity transfer to link f . Therefore, the quantity and resolution of non-interfered bandwidth routable to bin f on $\mathcal{P}_{r2}$ are maximized.

The beta tree routes the necessary beta transfers to eliminate interference on all F-zone links. The greedy nature of the algorithm minimizes interference and maximizes the available bandwidth quantity and resolution as described above. These aspects allow:

- (i) An optimal packing of $E_{a,r}$.
- (ii) Interference on B-zone borrow requests to be minimized.
- (iii) Minimization of the demand of borrow requests on feasible alpha routing sets.

There is a maximum of three beta transfers in the beta tree. Therefore, the beta tree has a maximum of 4 nodes. $\square$

E_list contains a maximum of m items. The computational complexity of sorting m requests is $O(m \log_2 m)$.

RLOAD2-PART[2] is an $O(m)$ algorithm without sorting. FIND-TRANSFER requires $O(m)$ for the compilation of B_list and C_list plus 1 scan of these lists for a beta transfer.

The maximum of 4 nodes in the beta tree may be regarded as a constant. For each node, RLOAD2-PART[2] is executed and for each descendant node. Find-Transfer is required for its ancestor. The total complexity without sorting remains at $O(m)$. With sorting the complexity is $O(m \log_2 m)$.

6.10 Rload2[2]

The alpha tree has one tree node *anode* for each alpha routing r . The root node denotes the initial routing r_{init} . Each edge represents an alpha routing set $A_{p \rightarrow r}$ which maps to routing r from the routing p represented by its parent node. $A_{p \rightarrow r}$ is stored in *anode*.

An overview of RLOAD2[2] is:

- 1 - **initialization:** Sort the requests in Q . Add the root node to the alpha tree. Assign r to r_{init} . Assign p to NIL. Assign $A_{p \rightarrow r}$ to NIL. Calculate the load array L for r_{init} .
- 2 - **Route from p to r :** Determine for r :
 - (i) The load array L .
 - (ii) The maximum load link a .
 - (iii) The routing zones, the B-zone maximum load link b and the F-zone links.
 - (iv) The eligible request set $E_{a,r}$.
 - (v) The capacity array C . Initialize the execution capacity array c to C .
 - (vi) The Lemma 3.5.1 limits on alpha routing request demands.
- 3 - **Rload2[2]:** Call RLOAD2[2] to find $L_{min,r}$:
 - 3.1 - **r feasibility:** Check that r is feasible according to Observation 2.3.1 and Lemma 2.4.1. If yes, proceed at step 3.2. If not, prune nonfeasible alpha tree descendants of p according to Lemma 6.7.4 and do nothing more.
 - 3.2 - **Rload2-Bpack[2]:** Call RLOAD2-BPACK[2] to perform the RLP2[2] bin packing function for r . Update the minimum attained congestion to date $Lmin$.
 - 3.3 - **terminal routing:** From the beta tree, read and store the optimal terminal routing from step 3.2.
 - 3.4 - **feasible alpha routing sets:** From the beta tree, read the feasible minimum and alternate alpha routing sets $A_{r \rightarrow t}$. For each feasible $A_{r \rightarrow t}$:

- 3.4.1 Save p as s . Assign p to r . Assign r to t .
 - 3.4.2 Add node r to the alpha tree as a descendant of node p .
 - 3.4.3 Repeat steps 2 and 3 for r (recursive algorithm).
 - 3.4.4 Prune nonfeasible alpha tree descendants of node p based upon the results of step 3.4.3.
 - 3.4.5 Restore r to p and p to s . Recalculate the L array for p .
- 4 - r_{opt} : Read the approximation to optimal routing r_{opt} from the alpha tree.

Parameters

Table 6.3 describes the RLOAD2[2] input/output parameters.

Data Item Name	Data Item Description	Remarks
n	input	number of optical nodes (links) on $\mathcal{R}$
m	input	number of requests in Q
$i[m]$	input	array of request terminal i nodes
$j[m]$	input	array of request terminal j nodes
$d[m]$	input	array of request bandwidth demands
w	input	$d_{max} = 2^w$
$\phi[m]$	output	path assignment for request q ; 0 = shortest path. 1 = alternate path

Table 6.3: RLOAD2[2] Input/Output Parameters

Global data

This data is applicable to routing r represented by the currently accessed alpha tree node, $anode$. It is updated each time a new alpha tree node (descendant or return to ancestor) is accessed. Therefore, only a single copy of the data is required. Table 6.4 describes the RLOAD2[2] global data.

Data Item Name	Data Item Description	Remarks
$\phi[m]$	routing r for $anode$; $r = \phi$	
$L[n]$	total load (congestion) on link ℓ	

Table 6.4: RLOAD2[2] Global Data

Alpha tree

Table 6.5 lists the RLOAD2[2] alpha tree node data items.

Description

RLOAD2[2] is now described in a modular format.

Rload2[2] Find RLP2[2] optimal routing.

Inputs RLOAD2[2] inputs n, k, i, j, d, w

Outputs ϕ RLP2[2] optimal routing

RLOAD2[2]-ROOT(n, m, i, j, d, w)

1 INIT- $\mathcal{R}(n, m, i, j, d, w)$:

2 ADD-ALPHA-ROOT($roota$):

3 ROUTE($roota$):

4 RLOAD2[2]($roota$):

5 READ-ROPT($roota$):

6 Deallocate the alpha tree:

Return:

Line 1 initializes the global data structures. Line 2 adds the alpha tree root node. Line 3 routes to r_{init} by initializing the root node data structures. Line 4 calls RLOAD2[2] to find $L_{min,init}$. Line 5 assigns the optimal routing to the final value of ϕ . Line 6 deletes the alpha tree.

Data Item Name	Data Item Description	Remarks
Rt.qa	$A_{p \rightarrow r}$ alpha request	not defined on root node
Rt.A_list	$A_{p \rightarrow r}$ reduction set	not defined on root node
Rt.C[l, r, b]	$C_{ab,r}, C_{al,r}, C_{ar,r}$	
Rt.term	terminal routing set	
pnode	pointer to ancestor node	not defined on root node
dcount	total number of descendant nodes	
dnode[]	pointers to descendant nodes	one for each descendant
lnk[a]	$\ell_1 = a$ for r	
lnk[b]	$\ell_2 = b$ for r	
lnk[l]	ℓ_l for r	
lnk[r]	ℓ_r for r	
plnkf	PI alpha routing set interference link	
Zf	F-zone link set	
Zb	B-zone link set	
E_list	eligible request list for r	
Lmin	minimum overall congestion to this point by the algorithm	serves as a bound to evaluate the feasibility of subsequent routings
Lminr	minimum routing r congestion	allows <i>anode</i> to be pruned if its subtree does not reduce the overall congestion
plnka	ℓ_1 for <i>anode.pnode</i>	not defined on root node
alimit.ind	maximum demand for a feasible alpha request qa : $anode.plnka \notin P^-(qa)$	upper limit on feasible alpha request
alimit.com	maximum demand for a feasible alpha request qa : $anode.plnka \in P^-(qa)$	upper limit on feasible alpha request

Table 6.5: RLOAD2[2] Alpha Tree Node

Init- $\mathcal{R}$ Initialize global data

Inputs RLOAD2[1] inputs n, k, i, j, d, w

Outputs Global data initialized as below

INIT- $\mathcal{R}(n, m, i, j, d, w)$

1 Clear each $\phi[q]$ to 0;

2 Sort $(i[q], j[q])$ into ascending order of $i[q]$;

For $q_1, q_2, i[q_1] = i[q_2]$, sort into descending order of $\delta(q)$;

For $q_1, q_2, i[q_1] = i[q_2], j[q_1] = j[q_2]$, sort into descending order of $d(q)$;

3 From the i, j, d and ϕ vectors, calculate $L[\ell]$ for all $\ell \in \mathcal{R}$;

Return:

Line 1 assigns ϕ to the initial routing r_{init} . Line 2 sorts the requests, first into ascending order of $i[q]$, then into descending order of shortest path length, then into descending order of $d[q]$. Request sorting facilitates the calculation of link loads and the determination of eligible sets. Line 3 calculates the link loads for r_{init} .

Add-alpha-root Add root node to the alpha tree

Inputs none

Outputs $anode$ alpha tree root node.

ADD-ALPHA-ROOT($anode$)

1 ALLOCATE($anode$):

2 $anode.Rt.qa \leftarrow \text{NIL}$;

3 $anode.Rt.A_list, anode.Rt.term \leftarrow \text{NIL}$;

4 $anode.pnode \leftarrow \text{NIL}$;

5 $anode.plnka \leftarrow \text{NIL}$;

6 $anode.Lmin, anode.Lminr \leftarrow \infty$;

Return:

This module allocates and initializes the alpha tree root node.

Read-ropt Read optimal routing

Inputs *anode* alpha tree root node.

Outputs *phi* r_{opt} flag settings

Remarks recursive subroutine

```

READ-ROPT(anode)
1  if anode has a descendant dnode
2  then anode  $\leftarrow$  dnode;
3      for  $q \in dnode.Rt.A\_list$ 
4           $phi[q] \leftarrow 1$ ;
5       $phi[dnode.Rt.qa] \leftarrow 1$ ;
6      READ-ROPT(anode);
7  else For  $q \in anode.Rtterm$   $phi[q] \leftarrow 1$ ;
    Return:

```

This module traverses the optimal routing sequence. Lines 3–4 set the *phi* flags for non-terminal routing set requests. Line 5 recursively invokes the procedure for a non-terminal node. Line 6 sets the *phi* flags for terminal routing set requests.

Route Route from routing *p* to routing *r*

Inputs *anode* represents routing *r*

Outputs *L* with *r* link loads:
 r eligible list, capacity array, routing zones stored in *anode*

```

Route(anode)
1  $R\_list \leftarrow anode.Rt.A\_list + anode.Rt.qa$ ;
2 From  $R\_list$ , calculate  $L$  array for  $r$ ;
3  $anode.lnk[a] \leftarrow a: L[a] \geq L[\ell]: \ell \in \mathcal{R}$ ;
4 READ-E_LIST( $anode$ );
5 READ-ROUTING-ZONES( $anode$ ):
6 READ-ROUTING-C( $anode$ ):
7 READ-ALIMITS( $anode$ ):
Return:

```

This module routes from p to r using $A_{p \rightarrow r}$ stored in $anode$. Line 1 assigns $A_{p \rightarrow r}$ to R_list . Line 2 updates the L array for r . Line 3 assigns the maximum load link for r ; ties are broken arbitrarily. Line 4 reads the eligible request set E_list . Line 5 reads the routing zones. Line 6 reads the capacity array for r . Line 7 reads the alpha request demand limits.

Read-E_list Route eligible request set

Inputs $anode$ represents routing r .

Outputs $anode$ with routing r eligible list

```

READ-E_LIST( $anode$ )
1  $a \leftarrow anode.lnk[a]; qa \leftarrow anode.Rt.qa; pi \leftarrow anode.pilnk$ ;
2  $anode.E\_list \leftarrow \{q \in \mathcal{Q}: a \in P^-(q) \text{ and } d(q) < d(qa)\}$ ;
3 if  $pi \neq \text{NIL}$ 
4    then Add to  $anode.E\_list$  1  $q: \{a, pi\} \in P^-(q)$  and
             $d(q) = d(qa)$ ;
Return

```

Line 2 reads $E_{a,r}$. The eligible request demands meet the Lemma 3.5.1 bounds. Line 3 checks if $R_{p \rightarrow r}$ is a PI alpha routing set. If yes, line 4 adds a transfer request of equal demand to $anode.Rt.qa$. Routing this transfer

request violates the Lemma 3.5.1 limit. However, it routes a PI transfer when no other transfer exists.

Read-routing-zones Read routing zones

Inputs *anode* represents routing *r*

Outputs *anode* with routing *r* lnk array, routing zones:
lnk array is defined in Table 6.5

```

READ-ROUTING-ZONES(anode)
1 anode.lnk[l]  $\leftarrow$  (anode.lnk[a] + 1) MOD(n);
  anode.lnk[r]  $\leftarrow$  (anode.lnk[a] + n - 1) MOD(n);
2 for lnk = 'l', 'r'
3   if  $\nexists q \in \text{anode.E\_list}: \text{anode.lnk[lnk]} \in P^-(q)$ 
4     then anode.lnk[lnk]  $\leftarrow$  NIL;
5 anode.Zf  $\leftarrow$  {anode.lnk[l]  $\div$  anode.lnk[r]};
6 anode.Zb  $\leftarrow$  {R - anode.lnk[a] - anode.Zf};
7 anode.lnk[b]  $\leftarrow$  b: b  $\in$  anode.Zb and
  L[b]  $\geq$  L[l]: l  $\in$  anode.Zb;
Return:

```

Lines 1-4 determine the F-zone links. NIL denotes a nonexistent F-zone link. Line 5 assigns the F-zone. Line 6 assigns the B-zone. Line 7 determines the B-zone maximum load link: ties are broken arbitrarily.

Read-routing-C Read *r* routing capacity

Inputs *anode* represents routing *r*

Outputs *anode* with *r* capacity array *C*

```

READ-RT-C(anode)
1 anode.Rt.C[b]  $\leftarrow \lfloor \frac{L[a] - L[b]}{2} \rfloor$ ;
2 for lnk = 'l', 'r'
3   if lnk  $\in$  anode.Zf
4     then anode.Rt.C[lnk]  $\leftarrow \lfloor \frac{L[a] - L[lnk]}{2} \rfloor$ ;
5     else anode.Rt.C[lnk]  $\leftarrow \infty$ ;
Return:

```

Line 1 reads the B-zone capacity. Lines 2-5 read the F-zone capacities. If an F-zone link does not exist, its capacity is assigned to infinity.

Read-alimits Read alpha request demand limits

Inputs *anode* represents routing *r*

Outputs *anode* with alpha limits as defined by Table 6.5.

Remarks This module establishes an exclusive upper bound on the demand of alpha requests in alpha routing sets routed from $\mathcal{P}_r$. This bound is based upon Lemma 3.5.1. These bounds are based upon links *a* and *plnka*, the maximum load link for the ancestor of *anode*. There are two bounds:

- (i) the independent bound for requests *q*: *plnka* $\notin P^-(q)$
- (ii) the common bound for requests *q*: *plnka* $\in P^-(q)$.

For RLP2[2], the common bound applies only to requests for which the shortest path includes *a* and *plnka*.

```

READ-ALIMITS(anode)
1 if pnode  $\leftarrow$  anode.pnode  $\neq$  NIL
2   then  $\ell \leftarrow$  anode.plnka:
3     READ-ALIMIT(d(anode.Rt.qa), anode.Rt.c[ $\ell$ ], anode.alimit.ind):
4     for lf  $\in$  anode.Zf
5       if anode.lnk[lf] =  $\ell$ 

```

```

6         thenif  $anode.lnk[a] = pnode.plnka$ 
7             then  $anode.alimit.com \leftarrow pnode.alimit.com$ ;
8             else  $anode.alimit.com \leftarrow pnode.alimit.ind$ ;
9     else  $anode.alimit.ind, anode.alimit.com \leftarrow \infty$ ;
    Return:

```

Line 1 checks for the root node for which line 9 sets both limits to infinity. Line 3 determines the independent limit. Lines 4-5 check if $plnka$ is in the F-zone. If yes, lines 6-8 set the common limit based upon the ancestor node limits.

```

READ-ALIMIT( $dalpha, c, alimit$ )
1  $i \leftarrow DEG2(dalpha - c)$ ;
2  $alimit \leftarrow 2^i$ ;
    Return:

```

Line 1 finds the binary degree i of $dalpha - c$. Line 2 establishes the alpha request demand as the maximum term of $dalpha - c$.

Reverse-route Route from r to the parent routing p .

Inputs $anode$ represents routing r .

Outputs L updated for ancestor of $anode$

Remarks Reverse routing is used only to traverse the alpha tree.
Reverse routing does not occur as part of a feasible routing sequence.

```

REVERSE-ROUTE( $anode$ )
1  $Rlist \leftarrow anode.Rt.Alist + anode.Rt.qa$ ;
2 From  $Rlist$ , calculate  $L$  array for  $anode.pnode$ ;
    Return:

```

Lines 1-2 restore the link loads to the p values.

Rload2[2] RLP2[2] algorithm

Inputs $anode$ represents routing r

Outputs $anode.Lminr$ minimum congestion for routing r

Remarks recursive subroutine

```

RLOAD2[2]( $anode$ )
1  $a.l.r.b \leftarrow anode.lnk[a.l.r.b]$ ;
2 if  $\lceil \frac{L[a] + L[b]}{2} \rceil < anode.Lmin$  and
    $\lceil \frac{L[l] + L[r]}{2} \rceil < anode.Lmin$ 
3   then RLOAD2-BPACK[2]( $anode.rootb$ ):
4       READ-RT-TERM( $rootb.anode.Rt.term$ ):
5       ADD-ALPHA-BRANCHES( $anode.rootb$ ):
6       deallocate  $rootb$  and its subtree:
7       for each descendant  $dnode$  of  $anode$ 
8           ROUTE( $dnode$ ):
9           RLOAD2[2]( $dnode$ ):
10          REVERSE-ROUTE( $dnode$ ):
11          PRUNE-NONFEASIBLE-BRANCHES( $anode.dnode$ ):
12 else From  $anode.pnode$ :
    Prune all subtrees for alpha routing sets not feasible
    according to Lemma 6.7.4.
Return:
```

Line 2 uses checks the minimum achievable congestion for r versus the currently attained level. If no improvement is possible. Lemma 6.7.4 is applied to remove other nonfeasible subtrees from the parent of $anode$ and the algorithm terminates. Otherwise. line 3 executes RLOAD2-BPACK[2] for r .

Line 4 reads the optimal terminal routing set in the beta tree. Line 5 uses the information in the beta tree to add an alpha tree branch for each feasible alpha routing set. Line 6 deallocates the beta tree now that it is no longer needed. For each descendant node. Lines 7-10 repeat RLOAD2[2] recursively. Line 11 prunes branches with non-minimum congestion.

Add-alpha-branches Add branches to alpha tree

Inputs *anode* represents routing r
 bnode represents beta routing rx

Outputs *anode* alpha tree with added branches

Remarks recursive subroutine

```
Add-alpha-branches(anode, bnode)
1 if bnode.minr FALSE
2   thenif bnode.min FALSE
3     READ-MINIMUM-ALPHAS(anode, bnode, mintab):
4     READ-ALTERNATE-ALPHAS(anode, bnode, alttab):
5     ADD-ALPHA-BRANCH-NODES(anode, bnode, mintab, alttab):
6     for each descendant dnode of bnode
7       ADD-ALPHA-BRANCHES(anode, dnode):
8   else READ-QMINB(anode, bnode, mintab):
9     ADD-ALPHA-BRANCH-NODES(anode, bnode, mintab, NIL):
Return:
```

Line 1 checks if the minimum r congestion has already been attained. If yes, no branch nodes are added and the subroutine terminates. Line 2 checks if a minimum packing of $E_{a,r}$ has occurred. If yes, lines 8-9 add a branch for the B-zone alpha routing set. Otherwise, the beta tree is traversed for all feasible alpha routing sets. Lines 3-4 read the minimum and alternate alpha routing sets on the beta subtree root node. Line 5 adds and initializes an alpha tree branch node for each required alpha routing set. Lines 6-7 repeat this process recursively for each descendant of *bnode*.

Read-minimum-phas Read minimum alpha requests

Inputs *anode* represents routing *r*
 bnode represents beta routing *rx*

Outputs *mintab* minimum alpha request table

```

READ-MINIMUM-ALPHAS(anode, bnode, mintab)
1 for lf ∈ anode.Zf
2   qa ← bnode.qbmin[lf];
3   if qa = NIL
4     then qa ← bnode.qfmin[lf];
5     if qa is not a borrow request
6       then qa ← NIL;
7   if qa ≠ NIL
8     then CHECK-ALPHA-VERSUS-LIMITS(anode, bnode, qa, lf, valid);
9     if valid = FALSE
10      then qa ← NIL;
11  mintab[lf].qa ← qa;
      mintab[lf].pi, mintab[lf].msi ← FALSE;
12 Eliminate one of mintab[l].qa or mintab[r].qa if both are terminal
      set 't' requests.
Return:

```

Line 1 repeats the search for each F-zone link. Lines 2-6 read the minimum alpha request for *f*. Lines 5-6 verify that an interfered request is a borrow request (in the *B_list* subtable of *X_list*). Line 8 checks that *qa* meets the Lemma 3.5.1 limits laid out in the *alimit* table. Line 11 places *qa* into the *mintab* table and clears the MSI and PI transfer flags. Line 12 ensures that only 1 terminal set 't' request is placed into *mintab*.

Read-alternate-alphas Read alternate alpha requests

Inputs *anode* represents routing r
 bnode represents beta routing rx

Outputs *alttab* alternate alpha request table

```

READ-ALTERNATE-ALPHAS(anode, bnode, alttab)
1 for  $lf \in anode.Z_f$ 
2    $qf \leftarrow bnode.qfmin[lf]; i \leftarrow 0;$ 
3   if  $qf \neq NIL$ 
4     then  $X' \leftarrow \{q \in bnode.X\_list: lf \in P^-(q) \text{ and } d(q) > d(qf)\};$ 
        Reduce  $X'$  to 1 request of each bandwidth denomination:
         $dmin \leftarrow d(qf);$ 
5     while  $X' \neq NIL$ 
6       Assign to  $qa$  the minimum demand request in  $X'$ ;
7       READ-MAXIMAL-BORROW-REQUEST(bnode,  $qa$ ,  $lf$ ,  $p$ ):
8       VALIDATE-MSI-ALPHA(anode, bnode,  $qa$ ,  $dmin$ ,  $lf$ ,  $msinvalid$ ):
9       VALIDATE-PI-ALPHA(anode, bnode,  $qa$ ,  $lf$ ,  $pivalid$ ):
10      if  $msinvalid = TRUE$  or  $pivalid = TRUE$ 
11        then CHECK-ALPHA-VERSUS-LIMITS(anode, bnode,  $qa$ ,  $lf$ ,  $valid$ ):
12        if  $valid = TRUE$ 
13          then  $alttab[i][lf].qa \leftarrow qa; dmin \leftarrow d(qa);$ 
               $alttab[i][lf].pi \leftarrow pivalid; alttab[i][lf].msi \leftarrow msinvalid;$ 
              increment  $i;$ 
14          else  $X' \leftarrow NIL;$ 
15      Remove from  $X'$  all  $q'$ :  $d(q') \leq d(qa);$ 
16   $alttab[lf].count \leftarrow i;$ 
  Return:

```

Line 1 repeats the search for each F-zone link. Line 2 reads the minimum demand rejected request on f . The alternate alpha request count is cleared to 0. Line 4 reads into X' the list of candidate alpha requests. Line 5 repeats lines 6-15 to check the feasibility of each candidate:

Line 6 reads the minimum demand candidate qa . Line 7 ensures that qa is feasible according to Lemma 3.6.4. If not qa is replaced by a feasible

candidate. Line 8 checks qa for MSI alternate alpha routing set feasibility. Line 9 checks qa for PI alternate alpha routing set feasibility. If qa qualifies, line 10 checks that qa meets the Lemma 3.5.1 limits in the *alimit* table. If valid, all necessary information is placed into the *alttab* table and the count is updated. If qa violates the Lemma 3.5.1 limit, all remaining candidates will. Therefore, line 14 clears X' to terminate the search. Line 15 updates X' for the next iteration. Line 16 places the count into *alttab* at the end of the search.

validate-pi-alpha validate PI alpha request

Inputs	<i>anode</i>	represents routing r
	<i>bnode</i>	represents beta routing rx
	qa	alpha request
	lf	qf interference link f

Outputs	<i>valid</i>	True for valid request
----------------	--------------	------------------------

```

VALIDATE-PI-ALPHA(anode, bnode,  $qa$ ,  $lf$ , pivalid)
1 pivalid  $\leftarrow$  FALSE
2 if anode.lnk[ $lf$ ] and anode.lnk[ $b$ ] are adjacent
3   thenif bnode.clist[ $\log_2 d(qa)$ ][ $lf$ ] $p$ ]  $\geq d(qa)$ 
4     then pivalid  $\leftarrow$  TRUE:
  Return:

```

Lines 2-3 check if qa meets the Lemma 5.6.1 requirements.

validate-msi-alpha validate MSI alpha request

Inputs

<i>anode</i>	represents routing r
<i>bnode</i>	represents beta routing rx
<i>qa</i>	alpha request
<i>lf</i>	qf interference link f
<i>dmin</i>	$d(qf)$ or demand of last candidate checked

Outputs *valid* True for valid request

```

VALIDATE-MSI-ALPHA(anode, bnode, qa, dmin, lf, msinvalid)
1 msinvalid  $\leftarrow$  FALSE;
2  $S' \leftarrow \{q \in \textit{bnode.S\_list} : \delta(q) = 1 \text{ and } dmin < d(q) < d(qa)\}$ ;
    $S'' \leftarrow \emptyset$ ;
3 if  $\textit{bnode.clist}[\log_2 d(qa)][lf] \geq \textit{bnode.clist}[\log_2 d(qa)][b]$ 
4   then  $S'' \leftarrow \{q \in \textit{bnode.S\_list} : lf'p \in P^-(q) \text{ and } dmin \leq d(q) < d(qa)\}$ ;
5       if  $S' + S'' \neq \text{NIL}$ 
6       then msinvalid  $\leftarrow$  TRUE;
Return:

```

This module checks the feasibility of an af MSI transfer initiated by an alternate alpha routing set with borrow request qa . Lines 2-6 list requests on S eliminated by the MSI transfer (Lemma 3.6.3). To be on the list, $d(q)$ must be greater than $d(qf)$ (Lemma 5.5.1). Also the alternate link must not be included on $P^-(q)$ (Lemma 3.6.1). Line 3 makes this check for $q: f' \in P^-(q)$. Line 4 is executed only if the alternate link is not f' . Requests listed for previous candidates are not listed for qa . The *dmin* parameter excludes these requests.

check-alpha-versus-limits check alpha request against
bandwidth limits

Inputs *anode* represents routing r
 bnode represents beta routing rx
 qa alpha request
 lf qf interference link f

Outputs *valid* True for valid request

```

CHECK-ALPHA-VERSUS-LIMITS(anode.bnode.qa.lf.valid)
1 valid  $\leftarrow$  TRUE;
2 alimit  $\leftarrow$  anode.alimit.ind;
3 if anode.plnka  $\in P^-(qa)$ 
4   then alimit  $\leftarrow$  anode.alimit.com
5 if  $d(qa) > alimit$ 
6   then valid  $\leftarrow$  FALSE;
7 CHECK-ALPHA-VERSUS-PI-LIMIT(anode.bnode.qa.lf.valid):
8 if bnode.tlimit[lf] = TRUE
9   thenif  $d(qa) \geq bnode.clst[\log_2 d(qa)][lf]$ 
10    then valid  $\leftarrow$  FALSE;
Return:

```

Lines 1-6 check if $d(qa)$ exceeds the limit imposed by *alimit*. If $P^-(qa)$ includes the maximum load link of the ancestor node, the common limit applies. Otherwise, the independent limit is used. Line 7 checks *qa* against the PI limit. Lines 8-10 check *qa* against the beta transfer limit.

```

CHECK-ALPHA-VERSUS-PI-LIMIT(anode.bnode.qa.lf.valid)
1 if valid = FALSE
2   thenif lf  $\in P^-(anode.Rt.qa)$ 
3     then Let  $\mathcal{A}$  be the alpha routing set for borrow request qa
4       if There exists  $q \in \mathcal{A}$ : lf  $\in P^-(q)$ 
5         then valid  $\leftarrow$  TRUE;
Return:

```

This module checks for the possibility that the alpha routing set $A_{p \rightarrow r}$, routed by $anode.pnode$, is a PI alternate alpha routing set. In that case, if qa forms a PI alpha routing set which routes the transfer requests for this PI transfer, the lemma 3.5.1 bounds are relaxed on $d(qa)$.

Line 1 checks the current status. Because the PI limit ($d(anode.Rt.qa)$) exceeds the $alimit$ (lemma 3.5.1) bound, this check applies only when qa already has an invalid status. Line 2 checks if f qualifies as the PI interference link for $A_{p \rightarrow r}$.

If yes, lines 3-5 check if qa forms a PI alpha routing set A with a minimum of 1 transfer request for $A_{p \rightarrow r}$. If yes, qa is reassigned valid status. A routes a maximum of $d(anode.Rt.qa)$ bandwidth so that no further checks are needed.

Read-qminb read minimum B-zone alpha request

Inputs $anode$ represents routing r
 $bnode$ represents beta routing rx

Outputs $mintab$ minimum alpha request table

Remarks This module searches the beta tree for and reads a minimum demand rejected request that is a non-interfered B-zone borrow request.

READ-QMINB($anode.bnode.mintab$)

- 1 From $bnode$ find a Beta Tree terminal node $mnode$
 such $mnode.mrn$ is TRUE:
 - 2 Place $mnode.qminb$ into $mintab$ in the appropriate format:
- Return:

Line 1 searches the beta tree for a node that represents an optimal packing of $E_{a,r}$. Line 2 places $mnode.qminb$ into the minimum alpha request table.

Add-alpha-branch-nodes Add alpha tree branch nodes

Inputs *anode* ancestor of branch node.
 bnode represents beta routing rx
 mintab minimum alpha request table
 alttab alternate alpha request table

Outputs *anode* alpha tree with added branch nodes.

```

ADD-ALPHA-BRANCH-NODES(anode, bnode, mintab, alttab)
1 for  $lf \in anode.Zf$ 
2   if mintab[lf].qa exists
3     then ADD-ALPHA-NODE(anode, bnode, mintab[lf]);
4   for  $i \leftarrow 0; i < alttab[lf].count; i \leftarrow i + 1$ 
5     ADD-ALPHA-NODE(anode, bnode, alttab[i][lf]);
  Return;
```

Line 1 repeats for each F-zone link f . Line 2 adds an alpha tree node for the minimum alpha request on f . Line 5 adds an alpha tree node for each alternate alpha request on f .

```

ADD-ALPHA-NODE(anode, bnode, atab)
1 ALLOCATE(dnode):
2 dnode.Rt.qa  $\leftarrow atab.qa$ ;
3 Alist  $\leftarrow \{q : q \in bnode.S\_list \text{ and } d(q) \geq d(atab.qa)\} \div bnode.A\_list$ ;
4 dnode.Rt.Alist  $\leftarrow A\_list$ ;
5 dnode.Rt.term  $\leftarrow NIL$ ;
6 dnode.plnka  $\leftarrow anode.lnk[a]$ ;
7 dnode.Lmin  $\leftarrow anode.Lmin$ ;
8 dnode.Lminr  $\leftarrow \infty$ ;
9 In anode add pointer to dnode and increment number of descendants;
10 dnode.pnode  $\leftarrow anode$ ;
11 From atab.qa, atab.pi, determine dnode.pilnkf;
  Return;
```

This module allocates and initializes an alpha tree node based upon the information in the alpha request table.

Read-Rt-term Read terminal routing

Inputs *rootb* beta tree root node

Outputs *T_list* terminal routing request set

READ-RT-TERM(*rootb*, *T_list*)

- 1 From *rootb* traverse the Beta tree for *tnode* such that *tnode* has the minimum value *bnode.Rt.c[b]* or *bnode.alterm.cb* in the tree.
- 2 Assign *T_list* to the minimum *tnode* terminal routing.

Return:

Line 1 finds the beta tree node with the terminal routing or alternate terminal routing which represents the best packing of $E_{a,r}$. Line 2 places this terminal routing into *T_list*.

Prune-nonfeasible-branches Prune nonfeasible branches

Inputs *anode* represents routing *r*
 dnode descendant of *anode*: branch root node

Outputs *anode* alpha tree with nonfeasible branch pruned

PRUNE-NONFEASIBLE-BRANCHES(*anode*, *dnode*)

- 1 if *dnode.Lminr* < *anode.Lmin*
- 2 **then** *anode.Lmin*, *anode.Lminr* ← *dnode.Lminr*:
- 3 Find descendant *xnode* of *anode*: *xnode.Lminr* > *dnode.Lminr* and *xnode.Lminr* < ∞:
- 4 Delete *xnode* and all its subtree:
- 5 **else** Delete *dnode* and all its subtree:

Return:

Line 1 checks if the *dnode* subtree improves the current congestion. If yes, the minimum congestion and $L_{min,r}$ are updated and any other subtree is deleted. Otherwise the *dnode* subtree is deleted.

RLOAD2[2] is a heuristic algorithm. A formal proof of correctness is not provided for RLOAD2[2]. However, these points indicate the efficacy of RLOAD2[2] and the concepts discussed by this thesis:

- (i) RLOAD2-BPACK[2] finds an optimal RLP2[2]-BPP bin packing for routing r (Lemma 6.9.1). In addition, RLOAD2-BPACK[2] finds at least one feasible alpha routing set to a subsequent routing t (Lemma 6.9.1).
- (ii) For $A_{r \rightarrow t}$ which relocates the maximum load link from a to y , the reduction set A_t represents an exact packing of $\lfloor \frac{C_{ay}}{d(\alpha_t)} \rfloor$ (Lemma 3.3.1). When $A_{r \rightarrow t}$ is routed, it is feasible to route the selected blocks of bandwidth, common to links a and y , of demand greater than or equal to $d(\alpha_t)$ (Lemma 3.3.2).
- (iii) Lemma 2.5.1 shows that routing two equal demand routing sets can reduce the congestion only on their path overlap zone. This, in addition to rationale presented by the above item, indicates the feasibility of minimum alpha routing sets. Therefore, all feasible minimum alpha routing sets are routed. However, the routing of alternate alpha routing sets may be necessary for transfers to eliminate interference on interfered requests. The concepts discussed by this thesis attempt to minimize number of necessary alpha routing sets.
- (iv) If PI common request sets are added to RLOAD2[2], the algorithm may approach more closely an optimal algorithm. However, this approach may not be justified for reasons already discussed.

6.11 RLP2[2] Computational Complexity

This section provides an approximation of the RLOAD2[2] computational complexity. A formal proof is not provided.

The key to the approximation is to show that the alpha tree has a maximum of $c2^w$ nodes where 2^w is the maximum request demand and c an integer constant. For the discussion to follow, let $\alpha node$ be an alpha tree node which represents routing r . Assume the Lemma 6.9.1 beta tree notation. Figure 6.2 illustrates the beta tree.

First, consider the case where the beta tree has more than two nodes. There are no interfered requests on beta routings $r2$ and $r3$ (Lemma 6.9.1). Therefore, the minimum demand rejected request on $r2$ or $r3$ is always an non-interfered B-zone borrow $qb1$. If:

$qb1$ exists Then $qb1$ represents an optimal packing of $C_{b,r}$ (Lemma 6.7.2 Corollary 6.7.2.1 or Lemma 6.7.3 Corollary 6.7.3.1). If $qb1$ indicates the attainment of $r_{min,r}$ according to Lemma 3.4.1 Corollary 3.4.1.1 then no alpha routing sets are routed for r . Else $qb1$, being non-interfered, represents the only feasible alpha routing set for r .

$qb1$ does not exist Then the rejected set is null and no alpha routing sets exist for $r2$ and $r3$.

Therefore, to calculate the number of alpha nodes, only beta routings r and $r1$ need to be considered. A request qf is interfered on one of two links $f \in Z_F$. For qf and interference link f , the alpha request for a transfer must belong to terminal set T_f . First, consider the case where no PI alternate alpha routing sets are required. Then consider the case in which a PI alternate alpha routing set is required. Let 2^x , $1 \leq x \leq w$ be the maximum request demand in $E_{a,r}$.

For the first case, only MSI alternate alpha routing sets are feasible. For interference link f on partition $\mathcal{P}$, Lemma 5.5.1 (ii) ensures that there exists at most one feasible alpha routing set for each alternate bandwidth denomination. Routing a $d(2^0)$ alpha routing request cannot reduce the congestion. Therefore, a maximum of $\left\lceil \frac{x}{2} \right\rceil$ alpha routing sets may occur for each of f and f' . For purposes of calculation it is reasonable to assume a maximum of 1 alpha routing set for each bandwidth denomination 2^y , $1 \leq y \leq x$. With further analysis, there is the possibility that this upper bound is provable. If $\mathcal{P}_{r1}$ exists, it is verifiable from the beta tree that this limit is preserved for the total number of alpha routing sets for the two nodes. Therefore, *anode* has one possible alpha tree node *dnode* for each value of y . The structure repeats itself recursively for each descendant node of *anode*. It is then easily proven, inductively, that the routing r alpha tree has at most 2^x nodes. For maximum bandwidth denomination w , the maximum value is 2^w .

Next, consider PI alternate alpha sets. For qf , interfered on f , consider the case where 2 or more PI alternate alpha sets A_{r-t} and $A_{r-t'}$: $\alpha_{t'} > \alpha_t$ are

feasible. Using Lemmas 6.4.1, 3.6.4 and 6.7.2, it is possible to prove that no interfered request exists on f' . In this case, Lemma 5.6.1 allows a maximum of $x - 1$ PI feasible alternate alpha sets. These alternate alpha sets include all feasible MSI alternate alpha candidates. If $\mathcal{P}_{r,1}$ exists, it is probably verifiable that the x limit holds on the total number of alpha routing sets. Even if not, the total number of nodes added to the alpha tree is anticipated to be less than 2^w . For the case where only a single feasible PI alternate alpha set exists, the PI alpha routing set can be counted with the x feasible alpha routing sets. Therefore, a $c2^w$ overall upper bound holds for both cases.

After PI alternate alpha routing set $A_{r \rightarrow t}$ routes to t , it is provable using a combination of Lemma 6.4.1, Lemma 2.5.1 Corollary 2.5.1.1 and Lemma 6.7.2 that the routing of $A_{r \rightarrow t}$ eliminates subsequent interference on f for $\ell_{1,t} = b$. Therefore a maximum of $x - 1$ alpha routing sets $A_{t \rightarrow r'} = A_{r'} + \alpha_{r'}$: $d(\alpha_{r'}) < d(\alpha_t)$ occur. As the Lemma 3.5.1 bounds do not apply to PI alternate alpha routing sets, an additional $A_{t \rightarrow r'}$: $d(\alpha_{r'}) = d(\alpha_t)$ is allowed. The maximum number of alpha routing sets is x to preserve the $c2^w$ overall limit.

An extensive analysis of the search for a case where common request set $A_{\alpha,t}$ replaces α_t in $A_{r \rightarrow t}$ is not performed here. The search for $A_{\alpha,t}$ and $qi \in E_{ba,r}$: $d(qi) = d(A_{\alpha,t})$ and $f \in P^-(qi)$ involves a search of $E_{ab,r}$ and $E_{ba,r}$ which probably is of $O(m)$ computational complexity. The overall number of PI alpha routing sets added to the alpha tree is estimated to be within and likely much less than 2^w . Therefore, the $c2^w$ overall limit holds.

Therefore, the overall computation complexity consists of:

- (i) $c2^w$ alpha routing sets. From the above analysis c is probably a small integer less than or equal to 5. Therefore, there are $O(2^w)$ alpha routing sets. Therefore, the alpha tree has $O(2^w)$ nodes.
- (ii) The RLOAD2-BPACK[2] computational complexity is $O(m + n)$. This algorithm is executed once for each alpha tree node. For the alpha tree, the total complexity is $O(2^w(m + n))$.
- (iii) Finally, there is the $O(m \log_2 m)$ sorting complexity during RLOAD2[2] initialization. This brings the RLOAD2[2] overall computational complexity to $O(2^w(m + n) + m \log_2 m)$.

With a more complicated proof, it may be possible to limit the alpha tree to 2^w nodes. However, the RLOAD2[2] algorithm is expected to find a solution to RLP2[2] with significantly less than 2^w alpha tree nodes.

6.12 Conclusion

The RLOAD2[2] algorithm is an $O(2^w(m + n) + m \log_2 m)$ approximate solution to RLP2[2]. The concepts presented in this chapter and in this thesis allow this algorithm to be fast by minimizing the number of required alpha and beta tree nodes. These concepts suggest other heuristics to speed up RLP2[2] solutions by further limiting the alpha routing sets and therefore, the number of alpha tree nodes.

RLOAD2[2] uses RLOAD2-BPACK[2] to obtain an optimal packing of $E_{a,r}$ within the RLP2[2] bin constraints. RLOAD2-BPACK[2] calls RLOAD2-PART[2] to perform the actual bin packing process by partitioning the eligible subset E of $E_{a,r}$ into the selection and rejection request sets. RLOAD2-BPACK[2] requires up to four partitions of $E_{a,r}$.

RLOAD2-PART[2] is a single pass algorithm that partitions a sorted eligible request set E in $O(m)$ time. This is made possible by the fact that all request demands are powers of 2, which allows the optimal packing of a bin by a greedy process in one pass.

Chapter 7

Conclusion and Open Problems

This thesis studies the RLP2 problem. RLP2 is an RLP problem that restricts request demands to integer powers of 2. The sequence $2^0, 2^1, \dots, 2^x$ is a superincreasing sequence such that $2^x = 2^{x-1} + 2^{x-2} + \dots + 2^0$ where $y \geq 0$. Therefore, a greedy packing sequence gives an optimal packing of a 2^x bin from a set of items of value less than or equal to 2^x . For a 2^x demand block, an equivalent demand block is easily identified. This facilitates identification of:

- (i) Equal demand path independent request sets not feasible for routing (Lemma 2.5.1 Corollary 2.5.1.1).
- (ii) A replacement block for the beta request on beta transfers.
- (iii) Alternate link reduction sets for alternate routing set transfers.
- (iv) Upper bounds on bandwidth routed after an alternate alpha routing set (Lemma 3.5.1).
- (v) Upper bounds on bandwidth routed after a transfer (Lemmas 6.4.2, 6.5.1 and 6.6.2).

In addition, this packing scheme allows RLOAD2-PART[K] to partition an eligible request set in a single pass. The disadvantage of the 2^x demand restriction is that some bandwidth may be wasted. This bandwidth under utilization may be restricted if it is possible to specify a limited number of bandwidth requests as two or more independent requests. Therefore, the user may achieve a degree of demand splitting to relax the integrality constraint.

The algorithms presented are not intended for practical application. Rather, they are intended as to suggest RLP2[k] solutions using concepts developed by this thesis. They may be upgraded to gather experimental data relating to RLP2[k] problems. As demonstrated by the relative complexity of the RLP2[2] solution, the development of an RLP2 algorithm that provides an optimal solution all the time is probably not feasible from either a practical or a computational standpoint. Rather, the requirement stated by Schrijver for a computationally fast algorithm that achieves a near optimum solution most of the time is more realistic. The computational complexity of RLP2[k] approximations may be improved by:

- (i) Limiting PI transfers. It has been shown that the Lemma 3.5.1 bounds on routed bandwidth apply to alpha routing sets except when a PI transfer is feasible. Therefore, heuristics that quickly limit the PI transfers required are likely to be useful. One possible solution is to disregard or limit the implementation the usage of common request sets as presented by Chapter 5 Section 5.6. Lemmas 5.6.2 and 5.6.3 suggest ways to limit PI transfers.
- (ii) Limiting the number of alternate alpha routing sets. As the demand of an alternate alpha request increases in proportion to the demand of a minimum alpha request, the probability of an alternate alpha routing set being feasible most likely decreases significantly. For RLP2[2], a solution is a limit on the number of borrow requests routed to either F-zone link and the B-zone link. For example, the minimum alpha request set and a maximum of one additional alternate alpha routing set may be allowed.
- (iii) Limiting beta transfers. For RLP2[2], the number of beta transfers is not perceived to be a significant problem (Lemma 6.9.1). For more general RLP2[k], the computational complexity of finding the necessary beta transfers will increase. Therefore, additional restrictions on beta transfers should be researched. It should be useful to investigate the revision of Lemma 6.6.1 for the RLP2[k] application.

SONET ring designs are selected to provide the bandwidth required to meet a set of optical demands. The large number of calls to RLP subroutines to evaluate proposed ring designs during ring planning remains computationally intensive. Khanna[15] states that there are multiple stages to SONET

ring planning. One stage consists determining good routings for requests on a large number of rings. The second finds better routings for the best candidates from the first stage. The heuristics suggested above may be parametrically adjusted to the accuracy requirements of each planning stage.

SONET rings are configured to route a relatively fixed set of optical demands over a relatively long time period. This suggests a static routing which may be inflexible in the long term. Traffic demands are likely to increase significantly. New technology and reduced equipment costs may increase available capacity. Network fault protection and survivability will remain critical. Therefore, some degree of dynamic routing of demands may be necessary to facilitate these developments. The concepts elaborated by this thesis may be used to provide computationally fast algorithms to dynamically reroute demands. The rapid greedy packing of 2^w demand blocks is one asset. The single pass RLOAD2-PART[K] algorithm is a second asset. The ability to restrict the routing of not practically feasible alpha routing sets and transfers should also reduce the computational requirement.

The protection bandwidth provided by SONET SHRs represents potential capacity under utilization. Dynamic routing schemes may permit some usage of this redundant capacity in the absence of faults. This added capacity may be used to alleviate heavily loaded links on the operational facility. The tradeoff may be increased blocking of connections when a fault occurs.

A possible open problem is computer aided design of SONET networks. Artificial intelligence and genetic algorithms may be used to propose a network design for a given set of traffic demands. The design software to evaluate proposed designs would use RLP2 subroutines.

For each algorithm, separate computational times were calculated to exclude and include request sorting. For the SONET ring case, where the same request set is used for all runs, the requests may be presorted. Therefore, the non sorting times stand. For RLOAD2[1], this value is $O(m)$. For RLOAD2[2], the non sorting complexity is approximated at $O(2^w(m + n))$. Sorting times add $O(m \log_2 m)$ to each of these values.

The $O(2^w(m + n))$ RLOAD2[2] value is exponential in w . Also a RLP2[k] solution will have higher order n and m values. One reason is the expanded the F-zone. However, exponential or high order polynomial theoretical complexities may be disregarded if accurate and fast algorithms are found for the practical application.

Appendix A

Notation

$\mathcal{R}$ denotes an optical ring. $\mathcal{R}$ is also the set of all optical nodes or optical links on the ring.

n the number of nodes on $\mathcal{R}$.

ℓ the optical link $0 \leq \ell \leq n-1$ joining adjacent nodes ℓ and $\ell+1 \bmod n$ on $\mathcal{R}$.

q a connection request $1 \leq q \leq m$ on $\mathcal{R}$.

$\mathcal{Q}$ the set of all optical requests on $\mathcal{R}$.

m the number of connection requests on $\mathcal{R}$. $m = |\mathcal{Q}|$.

$d(q)$ the bandwidth requirement or demand for request q .

$d(S)$ the total bandwidth requirement or demand for set S of requests.

d_{max} the maximum allowable demand for a RLP2 request. $d_{max} = 2^w$.

w $d_{max} = 2^w$.

δ_q the length of the shortest path between i_q and j_q .

k the upper bound on δ_q . $q \in \mathcal{Q}$.

I_ℓ the link path set for link ℓ . For routing r , the explicit form is $I_{\ell,r}$.

E an eligible request set:

E_ℓ the eligible request set on link ℓ .

E_{xy} the interdependent eligible request set for links x and y .

$\bar{E}_{xy}$ the independent eligible request set for link x relative to link y .

$E_{\mathcal{R}}$ eligible request set on $\mathcal{R}$.

Explicit forms for routing r : $E_{\ell,r}$, $E_{xy,r}$, $E_{\mathcal{R},r}$, etc.

ϕ the routing vector for Q on $\mathcal{R}$.

r a value r of ϕ which defines a routing of Q . Also:

r_{init} the initial routing of Q .

r_{opt} an optimal routing of Q .

$r_{min,r}$ a minimum routing of Q for initial routing r .

$r_{term,r}$ a terminal routing of Q for initial routing r .

$R_{r \rightarrow s}$ a routing set of requests which maps routing r to routing s . Particular routing sets are:

$R_{r \rightarrow min}$ a routing set mapping routing r to routing $r_{min,r}$.

$R_{r \rightarrow term}$ a terminal routing set for routing r .

$A_{r \rightarrow s}$ alpha routing set which maps routing r to routing s .

$B_{rx \rightarrow ry}$ beta routing set which maps routing rx to routing ry .

L load or congestion

L_ℓ the link load on link ℓ .

$\bar{L}_{xy}$ the independent congestion on link pair $\{x, y\}$.

L_{xy} the common congestion on link pair $\{x, y\}$.

$L_{min,xy}$ the minimum congestion attainable on $\{x, y\}$.

$L_{\mathcal{R}}$ the ringload or congestion on $\mathcal{R}$.

L_{min} the minimum congestion attainable on $\mathcal{R}$.

L_{opt} the optimal congestion on $\mathcal{R}$.

Explicit forms for routing r : $L_{\ell,r}$, $L_{xy,r}$, $L_{\mathcal{R},r}$, etc. $L_{opt} = L_{min,init}$.

Lq aggregate loads

$\hat{L}_{E,R,\ell}$ the aggregate link load for eligible request set E on link ℓ for $R \subseteq E$ routed to its alternate path.

$\hat{L}_{E,R}$ the array of aggregate link loads for eligible request set E on link ℓ for $R \subseteq E$ routed to its alternate path.

R may be replaced by request q in the above definitions.

C_{xy} the capacity of x with respect to y . $C_{xy} = \left\lfloor \frac{L_x - L_y}{2} \right\rfloor$.

Explicit forms for routing r : $C_{xy,r}$, $C_{ab,r}$, etc.

c_{xy} the RLOAD2-PART[K] execution capacity of x with respect to y .
The C_{xy} variants may apply to c_{xy} .

$cr(qx)$ The array of reduction capacities for rejected request qx . $cr(qx, y)$ is the reduction capacity on bin y for qx .

$crb(qb)$ The array of beta reduction capacities for beta request qb .

ct The array of final bin capacities for RLOAD2-PART[K]. $ct(y)$ is the final RLOAD2-PART[K] capacity on bin y .

$P(q)$ the optical path on $\mathcal{R}$ currently assigned to request q .

$P(q) = P^-(q)$ the set of optical links comprising the shortest path between i_q and j_q .

$P(q) = P^+(q)$ the set of links comprising the alternate path between i_q and j_q .

$P(S)$ the aggregate path of a set S of requests.

$P_{S_1 \cap S_2}$ the overlap zone of the aggregate paths of requests sets S_1 and S_2 .
Either of these sets may be replaced by a single request q .

Z_z routing zone z of $\mathcal{R}$. The routing zones are:

Z_A the maximum load link zone. $Z_A = \ell_1$, the maximum load link on $\mathcal{R}$.

Z_B the independent zone of ℓ_1 . $\ell_2 \in Z_B$ is the independent zone maximum load link on $\mathcal{R}$.

Z_F the interdependent zone of ℓ_1 . Z_F is partitioned into:

Z_L the left interdependent zone of ℓ_1 . $\ell_\ell \in Z_L$ is the left interdependent zone link for RLP2[2].

Z_R the right interdependent zone of ℓ_1 . $\ell_r \in Z_R$ is the right interdependent zone link for RLP2[2].

For routing r the routing zone is $Z_{z,r}$.

$\mathcal{P}$ an RLOAD2-PART[K] partition of eligible request set E_a . $\mathcal{P}_r$ is a partition

S the RLOAD2-PART[K] selection set. S_{ay} denotes the requests selected from $\bar{E}_{ay}$ to pack bin y .

X the RLOAD2-PART[K] rejection set. X is divided into two subsets:

B the RLOAD2-PART[K] borrow set. B_{ay} is the set of bin y borrows.

D the RLOAD2-PART[K] non-maximal shortest path interval set. D_{ay} is the set of bin y non-maximal shortest path interval requests.

$\tilde{S}(qx)$ the RLOAD2-PART[K] reduction set for request qx .

$\tilde{S}(qx, y)$ the RLOAD2-PART[K] independent reduction set for request qx on link y .

$\tilde{B}(qb)$ the RLOAD2-PART[K] beta reduction set for beta request qb .

$\tilde{T}$ the RLOAD2-PART[K] terminal reduction set

$\tilde{T}(y)$ the set of all selected requests in S_{ay} .

Index

- RLP2[k] Bin Packing Problem. 12
- A-zone. 13
- Add Drop Multiplexors. 2
- aggregate link load. 21
- aggregate load. 21
- aggregate path. 27
- alpha routing. 86
- alpha routing set. 37
- alpha tree. 15
- alternate alpha routing set. 51
- alternate link reduction set. 47
- alternate path. 9
- alternate routing set. 46
- alternate routing set transfer. 72
- alternate terminal routing. 90
- beta reduction capacity. 93
- beta reduction set. 93
- beta request. 84
- beta routing. 86
- beta routing set. 93
- beta transfer. 83
- beta tree. 15
- bin identifier. 35
- borrow request. 35
- borrow set. 35
- capacity. 23
- capacity transfer. 16
- chromatic number. 5
- common request. 72
- common request set. 77
- common subset. 22
- congestion. 2
- current bin capacity. 35
- Digital Cross Connect Systems. 2
- Directed Ring Loading problem. 6
- Edge-Disjoint Paths problem. 6
- eligible request set. 20
- eligible request set on $\mathcal{R}$. 20
- eligible request set on link ℓ . 20
- eligible set. 20
- feasible routing. 20
- feasible routing set. 20
- independent link pair. 23
- independent reduction set. 36
- independent subset. 22
- independent terminal reduction set. 36
- independent zone. 13
- independent zone capacity. 26
- independent zone maximum load link. 19
- initial routing. 12
- integer linear program. 5
- Integer Multicommodity Flow problem. maximum demand = 1. 6

- integrality constraint. 10
- interdependent subset. 22
- interdependent zone. 13
- interfered request. 14. 70. 71
- interference link. 14
- intermediate routing. 15

- left interdependent subzone. 69
- left interference link. 70
- left terminal set. 88
- Linear Multicommodity Flow problem. 7
- linear programming. 5
- link load. 9
- link pair independence. 23
- link path set. 19

- maximal shortest path interval. 28
- maximal shortest path interval (MSI) feasibility requirement. 50
- maximal shortest path interval alpha routing set. 39
- maximal shortest path interval constraint. 30
- maximum load link. 19
- maximum load link eligible request set. 20
- maximum load link zone. 13
- minimal routing set. 21
- minimum alpha routing set. 39
- Minimum Path Colouring problem. 1
- minimum routing. 20
- minimum routing set. 20
- MSI transfer. 73

- non-maximal shortest path interval set. 35
- non-terminal routing. 15

- non-terminal routing set. 15

- optimal routing of Q . 10
- optional T_i selection. 89

- partial ordering of aggregate loads. 21
- path independence of requests. 26
- path interdependent (PI) feasibility requirement. 50
- path set. 19
- permutation. 5
- PI transfer. 75
- Power 2 Demand k Shortest Path Ring Loading Problem . 4
- Power 2 Demand Ring Loading Problem. 10

- reduction capacity. 36
- reduction set. 36
- rejection set. 34
- repartition. 72
- request set path independence. 27
- right interdependent subzone. 69
- right terminal set. 88
- ring load. 9
- Ring Loading problem. 3. 6
- routing and wavelength assignment RWA. 1
- routing of Q . 9
- routing sequence. 16
- routing set. 20
- routing zones. 12

- selection set. 34
- shortest path. 9
- SONET self-healing ring. 2
- super increasing sequence. 34
- Synchronous Digital Hierarchy. 2

Synchronous Optical NETwork. 2

t1 terminal set, 88

terminal reduction capacity, 36

terminal reduction set, 36

terminal routing, 15

terminal routing set, 15

terminal set, 87

transfer, 16

transfer gain, 72

transfer request, 72

wave division multiplexing WDM.

1

wavelength continuity, 1

Bibliography

- [1] Y. Aumann and Y. Rabani. Improved bounds for all-optical routing. *SODA95*, pp. 567-576.
- [2] D. Banerjee and B. Mukherjee. A practical approach to routing and wavelength assignment in large wavelength routed optical networks. *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 5, pp. 903-908, June, 1996.
- [3] T. Cormen, C. Lierson and R. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 1993.
- [4] S. Cosares, I. Saniee and O. Wasem. Network planning with the SONET Toolkit. *Bellcore EXCHANGE*, Sept./Oct. 1992, pp. 8-13.
- [5] S. Cosares and I. Saniee. An optimization problem related to balancing loads on SONET rings. *Telecommunications Systems*, vol. 3, pp. 165-181, 1994.
- [6] T. Erlebach and K. Jansen. Scheduling of virtual connections in fast networks. *Proceedings of the 4th Parallel Systems and Algorithms Workshop*, pp. 13-32, 1996.
- [7] A. Frank. Edge-disjoint paths in planar graphs. *Journal of Combinatoric Theory, Series B*, vol. 38, pp. 164-178, 1985.
- [8] A. Frank, T. Nishizeki, N. Saito, H. Suzuki and E. Tardos. Algorithms for routing around a rectangle. *Discrete Applied Math*, vol. 40, pp. 363-378, 1992.
- [9] A. Frank. Connectivity and network flows. *Handbook of Combinatorics*. Elsevier Science, pp. 111-177, Amsterdam, 1995.

- [10] M.R. Garey and D.S. Johnson. *Computers and Intractability. A Guide to the Theory of NP -Completeness*. W. H. Freeman. 1979.
- [11] M. R. Garey, D. S. Johnson, G. L. Miller and C .H Papadimitriou. The complexity of colouring circular arcs and chords. *SIAM Journal of Algorithmic Discrete Methods*, vol. 1, part 2, pp. 216-227. 1980.
- [12] O. Gerstel. Deterministic models and algorithms for wavelength routing optical networks. Optical Networking Group, Tellabs. 1997.
- [13] M. Gondran and M. Minoux. *Graphs and Algorithms*. New York: Wiley, 1986.
- [14] D.S. Johnson. The NP-completeness Column: An ongoing guide. *Journal Algorithms*, vol. 3, pp. 381-395. 1982.
- [15] S. Khanna. A polynomial-time approximation for the SONET ring load-ing problem. *Bell Laboratories Technical Journal*, vol. 2, pp 36-41. 1997.
- [16] F.T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approx-imation algorithms. *FOCS88*, pp. 422-431. 1988.
- [17] M. Mihail, C. Kaklamanis and S. Rao. Efficient access to optical band-width. *FOCS95*, pp. 548-555.
- [18] H. Okamura and P. Seymour. Multicommodity flows in planar graphs. *Journal Combinatoric Theory, Series B*, vol. 31, pp.75-81. 1981.
- [19] Y.Rabani. Path colouring on the mesh. *FOCS96*, pp. 400-409.
- [20] P. Raghavan and C.D. Thompsom. Randomized rounding: A technique for probably good algorithms and algorithmic proofs. *Combinatorica*, vol. 7, no. 4, pp. 365-374. 1987.
- [21] R. Ramaswami and K. N. Sirarajan. Routing and wavelength assignment in all-optical networks. *IEEE/ACM Transactions on Networking*, vol. 3, no. 5, pp. 489-500. Oct., 1995.
- [22] T. Stern and K.Bala. *Multiwavelength Optical Networks: A Layered Ap-proach*. Addison Wesley, 1999.

- [23] H. Suzuki, A. Ishiguro and T. Nishizeki. Variable priority queue and doughnut routing. *Journal of Algorithms*, vol. 13, pp. 606-635, 1992.
- [24] A. Schrijver and P. Seymour and P. Winkler. The ring loading problem. *SIAM Review*, vol. 41, pp. 777-791, December, 1999.
- [25] R. Vachani, A. Shulman, P. Kubat and J. Ward. Multicommodity flows in ring networks. *INFORMS Journal on Computing*, pp. 235-242, 1996.
- [26] G. Wilfong and P. Winkler. Ring routing and wavelength translation. *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 333-341, 1997.