

## **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.**

**Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.**

**Bell & Howell Information and Learning  
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA  
800-521-0600**

**UMI<sup>®</sup>**



**A Study on Distributed Structures**

by

**Costas Busch**

**B.Sc., University of Crete, Greece, 1992**

**M.Sc., University of Crete, Greece, 1995**

**M.Sc., Brown University, 1997**

**A dissertation submitted in partial fulfillment of the  
requirements for the Degree of Doctor of Philosophy  
in the Department of Computer Science at Brown University**

**Providence, Rhode Island**

**May 2000**

UMI Number: 9987735

UMI<sup>®</sup>

---

UMI Microform 9987735

Copyright 2000 by Bell & Howell Information and Learning Company.

All rights reserved. This microform edition is protected against  
unauthorized copying under Title 17, United States Code.


---

Bell & Howell Information and Learning Company  
300 North Zeeb Road  
P.O. Box 1346  
Ann Arbor, MI 48106-1346

© Copyright 1998,1999,2000 by Costas Busch


This dissertation by Costas Busch is accepted in its present form by the Department of Computer Science as satisfying the dissertation requirement for the degree of Doctor of Philosophy.

Date 3 MAY 00

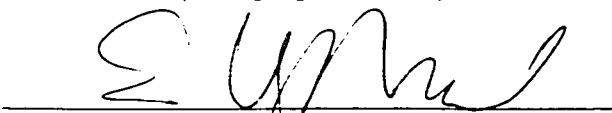
  
Maurice Herlihy, Director

Recommended to the Graduate Council

Date 5/3/2000

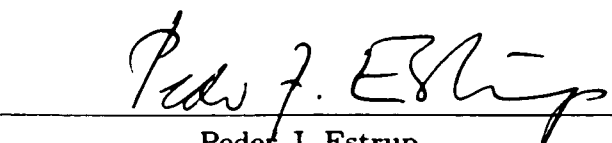
  
Mark Tuttle, Reader  
(Compaq Research)

Date 5/3/2000

  
Eli Upfal, Reader

Approved by the Graduate Council

Date 5/10/00

  
Peder J. Estrup  
Dean of the Graduate School and Research

# Vita

Costas Busch was born on November 2, 1969, in Annedal, Sweden. He spent the first 5 years of his life in Sweden and Germany, and then he moved to Greece. In 1987 he was admitted in the Computer Science Department of University of Crete, Greece, and graduated in 1992. In 1992 he entered the Master's program in the same department and graduated in 1995. In 1995 he entered the Ph.D. program in the Computer Science Department of Brown University.

# References

## Refereed Journal Articles:

- W. Aiello, C. Busch, M. Herlihy, M. Mavronicolas, N. Shavit and D. Touitou, "Supporting Increment and Decrement Operations in Balancing Networks," to appear in the *Chicago Journal of Theoretical Computer Science*. (A preliminary version appears in STACS'99.)

## Refereed Conference Articles:

- C. Busch, N. Demetriou, M. Herlihy and M. Mavronicolas, "A Combinatorial Characterization of Properties Preserved by Antitokens," to appear in the *European Conference on Parallel Computing (Euro-Par 2000)*, Munich, Germany, August/September 2000.
- C. Busch, M. Herlihy and R. Wattenhofer, "Hard-Potato Routing," to appear in the *32nd Annual ACM Symposium on Theory of Computing (STOC'00)*. Portland, Oregon, May 2000.
- C. Busch, M. Herlihy, and R. Wattenhofer, "Randomized Greedy Hot-Potato Routing," in *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '00)*, pp. 458–466, San Francisco, California, January 2000.
- C. Busch, N. Demetriou, M. Herlihy and M. Mavronicolas. "Threshold Counters with Increments and Decrements," in *Proceedings of the 6th International Colloquium on Structural Information and Communication Complexity (SIROCCO'99)*, pp. 47–61, Lacanau, France, July 1999.
- C. Busch and M. Herlihy, "Sorting and Counting Networks of Small Depth and Arbitrary Width," in *Proceedings of the 11th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '99)*, pp. 64–73, Saint-Malo, France, June 1999.



- C. Busch and M. Mavronicolas, "An Efficient Counting Network," in *Proceedings of the 1st Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing (IPPS/SPDP'98)*, pp. 380–384, Orlando, Florida, March/April 1998.

# Preface

In distributed and parallel computing environments, there are many computational problems that require efficient communication and coordination between the processes. For such problems there have been proposed several *distributed structures*, each designed to solve a particular problem or a class of problems. The distributed structures take into account the particularities of the specific problems and provide efficient solutions for them.

In this work we study two distributed structures: *counting networks* and the *mesh network*. Counting networks are used for implementing distributed counters; mesh is a communication network. So far, counting networks have been used for increment operations only. We demonstrate that counting networks can be extended to support decrement operations as well. Furthermore, we present new efficient counting network constructions. For the mesh network we study the communication problem, known as *hot-potato routing*, and we provide novel algorithms for the class of hot-potato algorithms known as “greedy.”

Although counting networks and the hot-potato routing in the mesh are used for different purposes, their underlying distributed structures share a common characteristic: the basic structural elements contain a very limited amount of memory. Specifically, in counting networks each structural element contains either a single bit of information or a small number of bits; in the mesh, with greedy hot-potato routing, each node doesn't have any amount of memory for storing communication packets in transit.

The present thesis is structured as follows. The first chapters are devoted to counting networks; while the last ones to hot-potato routing in the mesh. In particular, in Chapters 1, 2, 3, and 4 we study counting networks, and in Chapters 5, 6 and 7 we study hot-potato routing in the mesh. We give our conclusions in Chapter 8.

# Acknowledgements

I would like to thank all people that have helped in the preparation of this thesis. Especially, I would like to thank my advisor Maurice Herlihy for leading me into the world of research and for spending countless hours with me during the quest of solving research problems. I would also like to thank my thesis committee members, Mark Tuttle, and Eli Upfal, for their invaluable comments and suggestions. I deeply thank Marios Mavronicolas for introducing me to research in computer science and for helping me enter the Ph.D. program in the States. I would also like to thank Roger Wattenhofer for giving me the opportunity to work with him.

I would like to thank my peers in the computer science department at Brown University. Especially, I would like to thank Vaso Chatzi for helping me overcome all the obstacles that have appeared in my 5 years in the Ph.D. program. I would also like to thank my former officemate Laurent Michel, and his advisor Pascal Van Hentenryck, for helping me in crucial moments during my graduate studies. I would like to thank Manos Renieris for his invaluable help as Tex wizard, and his help in the preparations of many talks. I deeply thank the following people for their support: Mike Benjamin, Sharon Carabalo, Jose Castanos, Dimitris Michailidis, Luis Ortiz, Gopal Pandurangan, Galina Shubina, and Ioannis Tsochandaridis.

A special thank you goes to my friend and roommate Antonios Augoustakis, whose most invaluable gift was living with him and sharing a friendship that has changed my life for ever.

A big thank you to my parents Kalliopi and Nikos and my brother Lysandros for their moral support and understanding. To them this thesis is dedicated.

# Contents

<b>List of Tables</b>	<b>xii</b>
<b>List of Figures</b>	<b>xiii</b>
<b>1 Introduction to Counting Networks</b>	<b>1</b>
1.1 The Counting Problem . . . . .	1
1.2 Counting Networks . . . . .	2
1.3 Contributions . . . . .	6
1.3.1 Supporting Decrements . . . . .	6
1.3.2 New Constructions . . . . .	6
1.4 Preliminaries for Sequences . . . . .	6
1.4.1 Notations . . . . .	6
1.4.2 Properties . . . . .	7
1.5 Preliminaries for Balancing Networks . . . . .	7
1.5.1 Notations . . . . .	8
1.5.2 Output Function . . . . .	9
1.5.3 Families . . . . .	10
1.5.4 A Construction Technique . . . . .	11
<b>2 Decrements in Balancing Networks</b>	<b>15</b>
2.1 Preliminaries . . . . .	17
2.1.1 Sequences . . . . .	17
2.1.2 Balancing Networks . . . . .	17
2.1.3 Fooling Pairs . . . . .	18
2.1.4 Null Sequences . . . . .	22
2.2 Main Result . . . . .	24
2.3 Applications . . . . .	26

2.3.1	Boundedness Properties . . . . .	26
2.3.2	Threshold Property . . . . .	28
2.4	Discussion . . . . .	30
<b>3</b>	<b>Counting Networks of Arbitrary Width</b>	<b>31</b>
3.1	Preliminaries . . . . .	33
3.1.1	Sequences . . . . .	33
3.1.2	Balancing Networks . . . . .	33
3.2	A Counting Network Construction Framework . . . . .	34
3.2.1	A Counting Network . . . . .	36
3.2.2	A Merger Network . . . . .	37
3.2.3	A Staircase-Merger . . . . .	39
3.2.4	A Two-Merger and a Bitonic-Converter . . . . .	44
3.3	Specific Counting Network Constructions . . . . .	47
3.3.1	The Counting Network $\mathcal{K}$ . . . . .	47
3.3.2	The Counting Network $\mathcal{R}(p, q)$ . . . . .	48
3.3.3	The Counting Network $\mathcal{L}$ . . . . .	50
3.4	Related Work . . . . .	51
3.5	Discussion . . . . .	51
<b>4</b>	<b>Irregular Counting Networks</b>	<b>52</b>
4.1	Preliminaries . . . . .	54
4.1.1	Sequences . . . . .	54
4.1.2	Balancing Networks . . . . .	57
4.1.3	Contention . . . . .	59
4.2	The Counting Network $\mathcal{C}(w, t)$ . . . . .	63
4.3	The Difference Merging Network $\mathcal{M}(t, \delta)$ . . . . .	68
4.4	Contention Analysis of $\mathcal{C}(w, t)$ . . . . .	73
4.5	Related Work . . . . .	75
4.6	Discussion . . . . .	76
<b>5</b>	<b>Introduction to Greedy Hot-Potato Routing in the Mesh</b>	<b>77</b>
5.1	Greedy Hot-Potato Routing . . . . .	77
5.2	Contributions . . . . .	79
5.3	Preliminaries . . . . .	80
5.3.1	Mesh . . . . .	80

5.3.2	Deflections . . . . .	80
5.3.3	Equations . . . . .	80
<b>6</b>	<b>An One-Bend Greedy Hot-Potato Algorithm</b>	<b>82</b>
6.1	Preliminaries . . . . .	83
6.2	The Algorithm . . . . .	83
6.3	Time Analysis . . . . .	86
6.3.1	Succeeding in a Home Run . . . . .	88
6.3.2	Expected Case Analysis . . . . .	92
6.3.3	Analysis “with High Probability” . . . . .	93
6.4	Applications . . . . .	96
6.5	Related Work . . . . .	98
6.6	Discussion . . . . .	99
<b>7</b>	<b>A Multi-Bend Greedy Hot-Potato Algorithm</b>	<b>100</b>
7.1	Preliminaries . . . . .	102
7.2	Algorithm . . . . .	103
7.3	Time Analysis . . . . .	106
7.3.1	The Excited State . . . . .	107
7.3.2	The Running State . . . . .	107
7.3.3	One Packet . . . . .	110
7.3.4	All Packets . . . . .	111
7.4	Lower Bound . . . . .	114
7.5	Related Work . . . . .	115
7.6	Discussion . . . . .	116
<b>8</b>	<b>Conclusions</b>	<b>117</b>
<b>A</b>	<b>Counting Networks of Arbitrary Width</b>	<b>119</b>
<b>B</b>	<b>Irregular Counting Networks</b>	<b>121</b>
	<b>Bibliography</b>	<b>123</b>

\* Part of this thesis appears in [3, 15, 16, 18, 19, 20, 23].

# List of Tables

4.1	The case $a = b$ . . . . .	70
4.2	The case $a = b + 1, k = 1, l = t/2 - 1$ . . . . .	70

# List of Figures

1.1	A balancer and a counting network . . . . .	2
1.2	The input and output sequences . . . . .	8
1.3	Construction of a counting network . . . . .	11
1.4	Construction of a merging network . . . . .	13
1.5	The bitonic network with input/output width 4 . . . . .	13
2.1	Tokens and antitokens . . . . .	16
3.1	Counting network of width $15 = 3 \times 5$ . . . . .	32
3.2	Matrix arrangements . . . . .	34
3.3	Construction of counting network $\mathcal{C}$ . . . . .	36
3.4	Construction of merger network $\mathcal{M}$ . . . . .	37
3.5	Construction of staircase-merger network . . . . .	40
3.6	Optimizing the construction of the staircase-merger . . . . .	42
3.7	Construction of two-merger network . . . . .	45
3.8	Construction of bitonic-converter network . . . . .	46
3.9	Construction of width- $pq$ counting network . . . . .	48
4.1	The three parts of network $\mathcal{C}(w, t)$ . . . . .	53
4.2	The butterfly network . . . . .	58
4.3	The counting network $\mathcal{C}(w, t)$ . . . . .	64
4.4	The difference merging network $\mathcal{M}(t, 2)$ . . . . .	68
4.5	The difference merging network $\mathcal{M}(t, \delta)$ . . . . .	69
5.1	The $n \times n$ mesh network . . . . .	78
6.1	Packet States and Priority . . . . .	84
6.2	A Home Run: column-first and row-first traversals . . . . .	84



6.3	A “pathological” batch problem . . . . .	97
7.1	Packets with destinations in a region . . . . .	101
7.2	The squares and bands of a node . . . . .	102
7.3	Packet states and priority . . . . .	103
7.4	Up: a deflected packet becomes <i>excited</i> . Down: a successful <i>running</i> path .	104

# Chapter 1

## Introduction to Counting Networks

### 1.1 The Counting Problem

Counting is a fundamental computational task in distributed, shared memory, multiprocessors. In the *counting problem*, there are many distributed processes that need to perform an *increment* operation. Each increment operation returns a unique integer, so that if there were  $m$  total increment operations performed, then each increment operation returns a unique integer between  $0, \dots, m - 1$ .

A simple solution to the counting problem is to use a single shared variable, e.g.  $v$ . Each increment operation can be implemented by an atomic *Fetch&Increment* operation to  $v$ . In the Fetch&Increment operation, a process first locks the variable  $v$ , then it reads its contents, then increments  $v$  by one, and finally it unlocks  $v$  and returns the value it read. Initially, the variable  $v$  contains the value 0.

Although the single shared variable solution is easy to implement, it suffers from two major drawbacks:

- **High contention.** If there are  $n$  processes in the system then all the processes may try to perform Fetch&Increment operations on  $v$  at the same time. In such a case, all the processes will contend to getting access to  $v$ . As a consequence, a process may have to wait until all the other processes finish their increment operations on  $v$ , which is a sequential bottleneck.
- **Starvation.** A process which performs a Fetch&Increment operation may fail while it has locked the variable  $v$ . In such a case, the failed process will not allow the rest of the processes to access variable  $v$ , causing these processes to starve, since they are unable to continue computations.

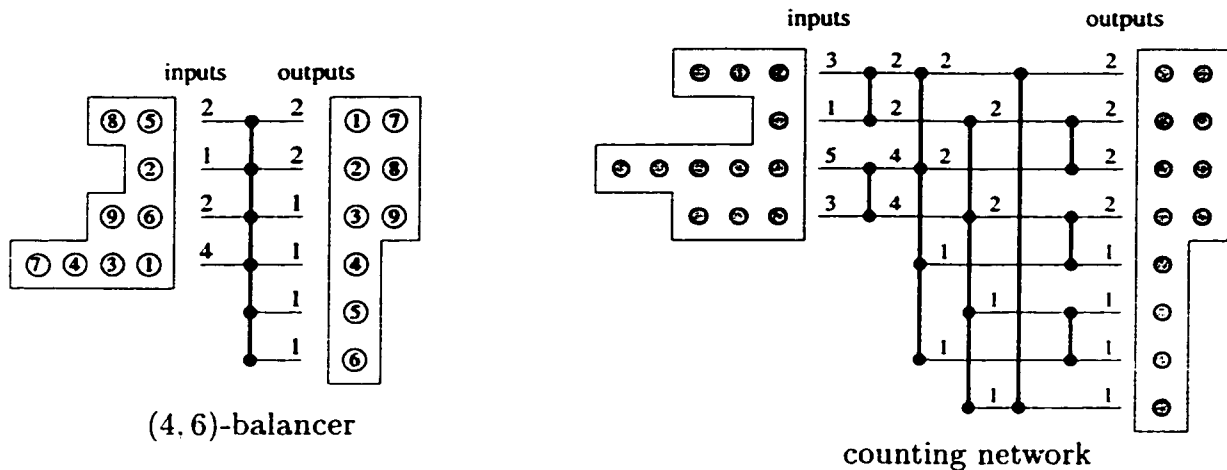


Figure 1.1: A balancer and a counting network

Therefore, the centralized solution of the single variable is not adequate to solve efficiently the counting problem: an alternative solution is needed.

## 1.2 Counting Networks

As an alternative solution to the counting problem, Aspnes, Herlihy and Shavit [6] have introduced *Counting networks*. Counting networks are highly distributed data structures which do not suffer from the problems of the centralized single shared variable. Namely, counting networks exhibit low contention and are *wait-free* (processes don't starve).

Counting networks are a subclass of *balancing networks*. A balancing network is constructed by connecting together elementary switches called *balancers*.<sup>1</sup> A  $(p, q)$ -balancer has  $p$  input wires and  $q$  output wires (see Figure 1.1, where  $p = 4$  and  $q = 6$ ). A *balancing network* is an acyclic network of balancers where output wires of some balancers are linked to input wires of others (see Figure 1.1). The network's *input wires* are those wires not connected to any of the balancer's output wires, and similarly for the network's *output wires*. The number of input wires is the network's *input width* and the number of output wires is the network's *output width*. We order the wires from top to bottom, so that for  $n$  input wires the top wire is wire 0 and the bottom wire is wire  $n - 1$ . Similarly, we order the output wires.

The distributed processes access a balancing network by issuing *tokens*. Each token

<sup>1</sup>Balancing networks are constructed in a similar way that comparator networks (e.g. sorting networks) are constructed from comparators.

corresponds to the operation that the process wishes to perform in the balancing network (for a counting network a token is an increment operation). A token enters the balancing network from any of its input wires. Then, the token traverses the network by moving from one balancer to the next balancer, following the interconnecting wires, until the token reaches an output wire. The tokens traverse the network asynchronously, and usually there can be many tokens in the network which are traversing it simultaneously.

Let us now examine how a balancer is traversed by tokens. A balancer is implemented as a shared variable whose value corresponds to the *state* of the balancer. The state points to an output wire of the balancer, namely, for a  $(p, q)$ -balancer the state is one of  $0, \dots, q - 1$ . Initially, the state is 0. Consider now a  $(p, q)$ -balancer, and as an example consider the balancer of Figure 1.1 where the tokens are drawn with circles. A token accesses atomically the balancer and performs the following operations. The token first locks the balancer. If the current state of the balancer is  $s$ , then the token reads the state  $s$  and increases the state by one so that the next state is  $(s + 1) \bmod q$  (notice that if the current state is  $q - 1$  then the next state is 0). The token, then, unlocks the balancer and exits the balancer from output wire  $s$ . This process is repeated for every token. Since a balancer is accessed atomically by the tokens, there is a sequential order imposed on the tokens of the balancer. In the example balancer of Figure 1.1, this sequential order of the tokens is drawn as a number inside each token's circle.

Let's assume now that the  $(p, q)$ -balancer has reached a *quiescent configuration*, in which all the tokens that have ever entered the balancer have left it, so that there are no more tokens traversing the balancer. The distribution of the tokens on the output wires of the balancer is almost uniform (see Figure 1.1). Namely, the number of tokens on each output wire is the same, and any excess tokens appear on the top wires. We say that the distribution of the tokens on the output wires of the balancer satisfies the *step property*. Formally speaking, let  $Y = y_0, \dots, y_{q-1}$  be a sequence such that  $y_i$  is equal to the total number of tokens that exit from the output wire  $i$  of the balancer. We call  $Y$  the output sequence of the balancer. Similarly, we define the input sequence of the balancer. In any quiescent configuration and for any input sequence, the output sequence  $Y$  of the balancer  $b$  satisfies the step property:

$$0 \leq y_i - y_j \leq 1, \text{ for all } 0 \leq i < j < q.$$

In Figure 1.1 we write on each wire the total number of tokens that appear on the wire.

We can generalize the above definitions for balancing networks. For instance, consider a balancing network of output width  $w$ . We say that the balancing network is in a quiescent

configuration if there are no more tokens traversing the network, that is, all the tokens that have entered the network have left it. In any quiescent configuration, the input and output sequences are defined in the same way as for the balancers. Namely, in the output sequence  $Y = y_0, \dots, y_{w-1}$ , the element  $y_i$  is equal to the total number of tokens that exit from output wire  $i$  of the network (and similarly for the input sequence).

A *counting network* is a balancing network whose output sequence satisfies the step property in any quiescent configuration and for any input sequence. In particular, consider a counting network of output width  $w$  which is in a quiescent configuration and has output sequence  $Y = y_0, \dots, y_{w-1}$ . Then for any input sequence, the output sequence  $Y$  satisfies the step property:

$$0 \leq y_i - y_j \leq 1, \text{ for all } 0 \leq i < j < w.$$

The balancing network of Figure 1.1 is a counting network. In the same figure above, we write on each wire of the network the total number of tokens that appear on that wire.

We describe now the way a counting network solves the counting problem. Each token corresponds to an increment operation issued by some process. Let's assume that we are given a counting network  $C$  with output width  $w$ . With each output wire  $i$  of the network  $C$  we associate a corresponding output shared variable  $v_i$ . Initially, each output variable  $v_i$  contains the value  $i$ . Whenever a process needs to perform an increment operation, it issues a token which traverses the network until it reaches an output wire. The token, then, returns the value that reads from the corresponding output variable of the output wire. In detail, consider the actions of a token when it exits from the output wire  $i$  of the network. The token first locks the variable  $v_i$ , so that the token accesses the variable atomically, in case other tokens exit from the same wire. If the current value of the variable  $v_i$  is  $a$  then the token reads the value  $a$ . Then, the token increases the value of the variable  $v_i$  by  $w$ , the output width, so that the next time the variable will contain  $a + w$ . Then, the token unlocks the variable and returns the value  $a$ . This way, each token performs an increment operation of some process.

The step property of the counting network guarantees that the increment operations are performed correctly. Namely, if there are  $m$  total tokens accessing the network, then each such token will return a unique integer in the range  $0, \dots, m - 1$ . As an example, consider a counting network of Figure 1.1 with width  $w = 8$  which is accessed by a total of  $m = 12$  tokens. Each token should return a unique value in the range  $0, \dots, 11$ . Because of the step property, from each of the top four output wires exit two tokens, and from each of the bottom four output wires exits one token. From wire 0, one token returns 0 and the other

8. From wire 1, one token returns 1 and the other 9. Similarly, for the rest of the four top wires. From wire 4, the token returns 4. From wire 5, the token returns 5. Likewise, for the rest bottom wires. Obviously, each token returns a unique value in the range  $0, \dots, 11$ , and therefore the counting is correct.

Similar to counting networks, there are other families of balancing networks which are used to solve other kinds of distributed problems. These balancing network families are distinguished by the properties their output sequences satisfy. Two such well known families of balancing networks are the *smoothing networks* and the *threshold networks*. Smoothing networks are used to solve load balancing problems, and threshold networks solve barrier synchronization problems.

All the above families of balancing networks do not suffer from the problems of centralized schemes. This is because balancing networks are highly distributed, since there are many shared variables, the balancers and the output variables, among which the processes are dispersed. As a consequence, the contention in the variables of a balancing network is small, since it is less likely for the processes to meet at the same variable at the same time. Moreover, balancing networks are wait-free: if one of the processes fails in one of the shared variables of the network, then the rest of the processes in the other variables can continue uninterrupted.

There are two important factors that determine the good performance of balancing networks:

- **Depth.** The depth of a balancing network is the maximum number of balancers a token has to traverse from an input wire to an output wire. For example, the network of Figure 1.1 has depth 3. The depth of a network determines its *latency*: the higher the depth the more the time a token spends in the network.
- **Contention.** The contention of a counting network determines its *throughput*: the smaller the contention the higher the number of processes that can access the counting network simultaneously.

Therefore, it is desirable to construct balancing networks which have both small depth and small contention.

We proceed as follows. In Section 1.3, we present our contributions, and in Sections 1.4 and 1.5 we give some necessary preliminaries on sequences and balancing networks, which are common to the subsequent chapters.

## 1.3 Contributions

### 1.3.1 Supporting Decrements

So far, counting networks have been used for increment operations only. Some distributed computing applications require the use of decrement operations as well. In Chapter 2, we show that counting networks, and in general a broad class of balancing networks, can be used to support decrement operations together with increment operations. In particular, we show that if we know that a balancing network “works correctly” for increments, then it will also work correctly for decrements. This makes balancing networks suitable for a broader range of applications.

### 1.3.2 New Constructions

Most of the previously known counting networks have equal input and output width and this width is a power of 2. It was an open question whether we can construct counting networks with arbitrary widths that have also small depth. In Chapter 3, we present a novel small-depth counting network construction, with equal input and output width, whose width is an arbitrary integer.

In Chapter 4, we present a novel counting network construction which improves in terms of contention over the previously known counting network constructions. In this construction, the output width is bigger or equal to the input width, and by increasing the output width we can increase the number of balancers in the network without increasing the depth. By having more balancers the contention decreases, since processes are less likely to meet in the same balancers.

## 1.4 Preliminaries for Sequences

### 1.4.1 Notations

We consider sequences of integers. We denote sequences in upper case and elements of a sequence in lower case. For example, consider the sequence  $X = x_0, \dots, x_{w-1}$ . The *length* (or *width*) of sequence  $X$  is  $|X| = w$ . Often, we denote the length of  $X$  with a superscript as  $X^{(w)}$ . The sum of the elements of  $X$  is  $\Sigma(X) = x_0 + \dots + x_{w-1}$ .

### 1.4.2 Properties

We say that a sequence  $X^{(w)}$  has the *step property* whenever

$$0 \leq x_i - x_j \leq 1, \text{ for all } 0 \leq i < j < w.$$

Alternatively, we say that  $X^{(w)}$  is a step sequence. If  $X^{(w)}$  has the step property then its step point is the unique index  $i$  such that  $x_i < x_{i-1}$ . If all  $x_i$  are equal then the step point can be defined to be either 0 or  $w$ , and we will explicitly specify this in the subsequent chapters, where we use the step points. The following equality holds for any element  $x_i$  of a step sequence  $X^{(w)}$ ,

$$x_i = \left\lfloor \frac{\Sigma(X^{(w)}) - i}{w} \right\rfloor. \quad (1.1)$$

We immediately have the following observation.

**Observation 1.4.1** *Any subsequence of a step sequence has the step property.*

We say that a sequence  $X^{(w)}$  has the *k-smoothing property*, for some  $k \geq 1$ , whenever

$$|x_i - x_j| \leq k, \text{ for all } 0 \leq i, j < w.$$

Alternatively, we say that the sequence  $X$  is a *k-smooth sequence*. The elements of a *k-smooth sequence* take values in a range  $a, a + 1, \dots, a + k$ , for some value  $a$ . Notice that any step sequence is also 1-smooth.

We say that a sequence  $X^{(w)}$  has the *threshold property* whenever

$$x_{w-1} = \left\lfloor \frac{\Sigma(X^{(w)})}{w} \right\rfloor.$$

Alternatively, we say that sequence  $X^{(w)}$  is a *threshold sequence*. Notice that any step sequence is also threshold.

## 1.5 Preliminaries for Balancing Networks

Henceforth (unless otherwise stated), for the rest of this chapter and the other chapters, we consider balancers and balancing networks in quiescent configurations, where all the tokens that have ever entered the balancing network have left it. We do this, because we want to reason about the distributions of the tokens on the output wires in quiescent configurations.



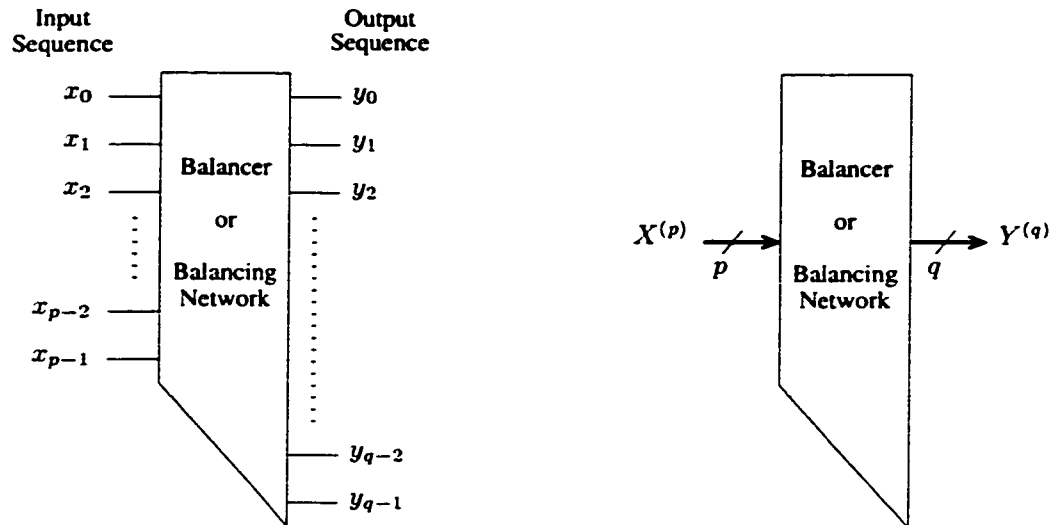


Figure 1.2: The input and output sequences

### 1.5.1 Notations

Often, for convenience, we draw balancers and balancing networks with boxes as shown in Figure 1.2. Consider now a  $(p, q)$ -balancer with input sequence  $X^{(p)}$  and output sequence  $Y^{(q)}$ . We draw the input and output sequences of the balancer as shown in Figure 1.2, where there are two alternative representations: one representation where all the elements of the sequence are drawn with wires and the other representation where the whole sequence is drawn with a single wire. In the same way we draw the input and output sequences for balancing networks.

Any balancing network with input sequence  $X^{(w)}$  and output sequence  $Y^{(t)}$  satisfies the *sum preservation property*:

$$\Sigma(X^{(w)}) = \Sigma(Y^{(t)}).$$

The depth of a balancing network  $\mathcal{B}$ , is denoted by  $\text{depth}(\mathcal{B})$ . Any balancing network can be decomposed into smaller components connected in series which are called *layers*, where each layer has depth 1. For example, the network of Figure 1.1 consists from three layers, and going from the input wires to the output wires, the first layer consists from two  $(2, 2)$ -balancers, the second layer from two  $(2, 4)$ -balancers, and the last layer by four  $(2, 2)$ -balancers.

### 1.5.2 Output Function

Consider a  $(p, q)$ -balancer  $b$ , with input sequence  $X^{(p)}$  and output sequence  $Y^{(q)}$ . By the definition of the balancer, the output sequence  $Y^{(q)}$  satisfies the step property for any input sequence  $X^{(p)}$ . By equation 1.1, and by the sum preservation property of the balancer, we have

$$y_i = \left\lceil \frac{\Sigma(X^{(p)}) - i}{q} \right\rceil.$$

This equation implies that the output sequence of the balancer is determined by the input sequence, so that each input sequence has only one respective output sequence. Therefore, the output sequence is a function of the input sequence. We can think of the balancer  $b$  as a function which maps input sequences to output sequences. Sometimes, for convenience, we write  $b(X^{(p)}) = Y^{(q)}$ .

We can generalize the above observation from balancers to balancing networks, and we show in the next lemma that any balancing network is a function from input sequences to output sequences.

**Lemma 1.5.1** *A balancing network  $\mathcal{B}$  is a function from input sequences to output sequences, so that each input sequence has only one respective output sequence.*

**Proof:** Let  $d$  be the depth of the network  $\mathcal{B}$ . We prove the result by induction on the depth  $d$ .

First, we consider the base case, where  $d = 1$ . In this case, the network consists from a single layer of balancers. The input sequence of the network is distributed and fed to the input sequences of the balancers, and the output sequence of the network is the combination of the output sequences of the balancers. Each balancer of the layer is a function from its input sequence to its respective output sequence. Combining the balancers, we have that the output sequence of the network is a function of the input sequence of the network.

For the induction step, let's assume that the result holds for all the networks with depths up to  $d - 1$ . We split the network  $\mathcal{B}$  into two smaller networks  $\mathcal{B}'$  and  $\mathcal{B}''$ , each with depth smaller or equal to  $d - 1$ , in such way that  $\mathcal{B}$  is formed by connecting in series  $\mathcal{B}'$  and  $\mathcal{B}''$ . The input sequence of network  $\mathcal{B}$  is the input sequence of network  $\mathcal{B}'$  and the output sequence of  $\mathcal{B}'$  is the input sequence of network  $\mathcal{B}''$ . By the induction hypothesis, each of the networks  $\mathcal{B}'$  and  $\mathcal{B}''$  is a function from input sequences to output sequences, since each has depth smaller or equal to  $d - 1$ . The output sequence of network  $\mathcal{B}$  is produced by the composition of the functions of networks  $\mathcal{B}'$  and  $\mathcal{B}''$ . Since the composition of two functions

is a function, we have that the network  $\mathcal{B}$  is a function from input sequences to output sequences. ■

From the above lemma we have that a balancing network  $\mathcal{B}$  is a function from input to output sequences. Let  $w$  be the input width of the network and  $t$  the output width. Let  $X^{(w)}$  be the input sequence and  $Y^{(t)}$  the respective output sequence. Sometimes, for convenience, we write  $\mathcal{B}(X^{(w)}) = Y^{(t)}$ .

When we construct a balancing network, we take advantage of the functional behavior of the network. For example, we build a counting network by combining smaller counting networks (see Section 1.5.4). We can easily prove the correctness of the counting network by analyzing the functional behavior of the smaller networks.

### 1.5.3 Families

As we have seen in Section 1.2, a counting network is any balancing network whose output sequence satisfies the step property, on any input sequence. In a similar way, we define other families of balancing networks, according to what kind of properties their output sequences satisfy. These balancing networks are used for solving different kinds of distributed computing problems.

A *k-smoothing network* [2, 6] is a balancing network whose output sequence satisfies the *k-smoothing property*, for any input sequence. In a smoothing network, the tokens are distributed in the output wires in such a way that the number of tokens on any two output wires differ by at most  $k$ . Smoothing networks are used for load balancing applications [36], where jobs need to be distributed in different processors in a balanced way so that all processors perform approximately an equal amount of work. Furthermore, smoothing networks are used as structural components for building counting networks [39].

A *threshold network* [6, 22] is any balancing network whose output sequence satisfies the *threshold property*, for any input sequence. Threshold networks can be used for a weak form of counting, where the only output wire of the network that counts is the bottom wire. The output variable  $v$  of the bottom output wire holds a value which corresponds to the “approximate” total number of tokens that entered the network. If the output width of the network is  $w$ , then the variable  $v$  counts chunks of  $w$  tokens that traverse the network. Namely, for every  $w$  tokens, one of these tokens leaves from the bottom output wire. Furthermore, threshold networks can be used for barrier synchronization. If there are  $w$  processes to be synchronized in the barrier, then the first  $w - 1$  processes that entered the network wait until the last process exits from the bottom wire. Threshold networks are

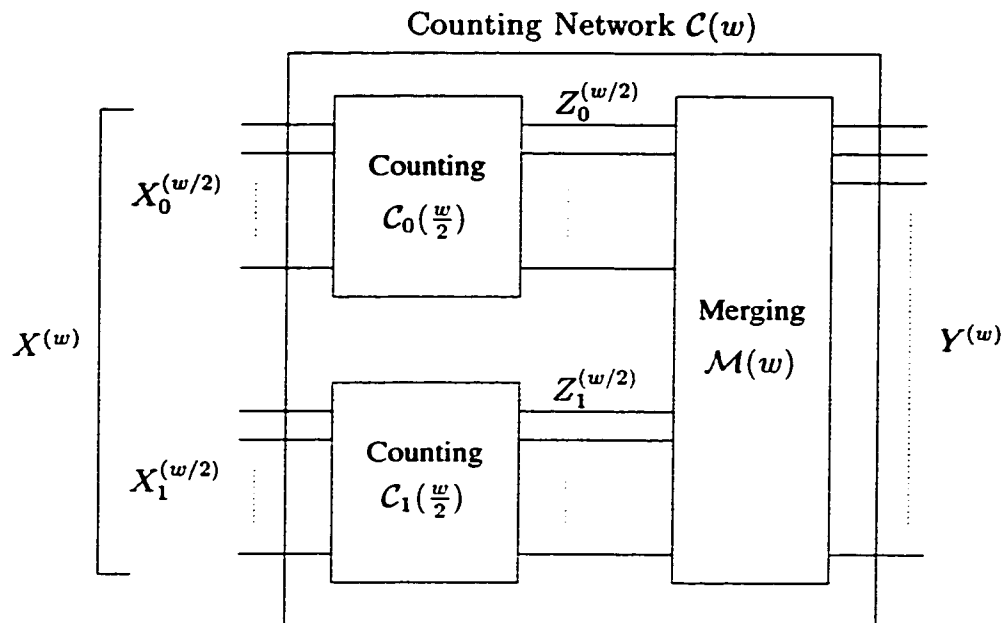


Figure 1.3: Construction of a counting network

interesting because they can be implemented more efficiently than counting networks (see [6]).

#### 1.5.4 A Construction Technique

In this section we present a simple technique for constructing counting networks. In particular, we show how to construct a counting network inductively, by first building smaller counting networks and then merging the outputs of the smaller networks, as shown in Figure 1.3. The counting network constructions of Chapters 3 and 4, are based on this construction technique.

The technique of constructing networks inductively was first introduced by Batcher [9], who used it to construct *sorting networks*. Sorting networks look like counting networks<sup>2</sup> and are used for parallel sorting. Using the inductive technique, Batcher constructed the *bitonic sorting network* and the *odd-even sorting network*. (For more information about sorting networks look in [40].) Aspnes *et al.* [6], showed that the same inductive technique can be used to construct counting networks. They showed that the isomorphic balancing network of the bitonic sorting network, is a counting network too. To demonstrate the inductive construction technique, we present here the construction of the bitonic counting

<sup>2</sup>In sorting networks the number of input and output wires of the network is the same.

network.

We will construct the bitonic counting network  $\mathcal{C}$  of input and output width  $w$ , where  $w$  is some power of 2 (see Figure 1.3). We denote the network as  $\mathcal{C}(w)$ .<sup>3</sup> Let  $X^{(w)}$  be the input sequence and  $Y^{(w)}$  the corresponding output sequence of the network.

The construction is by induction on the width  $w$ . For the base case, where  $w = 2$ , the network is simply a  $(2, 2)$ -balancer. Let's assume that we have constructed the network  $\mathcal{C}(w/2)$ . We will construct now the network  $\mathcal{C}(w)$ . We take two copies of the network  $\mathcal{C}(w/2)$  (given by induction hypothesis), which we denote  $\mathcal{C}_0(w/2)$  and  $\mathcal{C}_1(w/2)$ . We split the input sequence  $X^{(w)}$  into two sequences each of width  $w/2$ , which we denote  $X_0^{(w/2)}$  and  $X_1^{(w/2)}$ . We feed the sequence  $X_0^{(w/2)}$  to network  $\mathcal{C}_0(w/2)$  and the sequence  $X_1^{(w/2)}$  to network  $\mathcal{C}_1(w/2)$ . Let  $Z_0^{(w/2)}$  and  $Z_1^{(w/2)}$  denote the respective output sequences of these two networks. By the induction hypothesis, we have that each of the sequences  $Z_0^{(w/2)}$  and  $Z_1^{(w/2)}$  satisfies the step property.

Next, we use a *merging network*, which receives two input sequences with the step property, merges these sequences, and produces one output sequence which has the step property. Let  $\mathcal{M}(w)$  be the merging network of our construction. The network  $\mathcal{M}(w)$  accepts on its inputs the step sequences  $Z_0^{(w/2)}$  and  $Z_1^{(w/2)}$ , merges these sequences, and produces the final output sequence  $Y^{(w)}$ , which has the step property.

The merging network is also constructed by induction on  $w$ , as shown in Figure 1.4. For the base case where  $w = 2$ , the merging network is simply a  $(2, 2)$ -balancer. Let's assume now that we have constructed the merging network  $\mathcal{M}(w/2)$ . We will construct the merging network  $\mathcal{M}(w)$ .

The network  $\mathcal{M}(w)$  accepts the two input sequences  $Z_0^{(w/2)}$  and  $Z_1^{(w/2)}$ . We take two copies of the network  $\mathcal{M}(w/2)$  (given by induction hypothesis), which we denote  $\mathcal{M}_0(w/2)$  and  $\mathcal{M}_1(w/2)$ . We split the sequence  $Z_0^{(w/2)}$  into its even and odd subsequences, where the even subsequence contains the elements with even index  $0, 2, \dots$ , and the odd subsequence contains the elements with odd index  $1, 3, \dots$ . Similarly, we split the sequence  $Z_1^{(w/2)}$  into its even and odd subsequences. The two input sequences of network  $\mathcal{M}_0(w/2)$  are the even subsequence of  $Z_0^{(w/2)}$  and the odd subsequence of  $Z_1^{(w/2)}$ . The two input sequences of  $\mathcal{M}_1(w/2)$  is the odd subsequence of  $Z_0^{(w/2)}$  and the even subsequence of  $Z_1^{(w/2)}$ .

Since the sequences  $Z_0^{(w/2)}$  and  $Z_1^{(w/2)}$  have the step property, by Observation 1.4.1, their even and odd subsequences will also have the step properties. Subsequently, the merging networks  $\mathcal{M}_0(w/2)$  and  $\mathcal{M}_1(w/2)$  will merge step sequences, and thus their respective

<sup>3</sup>This notation should not be confused with the function notation of Section 1.5.2.

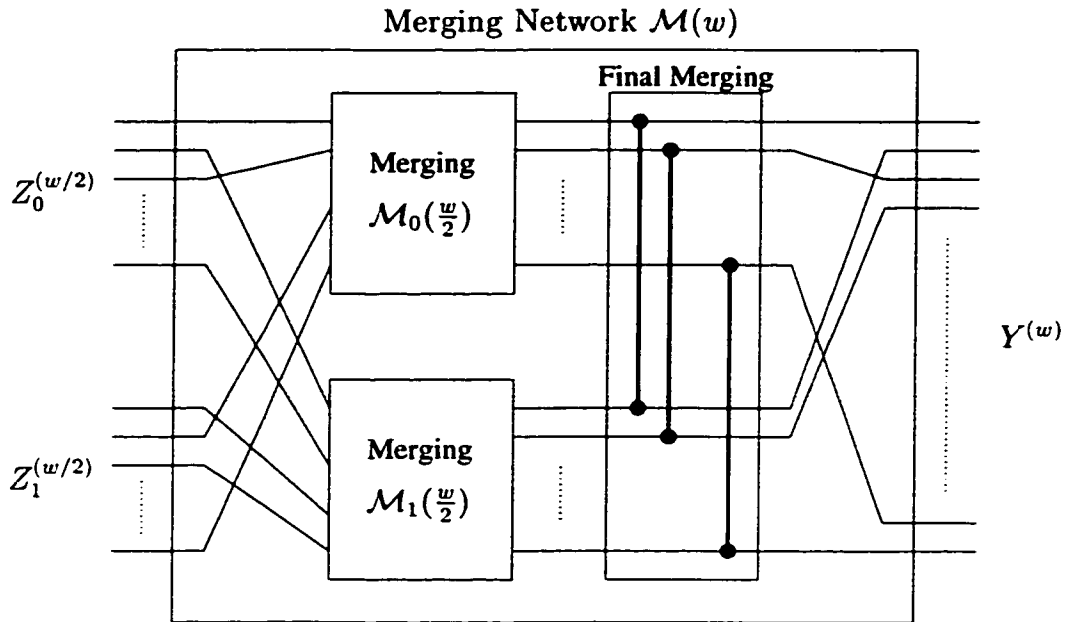


Figure 1.4: Construction of a merging network

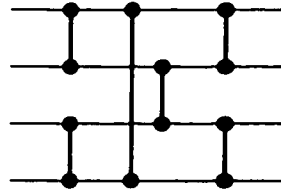


Figure 1.5: The bitonic network with input/output width 4

output sequences will have the step property too (by the induction hypothesis).

Consider now the sequence  $Z_0^{(w/2)}$ . From the step property of sequence  $Z_0^{(w/2)}$ , it is easy to observe that the sums of the elements of the even subsequence differ by at most one from the sums of the elements of the odd subsequence. A similar observation holds for sequence  $Z_1^{(w/2)}$ . Combining these two observations, it is easy to infer that the respective sums of the output sequences of the two merging networks,  $\mathcal{M}_0(w/2)$  and  $\mathcal{M}_1(w/2)$ , differ by at most one. Therefore, on the outputs of the two merging networks we obtain two step sequences whose sums differ by at most one. Because of this small difference, We can easily merge these two sequences with a single layer of balancers, denoted as “final merging” network in Figure 1.4, and the produced output sequence  $Y^{(w)}$  has the step property.

We can easily see that the depth of the inductive construction depends on the depth of

the “final merging” network. In the bitonic network, the “final merging” network has depth 1 (it is just one layer of balancers). As a consequence, the depth of the merging network  $\mathcal{M}(w)$  is  $O(\log w)$ , since there are  $O(\log w)$  induction steps and in each induction step the depth increases by only 1 (the depth of the “final merging” network). Furthermore, the depth of the counting network  $\mathcal{C}(w)$  will be  $O(\log^2 w)$ , since there are  $O(\log w)$  induction steps and in each induction step the depth increases by at most  $O(\log w)$ , the depth of the merging network. As a small example, for  $w = 4$ , we obtain the bitonic counting network  $\mathcal{C}(4)$  which is depicted in Figure 1.5, and has depth 3.

In general, we want to build counting networks with small depth. In order to achieve this, when we construct counting networks using the inductive technique, we want the “final merging” network to have a constant depth  $c$ . This way, the total depth of the network is at most  $O(c \log^2 w) = O(\log^2 w)$ . By increasing the depth of the “final merging” network we obtain networks with worse depths.

The constructions we present in Chapters 3 and 4 are variations of the inductive construction of the bitonic network. In particular, in Chapter 3 we modify the inductive construction so that the input and output width of the network is an arbitrary integer, and not only a power of 2. To achieve this, we use more than two copies of the smaller counting networks (given by the induction hypothesis) in the inductive construction of the counting network. Similarly, the merging network is made from more than two copies of the smaller merging networks (given by the induction hypothesis). In that construction, the “final merging” network has constant depth, which gives a total network depth  $O(\log^2 w)$ . The construction of Chapter 4, modifies the inductive construction of the bitonic network so that it handles a different number of input and output wires. The “final merging” network there has depth 1, and the total network depth is again  $O(\log^2 w)$ .

## Chapter 2

# Decrements in Balancing Networks

A limitation of balancing networks is that they are accessed by tokens only. In a balancing network, a token performs “increment” operations to the balancers it accesses (it increases the state of the balancer). By using tokens only, the capabilities of balancing networks are limited to solving distributed problems based on increment operations. However, many distributed problems require, the ability to perform “decrement” operations. For example, the classical synchronization constructs of *semaphores* [24], *critical regions* [34], and *monitors* [31] all rely on applying both increment and decrement operations on shared counters (see, e.g., [50, Chapter 6]).

In order to solve such kinds of problems Shavit and Touitou [48] invented the *antitoken*, the complementary entity of a token which corresponds to a “decrement” operation. As we have seen in Chapter 1, a token traverses a balancer by reading first the state of the balancer and then increasing the state by one. On the other hand, an antitoken first decreases the state of a balancer by one and then reads the state and exits from the corresponding wire. Informally, an antitoken “cancels” the effect of the most recent token on the balancer’s state, and vice versa. Furthermore, when an antitoken and a token meet while they traverse a network they can “eliminate” each other without needing to traverse the rest of the network.

We can think of a token and an antitoken as having complementary algebraic values so that a token corresponds to the quantity  $+1$ , and an antitoken to the quantity  $-1$ . As an example, tokens and antitokens are depicted in Figure 2.1, where an antitoken is drawn with an empty circle and a token is drawn with a full circle. On each wire we see the algebraic sum of tokens and antitokens that appear on that wire. Each element of the output sequence of a balancing network is now equal to the algebraic sum of tokens and antitokens that appear on the corresponding output wire. Subsequently, each element of the output sequence can



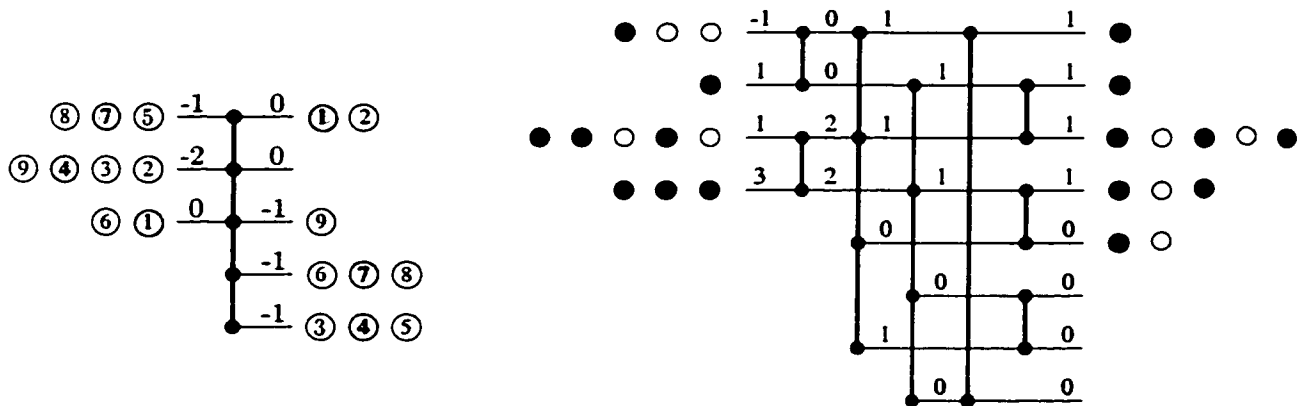


Figure 2.1: Tokens and antitokens

be either positive or negative, depending on whether the tokens or antitokens are of excess on the corresponding output wire. (Similarly for the input sequences.) This sum is shown on each wire of Figure 2.1.

For a specific kind of counting networks, which have the form of a binary tree, Shavit and Touitou [48] showed that these networks count correctly even when they are traversed by both tokens and antitokens. Namely, they showed that the output sequence of any such network satisfies the step property even when the antitokens are introduced.

For simplicity, we say that a network satisfies a property if its output sequence satisfies that property. Here, we answer the more general question: which properties of balancing networks are preserved by the introduction of antitokens? We give a new characterization of balancing network properties: *properties closed under the nullity of a balancing network*. We show that if a balancing network satisfies a property for tokens only and this property is closed under the nullity of the balancing network, then the network will still satisfy the same property under the introduction of antitokens.<sup>1</sup>

We show that all the known properties, namely, the step property, the  $k$ -smoothing property, and the threshold property, are all closed under the nullity of a balancing network. Subsequently, all these properties are preserved under the introduction of antitokens. As a result, for all the known counting,  $k$ -smoothing, and threshold network constructions which we know they satisfy their properties with tokens only, we can immediately infer that they can also support antitokens, namely, these networks will satisfy their corresponding properties with both tokens and antitokens. As an example, we know from Chapter 4 that the network in Figure 2.1 is a counting network with tokens. From what we show here, we

<sup>1</sup>The work of this chapter was first presented in [3, 16, 15].

can immediately infer that the step property of this network is still preserved even when antitokens are introduced.

We proceed as follows. In Section 2.1 we give necessary preliminaries. In Section 2.2 we prove our main result about properties closed under the nullity of a network. In Section 2.3 we apply our main result to the known properties of balancing networks. Finally, in Section 2.4 we give a discussion. (There is no related work other than the one mentioned above.)

## 2.1 Preliminaries

### 2.1.1 Sequences

We treat sequences as vectors, and we define the operations of addition and multiplication on sequences in the same way that are defined for vectors.

We use  $\mathbf{0}^{(g)}$  to denote the sequence  $\mathbf{0}^{(g)} = 0, 0, \dots, 0$ , a sequence with  $g$  zero entries. Similarly, we use  $\mathbf{1}^{(g)}$  to denote  $\mathbf{1}^{(g)} = 1, 1, \dots, 1$ , a sequence with  $g$  unit entries. A *constant sequence* is any sequence of the form  $c \cdot \mathbf{1}^{(g)}$ , for any constant  $c$ .

A *non-negative sequence*, is any sequence whose entries are non-negative integers.

### 2.1.2 Balancing Networks

Since, a balancing network may be traversed now by both tokens and antitokens, we consider balancing networks in quiescent configurations in which no tokens or antitokens are traversing the network (that is, all the tokens and antitokens that have entered the network have left it).

If a balancing network is traversed by tokens only and we are given the input sequence, then we can infer how many tokens have entered on each input wire: the number of tokens on the wire is equal to the value of the corresponding element in the input sequence. With both tokens and antitokens, knowing the input sequence doesn't necessarily mean that we know the number of tokens and antitokens that have entered from a specific input wire. This is because on an input wire of the network, there are infinite combinations of tokens and antitokens that can produce the same algebraic sum in the corresponding element of the input sequence. A similar observation holds for the output sequence. Here, we consider only algebraic sums on each input/output wire, and the corresponding input/output sequences, because these characterize the properties of the balancing networks.

It is easy to see that the output sequence of a balancer still has the step property even when antitokens are introduced. Furthermore, we can easily extend the observations and

results of section 1.5.2, about the functional behaviors of balancing networks, to hold even when antitokens are introduced. In particular, a balancer can be thought of as function from input to output sequences, and similarly, a balancing network can be thought of as a function from input to output sequences. Consider a balancing network  $\mathcal{B}$  of input width  $w$ , and output width  $t$ . Let  $X^{(w)}$  be an input sequence and  $Y^{(t)}$  be the corresponding output sequence. We write  $\mathcal{B}(X^{(w)}) = Y^{(t)}$ . Trivially,  $\mathcal{B}(\mathbf{0}^{(w)}) = \mathbf{0}^{(t)}$ .

The sum preservation property still holds in any balancing network, even though some tokens and antitokens may be eliminated in the network's balancers. That is, for any input sequence  $X^{(w)}$ ,

$$\Sigma(\mathcal{B}(X^{(w)})) = \Sigma(X^{(w)}).$$

In Section 1.2, we have defined the state of the balancer to always point to an output wire, so that if a token arrives, then the token will exit from that wire. Here, we keep the same definition even when antitokens are introduced. We can easily see that in quiescent configurations the state of the balancer is uniquely determined by the algebraic sum of all the elements in the input sequence. For any  $(p, q)$ -balancer  $b$  with input sequence  $X^{(p)}$ , we denote the state of the balancer as  $\text{state}_b(X^{(p)})$ . Furthermore, we have

$$\text{state}_b(X^{(p)}) = \Sigma(X^{(p)}) \bmod q.$$

We immediately obtain the following “linearity” lemma for the state of a balancer.

**Lemma 2.1.1** *For any two input sequences  $X_1^{(p)}$  and  $X_2^{(p)}$  of a  $(p, q)$ -balancer  $b$ ,*

$$\text{state}_b(X_1^{(p)} + X_2^{(p)}) = (\text{state}_b(X_1^{(p)}) + \text{state}_b(X_2^{(p)})) \bmod q.$$

We generalize the definition of the state to balancing networks. For any balancing network  $\mathcal{B}$  of input width  $w$ , and any input sequence  $X^{(w)}$ , we define the *state* of the network, denoted by  $\text{state}_{\mathcal{B}}(X^{(w)})$ , to be the collection of the states of its individual balancers after the specific input sequence. The *initial state* of the balancing network is  $\text{state}_{\mathcal{B}}(\mathbf{0}^{(w)})$ . (Initially, when the network started operating, it hasn't received any input sequence, which corresponds to the input sequence  $\mathbf{0}^{(w)}$ .)

### 2.1.3 Fooling Pairs

For any balancing network  $\mathcal{B}$ , with input width  $w$ , we say that two input sequences  $X_0^{(w)}$  and  $X_1^{(w)}$  are a *fooling pair* to network  $\mathcal{B}$  if

$$\text{state}_{\mathcal{B}}(X_0^{(w)}) = \text{state}_{\mathcal{B}}(X_1^{(w)}).$$

Roughly speaking, a fooling pair “drives” all balancers of the network to identical states.

We continue by showing a “linearity” result for fooling pairs. First we show the “linearity” result for balancers, and then for balancing networks.

**Lemma 2.1.2** *Consider a  $(p, q)$ -balancer  $b$ . Take any input sequences  $X_1^{(p)}$  and  $X_2^{(p)}$  that are a fooling pair to balancer  $b$ . Then, for any input sequence  $X^{(p)}$ ,*

(1) *the input sequences  $X_1^{(p)} + X^{(p)}$  and  $X_2^{(p)} + X^{(p)}$  are a fooling pair to balancer  $b$ ;*

(2)  $b(X_1^{(p)} + X^{(p)}) - b(X_1^{(p)}) = b(X_2^{(p)} + X^{(p)}) - b(X_2^{(p)})$ .

**Proof:** We first show (1). Clearly, by Lemma 2.1.1,

$$\text{state}_b(X_1^{(p)} + X^{(p)}) = (\text{state}_b(X_1^{(p)}) + \text{state}_b(X^{(p)})) \bmod q,$$

and

$$\text{state}_b(X_2^{(p)} + X^{(p)}) = (\text{state}_b(X_2^{(p)}) + \text{state}_b(X^{(p)})) \bmod q.$$

Since  $X_1^{(p)}$  and  $X_2^{(p)}$  are a fooling pair to  $b$ ,  $\text{state}_b(X_1^{(p)}) = \text{state}_b(X_2^{(p)})$ , it follows that

$$\text{state}_b(X_1^{(p)} + X^{(p)}) = \text{state}_b(X_2^{(p)} + X^{(p)});$$

thus,  $X_1^{(p)} + X^{(p)}$  and  $X_2^{(p)} + X^{(p)}$  are a fooling pair to  $b$ , as needed.

We continue to show (2). By Equation 1.1, for each  $i$ ,  $0 \leq i < q$ .

$$\begin{aligned}
& (b(X_1^{(p)} + X^{(p)}) - b(X_1^{(p)}))_i \\
&= \left\lceil \frac{\Sigma(X_1^{(p)} + X^{(p)}) - i}{q} \right\rceil - \left\lceil \frac{\Sigma(X_1^{(p)}) - i}{q} \right\rceil \\
&= \left\lceil \frac{\Sigma(X_1^{(p)}) + \Sigma(X^{(p)}) - i}{q} \right\rceil - \left\lceil \frac{\Sigma(X_1^{(p)}) - i}{q} \right\rceil \\
&= \left\lceil \frac{(\Sigma(X_1^{(p)}) \operatorname{div} q) \cdot q + (\Sigma(X_1^{(p)}) \bmod q) + \Sigma(X^{(p)}) - i}{q} \right\rceil \\
&\quad - \left\lceil \frac{(\Sigma(X_1^{(p)}) \operatorname{div} q) \cdot q + (\Sigma(X_1^{(p)}) \bmod q) - i}{q} \right\rceil \\
&= (\Sigma(X_1^{(p)}) \operatorname{div} q) + \left\lceil \frac{(\Sigma(X_1^{(p)}) \bmod q) + \Sigma(X^{(p)}) - i}{q} \right\rceil \\
&\quad - (\Sigma(X_1^{(p)}) \operatorname{div} q) - \left\lceil \frac{(\Sigma(X_1^{(p)}) \bmod q) - i}{q} \right\rceil \\
&= \left\lceil \frac{(\Sigma(X_1^{(p)}) \bmod q) + \Sigma(X^{(p)}) - i}{q} \right\rceil - \left\lceil \frac{(\Sigma(X_1^{(p)}) \bmod q) - i}{q} \right\rceil \\
&= \left\lceil \frac{\operatorname{state}_b(X_1^{(p)}) + \Sigma(X^{(p)}) - i}{q} \right\rceil - \left\lceil \frac{\operatorname{state}_b(X_1^{(p)}) - i}{q} \right\rceil \\
&\quad \text{(by definition of the state).}
\end{aligned}$$

Similarly, we can show that

$$(b(X_2^{(p)} + X^{(p)}) - b(X_2^{(p)}))_i = \left\lceil \frac{\operatorname{state}_b(X_2^{(p)}) + \Sigma(X^{(p)}) - i}{q} \right\rceil - \left\lceil \frac{\operatorname{state}_b(X_2^{(p)}) - i}{q} \right\rceil.$$

Since  $X_1^{(p)}$  and  $X_2^{(p)}$  are a fooling pair to  $b$ ,  $\operatorname{state}_b(X_1^{(p)}) = \operatorname{state}_b(X_2^{(p)})$ . It follows that

$$b(X_1^{(p)} + X^{(p)}) - b(X_1^{(p)}) = b(X_2^{(p)} + X^{(p)}) - b(X_2^{(p)}),$$

as needed. ■

**Proposition 2.1.3** *Consider a balancing network  $\mathcal{B}$ , of input width  $w$ . Take any input sequences  $X_1^{(w)}$  and  $X_2^{(w)}$  that are a fooling pair to network  $\mathcal{B}$ . Then, for any input sequence  $X^{(w)}$ ,*

- (1) *the input sequences  $X_1^{(w)} + X^{(w)}$  and  $X_2^{(w)} + X^{(w)}$  are a fooling pair to network  $\mathcal{B}$ ;*
- (2)  $\mathcal{B}(X_1^{(w)} + X^{(w)}) - \mathcal{B}(X_1^{(w)}) = \mathcal{B}(X_2^{(w)} + X^{(w)}) - \mathcal{B}(X_2^{(w)})$ .

**Proof:** Let  $d$  be the depth of network  $\mathcal{B}$ . The proof is by induction on the depth  $d$ . For the basis case, where  $d = 1$ ,  $\mathcal{B}$  is a single layer of balancers, and the claim follows immediately by applying Lemma 2.1.2 to each of the balancers in the layer.

Let's assume inductively that the claim holds for all networks of depth at most  $d - 1$ . For the induction step, let  $\mathcal{B} = \mathcal{B}'\mathcal{B}''$ , where  $\mathcal{B}'$ , and  $\mathcal{B}''$  are networks of depth at most  $d - 1$ ; that is,  $\mathcal{B}$  is the "cascade" of  $\mathcal{B}'$  and  $\mathcal{B}''$ .

For the induction step, we start by showing (1). Since the input sequences  $X_1^{(w)}$  and  $X_2^{(w)}$  are a fooling pair to  $\mathcal{B}$ , it follows that they are a fooling pair to  $\mathcal{B}'$ . Thus, by induction hypothesis (1) for  $\mathcal{B}'$ , for any input sequence  $X^{(w)}$ , the input sequences  $X_1^{(w)} + X^{(w)}$  and  $X_2^{(w)} + X^{(w)}$  are a fooling pair to  $\mathcal{B}'$ . It remains to show that the sequences  $\mathcal{B}'(X_1^{(w)} + X^{(w)})$  and  $\mathcal{B}'(X_2^{(w)} + X^{(w)})$  are a fooling pair to  $\mathcal{B}''$ .

By induction hypothesis (2) for  $\mathcal{B}'$ ,

$$\mathcal{B}'(X_1^{(w)} + X^{(w)}) = \mathcal{B}'(X_1^{(w)}) + \mathcal{B}'(X_2^{(w)} + X^{(w)}) - \mathcal{B}'(X_2^{(w)}),$$

and

$$\mathcal{B}'(X_2^{(w)} + X^{(w)}) = \mathcal{B}'(X_2^{(w)}) + \mathcal{B}'(X_1^{(w)} + X^{(w)}) - \mathcal{B}'(X_1^{(w)}).$$

Furthermore, by induction hypothesis (2) for  $\mathcal{B}'$ ,

$$\mathcal{B}'(X_2^{(w)} + X^{(w)}) - \mathcal{B}'(X_2^{(w)}) = \mathcal{B}'(X_1^{(w)} + X^{(w)}) - \mathcal{B}'(X_1^{(w)}),$$

while, by assumption,  $\mathcal{B}'(X_1^{(w)})$  and  $\mathcal{B}'(X_2^{(w)})$  are a fooling pair for  $\mathcal{B}''$ . We apply induction hypothesis (1) to  $\mathcal{B}''$ , taking  $\mathcal{B}'(X_1^{(w)})$  for  $X_1^{(w)}$ ,  $\mathcal{B}'(X_2^{(w)})$  for  $X_2^{(w)}$ , and  $\mathcal{B}'(X_2^{(w)} + X^{(w)}) - \mathcal{B}'(X_2^{(w)}) = \mathcal{B}'(X_1^{(w)} + X^{(w)}) - \mathcal{B}'(X_1^{(w)})$  for  $X^{(w)}$ ; it implies that the input sequences

$$\mathcal{B}'(X_1^{(w)}) + \mathcal{B}'(X_2^{(w)} + X^{(w)}) - \mathcal{B}'(X_2^{(w)}) = \mathcal{B}'(X_1^{(w)} + X^{(w)}),$$

and

$$\mathcal{B}'(X_2^{(w)}) + \mathcal{B}'(X_1^{(w)} + X^{(w)}) - \mathcal{B}'(X_1^{(w)}) = \mathcal{B}'(X_2^{(w)} + X^{(w)})$$

are a fooling pair to  $\mathcal{B}''$ , as needed.

We continue to show (2). Since the input sequences  $X_1^{(w)}$  and  $X_2^{(w)}$  are a fooling pair to  $\mathcal{B}$ , it follows that they are a fooling pair to  $\mathcal{B}'$ . So, by induction hypothesis (2) for  $\mathcal{B}'$ ,

$$\mathcal{B}'(X_1^{(w)} + X^{(w)}) = \mathcal{B}'(X_1^{(w)}) + \mathcal{B}'(X_2^{(w)} + X^{(w)}) - \mathcal{B}'(X_2^{(w)}).$$

Thus,

$$\begin{aligned}
& \mathcal{B}(X_1^{(w)} + X^{(w)}) - \mathcal{B}(X_1^{(w)}) \\
&= \mathcal{B}''(\mathcal{B}'(X_1^{(w)} + X^{(w)})) - \mathcal{B}''(\mathcal{B}'(X_1^{(w)})) \\
&= \mathcal{B}''(\mathcal{B}'(X_1^{(w)}) + \mathcal{B}'(X_2^{(w)} + X^{(w)}) - \mathcal{B}'(X_2^{(w)})) - \mathcal{B}''(\mathcal{B}'(X_1^{(w)})).
\end{aligned}$$

Since the input sequences  $X_1^{(w)}$  and  $X_2^{(w)}$  are a fooling pair to  $\mathcal{B}$ , it follows that the sequences  $\mathcal{B}'(X_1^{(w)})$  and  $\mathcal{B}'(X_2^{(w)})$  are a fooling pair to  $\mathcal{B}''$ . Thus, we apply induction hypothesis (2) for  $\mathcal{B}''$  by taking  $\mathcal{B}'(X_1^{(w)})$  for  $X_1^{(w)}$ ,  $\mathcal{B}'(X_2^{(w)})$  for  $X_2^{(w)}$ , and  $\mathcal{B}'(X_2^{(w)} + X^{(w)}) - \mathcal{B}'(X_2^{(w)})$  for  $X^{(w)}$ , to obtain that

$$\begin{aligned}
& \mathcal{B}''(\mathcal{B}'(X_1^{(w)}) + \mathcal{B}'(X_2^{(w)} + X^{(w)}) - \mathcal{B}'(X_2^{(w)})) - \mathcal{B}''(\mathcal{B}'(X_1^{(w)})) \\
&= \mathcal{B}''(\mathcal{B}'(X_2^{(w)}) + \mathcal{B}'(X_2^{(w)} + X^{(w)}) - \mathcal{B}'(X_2^{(w)})) - \mathcal{B}''(\mathcal{B}'(X_2^{(w)})) \\
&= \mathcal{B}''(\mathcal{B}'(X_2^{(w)} + X^{(w)})) - \mathcal{B}''(\mathcal{B}'(X_2^{(w)})) \\
&= \mathcal{B}(X_2^{(w)} + X^{(w)}) - \mathcal{B}(X_2^{(w)}).
\end{aligned}$$

It follows that  $\mathcal{B}(X_1^{(w)} + X^{(w)}) - \mathcal{B}(X_1^{(w)}) = \mathcal{B}(X_2^{(w)} + X^{(w)}) - \mathcal{B}(X_2^{(w)})$ , which completes the proof of (2). ■

#### 2.1.4 Null Sequences

For any balancing network  $\mathcal{B}$ , with input width  $w$ , we say that an input sequence  $X^{(w)}$  is a *null sequence* to network  $\mathcal{B}$  if the input sequences  $X^{(w)}$  and  $\mathbf{0}^{(w)}$  are a fooling pair to  $\mathcal{B}$ .

Intuitively, a null sequence leaves the network to its initial state. By the definition of the null sequence, if in a fooling pair one sequence is null then the other sequence must be null too.

Let's assume that the output width of network  $\mathcal{B}$  is  $t$ . Since  $\mathcal{B}(\mathbf{0}^{(w)}) = \mathbf{0}^{(t)}$ , from Proposition 2.1.3, by taking  $Z^{(w)}$  for  $X_1^{(w)}$  and  $\mathbf{0}^{(w)}$  for  $X_2^{(w)}$ , we obtain the following “linearity” corollary for a null sequence.

**Corollary 2.1.4** *Consider a balancing network  $\mathcal{B}$ , of input width  $w$ . Take any null input sequence  $Z^{(w)}$ . Then, for any input sequence  $X^{(w)}$ ,*

- (1) *the input sequences  $X^{(w)}$  and  $X^{(w)} + Z^{(w)}$  are a fooling pair to network  $\mathcal{B}$ ;*
- (2)  $\mathcal{B}(X^{(w)} + Z^{(w)}) = \mathcal{B}(X^{(w)}) + \mathcal{B}(Z^{(w)})$ .

We continue by showing various lemmas for null sequences.

**Lemma 2.1.5** *Consider a balancing network  $\mathcal{B}$  of input width  $w$ . Take any input sequence  $X^{(w)}$  that is null to  $\mathcal{B}$ . Then, for any integer  $k \geq 0$ ,*

- (1)  $kX^{(w)}$  is a null sequence to  $\mathcal{B}$ ;
- (2)  $\mathcal{B}(kX^{(w)}) = k\mathcal{B}(X^{(w)})$ .

**Proof:** We proceed by induction on  $k$ . For the basis case where  $k = 0$ , the claim holds trivially.

Let's assume inductively that the claim holds for all integers  $k'$  such that  $k' \leq k - 1$ . For the induction step consider the integer  $k$ .

We start by showing (1). Clearly,

$$kX^{(w)} = X^{(w)} + (k - 1)X^{(w)}.$$

By the induction hypothesis,  $(k - 1)X^{(w)}$  is a null sequence. We apply Corollary 2.1.4(1) by taking  $X^{(w)}$  for  $X^{(w)}$  and  $(k - 1)X^{(w)}$  for  $Z^{(w)}$ . We get that  $X^{(w)}$  and  $X^{(w)} + (k - 1)X^{(w)}$  are a fooling pair. Since,  $X^{(w)}$  is a null sequence,  $kX^{(w)}$  is a null sequence too, as needed.

We continue by showing (2). We apply Corollary 2.1.4(2) by taking  $X^{(w)}$  for  $X^{(w)}$  and  $(k - 1)X^{(w)}$  for  $Z^{(w)}$  to obtain

$$\begin{aligned} \mathcal{B}(kX^{(w)}) &= \mathcal{B}(X^{(w)} + (k - 1)X^{(w)}) \\ &= \mathcal{B}(X^{(w)}) + \mathcal{B}((k - 1)X^{(w)}) \\ &= \mathcal{B}(X^{(w)}) + (k - 1)\mathcal{B}(X^{(w)}) \\ &\quad \text{(by induction hypothesis)} \\ &= k\mathcal{B}(X^{(w)}), \end{aligned}$$

as needed. ■

For any balancing network  $\mathcal{B}$ , denote by  $P(\mathcal{B})$ , the product of the fan-outs of balancers of  $\mathcal{B}$ . The next claim establishes a sufficient condition involving  $P(\mathcal{B})$  for a sequence to be null to  $\mathcal{B}$ .

**Proposition 2.1.6** *Consider a balancing network  $\mathcal{B}$  of input width  $w$ . Take an input sequence  $X^{(w)}$  such that each of its elements is a non-negative multiple of  $P(\mathcal{B})$ . Then,  $X^{(w)}$  is a null sequence to  $\mathcal{B}$ .*



**Proof:** We prove the claim by induction on the depth  $d$  of  $\mathcal{B}$ . For the basis case where  $d = 1$ ,  $\mathcal{B}$  is a single layer. Take any single  $(p, q)$ -balancer  $b$  of the layer. By definition,  $P(\mathcal{B})$  is a non-negative multiple of  $q$ . It follows that the restriction of input sequence  $X^{(w)}$  to balancer  $b$  is a null sequence to  $b$ . Since  $b$  was chosen arbitrarily, this implies that  $X^{(w)}$  is a null sequence to the layer  $\mathcal{B}$ , as needed.

Let's assume inductively that the claim holds for all balancing networks of depth at most  $d - 1$ . Write  $\mathcal{B} = \mathcal{B}'\mathcal{B}''$ , where each of  $\mathcal{B}'$  and  $\mathcal{B}''$  is a balancing network of depth at most  $d - 1$ .

We only need to show that  $X^{(w)}$  is a null sequence to network  $\mathcal{B}'$ , and  $\mathcal{B}'(X^{(w)})$  is a null sequence to network  $\mathcal{B}''$ .

By definition,  $P(\mathcal{B}) = P(\mathcal{B}')P(\mathcal{B}'')$ . Since, each element of  $X^{(w)}$  is a non-negative multiple of  $P(\mathcal{B}')$ , by induction hypothesis for network  $\mathcal{B}'$ , we have that  $X^{(w)}$  is a null sequence to network  $\mathcal{B}'$ .

It remains to show that  $\mathcal{B}'(X^{(w)})$  is a null sequence to network  $\mathcal{B}''$ . The sequence  $X^{(w)}$  can be rewritten as the product  $X^{(w)} = P(\mathcal{B}'')G^{(w)}$ , where each element of the sequence  $G^{(w)}$  is a non-negative multiple of  $P(\mathcal{B}')$ . By induction hypothesis for network  $\mathcal{B}'$ , we have that  $G^{(w)}$  is a null sequence to network  $\mathcal{B}'$ . By Lemma 2.1.5(2) we have

$$\mathcal{B}'(X^{(w)}) = \mathcal{B}'(P(\mathcal{B}'')G^{(w)}) = P(\mathcal{B}'')\mathcal{B}'(G^{(w)}).$$

Therefore, each element of  $\mathcal{B}'(X^{(w)})$  is a non-negative multiple of  $P(\mathcal{B}'')$ . Thus, by induction hypothesis for network  $\mathcal{B}''$ ,  $\mathcal{B}'(X^{(w)})$  is a null sequence to network  $\mathcal{B}''$ , which completes the proof.  $\blacksquare$

## 2.2 Main Result

In this section, we present a necessary condition of properties of balancing networks that are preserved under the introduction of antitokens.

A property  $\Pi$  is a predicate on finite integer sequences. For a sequence  $X$  we denote  $X \in \Pi$  if  $X$  satisfies property  $\Pi$ . For example,  $\Pi$  could be either the step property, the  $k$ -smoothing property, or the threshold property.

For any balancing network  $\mathcal{B}$  of input width  $w$ , we say that  $\mathcal{B}$  *satisfies property  $\Pi$  on input sequence  $X^{(w)}$* , if simply  $\mathcal{B}(X^{(w)}) \in \Pi$ . Namely, a network satisfies a property on an input sequence if the corresponding output sequence satisfies that property.

We give the following characterization for properties.

**Definition 2.2.1** Consider a balancing network  $\mathcal{B}$  of input width  $w$ , and a property  $\Pi$ . We say that a property  $\Pi$  is closed under the nullity of network  $\mathcal{B}$ , if for any non-negative input sequence  $X^{(w)}$ , and for any non-negative null input sequence  $Z^{(w)}$  to  $\mathcal{B}$ , it holds that whenever  $\mathcal{B}(X^{(w)}) \in \Pi$  then  $\mathcal{B}(X^{(w)}) \pm \mathcal{B}(Z^{(w)}) \in \Pi$ .

In the next claim, we will establish that if a network satisfies a property when it is accessed by tokens only then it will also satisfy the same property when it is accessed by both tokens and antitokens. When a network is accessed by tokens only then all the possible input sequences are non-negative. If a network is accessed by both tokens and antitokens an input sequence may contain entries which are negative (in case the algebraic sum of tokens and antitokens on the corresponding wires is negative). Therefore, if a network satisfies a property on all non-negative input sequences then we will prove that this network will satisfy the same property on any input sequence (that may contain negative entries). This claim will hold for any property that is closed under the nullity of the network. (Notice that Definition 2.2.1, has to do only with non-negative input sequences.)

**Theorem 2.2.1** Consider any balancing network  $\mathcal{B}$  that satisfies a property  $\Pi$  on all non-negative input sequences. If  $\Pi$  is closed under the nullity of  $\mathcal{B}$ , then network  $\mathcal{B}$  satisfies property  $\Pi$  on any input sequence (that may contain negative entries).

**Proof:** Consider any arbitrary input sequence  $X^{(w)}$  (with either positive or negative entries). We will show that  $\mathcal{B}(X^{(w)}) \in \Pi$ .

Construct from  $X^{(w)}$  an non-negative input sequence  $Z^{(w)}$  such that for each index  $i$ ,  $0 \leq i < w$ ,  $z_i$  is the least non-negative multiple of  $P(\mathcal{B})$  so that  $0 \leq x_i + z_i$ .

Obviously,  $X^{(w)} + Z^{(w)}$  is a non-negative input sequence, and thus

$$\mathcal{B}(X^{(w)} + Z^{(w)}) \in \Pi.$$

Since each element of  $Z^{(w)}$  is a non-negative multiple of  $P(\mathcal{B})$ , it follows by Proposition 2.1.6, that  $Z^{(w)}$  is a null sequence of  $\mathcal{B}$ . Furthermore,  $Z^{(w)}$  is a non-negative null input sequence, since all the entries in this sequence are non-negative.

Since the property  $\Pi$  is closed under the nullity of  $\mathcal{B}$ , we have by Definition 2.2.1 that

$$\mathcal{B}(X^{(w)} + Z^{(w)}) - \mathcal{B}(Z^{(w)}) \in \Pi.$$

Since  $Z^{(w)}$  is a null sequence, from Lemma 2.1.4(2) we have

$$\mathcal{B}(X^{(w)}) = \mathcal{B}(X^{(w)} + Z^{(w)}) - \mathcal{B}(Z^{(w)}).$$

Subsequently,

$$\mathcal{B}(X^{(w)}) \in \Pi,$$

as needed. ■

## 2.3 Applications

In this section we consider most of the known kinds of balancing networks, namely, counting networks, smoothing networks, and threshold networks. We prove that whenever such a balancing network satisfies a property with tokens alone, it will still satisfy the same property for both tokens and antitokens.

The respective properties satisfied by the counting, smoothing, and threshold networks, when these networks are accessed by tokens alone, are the step property,  $k$ -smoothing property, and threshold property. In order to show that the above properties are preserved with the introduction of antitokens, we use theorem 2.2.1. According to this theorem, we only need to show that all of the above properties are closed under the nullity of the respective balancing networks.

We proceed as follows. We first show that the step property and the  $k$ -smoothing property are closed under the nullity of a network. In fact, we show the stronger result that a broader class of properties, which we call *boundedness properties*, are closed under the nullity of a network. Then we continue by showing that the threshold property is closed under the nullity of a network.

### 2.3.1 Boundedness Properties

A  $k$ -boundedness property, for some integer  $k \geq 1$ , is any property that is  $k$ -smooth, and is also closed under the addition (or subtraction) with a constant sequence.

Clearly, there are infinitely many boundedness properties. Obviously, the  $k$ -smoothing property is a  $k$ -boundedness property.

For step sequences we have the following observation.

**Observation 2.3.1** *The addition (or subtraction) of a step sequence with a constant sequence gives a step sequence.*

The step property is a 1-boundedness property, since any step sequence is 1-smooth, and, by Observation 2.3.1, the step property is closed under the addition (or subtraction) with a constant sequence.

Now, we show that if a balancing network satisfies a boundedness property, then for any non-negative null input sequence the corresponding output sequence is constant.

**Lemma 2.3.1** *Consider a balancing network  $\mathcal{B}$  of input width  $w$ , that satisfies a  $k$ -boundedness property  $\Pi$  on all non-negative input sequences. Then, for any non-negative null input sequence  $Z^{(w)}$  the corresponding output sequence  $\mathcal{B}(Z^{(w)})$  is constant.*

**Proof:** Let's assume that the output width of  $\mathcal{B}$  is  $t$ . Consider the non-negative null input sequence  $Z^{(w)}$  to  $\mathcal{B}$ . We will show that the sequence  $\mathcal{B}(Z^{(w)})$  is constant.

Let  $Y^{(t)} = \mathcal{B}(Z^{(w)})$ . Let's assume, by way of contradiction, that  $Y^{(t)}$  is not a constant sequence. Then, there exist entries  $y_i$  and  $y_j$  of  $Y^{(t)}$  such that  $y_i \neq y_j$ . Thus, clearly,  $|y_j - y_i| \geq 1$ .

Since  $Z^{(w)}$  is a null sequence to network  $\mathcal{B}$ , it follows by Proposition 2.1.5(2),

$$\mathcal{B}((k+1)Z^{(w)}) = (k+1)\mathcal{B}(Z^{(w)}) = (k+1)Y^{(t)}.$$

Since the input sequence  $(k+1)Z^{(w)}$  is a non-negative input sequence, the output sequence  $(k+1)Y^{(t)}$  satisfies the  $k$ -boundedness property  $\Pi$ . Subsequently, the sequence  $(k+1)Y^{(t)}$  satisfies the  $k$ -smoothing property. The  $i$ -th entry of sequence  $(k+1)Y^{(t)}$  is equal to  $(k+1)y_i$  and the  $j$ -th entry is equal to  $(k+1)y_j$ . By the  $k$ -smoothing property we have

$$|(k+1)y_i - (k+1)y_j| \leq k,$$

and thus

$$(k+1)|y_i - y_j| \leq k,$$

which implies that  $|y_i - y_j| = 0$ , a contradiction. Therefore,  $Y^{(t)}$  is a constant sequence, as needed. ■

We are now ready to show that any  $k$ -boundedness property is closed under the nullity of a network.

**Theorem 2.3.2** *If a network  $\mathcal{B}$  satisfies a  $k$ -boundedness property  $\Pi$  on all non-negative input sequences, then  $\Pi$  is closed under the nullity of network  $\mathcal{B}$ .*

**Proof:** Let's assume that the input width of  $\mathcal{B}$  is  $w$ . Let  $X^{(w)}$  be any non-negative input sequence. Clearly,  $\mathcal{B}(X^{(w)}) \in \Pi$ . Let  $Z^{(w)}$  be any non-negative null input sequence to  $\mathcal{B}$ . By Lemma 2.3.1, we have that  $\mathcal{B}(Z^{(w)})$  is a constant sequence. Since the property  $\Pi$  is a

$k$ -boundedness property, the property  $\Pi$  is closed under the addition (or subtraction) with a constant vector and we immediately have

$$\mathcal{B}(X^{(w)}) \pm \mathcal{B}(Z^{(w)}) \in \Pi.$$

Therefore, by Definition 2.2.1, the property  $\Pi$  is closed under the nullity of network  $\mathcal{B}$ , as needed. ■

Subsequently, if we know that a network satisfies a boundedness property for tokens only, then we can immediately infer that it will satisfy the same property even when antitokens are introduced. Therefore, any network that we know it is a smoothing or counting network for tokens only, then this network will remain smoothing or counting with both tokens and antitokens.

### 2.3.2 Threshold Property

We will show that the threshold property is closed under the nullity of a balancing network. The threshold property is not a boundedness property because any threshold sequence does not necessarily satisfy the  $k$ -smoothing property, for any  $k \geq 1$ . Therefore, we cannot apply the result of the previous section.

To show the desired result we use similar techniques to the previous section. We use a special kind of sequences called *saturated sequences*. A saturated sequence  $X^{(w)}$  is any sequence such that

$$x_{w-1} = \frac{\Sigma(X^{(w)})}{w}.$$

Remember that  $x_{w-1}$  must be an integer all the times. Clearly, any saturated vector is a threshold vector, but not vice versa.

We have the following lemma that combines a threshold and a saturated sequence.

**Lemma 2.3.3** *If  $X^{(w)}$  is a threshold sequence and  $Y^{(w)}$  is a saturated sequence, then the sequence  $X^{(w)} \pm Y^{(w)}$  is threshold.*

**Proof:** Since  $X^{(w)}$  is threshold, we have

$$x_{w-1} = \left\lfloor \frac{\Sigma(X^{(w)})}{w} \right\rfloor.$$

Since  $Y^{(w)}$  is saturated, we have

$$y_{w-1} = \frac{\Sigma(Y^{(w)})}{w}.$$

Adding the two above terms, and since  $y_{w-1}$  is integer, we get

$$\begin{aligned} x_{w-1} \pm y_{w-1} &= \left\lfloor \frac{\Sigma(X^{(w)})}{w} \right\rfloor \pm \frac{\Sigma(Y^{(w)})}{w} \\ &= \left\lfloor \frac{\Sigma(X^{(w)})}{w} \pm \frac{\Sigma(Y^{(w)})}{w} \right\rfloor \\ &= \left\lfloor \frac{\Sigma(X^{(w)} \pm Y^{(w)})}{w} \right\rfloor, \end{aligned}$$

as needed. ■

Now, we show that for any threshold network and for any non-negative null input sequence the corresponding output sequence is a saturated sequence.

**Lemma 2.3.4** *Consider a balancing network  $\mathcal{B}$  of input width  $w$  that satisfies the threshold property on all non-negative input sequences. Then, for any non-negative null input sequence  $Z^{(w)}$ , the output sequence  $\mathcal{B}(Z^{(w)})$  is saturated.*

**Proof:** Let's assume that the output width of  $\mathcal{B}$  is  $t$ . Consider the non-negative null sequence  $Z^{(w)}$ . We will show that  $\mathcal{B}(Z^{(w)})$  is saturated.

Let  $Y^{(t)} = \mathcal{B}(Z^{(w)})$ . Since  $Z^{(w)}$  is a null sequence to network  $\mathcal{B}$ , it follows by Proposition 2.1.5(2),

$$\mathcal{B}(tZ^{(w)}) = t\mathcal{B}(Z^{(w)}) = tY^{(t)}.$$

Clearly, the sequence  $tZ^{(w)}$  is non-negative and thus the sequence  $tY^{(t)}$  satisfies the threshold property. The  $(t-1)$ th entry of sequence  $tY^{(t)}$  is equal to  $ty_{t-1}$ . By the threshold property of  $tY^{(t)}$  we have

$$ty_{t-1} = \left\lfloor \frac{\Sigma(tY^{(t)})}{t} \right\rfloor = \left\lfloor \frac{t\Sigma(Y^{(t)})}{t} \right\rfloor = \Sigma(Y^{(t)}).$$

and subsequently  $y_{t-1} = \Sigma(Y^{(t)})/t$ . Therefore,  $Y^{(t)}$  is saturated, as needed. ■

We are now ready to show that the threshold property is closed under the nullity of a network.

**Theorem 2.3.5** *If a network  $\mathcal{B}$  satisfies the threshold property on all non-negative input sequences, then the threshold property is closed under the nullity of network  $\mathcal{B}$ .*

**Proof:** Let's assume that the input width of  $\mathcal{B}$  is  $w$ . Let  $X^{(w)}$  be any non-negative input sequence. Clearly,  $\mathcal{B}(X^{(w)})$  is a threshold sequence. Let  $Z^{(w)}$  be any non-negative null

sequence to  $\mathcal{B}$ . By Lemma 2.3.4, we have that  $\mathcal{B}(Z^{(w)})$  is a saturated sequence. Furthermore, since  $\mathcal{B}(X^{(w)})$  is threshold and  $\mathcal{B}(Z^{(w)})$  is saturated, by Lemma 2.3.3, we have that the sequence

$$\mathcal{B}(X^{(w)}) \pm \mathcal{B}(Z^{(w)})$$

is threshold. By Definition 2.2.1, we have that the threshold property is closed under the nullity of network  $\mathcal{B}$ , as needed. ■

Subsequently, if we know that a network is threshold for tokens only, then we can immediately infer that it will remain a threshold network even when antitokens are introduced.

## 2.4 Discussion

We have shown that any balancing network that satisfies some property on all non-negative input sequences, then the network will also satisfy the same property for any arbitrary input sequence (with either negative or positive entries), if this property is closed under the nullity of the network. Interesting examples of such properties are the step property, the  $k$ -smoothing property, and the threshold property. A significant consequence of our result is that for all known (deterministic) constructions of counting, smoothing, and threshold networks (see [2, 4, 6, 17, 23, 28, 32, 38, 39, 49], and the constructions of Chapters 3 and 4) which already are proven to work correctly for tokens only, they will work correctly with both tokens and antitokens.

Aiello *et al.* [4] present an interesting construction of a *randomized* counting network; they use *randomized balancers*, which distribute tokens on output wires according to some random permutation. Does this network operate correctly when it is simultaneously traversed by both tokens and antitokens? It seems that the randomized balancers need to somehow “remember” the entire history of the random permutations in order for antitokens to trace back the paths of tokens.

## Chapter 3

# Counting Networks of Arbitrary Width

The design of a counting network is a trade-off between balancer width and network depth. Wider balancers produce shallower networks (with smaller depth) where the contention-related delay may increase as tokens queue up in the same balancer. Deeper networks produce more latency.

We present a new counting network construction that illuminates how network width, depth, and balancer widths can be traded off in counting networks.<sup>1</sup> Our network has the same input and output width, that we simply call the network's width, and it is built from balancers with the same input and output width, that we simply call the balancer's width.

More specifically, we present the first network construction of arbitrary width  $w$  that requires both small depth and has small constant factors in the depth expression. In detail, let  $w$  be the product  $w = p_0 \cdots p_{n-1}$ , whose factors are not necessarily prime. We construct a network of width  $w$  and depth  $O(n^2) = O(\log^2 w)$ , using balancers of width at most  $\max(p_i)$ . This construction is practical in the sense that the asymptotic notation does not hide any large constants. As an example, our counting network for width  $15 = 3 \times 5$  is depicted in Figure 3.1, built from balancers of widths 3 and 5.

An interesting aspect of this construction is that it establishes a family of counting networks of width  $w$ , one family for each distinct factorization of  $w$ . A factorization in which  $\max(p_i)$  is large and  $n$  is small yields a network that trades small depth for large balancers, and a factorization where  $\max(p_i)$  is small and  $n$  is large makes the opposite trade-off. For example, consider the width  $w = 60$ . We have a factorization  $w = 2 \cdot 2 \cdot 3 \cdot 5$  which gives

---

<sup>1</sup>The work of this chapter was first presented in [18].



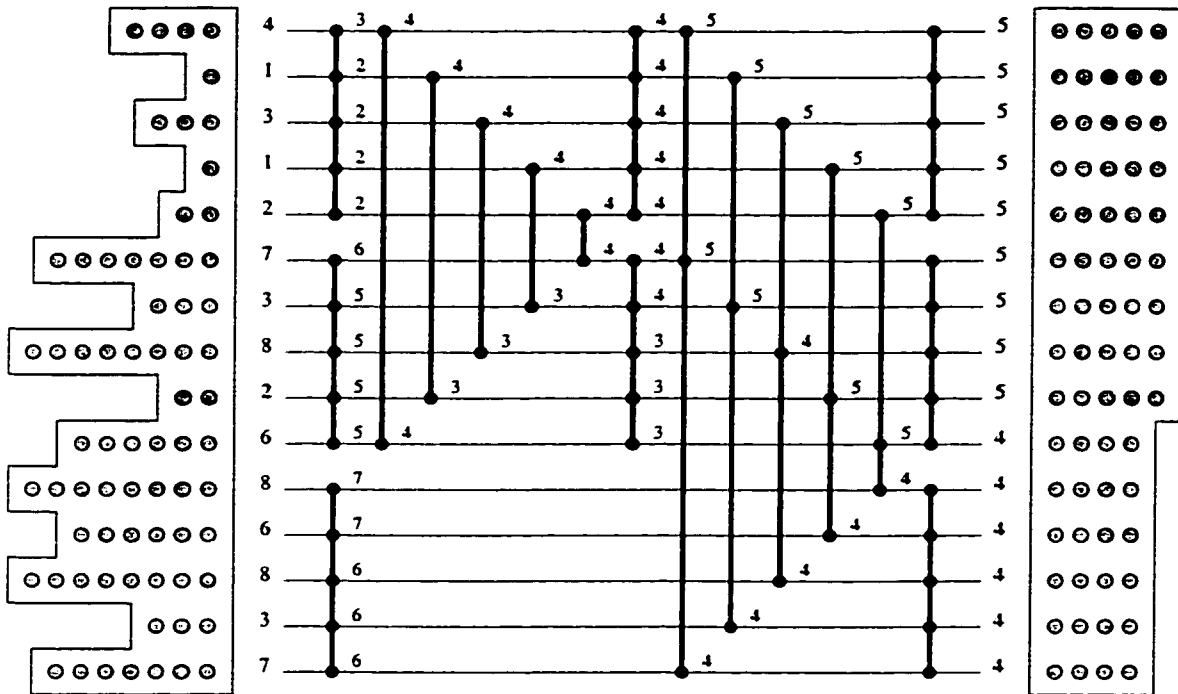


Figure 3.1: Counting network of width  $15 = 3 \times 5$

a network with depth 105 and uses balancers of maximum width 5. Another factorization is  $w = 4 \cdot 15$  which gives a network with depth 28 and uses balancers of maximum width 15. This flexibility may be useful in practice, since experimental evidence [28] suggests that for shared-memory implementations of counting networks, optimal performance for a fixed  $w$  is achieved by balancers of intermediate width. (Each distinct ordering of a fixed set of factors also yields a different counting network, but all such networks have the same depth.)

We proceed as follows. In Section 3.1 we present necessary preliminaries for our constructions. In Section 3.2 we give a top-down description of the framework of a counting network construction. Using this description, we present our counting network construction in Section 3.3. In Section 3.4 we present related work and we conclude in Section 3.5 by giving a discussion. During our presentation, we focus on the modular decomposition of the network. Where alternative constructions exist, we focus on the simplest, adding descriptions of more complicated optimizations. Readers are encouraged to consult the illustrations.

## 3.1 Preliminaries

### 3.1.1 Sequences

For any sequence  $X$ , the subsequence  $X[i, p]$  is the sequence  $x_i, x_{i+p}, x_{i+2p}, \dots$ . By Observation 1.4.1, if  $X$  has the step property then all the sequences  $X[i, p]$  have the step property too.

For any step sequence  $X$ , the step point is defined as in Section 1.4, where now if all  $x_i$  are equal then the step point is equal to 0.

In any sequence  $X$  we say that there is a *transition* between two consecutive elements  $x_i$  and  $x_{i+1}$  if their values are different. We say that a sequence  $X$  has the *bitonic property* whenever it is 1-smooth and has at most two transitions.

We say that the sequences  $X_0, \dots, X_{p-1}$  have the *k-staircase property* whenever

$$0 \leq \Sigma(X_i) - \Sigma(X_j) \leq k, \text{ for all } 0 \leq i < j < w.$$

It is often convenient to express a sequence  $X$  of length  $rc$  as an  $r \times c$  matrix. There are four ways to arrange the elements of  $X$  in a matrix, as shown by the following table.

$x_i$ goes to	row	column
row major	$\lfloor \frac{i}{c} \rfloor$	$i \bmod c$
reverse row major	$r - \lfloor \frac{i}{c} \rfloor - 1$	$c - (i \bmod c) - 1$
column major	$i \bmod r$	$\lfloor \frac{i}{r} \rfloor$
reverse col. major	$r - (i \bmod r) - 1$	$c - \lfloor \frac{i}{r} \rfloor - 1$

As an example, these arrangements are illustrated in Figure 3.2, for a sequence that has the step property. In all figures, the dark region labeled “1” represents the subsequence of higher values, and the light region labeled “0” the lower values.

### 3.1.2 Balancing Networks

For simplicity, we denote a  $(p, p)$ -balancer as a  $p$ -balancer, for any  $p$ .

We consider the following balancing network families.

- A *counting network*  $\mathcal{C}(p_0, \dots, p_{n-1})$  has input and output sequence of length  $w = p_0 \cdots p_{n-1}$ . The output sequence has the step property.
- A *merger network*  $\mathcal{M}(p_0, \dots, p_{n-1})$  has input sequences  $X_0, \dots, X_{p_{n-1}-1}$ , where each  $|X_i| = p_0 \cdots p_{n-2}$ , and output sequence of length  $p_0 \cdots p_{n-1}$ . If each  $X_i$  satisfies the step property, so does the output sequence.

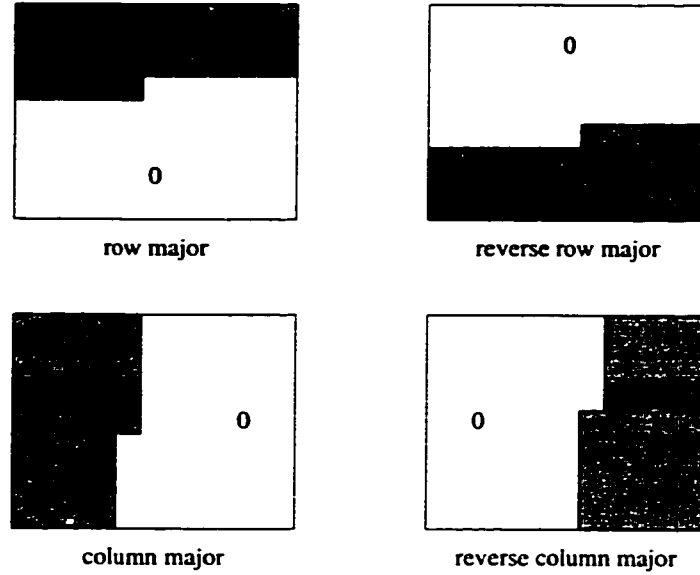


Figure 3.2: Matrix arrangements

- A *staircase-merger* network  $\mathcal{S}(r, p, q)$  has input sequences  $X_0, \dots, X_{q-1}$ , where each  $|X_i| = rp$ , and output sequence of length  $rpq$ . If each  $X_i$  satisfies the step property, and the sequences  $X_0, \dots, X_{q-1}$  satisfy the  $p$ -staircase property, then the output sequence satisfies the step property.
- A *two-merger* network  $\mathcal{T}(p, q_0, q_1)$  has input sequences  $X_0$  and  $X_1$ , where  $|X_0| = pq_0$  and  $|X_1| = pq_1$ , and output sequence of length  $p(q_0 + q_1)$ . If  $X_0$  and  $X_1$  each satisfies the step property, so does the output sequence.
- A *bitonic-converter* network  $\mathcal{D}(p, q)$  has input and output sequence of length  $pq$ . If the input sequence satisfies the bitonic property then the output sequence satisfies the step property.

We use  $\mathcal{C}$  to refer to the family  $\mathcal{C}(p_0, \dots, p_{n-1})$ , when the exact values of the  $p_i$  are unimportant, and similarly for the other balancing network families.

### 3.2 A Counting Network Construction Framework

Let  $w = p_0 \cdots p_{n-1}$ , and  $w_i = p_0 \cdots p_i$ , for  $0 \leq i < n$ , where  $p_i \geq 2$  and  $n \geq 2$ .

In this section, we give the construction of a counting network  $\mathcal{C}(p_0, \dots, p_{n-1})$  of width  $w$ . This network will serve as the framework for the desired counting network construction

$\mathcal{L}$ . presented in Section 3.3.3, which has width  $w$  and is built from balancers of width at most  $\max(p_i)$ .

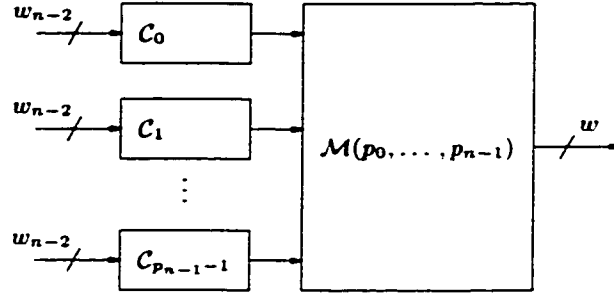
The construction of the network  $\mathcal{C}$  uses the same principles as the inductive construction of the bitonic network presented in Section 1.5.4. In particular, the counting network  $\mathcal{C}$  is built inductively by using smaller counting networks. Then, the outputs of the smaller networks are merged by using a merger network  $\mathcal{M}$ , as shown in Figure 3.3. The merger network is also built recursively, as shown in Figure 3.4. In the induction of the merger network, we use a staircase-merger network  $\mathcal{S}$ . (The staircase-merger network  $\mathcal{S}$ , corresponds to the “final merging” network in the construction of the bitonic network in Section 1.5.4.)

Since the construction of network  $\mathcal{C}$  is inductive, we need an induction basis case. In this section, we will not provide the basis case of the construction, and we will assume for the moment that we are given the network  $\mathcal{C}(p_i, p_j)$  of the basis case. We will also assume that the depth of network  $\mathcal{C}(p_i, p_j)$  is fixed and at most  $d$ .

By unfolding the inductive construction of  $\mathcal{C}$ , we show in Proposition 3.2.1, that the depth of the network  $\mathcal{C}$  depends on two parameters, the depth  $d$ , and the  $\text{depth}(\mathcal{S})$ , so that  $\text{depth}(\mathcal{C}) = O(nd + n^2 \cdot \text{depth}(\mathcal{S}))$ . Using the network  $\mathcal{C}(p_i, p_j)$ , we construct in Section 3.2.3 the staircase-merger  $\mathcal{S}$ , with depth that depends on  $d$ , so that  $\text{depth}(\mathcal{S}) = O(d)$ . Subsequently, we get  $\text{depth}(\mathcal{C}) = O(n^2d)$ .

Our goal is to obtain a counting network with depth  $O(n^2)$ . In order to achieve this, we must construct a network  $\mathcal{C}(p, q)$  which has depth  $d$  equal to a constant, since  $\text{depth}(\mathcal{C}) = O(n^2d)$ . In the same time, we want the network  $\mathcal{C}(p_i, p_j)$  to be constructed from balancers of width at most  $\max(p_i, p_j)$ , so that the whole network construction is made from balancers of width at most  $\max(p_i)$ .

To achieve our goal, we do the following in Section 3.3. In Section 3.3.1, we substitute in  $\mathcal{C}$  every instance of the network  $\mathcal{C}(p_i, p_j)$  with a single  $p_i \cdot p_j$ -balancer, which trivially has depth  $d = 1$ . and we obtain the counting network  $\mathcal{K}$  of width  $w$  and depth  $O(n^2)$ , made from balancers of width at most  $\max(p_i \cdot p_j)$ . Using the network  $\mathcal{K}$ , we build in Section 3.3.2, the novel counting network  $\mathcal{R}(p_i, p_j)$ , which has constant depth and is made from balancers of width at most  $\max(p_i, p_j)$ . When we substitute in  $\mathcal{C}$  every instance of the network  $\mathcal{C}(p_i, p_j)$  with the  $\mathcal{R}(p_i, p_j)$  network, which has constant depth, we obtain in Section 3.3.3, the desired counting network  $\mathcal{L}$  of width  $w$  and depth  $O(n^2)$ , made from balancers of width at most  $\max(p_i)$ .

Figure 3.3: Construction of counting network  $\mathcal{C}$ 

### 3.2.1 A Counting Network

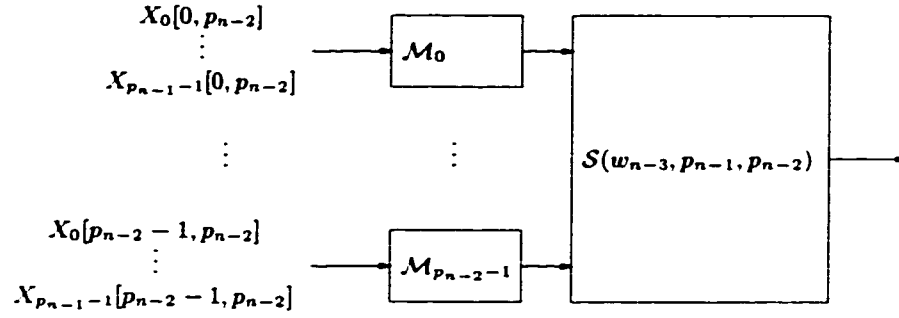
For the construction of  $\mathcal{C}(p_0, \dots, p_{n-1})$ , we argue by induction on  $n$ , the length of the factorization. For the base case, where  $n = 2$ , the network  $\mathcal{C}(p_0, p_1)$  is given by assumption.

Let  $n > 2$ , and  $\mathcal{C}(p_0, \dots, p_{n-2})$  the counting network guaranteed by the induction hypothesis. Our construction relies on the merger network  $\mathcal{M}(p_0, \dots, p_{n-1})$  constructed below in Section 3.2.2 (see Figure 3.3). Take  $p_{n-1}$  copies of  $\mathcal{C}(p_0, \dots, p_{n-2})$ , denoted  $\mathcal{C}_0, \dots, \mathcal{C}_{p_{n-1}-1}$ . Split the input sequence  $X$  of length  $w$  into subsequences  $X_0, \dots, X_{p_{n-1}-1}$ , each of length  $w_{n-2}$ . Direct each  $X_i$  to  $\mathcal{C}_i$ , and let  $Y_i$  be the corresponding output sequence. Each  $Y_i$  has the step property. Direct the  $Y_0, \dots, Y_{p_{n-1}-1}$  to  $\mathcal{M}(p_0, \dots, p_{n-1})$ . The resulting output has the step property.

Next, we compute the depth of  $\mathcal{C}$  in terms of the depth of the staircase-merger  $\mathcal{S}$ , presented in Section 3.2.3, and the depth  $d$  of  $\mathcal{C}(p_0, p_1)$ .

#### Proposition 3.2.1

$$\text{depth}(\mathcal{C}(p_0, \dots, p_{n-1})) = (n-1)d + \left(\frac{n^2}{2} - \frac{3n}{2} + 1\right) \cdot \text{depth}(\mathcal{S}).$$

Figure 3.4: Construction of merger network  $\mathcal{M}$ 

**Proof:** From the inductive construction of  $\mathcal{C}(p_0, \dots, p_{n-1})$  we have:

$$\begin{aligned}
& \text{depth}(\mathcal{C}(p_0, \dots, p_{n-1})) \\
&= \text{depth}(\mathcal{C}(p_0, \dots, p_{n-2})) + \text{depth}(\mathcal{M}(p_0, \dots, p_{n-1})) \\
&= \text{depth}(\mathcal{C}(p_0, \dots, p_{n-3})) + \text{depth}(\mathcal{M}(p_0, \dots, p_{n-2})) \\
&\quad + \text{depth}(\mathcal{M}(p_0, \dots, p_{n-1})) \\
&= \dots \\
&= \text{depth}(\mathcal{C}(p_0, p_1)) + \text{depth}(\mathcal{M}(p_0, p_1, p_2)) + \dots \\
&\quad + \text{depth}(\mathcal{M}(p_0, \dots, p_{n-1})) \\
&= d + (d + (3 - 2) \cdot \text{depth}(\mathcal{S})) + \dots \\
&\quad + (d + (n - 2) \cdot \text{depth}(\mathcal{S})) \\
&\quad \text{(by Proposition 3.2.3)} \\
&= (n - 1)d + ((3 + \dots + n) - 2(n - 2)) \cdot \text{depth}(\mathcal{S}) \\
&= (n - 1)d + \left( \left( \frac{n(n+1)}{2} - 3 \right) - 2(n - 2) \right) \cdot \text{depth}(\mathcal{S}) \\
&= (n - 1)d + \left( \frac{n^2}{2} - \frac{3n}{2} + 1 \right) \cdot \text{depth}(\mathcal{S}).
\end{aligned}$$

■

### 3.2.2 A Merger Network

We now show how to construct the merger network  $\mathcal{M}(p_0, \dots, p_{n-1})$ . This construction relies on the staircase-merger network  $\mathcal{S}$  constructed below in Section 3.2.3.

We argue by induction on  $n$ . For the base case, where  $n = 2$ , the network  $\mathcal{M}(p_0, p_{n-1})$  is the network  $\mathcal{C}(p_0, p_{n-1})$  (given by assumption).

Let  $X_0, \dots, X_{p_{n-1}-1}$  be the input sequences and let's assume that each satisfies the step property. Using the induction hypothesis, construct the merger network  $\mathcal{M}(p_0, \dots, p_{n-3}, p_{n-1})$ . Take  $p_{n-2}$  copies of this network, denoted  $\mathcal{M}_0, \dots, \mathcal{M}_{p_{n-2}-1}$ . Each  $\mathcal{M}_i$  has  $p_{n-1}$  input sequences  $X_0[i, p_{n-2}], \dots, X_{p_{n-1}-1}[i, p_{n-2}]$ . Each of these sequences satisfies the step property since it is a subsequence of a sequence that satisfies the step property. Denote the output sequence of  $\mathcal{M}_i$  by  $Y_i$ . Each  $Y_i$  has the step property. Now direct each  $Y_i$  to the staircase-merger  $\mathcal{S}(w_{n-3}, p_{n-1}, p_{n-2})$ . The final output sequence satisfies the step property. See Figure 3.4.

For the correctness of network  $\mathcal{M}$  we need only show that the input sequences to the staircase-merger  $\mathcal{S}$  satisfy the  $p_{n-1}$ -staircase property.

**Proposition 3.2.2** *The sequences  $Y_i$ , for  $0 \leq i < p_{n-2}$ , satisfy the  $p_{n-1}$ -staircase property.*

**Proof:** Since each  $X_i$  has the step property, for  $0 \leq j < k < p_{n-2}$ ,

$$0 \leq \Sigma(X_i[j, p_{n-2}]) - \Sigma(X_i[k, p_{n-2}]) \leq 1.$$

By construction,

$$\Sigma(Y_i) = \Sigma(X_0[i, p_{n-2}]) + \dots + \Sigma(X_{p_{n-1}-1}[i, p_{n-2}]).$$

It follows that for  $0 \leq i < j < p_{n-2}$

$$\begin{aligned} \Sigma(Y_i) - \Sigma(Y_j) &= \Sigma(X_0[i, p_{n-2}]) - \Sigma(X_0[j, p_{n-2}]) + \dots \\ &\quad + \Sigma(X_{p_{n-1}-1}[i, p_{n-2}]) - \Sigma(X_{p_{n-1}-1}[j, p_{n-2}]) \\ &\leq p_{n-1}. \end{aligned}$$

Similarly,  $\Sigma(Y_i) - \Sigma(Y_j) \geq 0$ . Subsequently, the  $Y_i$  satisfy the  $p_{n-1}$ -staircase property, as needed. ■

Next, we compute the depth of  $\mathcal{M}$  in terms of the depth of the staircase-merger  $\mathcal{S}$ , and the depth  $d$  of  $\mathcal{C}(p_0, p_{n-1})$ .

**Proposition 3.2.3**  $depth(\mathcal{M}(p_0, \dots, p_{n-1})) = d + (n - 2) \cdot depth(\mathcal{S})$ .

**Proof:** From the inductive construction of  $\mathcal{M}(p_0, \dots, p_{n-1})$  we have:

$$\begin{aligned}
& \text{depth}(\mathcal{M}(p_0, \dots, p_{n-1})) \\
&= \text{depth}(\mathcal{M}(p_0, \dots, p_{n-3}, p_{n-1})) + \text{depth}(\mathcal{S}) \\
&= \text{depth}(\mathcal{M}(p_0, \dots, p_{n-4}, p_{n-1})) + \text{depth}(\mathcal{S}) + \text{depth}(\mathcal{S}) \\
&= \dots \\
&= \text{depth}(\mathcal{M}(p_0, \dots, p_{n-k}, p_{n-1})) + (k-2) \cdot \text{depth}(\mathcal{S}) \\
&= \dots \\
&= \text{depth}(\mathcal{M}(p_0, p_{n-1})) + (n-2) \cdot \text{depth}(\mathcal{S}) \\
&= \text{depth}(\mathcal{C}(p_0, p_{n-1})) + (n-2) \cdot \text{depth}(\mathcal{S}) \\
&= d + (n-2) \cdot \text{depth}(\mathcal{S}).
\end{aligned}$$

■

### 3.2.3 A Staircase-Merger

We now show how to construct the staircase-merger network  $\mathcal{S}(r, p, q)$ , whose depends on the depth  $d$  of network  $\mathcal{C}(p, q)$  (given by assumption), where  $r, p, q \geq 2$ . This construction relies on the two-merger network  $\mathcal{T}$ , with depth 2, constructed below in Section 3.2.4.

Let  $X_0, \dots, X_{q-1}$  be the input sequences and let's assume that they satisfy the  $p$ -staircase property and each satisfies the step property. Let  $A$  be the  $rp \times q$  matrix such that column  $i$  is the sequence  $X_i$ , for all  $0 \leq i < q$ . Because the  $X_i$  satisfy the  $p$ -staircase property, the step points of the columns of  $A$  lie within  $p$  of one another, modulo  $rp$  since the step points can be at the top or bottom part of  $A$  (Figure 3.5 (a)).

The *dirty region* of  $A$  is the smallest rectangle region of  $A$  such that if it this region has the step property in row-major form then the whole matrix  $A$  has the step property in row-major form. Initially, in the worst case, the dirty region of matrix  $A$  spans all the step points of the columns and has size at most  $p \times q$ . We want to find the dirty region and correct it (make it have the step property in row-major form). Since we do not know the exact location of the dirty region, we try to approximate its location and correct it by breaking matrix  $A$  into smaller pieces as described below.

Partition  $A$  into  $p \times q$  consecutive and disjoint submatrices  $A_0, \dots, A_{r-1}$  ( $A_0$  is on the top and  $A_{r-1}$  on the bottom). The column step points all lie within two adjacent  $A_i$  and  $A_{(i+1) \bmod r}$ , for some  $0 \leq i < r$  (Figure 3.5 (b)). If the step points all lie within two consecutive  $A_i$  and  $A_{i+1}$ , for  $0 \leq i < r-1$ , then it is easy to see that the combination of the



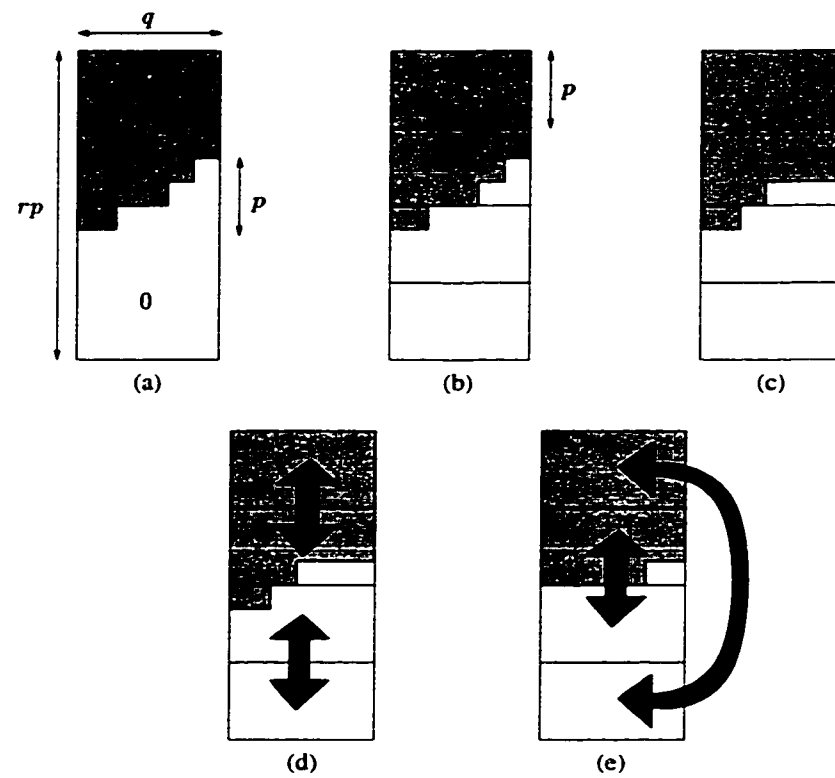


Figure 3.5: Construction of staircase-merger network

input sequences is 1-smooth, and the dirty region lies within these two matrices. (The case of Figure 3.5 (b)). If all the step points lie within  $A_0$  and  $A_{r-1}$  then the combination of the input sequences is 2-smooth and there are three possible values in array  $A$ , namely  $a$ ,  $a + 1$  and  $a + 2$ . The elements of  $A_0$  take values  $a + 2$  and  $a + 1$ , the elements of  $A_1, \dots, A_{r-2}$  all take values  $a + 1$  and the elements of  $A_{r-1}$  take values  $a + 1$  and  $a$ . Subsequently, the dirty region lies within matrices  $A_0$  and  $A_{r-1}$ .

Use  $\mathcal{C}(p, q)$  (given by assumption) to give each  $A_i$  the step property (see Figure 3.5 (c), where the sequences  $A_i$  are drawn in row major order). Notice that the dirty region still lies within two  $A_i$  and  $A_{(i+1) \bmod r}$ , but now it can easily be corrected by merging the two matrices, as described next.

Use a layer of two-mergers  $\mathcal{T}(p, q, q)$  to merge each  $A_{2i}$  and  $A_{2i+1}$  (Figure 3.5 (d)), and a second layer of  $\mathcal{T}(p, q, q)$  to merge each  $A_{2i+1}$  and  $A_{(2i+2) \bmod r}$  (Figure 3.5 (e)), for  $0 \leq i < \lfloor r/2 \rfloor$ . The first layer corrects the dirty region that is within two  $A_j$  and  $A_{(j+1) \bmod r}$  when  $j$  is even, and the second layer when  $j$  is odd. If  $r$  is odd we also need a third layer with one  $\mathcal{T}(p, q, q)$  to merge  $A_0$  and  $A_{r-1}$ . After the layers with the two-mergers, the dirty region is corrected and the resulting matrix  $A$  has the step property in row major order, and this is the output sequence of the staircase-merger  $\mathcal{S}$ .

Since each two-merger  $\mathcal{T}$  has depth two (see Section 3.2.4), and the depth of  $\mathcal{C}(p, q)$  is equal to  $d$  we have that  $\text{depth}(\mathcal{S}) \leq d + 6$ . The two-mergers use balancers of width  $2q$  and  $p$ . When we need to use balancers of width at most  $\max(p, q)$ , we can substitute each  $2q$ -balancer with a two-merger  $\mathcal{T}(q, 1, 1)$  (the input sequence of the  $2q$ -balancer consists of two smaller step sequences of length  $q$ ) which uses balancers of width 2 and  $q$ , yielding  $\text{depth}(\mathcal{S}) \leq d + 9$ .

## Optimizations

We can improve the depth of  $\mathcal{S}$  by replacing the two-mergers with the following construction (see Figure 3.6).

The construction is the same with the one described earlier until after the point where we apply the network  $\mathcal{C}(p, q)$  to each of the  $A_i$ . As we mentioned before, after the  $\mathcal{C}(p, q)$  networks, each  $A_i$  has the step property and the dirty region lies within two  $A_i$  and  $A_{i+1 \bmod r}$ , for some  $i$ ,  $0 \leq i < r$ .

We split each resulting  $A_i$  into two equal sized upper and lower parts. We use a layer of 2-balancers, which trivially have with depth 1, to connect the lower and upper part, respectively, of every two adjacent  $A_i$  and  $A_{(i+1) \bmod r}$  (see Figure 3.6). This moves the

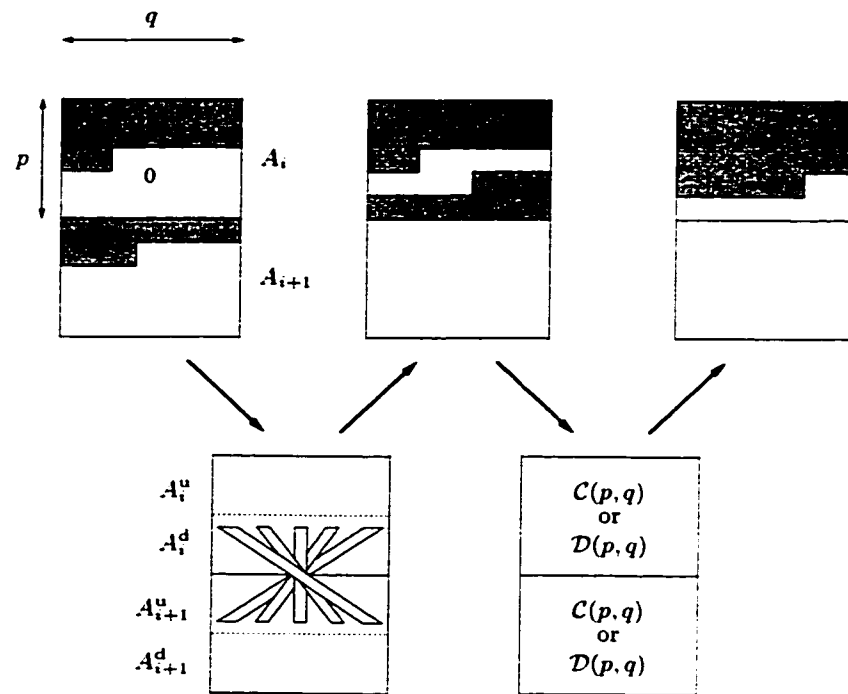


Figure 3.6: Optimizing the construction of the staircase-merger

dirty region into a single  $A_i$ , for some  $i$ ,  $0 \leq i < r$ , where now it has the form of a bitonic sequence. A final layer of  $\mathcal{C}(p, q)$  counting networks corrects the dirty region in the  $A_i$ . This construction gives  $\text{depth}(S) = 2d + 1$ . Alternatively, instead of the final layer of  $\mathcal{C}(p, q)$ , we can use the *bitonic-converter*  $\mathcal{D}(p, q)$ , described in section 3.2.4. This construction gives  $\text{depth}(S) = d + 3$ . Below we give a more detailed description of the construction.

In detail, after the first layer of  $\mathcal{C}(p, q)$  networks, we split each  $A_i$  into two subsequences  $A_i^u$  and  $A_i^d$  each of length  $s = \lfloor pq/2 \rfloor$ , such that they contain the first  $s$  and last  $s$  elements, respectively, of  $A_i$  (see Figure 3.6). Since the  $A_i$  have the step property, each of these subsequences has the step property too. A layer  $\ell$  of 2-balancers connects the sequences  $A_i^d$  and  $A_{(i+1) \bmod r}^u$ , for all  $i$ ,  $0 \leq i < r$ . Specifically, a 2-balancer connects the  $j$ th element of  $A_i^d$  with the  $(s - 1 - j)$ th element of  $A_{(i+1) \bmod r}^u$ , for all  $j$ ,  $0 \leq j < s$ , such that the first output of each 2-balancer is directed to north, with respect to matrix  $A$ .

For the correctness of the construction we only need to show that after layer  $\ell$  the dirty region lies within only one  $A_i$  and each  $A_i$  has the bitonic property.

**Proposition 3.2.4** *After layer  $\ell$ , the dirty region lies within only one  $A_i$ , for some  $i$ ,  $0 \leq i < r$ , and this  $A_i$  satisfies the bitonic property.*

**Proof:** Before layer  $\ell$ , each  $A_i$  has the step property and the dirty region lies within two  $A_i$  and  $A_{(i+1) \bmod r}$ , for some  $0 \leq i < r$ .

First, we consider the case  $i \neq r - 1$ , where the dirty region lies within two consecutive  $A_i$  and  $A_{i+1}$  (the other case is described below). These  $A_i$  and  $A_{i+1}$  are 1-smooth and for simplicity, we assume that their elements take values 0 and 1 (the case with higher values is similar). Denote by  $z_i$  and  $o_i$  the number of elements of  $A_i$  with value 0 and 1, respectively. Note that  $z_i + o_i = pq$ . By construction we have  $o_i \geq o_{i+1}$ . There are two possible cases: (a)  $0 \leq o_i + o_{i+1} \leq pq$ , and (b)  $pq < o_i + o_{i+1} \leq 2pq$ .

In case (a) (shown in Figure 3.6), we have  $o_{i+1} \leq s$  and  $z_i \geq o_{i+1}$ . All the  $o_{i+1}$  1s of  $A_{i+1}$  are in  $A_{i+1}^u$  and at least as many 0s are in  $A_i^d$ . Subsequently, the layer of 2-balancers  $\ell$ , that connects the  $A_i^d$  and  $A_{i+1}^u$ , moves all the 1s from  $A_{i+1}^u$  to  $A_i^d$ . The  $A_i^u$  and  $A_{i+1}^d$  remain unaffected. The result is that  $A_{i+1}$  contains only 0s, and  $A_i$  contains  $o_i$  1s followed by  $z_i - o_{i+1}$  0s followed by  $o_{i+1}$  1s, and thus  $A_i$  is bitonic. Therefore, the dirty region has moved to  $A_i$  with the form of a bitonic sequence.

In case (b), we have  $z_i \leq s$  and  $o_{i+1} \geq z_i$ . All the  $z_i$  0s of  $A_i$  are in  $A_i^d$  and at least as many 1s are in  $A_{i+1}^u$ . Subsequently, the layer of 2-balancers  $\ell$ , that connects the  $A_i^d$  and  $A_{i+1}^u$ , moves all the 0s from  $A_i^d$  to  $A_{i+1}^u$ . The  $A_i^u$  and  $A_{i+1}^d$  remain unaffected. The result is that  $A_i$  contains only 1s, and  $A_{i+1}$  contains  $z_i$  0s followed by  $o_{i+1} - z_i$  1s followed by  $z_{i+1}$

0s, and thus  $A_{i+1}$  is bitonic. Therefore, the dirty region has moved to  $A_{i+1}$  with the form of a bitonic sequence.

Next, consider the case  $i = r - 1$ , where the dirty region before layer  $\ell$  lies within the  $A_0$  and  $A_{r-1}$ . In this case, the combination of  $A_0$  and  $A_{r-1}$  is 2-smooth and each  $A_0$  and  $A_{r-1}$  is 1-smooth. For simplicity, we assume that the elements of these sequences take values 0, 1, and 2 (the case for higher values is similar). Specifically, the elements of  $A_0$  take values 2 and 1 and the elements of  $A_{r-1}$  take values 0 and 1. Denote by  $o_0$  and  $t_0$  the number of elements of  $A_0$  with value 1 and 2, respectively, and by  $z_{r-1}$  and  $o_{r-1}$  the number of elements of  $A_{r-1}$  with value 0 and 1, respectively. Note that  $o_0 + t_0 = pq$  and  $z_{r-1} + o_{r-1} = pq$ . It is easy to observe that  $o_{r-1} \geq t_0$  (since when the matrix  $A$  was originally constructed, each column had the step property). Again, there are two possible cases: (a)  $0 \leq t_0 + o_{r-1} \leq pq$ , and (b)  $pq < t_0 + o_{r-1} \leq 2pq$ .

In case (a) we have  $t_0 \leq s$  and  $z_{r-1} \geq t_0$ . All the  $t_0$  2s of  $A_0$  are in  $A_0^u$  and at least as many 0s are in  $A_{r-1}^d$ . Subsequently, the layer of 2-balancers  $\ell$ , that connects the  $A_0^u$  and  $A_{r-1}^d$ , transforms the 2s of  $A_0^u$  to 1s and the same number of 0s of  $A_{r-1}^d$  to 1s. The  $A_0^d$  and  $A_{r-1}^u$  remain unaffected. The result is that  $A_0$  contains only 1s, and  $A_{r-1}$  contains  $o_{r-1}$  1s followed by  $z_{r-1} - t_0$  0s followed by  $t_0$  1s, and thus  $A_{r-1}$  is bitonic. Therefore, the dirty region has moved to  $A_{r-1}$  with the form of a bitonic sequence.

In case (b), we have  $z_{r-1} \leq s$  and  $t_0 \geq z_{r-1}$ . All the  $z_{r-1}$  0s of  $A_{r-1}$  are in  $A_{r-1}^d$  and at least as many 2s are in  $A_0^u$ . Subsequently, the layer of 2-balancers  $\ell$ , that connects the  $A_0^u$  and  $A_{r-1}^d$ , transforms the 0s of  $A_{r-1}^d$  to 1s and the same number of 2s of  $A_0^u$  to 1s. The  $A_0^d$  and  $A_{r-1}^u$  remain unaffected. The result is that  $A_{r-1}$  contains only 1s, and  $A_0$  contains  $z_{r-1}$  1s followed by  $t_0 - z_{r-1}$  2s followed by  $o_0$  1s, and thus  $A_i$  is bitonic. Therefore, the dirty region has moved to  $A_0$  with the form of a bitonic sequence. ■

### 3.2.4 A Two-Merger and a Bitonic-Converter

First, we construct the *two-merger* network  $\mathcal{T}(p, q_0, q_1)$  of depth two from  $(q_0 + q_1)$ -balancers and  $p$ -balancers, where  $p \geq 2$  and  $q_0, q_1 \geq 1$ .

Let  $X_0$  and  $X_1$  be the input sequences with respective lengths  $pq_0$  and  $pq_1$ . As illustrated in Figure 3.7, arrange  $X_0$  as a  $p \times q_0$  matrix in column-major form,  $X_1$  as a  $p \times q_1$  matrix in reverse column major form, and align the two matrices side by side. If we place a  $(q_0 + q_1)$ -balancer across each row only one column is 1-smooth. If we then place a  $p$ -balancer across each column, the result has the step property (as a matrix in column-major form). More precisely, we have the following.

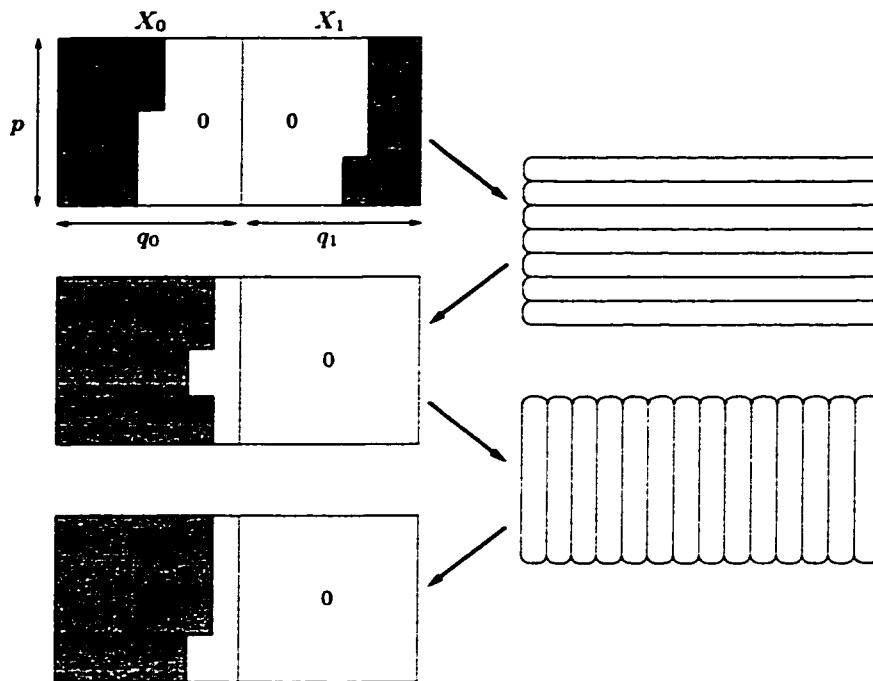


Figure 3.7: Construction of two-merger network

**Proposition 3.2.5** *The network  $\mathcal{T}(p, q_0, q_1)$  is a two-merger.*

**Proof:** let  $X_0$  assume values  $a_0$  and  $a_0 + 1$ , and let  $(r_0, c_0)$  be the step point's row and column in the  $p \times q_0$  matrix. Define  $a_1$  and  $(r_1, c_1)$  similarly for  $X_1$ . Suppose  $r_0 \leq r_1$  (the case of Figure 3.7, the other case is similar). Consider the row sums for the combined  $p \times (q_0 + q_1)$  matrix. Let  $s_r$  be the sum of the elements of row  $r$ , for  $0 \leq r \leq p - 1$ . We have

row $r$	sum $s_r$
$r < r_0$	$q_0 a_0 + (c_0 + 1) + q_1 a_1 + c_1$
$r_0 \leq r \leq r_1$	$q_0 a_0 + c_0 + q_1 a_1 + c_1$
$r_1 < r$	$q_0 a_0 + c_0 + q_1 a_1 + (c_1 + 1)$ .

The sequence  $s_0, \dots, s_{p-1}$  is 1-smooth. Therefore, after the first (horizontal) layer of balancers, all the step points of the balancers will appear in at most two consecutive columns (modulo  $q_0 + q_1$ ). As a result, the matrix has a single column  $c$  such that all elements of columns to the left have some value  $d + 1$ , all elements to the right have value  $d$ , and all elements of column  $c$  are 1-smooth with values  $d$  or  $d + 1$ . After the second (vertical) layer of balancers, columns to the left and right are unaffected, but column  $c$  has the step property, and so does the resulting matrix. ■

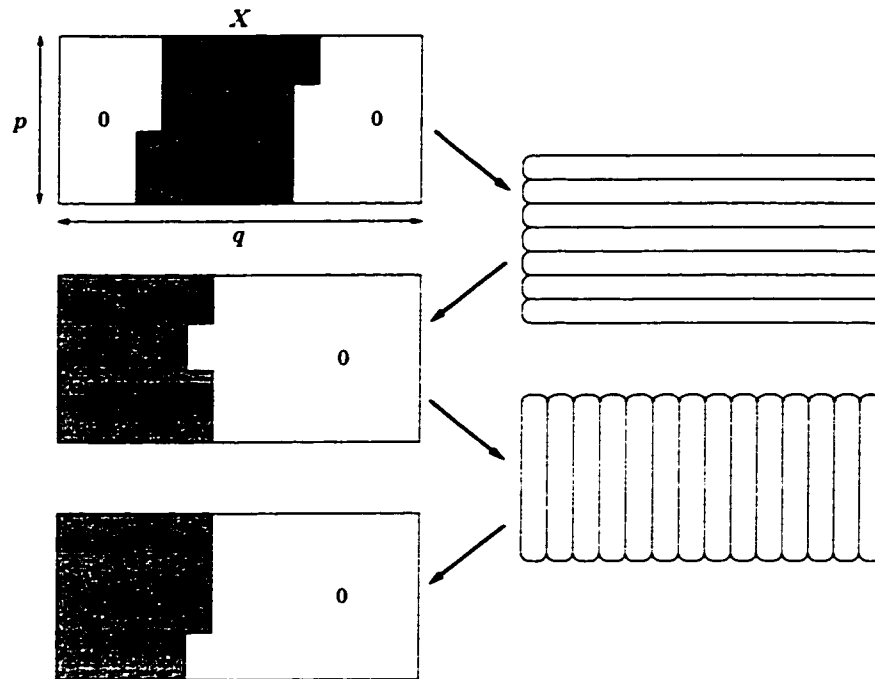


Figure 3.8: Construction of bitonic-converter network

Next, we construct the *bitonic-converter* network  $\mathcal{D}(p, q)$  of depth two from  $q$ -balancers and  $p$ -balancers, where  $p, q \geq 2$ .

Let  $X$  be the input sequence with the bitonic property. As illustrated in Figure 3.8, arrange  $X$  as a  $p \times q$  matrix in column-major form. If we place a  $q$ -balancer across each row only one column is 1-smooth. If we then place a  $p$ -balancer across each column, the result has the step property (as a matrix in column-major form). More precisely, we have the following.

**Proposition 3.2.6** *The network  $\mathcal{D}(p, q)$  is a bitonic-converter.*

**Proof:** Since  $X$  is bitonic it is 1-smooth and has at most two transitions. Let  $X$  assume values  $a$  and  $a + 1$ . We consider the case where  $X$  has two transitions (the case with one transition can be treated similarly). Furthermore, in sequence  $X$  let's assume that the transitions occur such that the first elements take value  $a$ , then the first transition occurs and the next elements take value  $a + 1$ , then the second transition occurs and the rest elements take value  $a$ . (The other case, where the elements of  $X$  take first values  $a + 1$  then  $a$  and then  $a + 1$ , is similar.)

Let's assume that the first transition occurs between elements  $x_a$  and  $x_{a+1}$  and the

second between elements  $x_{b-1}$  and  $x_b$ . Denote by  $(r_a, c_a)$  the row and column position of  $a$  in the array of  $X$ . Similarly define  $(r_b, c_b)$  for  $b$ . Suppose  $r_a \geq r_b$  (the case of Figure 3.8, the other case is similar). Consider the row sums for the matrix. Let  $s_r$  be the sum of the elements of row  $r$ , for  $0 \leq r \leq p-1$ . We have

row $r$	sum $s_r$
$r < r_b$	$qa + (c_b - c_a - 1) + 1$
$r_b \leq r \leq r_a$	$qa + (c_b - c_a - 1)$
$r_a < r$	$qa + (c_b - c_a - 1) + 1.$

The sequence  $s_0, \dots, s_{p-1}$  is 1-smooth. Therefore, after the first (horizontal) layer of balancers, all the step points of the balancers will appear in at most two consecutive columns (modulo  $q$ ). As a result, the matrix has a single column  $c$  such that all elements of columns to the left have value  $a + 1$ , all elements to the right have value  $a$ , and all elements of column  $c$  are 1-smooth with values  $a$  or  $a + 1$ . After the second (vertical) layer of balancers, columns to the left and right are unaffected, but column  $c$  has the step property, and so does the resulting matrix. ■

### 3.3 Specific Counting Network Constructions

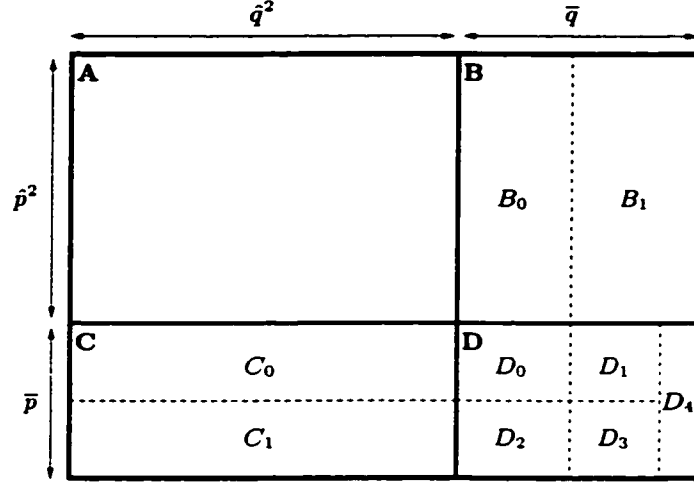
#### 3.3.1 The Counting Network $\mathcal{K}$

We construct  $\mathcal{K}(p_0, \dots, p_{n-1})$ , the counting network of depth  $O(n^2)$  from balancers of width at most  $\max(p_i p_j)$ , for  $0 \leq i, j < n$ , where  $p_i \geq 2$  and  $n \geq 2$ .

The construction is the same with the construction of  $\mathcal{C}$  described in Section 3.2, where in place of each instance of  $\mathcal{C}(p_i, p_j)$  we use a balancer of width  $p_i p_j$  with  $d = 1$ . For the staircase-merger  $\mathcal{S}$  we use the optimization described in Section 3.2.3 with  $\text{depth}(\mathcal{S}) = 2d + 1 = 3$ , and we get for the depth of  $\mathcal{K}$ :

**Proposition 3.3.1**  $\text{depth}(\mathcal{K}(p_0, \dots, p_{n-1})) = 1.5n^2 - 3.5n + 2.$



Figure 3.9: Construction of width- $pq$  counting network

**Proof:**

$$\begin{aligned}
& \text{depth}(\mathcal{K}(p_0, \dots, p_{n-1})) \\
&= \text{depth}(\mathcal{C}(p_0, \dots, p_{n-1})) \\
&= (n-1)d + \left( \frac{n^2}{2} - \frac{3n}{2} + 1 \right) \cdot \text{depth}(S) \\
&\quad \text{(by Proposition 3.2.1)} \\
&= (n-1)1 + \left( \frac{n^2}{2} - \frac{3n}{2} + 1 \right) 3 \\
&= 1.5n^2 - 3.5n + 2.
\end{aligned}$$

■

### 3.3.2 The Counting Network $\mathcal{R}(p, q)$

Let  $w = pq$ , where  $p, q \geq 2$ . We now construct a constant-depth counting network  $\mathcal{R}(p, q)$  of width  $w$  from balancers of width at most  $\max(p, q)$ . We rely on two subsidiary networks: the two-merger network  $\mathcal{T}$  described in Section 3.2.4, and the counting network  $\mathcal{K}$  described in Section 3.3.1.

Let  $\hat{p} = \lfloor \sqrt{p} \rfloor$ , and  $\bar{p} = p - \hat{p}^2$ . Similarly, we define  $\hat{q}$  and  $\bar{q}$ . The following inequalities hold (for a proof look in Appendix A):

$$\max(\hat{p}, \hat{q})^2 \leq \max(p, q) \quad (3.1)$$

$$\max(\hat{p}, \hat{q}) \left\lceil \frac{\max(\bar{p}, \bar{q})}{2} \right\rceil \leq \max(p, q) \quad (3.2)$$

$$\left\lfloor \frac{\max(\bar{p}, \bar{q})}{2} \right\rfloor \left\lceil \frac{\max(\bar{p}, \bar{q})}{2} \right\rceil \leq \max(p, q) \quad (3.3)$$

Let  $X$  be the input sequence to  $\mathcal{R}(p, q)$ . Because  $|X| = pq$ , we can arrange  $X$  as a  $p \times q$  matrix in arbitrary order. Divide  $X$  into four quadrants:  $A$  encompasses the first  $\hat{p}^2$  rows and  $\hat{q}^2$  columns,  $B$  the first  $\hat{p}^2$  rows and remaining  $\bar{q}$  columns,  $C$  the remaining  $\bar{p}$  rows and first  $\hat{q}^2$  columns, and  $D$  the remaining  $\bar{p}$  rows and  $\bar{q}$  columns. These divisions are shown as thick lines in Figure 3.9.

Area  $A$  is a sequence of length  $\hat{p}\hat{p}\hat{q}\hat{q}$ . We can use the constant-depth counting network  $\mathcal{K}(\hat{p}, \hat{p}, \hat{q}, \hat{q})$ , constructed from balancers of width at most  $\max(\hat{p}^2, \hat{q}^2, \hat{p}\hat{q}) \leq \max(p, q)$  (Equation 3.1), to transform  $A$  into a sequence  $A'$  satisfying the step property.

Let  $\bar{q}_0 = \lfloor \bar{q}/2 \rfloor$  and  $\bar{q}_1 = \lceil \bar{q}/2 \rceil$ . Partition  $B$  into disjoint submatrices  $B_0$  and  $B_1$  of respective dimensions  $\hat{p}^2 \times \bar{q}_0$  and  $\hat{p}^2 \times \bar{q}_1$ . (These divisions are shown as dotted lines in Figure 3.9.) We use the constant-depth counting network  $\mathcal{K}(\bar{q}_0, \hat{p}, \hat{p})$  and  $\mathcal{K}(\bar{q}_1, \hat{p}, \hat{p})$ , constructed from balancers of width at most  $\max(\hat{p}^2, \hat{p}\bar{q}_0)$  and  $\max(\hat{p}^2, \hat{p}\bar{q}_1)$ , that respectively transform  $B_0$  and  $B_1$  into sequences  $B'_0$  and  $B'_1$  satisfying the step property. By Equations 3.1 and 3.2, each of these networks is constructed from balancers of width at most  $\max(p, q)$ . Finally, the constant-depth two-merger network  $\mathcal{T}(\hat{p}^2, \bar{q}_0, \bar{q}_1)$  merges  $B'_0$  and  $B'_1$  to a single sequence  $B'$  satisfying the step property. This two-merger is constructed from balancers of width  $\hat{p}^2$  and  $\bar{q}$ , each less than or equal to  $\max(p, q)$  (Equation 3.1). In exactly the same way,  $C$  can be transformed to  $C'$  satisfying the step property.

Partition  $D$  into disjoint submatrices  $D_0, D_1, D_2, D_3$ , and  $D_4$ , with respective dimensions  $\bar{p}_0 \times \bar{q}_0, \bar{p}_0 \times \bar{q}_0, \bar{p}_1 \times \bar{q}_0, \bar{p}_1 \times \bar{q}_0$ , and  $\bar{p} \times 1$ . (See Figure 3.9.) Each of these regions can be given the step property by a single balancer of width less than or equal to  $\max(p, q)$  (Equation 3.3). The resulting sequences can then be merged in constant depth using several copies of the two-merger network  $\mathcal{T}$  to a sequence  $D'$  satisfying the step property. These two-mergers are constructed with balancers of width less than  $\max(p, q)$ . Notice that  $D_4$  exists only if  $\bar{q}_0 \neq \bar{q}_1$ , otherwise we do not include it in the above construction and we use the two-mergers accordingly.

We have shown that  $A, B, C$ , and  $D$  can be transformed to  $A', B', C'$ , and  $D'$  satisfying the step property by counting networks constructed from balancers of width less than  $\max(p, q)$ . In the same way, two-merger networks can merge  $A'$  and  $B'$ , and (in parallel)  $C'$  and  $D'$ . Finally, a two-merger network can merge their results. These two-mergers are constructed with balancers of width less than or equal to  $\max(p, q)$ .

The depth of the construction of  $\mathcal{R}$  is dominated by the depth of the counting network

of  $A$  plus the final two layers of two-mergers that each has depth 2. We have:

$$\begin{aligned}
\text{depth}(\mathcal{R}(p, q)) &= \text{depth}(\mathcal{K}(\hat{p}, \bar{p}, \hat{q}, \bar{q})) + 2 \cdot \text{depth}(\mathcal{T}) \\
&= 1.5 \cdot 4^2 - 3.5 \cdot 4 + 2 + 2 \cdot 2 \\
&\quad \text{(by Proposition 3.3.1)} \\
&= 16.
\end{aligned}$$

Some of the variables  $\hat{p}, \bar{p}, \bar{p}_0, \dots$  may take the extreme values 0 or 1. In these cases, for each of the affected  $A, B, B_0, \dots$  we either do not use any network or we use a single balancer, and then we use the two-mergers accordingly. Alternatively, we can combine two or more of the above areas. These extreme cases can give us a network that has depth smaller than 16. For example, consider the  $\mathcal{R}(3, 5)$  network. We have  $p = 3, \hat{p} = 1, \bar{p} = 2$  and  $q = 5, \hat{q} = 2, \bar{q} = 1$ . The areas  $A, B, C$  and  $D$  have sizes  $1 \times 4, 1 \times 1, 2 \times 4$ , and  $2 \times 1$ . By combining areas  $A$  and  $B$  and areas  $C$  and  $D$  and by using 5-balancers and two-mergers accordingly, we get the network of Figure 3.1 with depth 5.

Therefore, taking into consideration all the cases, we have  $\text{depth}(\mathcal{R}(p, q)) \leq 16$ .

### 3.3.3 The Counting Network $\mathcal{L}$

Finally, we construct  $\mathcal{L}(p_0, \dots, p_{n-1})$ , the desired counting network of depth  $O(n^2)$  from balancers of width at most  $\max(p_i)$ , for  $0 \leq i < n$ , where  $p_i \geq 2$  and  $n \geq 2$ .

The construction is the same with the construction of  $\mathcal{C}$  described in Section 3.2, where in place of each instance of network  $\mathcal{C}(p_i, p_j)$  we use the counting network  $\mathcal{R}(p_i, p_j)$ , described in Section 3.3.2 with  $d = \text{depth}(\mathcal{R}(p_i, p_j)) \leq 16$ . For the staircase-merger  $\mathcal{S}$  we use the optimization described in Section 3.2.3 with  $\text{depth}(\mathcal{S}) = d + 3 \leq 19$ , and we get for the depth of  $\mathcal{L}$ :

**Theorem 3.3.2**  $\text{depth}(\mathcal{L}(p_0, \dots, p_{n-1})) \leq 9.5n^2 - 12.5n + 3$ .

**Proof:**

$$\begin{aligned}
&\text{depth}(\mathcal{L}(p_0, \dots, p_{n-1})) \\
&= \text{depth}(\mathcal{C}(p_0, \dots, p_{n-1})) \\
&= (n-1)d + \left(\frac{n^2}{2} - \frac{3n}{2} + 1\right) \cdot \text{depth}(\mathcal{S}) \\
&\quad \text{(by Proposition 3.2.1)} \\
&\leq (n-1)16 + \left(\frac{n^2}{2} - \frac{3n}{2} + 1\right) 19 \\
&= 9.5n^2 - 12.5n + 3.
\end{aligned}$$



### 3.4 Related Work

The first counting network constructions [6] used 2-balancers, yielding networks of width  $2^n$  and depth  $O(n^2)$ . Aharonson and Attiya [2] constructed a counting network of width  $w = p2^k$  and depth  $O(\lg^3(w/p))$  from balancers of width 2 and  $p$ . They also construct networks of arbitrary width by taking a standard counting network and linking the excess output wires to the excess input wires, resulting in a cyclic network (our is acyclic). Busch, Hardavellas, and Mavronicolas [17] give a construction of  $w = p2^k$  and depth  $O(\lg^2(w/p))$  using balancers of width 2 and  $p$ . Felten, LaMarca, and Ladner [28] give a construction of width  $w = 2^k$  from balancers of width  $2^l$ , where the depth ranges from  $O(1)$  to  $O(\log^2 w)$  depending on the value of  $l$ , as well as a construction of width  $w = p2^k$ . Klugerman [38] gives a construction of arbitrary width  $w$  and depth  $O((\lg w) \lg \lg w)$  from  $p$ -balancers, where  $p$  ranges over the prime factors of  $w$ . This construction is based on the AKS sorting network [5], and it is impractical in the sense that the constant factors are enormous.

### 3.5 Discussion

We presented a new construction for a family of counting networks of width  $w = p_0 \cdots p_{n-1}$ , and depth at most  $9.5n^2 - 12.5n + 3$ , from balancers of width at most  $\max(p_i)$ . This is the first arbitrary-width construction without enormous constant factors.

The overall network structure (Figure 3.3) is similar but not identical to that of the bitonic counting network [6, 9]. The bitonic network, however, has smaller depth by a constant factor, suggesting that further improvement in our constant terms may be possible. It remains an open problem whether the asymptotic  $O(n^2)$  depth can be improved without introducing very large constants.

An interesting open question concerns the timing constraints necessary for counting networks built in this way to be linearizable (c.f., [41, 43, 44]).

## Chapter 4

# Irregular Counting Networks

Most of the known counting network constructions have the same input and output width, and are built from balancers with the same input and output width [2, 6, 17, 18, 28]. An example of such a well studied counting network is the *bitonic* counting network [6, 9], which has input and output width  $w = 2^k$ , for some  $k > 0$ , and is built from  $(2, 2)$ -balancers (see Section 1.5.4). The depth of the bitonic network is  $O(\lg^2 w)$ . Another well known construction is the *periodic* counting network [6, 25].

Here, we deviate from the “standard” approach of building counting networks with the same input and output width and we present a novel “irregular” counting network construction  $\mathcal{C}(w, t)$  where the input width  $w$  is smaller or equal than the output width  $t$ .<sup>1</sup> Specifically,  $w = 2^k$ ,  $t = p2^l$ , for some  $k, l, p > 0$ , and  $w \leq t$ . Our network is constructed from  $(2, 2)$ -balancers and  $(2, 2t/w)$ -balancers. The depth of the network depends only on the input width  $w$  and it is  $O(\lg^2 w)$ . As an example, the network presented in Figure 1.1 is the counting network  $\mathcal{C}(4, 8)$ .

Because of its structure, our network provides more flexibility in terms of contention than other networks. As a measurement of a network’s contention Dwork *et al.* [26] introduced the *amortized contention*. In a balancer, every time step a token has to wait for another token then a *stall* step incurs. The amortized contention measures the number of stalls that any token experiences while it traverses the counting network. For any network  $\mathcal{B}$ , which is accessed by  $n$  concurrent processes, we denote the amortized contention by  $\text{cont}(\mathcal{B}, n)$ .

In the other known networks the contention depends only on two parameters: the input/output width  $w$  and the concurrency  $n$ . For example, the amortized contention of the bitonic network is  $O(n \lg^2 w/w)$  (see [26]), and the amortized contention of the periodic

---

<sup>1</sup>The work of this chapter was first presented in [23].

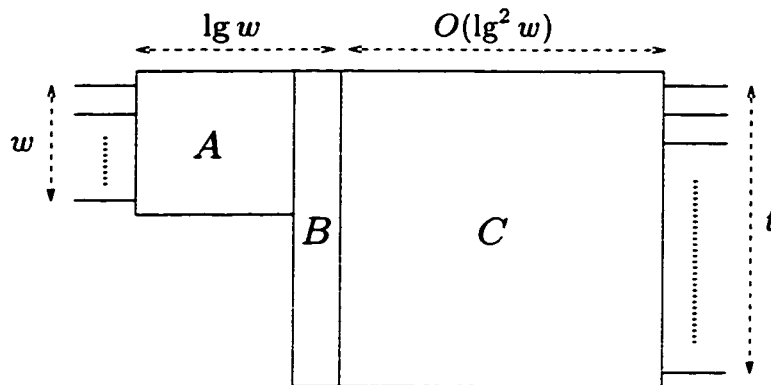


Figure 4.1: The three parts of network  $\mathcal{C}(w, t)$

network is  $O(n \lg^3 w/w)$  (see [26]). In our network, the amortized contention is determined by one more parameter: the output width  $t$ . Specifically, the contention in our network is given by the expression

$$\text{cont}(\mathcal{C}(w, t), n) = O\left(\frac{n \lg w}{w} + \frac{n \lg^2 w}{t} + \frac{w \lg^3 w}{t} + \lg^2 w\right).$$

Since the contention of our network is determined by three parameters, we have more flexibility in choosing the right network parameters for the specific needs of a counting problem.

To demonstrate the flexibility of our network and to compare it to the other network constructions, we fix the input width  $w$ , we take  $n \geq \lg w$ , and we adjust the output width  $t$ . When  $w = t$  we obtain a new “standard” counting network that has the same input and output width. This network has similar characteristics with the bitonic network. That is, both networks have input/output width  $w$ , exactly same depth  $O(\lg^2 w)$ , and contention  $O(n \lg^2 w/w)$ . By increasing the output width  $t$  the contention of network  $\mathcal{C}(w, t)$  decreases while the depth remains the same. Specifically, by taking  $t = w \lg w$  the amortized contention becomes  $O(n \lg w/w)$ , which is better by a logarithmic factor of  $w$  over the contention of the bitonic counting networks with the same input width and depth. Since the contention decreases, in our network we expect a higher throughput for the same latency, while in the other networks no such option is available.

The flexibility of our network is a result of its unique structure. When we look inside its structure we can identify three parts, as shown in Figure 4.1:

- Part A has input and output width  $w$ , depth  $\lg w - 1$ , and it is build from  $(2, 2)$ -balancers.

- Part  $B$  has input width  $w$ , output width  $t$ , depth 1, and it is build from  $(2, 2t/w)$ -balancers. This part serves as the transition from parts  $A$  to  $C$ .
- Part  $C$  has input and output width  $t$ , depth  $O(\lg^2 w)$ , and is build from  $(2, 2)$ -balancers.

The most significant part, in terms of depth, is part  $C$ . The tokens spend most of their time in this part and therefore the contention depends heavily on it. By increasing the output width  $t$ , part  $C$  becomes wider with more balancers, and thus the tokens have less chance to contend for the same balancer. Therefore, as  $t$  increases the contention of part  $C$  decreases, and subsequently the contention of the whole network decreases. By increasing arbitrarily  $t$  the contention of this part goes to 0. However, for fixed  $w$ , as  $t$  becomes large, part  $A$  remains the same, and thus this part will determine the network's contention when  $w \ll t$ . Nevertheless, since the depth of part  $A$  is only  $O(\lg w)$  it cannot affect the performance of the network very much, and thus the low contention is preserved.

Since by increasing the output width  $t$  the number of balancers in the network increases (balancers of part  $C$ ), we may have difficulties to implement this network in a real system. Therefore, there is an implementation tradeoff between the two extreme cases  $w = t$  and  $w \ll t$ , which depends on the particular needs of the counting problem we try to solve. A compromise where  $t = w \lg w$  seems to be a logical solution.

We proceed as follows. In Section 4.1 we give some necessary preliminaries. In Section 4.2 we give a top down description of the counting network construction. This network uses as a building block the novel *difference merging network* which we describe in Section 4.3. In Section 4.4 we compute the amortized contention of our counting network. In Section 4.5 we present related work and we conclude in Section 4.6 with a discussion.

## 4.1 Preliminaries

### 4.1.1 Sequences

For any step sequence  $X^{(w)}$ , the step point is defined as in Section 1.4, where now if all  $x_i$  are equal the step point is equal to  $w$ . Notice that now a step point cannot be zero in any case.

For any sequence, the *highest value* is the maximum value of any of its elements. Similarly, we define the *lowest value*.

For sequence  $X^{(w)}$ , the even subsequence is  $X_e^{(w/2)} = x_0, x_2, \dots$ , and the odd subsequence is  $X_o^{(w/2)} = x_1, x_3, \dots$ . We immediately have the following observation.

**Observation 4.1.1** *If the sequence  $X^{(w)}$ , where  $w \geq 2$ , has the step property then*

$$0 \leq \Sigma(X_e^{(w/2)}) - \Sigma(X_o^{(w/2)}) \leq 1.$$

We continue by showing some useful lemmas.

**Lemma 4.1.1** *If  $X^{(w)}$  and  $Y^{(w)}$ , where  $w \geq 2$ , are two step sequences with highest values  $a$  and  $b$ , respectively, such that*

$$0 \leq \Sigma(X^{(w)}) - \Sigma(Y^{(w)}) \leq d$$

then

$$0 \leq a - b \leq \left\lfloor \frac{d}{w} \right\rfloor + 1.$$

**Proof:** Since  $a$  and  $b$  represent the highest values of the step sequences  $X^{(w)}$  and  $Y^{(w)}$ , respectively, we have

$$\begin{aligned} w(a - 1) &< \Sigma(X^{(w)}) \leq wa, \text{ and} \\ w(b - 1) &< \Sigma(Y^{(w)}) \leq wb. \end{aligned}$$

By subtracting the two inequalities we get

$$w(a - b - 1) < \Sigma(X^{(w)}) - \Sigma(Y^{(w)}) < w(a - b + 1).$$

By inequality

$$0 \leq \Sigma(X^{(w)}) - \Sigma(Y^{(w)}) \leq d$$

we get:

$$\begin{aligned} w(a - b - 1) &< d, \text{ and} \\ w(a - b + 1) &> 0. \end{aligned}$$

Since  $w \geq 2$ , we get

$$\begin{aligned} a - b &< \frac{d}{w} + 1, \text{ and} \\ a - b &> -1. \end{aligned}$$

Since  $a$  and  $b$  are integers, we get

$$0 \leq a - b \leq \left\lfloor \frac{d}{w} \right\rfloor + 1,$$

as needed. ■



By a simple case analysis we can show the following. (For a proof look in Appendix B.)

**Lemma 4.1.2** *If  $X^{(w)}$  and  $Y^{(w)}$ , where  $w \geq 2$ , are step sequences with highest values  $a$  and  $b$ , respectively, and step points  $k$  and  $l$ , respectively, such that*

$$0 \leq \Sigma(X^{(w)}) - \Sigma(Y^{(w)}) \leq 2$$

*then either  $a = b$  and*

$$0 \leq k - l \leq 2,$$

*or  $a = b + 1$  and*

$$l = w \text{ and } 1 \leq k \leq 2, \text{ or}$$

$$l = w - 1 \text{ and } k = 1.$$

We continue by showing a lemma about the even and odd subsequences of step sequences.

**Lemma 4.1.3** *If  $X^{(w)}$  and  $Y^{(w)}$ , where  $w \geq 2$ , are step sequences such that*

$$0 \leq \Sigma(X^{(w)}) - \Sigma(Y^{(w)}) \leq \delta,$$

*where  $\delta$  is even, then*

$$0 \leq \Sigma(X_e^{(w/2)}) - \Sigma(Y_e^{(w/2)}) \leq \frac{\delta}{2}, \text{ and}$$

$$0 \leq \Sigma(X_o^{(w/2)}) - \Sigma(Y_o^{(w/2)}) \leq \frac{\delta}{2}.$$

**Proof:** Denote

$$A = \Sigma(X_e^{(w/2)}) - \Sigma(Y_e^{(w/2)}), \text{ and}$$

$$B = \Sigma(X_o^{(w/2)}) - \Sigma(Y_o^{(w/2)}).$$

We need to show that  $0 \leq A \leq \delta/2$  and  $0 \leq B \leq \delta/2$ .

We have

$$\Sigma(X^{(w)}) = \Sigma(X_e^{(w/2)}) + \Sigma(X_o^{(w/2)}), \text{ and}$$

$$\Sigma(Y^{(w)}) = \Sigma(Y_e^{(w/2)}) + \Sigma(Y_o^{(w/2)}).$$

By assumption, we have  $0 \leq \Sigma(X^{(w)}) - \Sigma(Y^{(w)}) \leq \delta$  and thus

$$0 \leq (\Sigma(X_e^{(w/2)}) + \Sigma(X_o^{(w/2)})) - (\Sigma(Y_e^{(w/2)}) + \Sigma(Y_o^{(w/2)})) \leq \delta \implies$$

$$0 \leq A + B \leq \delta.$$

By observation 4.1.1 we have

$$0 \leq \Sigma(X_e^{(w/2)}) - \Sigma(X_o^{(w/2)}) \leq 1, \text{ and}$$

$$0 \leq \Sigma(Y_e^{(w/2)}) - \Sigma(Y_o^{(w/2)}) \leq 1.$$

By subtracting the above inequalities we get

$$-1 \leq (\Sigma(X_e^{(w/2)}) - \Sigma(Y_e^{(w/2)})) - (\Sigma(X_o^{(w/2)}) - \Sigma(Y_o^{(w/2)})) \leq 1 \implies$$

$$-1 \leq A - B \leq 1.$$

By adding the inequalities

$$0 \leq A + B \leq \delta, \text{ and}$$

$$-1 \leq A - B \leq 1,$$

we get

$$-\frac{1}{2} \leq A \leq \frac{\delta}{2} + \frac{1}{2}, \text{ and}$$

$$-\frac{1}{2} \leq B \leq \frac{\delta}{2} + \frac{1}{2}.$$

Since A and B are integers we get

$$0 \leq A \leq \frac{\delta}{2}, \text{ and}$$

$$0 \leq B \leq \frac{\delta}{2},$$

as needed. ■

### 4.1.2 Balancing Networks

We consider the following balancing network families

- A *counting network*  $\mathcal{C}(w, t)$  has input sequence of length  $w$  and output sequence of length  $t$ . For any input, the output sequence satisfies the step property.
- A *difference merging network*  $\mathcal{M}(w, \delta)$  has two input sequences, where the first input sequence is  $X^{(w/2)}$  and the second is  $Y^{(w/2)}$ , and output sequence of length  $w$ . If both  $X^{(w/2)}$  and  $Y^{(w/2)}$  satisfy the step property and  $0 \leq \Sigma(X^{(w/2)}) - \Sigma(Y^{(w/2)}) \leq \delta$  then the output sequence satisfies the step property. That is, a difference merging network merges two step sequences when their sums differ by at most  $\delta$  (and the sum of the first sequence is bigger or equal to the sum of the second sequence).

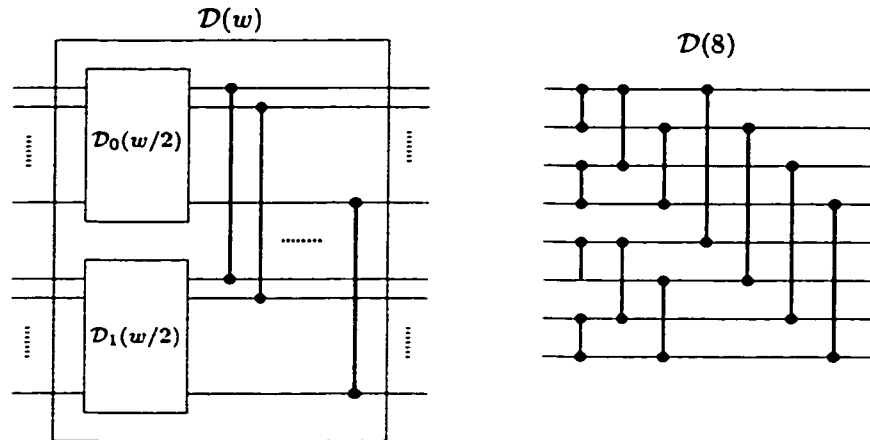


Figure 4.2: The butterfly network

A balancing network of interest, that will be useful when we calculate the contention of our counting network, is the *butterfly* balancing network  $\mathcal{D}(w)$  that we describe next.

The butterfly balancing network  $\mathcal{D}(w)$  has input and output width  $w$ , where  $w = 2^k$ ,  $k \geq 1$ , and is constructed by induction on  $w$  from  $(2, 2)$ -balancers (See Figure 4.2). For  $w = 2$ ,  $\mathcal{D}(w)$  is simply a  $(2, 2)$ -balancer. Let's assume that we have constructed  $\mathcal{D}(w')$  for all  $2 \leq w' < w$ , we will construct  $\mathcal{D}(w)$ . Take two copies of  $\mathcal{D}(w/2)$ , denoted  $\mathcal{D}_0(w/2)$  and  $\mathcal{D}_1(w/2)$ , with respective output sequences  $X^{(w/2)}$  and  $Y^{(w/2)}$ . The input sequences to  $\mathcal{D}_0(w)$  and  $\mathcal{D}_1(w)$  are the first and the second half, respectively, of the input sequence to  $\mathcal{D}(w)$ . Take  $w/2$  copies of the  $(2, 2)$ -balancer, denoted  $b_0, \dots, b_{w/2-1}$ . The first and second input wires of  $b_i$  are connected to elements  $x_i$  and  $y_i$ , respectively, for all  $0 \leq i < w/2$ . Let  $Z^{(w)}$  be the output sequence of  $\mathcal{D}(w)$ . We have that the first and second output wires of balancer  $b_i$ , are connected to the elements  $z_i$  and  $z_{i+w/2}$ , respectively, for all  $0 \leq i < w/2$ .

By the construction of the butterfly network we immediately have that  $\text{depth}(\mathcal{D}(w)) = \lg w$ . We can show the following smoothing property for a butterfly.

**Lemma 4.1.4** *The output sequence of the butterfly network  $\mathcal{D}(w)$  has the  $\lg w$ -smooth property.*

**Proof:** The proof is by induction of  $w$ . For the base case  $w = 2$ , the network is just a  $(2, 2)$ -balancer that trivially has the 1-smooth property. Let's assume that the claim holds for all  $2 \leq w' < w$ , we will show that it holds for  $w$ . From the inductive construction of  $\mathcal{D}(w)$ , described above, we have that each of  $\mathcal{D}_0(w/2)$  and  $\mathcal{D}_1(w/2)$  has the  $\lg(w/2)$ -smooth property. Let  $c_0$  and  $d_0$  be the smallest and biggest values of the output sequence

of  $\mathcal{D}_0(w/2)$ . Similarly, define  $c_1$  and  $d_1$  for  $\mathcal{D}_1(w/2)$ .

By the  $\lg(w/2)$ -smooth property we have  $d_0 - c_0 \leq \lg(w/2)$ , and similarly for  $c_1$  and  $d_1$ . Each balancer  $b_i$  at the final layer of  $\mathcal{D}(w)$  receives one input from  $\mathcal{D}_0(w/2)$  and the other from  $\mathcal{D}_1(w/2)$ . The balancer will output the smallest value when it receives the smallest values in its inputs which are  $c_0$  and  $c_1$ . In this case the smallest value on the output of the balancer will appear on the bottom output wire and is  $c = \lceil (c_0 + c_1 - 1)/2 \rceil$ . Similarly, the highest value is  $d = \lceil (d_0 + d_1)/2 \rceil$ . We have:

$$\begin{aligned} d &= \left\lceil \frac{d_0 + d_1}{2} \right\rceil \\ &\leq \left\lceil \frac{c_0 + c_1 + 2 \lg \frac{w}{2}}{2} \right\rceil \\ &= \left\lceil \frac{c_0 + c_1}{2} \right\rceil + \lg \frac{w}{2}. \end{aligned}$$

Furthermore,

$$\left\lceil \frac{c_0 + c_1}{2} \right\rceil - \left\lceil \frac{c_0 + c_1 - 1}{2} \right\rceil \leq 1.$$

Therefore,

$$d - c \leq 1 + \lg \frac{w}{2} = \lg w,$$

as needed. ■

### 4.1.3 Contention

In this section, we first give formal definitions for contention, then we give a general formula for computing the amortized contention of any layer of a balancing network, and finally we use this formula to compute the contention of the butterfly network. Parts of the following discussion are adapted from Section 3.2 of [26].

Each time a token passes through a balancer, it incurs a *stall* step to all other tokens pending at this balancer, or equivalently, every time step a token has to wait for another token a stall step incurs. The number of stall steps has been introduced by Dwork *et al.* [26] as a measurement for contention. The *contention* incurred by the traversal of  $m$  tokens through the network  $\mathcal{B}$  at concurrency  $n$ , denoted  $\text{cont}(\mathcal{B}, n, m)$ , is the maximum number of stalls, over all possible executions, induced by an adversary scheduler. The *amortized contention* of the network  $\mathcal{B}$  at concurrency  $n$ , denoted  $\text{cont}(\mathcal{B}, n)$ , is the limit supremum of  $\text{cont}(\mathcal{B}, n, m)$  divided by  $m$ , as  $m$  goes to infinity.

The amortized contention is a simple measure of stalls that any token experiences while it traverses the counting network. Dwork *et al.* [26] compute the amortized contention of a

network as follows: they partition the tokens into equally sized groups and they count the average number of stalls that occur between any two groups; then they simply divide this number of stalls by the size of the groups. This measurement is simple and practical in the sense that the only parameters required for its computation are the number of concurrent processes  $n$  and the width of the network. Furthermore, it doesn't require any timing information for the arrival and departure time of tokens, as more complicated queueing theory models do.

Let  $l$  be a layer of a balancing network  $\mathcal{B}$ . Let's assume that layer  $l$  is made of balancers of output width at most  $q$ . Denote by  $w$  the output width of  $l$ . Let's also assume that at any quiescent state the output of  $l$  has the  $k$ -smooth property. Let the concurrency of  $\mathcal{B}$  be  $n$ .

We will compute the amortized contention  $\text{cont}(l, n)$  of layer  $l$ . In order to do so, we will partition the tokens arriving at layer  $l$ , over the lifetime of the system, into *generations* of size  $w$ . We will show that as a group, each generation of tokens at layer  $l$  causes  $O(qn + qkw)$  stalls to other tokens. It then follows that an average generation receives  $O(qn + qkw)$  stalls. (If 10 people each throw 5 balls into the air, and all the balls are caught then the average person catches 5 balls.) Dividing by the number of tokens in a generation, it follows that the average token passing through  $l$  receives (or causes)  $O(qn/w + qk)$  stalls.

Let  $b$  be a balancer of  $l$  with output width  $r$ , where  $2 \leq r \leq q$ . We say that a token belongs to the  $g$ th generation of tokens arriving at  $b$  if it is one of the  $((g - 1)r + 1)$ th,  $\dots, ((g - 1)r + r)$ th tokens to arrive at  $b$ . Note that the  $g$ th generation of  $b$  has  $r$  tokens. The  $g$ th generation of  $l$  is the set of  $g$ th generation tokens of the balancers at layer  $l$ . Note that the  $g$ th generation of  $l$  has  $w$  tokens. We say that by time  $t$ , the  $g$ th generation has completed its arrival at  $l$  if for each balancer in  $l$  all the tokens of the  $g$ th generation have already arrived by that time. Finally, we say that at time  $t$  there are  $f$  tokens of the  $g$ th generation missing at layer  $l$  if by time  $t$  exactly  $w - f$  tokens of generation  $g$  have arrived at  $l$ .

**Lemma 4.1.5** *Let  $\mathcal{B}$  be in a quiescent state, and let  $g$  be the maximum generation such that some balancer  $b$  in layer  $l$  has received at least one generation  $g$  token. Then all balancers in  $l$  have received at least one generation  $g - k$  token.*

**Proof:** Since balancer  $b$  has received a generation  $g$  token the highest value in the output sequence of  $b$  is at least  $g$ . We will assume for contradiction that there is a balancer  $b'$  that hasn't received any generation  $g - k$  token. The highest value on the output sequence of  $b'$  is at most  $g - k - 1$ . Therefore there is an output wire of  $b$  and an output wire of  $b'$  with

difference at least  $g - (g - k - 1) = k + 1$ . This is a contradiction, since the output sequence of  $l$  is  $k$ -smooth. ■

**Lemma 4.1.6** *Let  $t$  be the time at which the first  $g$ th generation token arrives at  $l$ . Then the number of tokens of generations strictly less than  $g - k$  stuck at  $l$ , plus the number of tokens of generations strictly less than  $g - k$  still missing from layer  $l$ , is at most  $n$ .*

**Proof:** Run the network  $B$  to quiescence from its state at time  $t$ . Let  $g'$  be the maximum generation such that some balancer in layer  $l$  has received at least one generation  $g'$  token. Clearly  $g' \geq g$ . By Lemma 4.1.5, every balancer has received at least one token from generation  $g' - k \geq g - k$ . Thus, the lemma follows from the fact that at most  $n$  tokens (the maximum number of tokens in  $B$  at any time) were involved in moving  $B$  to a quiescent state. ■

Recall that when a token passes through a balancer it causes stalls to all tokens that are waiting at this balancer. By *stalls caused at layer  $l$  by generation  $g$  to generation  $g'$*  we refer to stalls incurred by tokens of generation  $g'$  when they are waiting at some balancer of layer  $l$  and some token of generation  $g$  passes. By *stalls caused at layer  $l$  between generation  $g$  and generation  $g'$*  we refer to stalls caused by generation  $g$  to generation  $g'$ , and vice versa.

**Lemma 4.1.7** *Consider the  $g$ th generation passing through layer  $l$ . The maximal number of stalls caused between this generation and generations less than or equal to  $g$  at this layer is at most  $qn + q(k + 1)w$ .*

**Proof:** Consider the first token of generation  $g$  to arrive at  $l$ . Say it arrives at time  $t$ . A generation  $g$  token can encounter (and hence cause a stall to or be stalled by)

- (1) tokens of generation strictly less than  $g - k$ ,
- (2) generation  $g - k, \dots, g$  tokens.

By Lemma 4.1.6, the total number of tokens of generation strictly less than  $g - k$  stuck at  $l$  or missing from  $l$  is at most  $n$ . Therefore, the type (1) tokens are at most  $n$ . The tokens of type (2) are at most  $(k + 1)w$ , since each generation has  $w$  tokens.

The number of stalls occurring between each token of generation  $g$  and tokens of generation less than or equal to  $g$  are at most the number of tokens of these generations that this token encounters at its balancer. Each token of generation less than or equal to  $g$  can be encountered by at most  $q$  tokens of generation  $g$  (since  $q$  is the maximum balancer width in  $l$ ). Therefore, we get  $qn$  stalls of type (1) and  $q(k + 1)w$  stalls of type (2). Summing, we get  $qn + q(k + 1)w$  stalls. ■

Since there are  $w$  tokens at any generation  $g$ , we have by Lemma 4.1.7 that the amortized contention endured by a token at layer  $l$  is at most  $(qn + q(k+1)w)/w$ . Subsequently, we have the following corollary.

**Corollary 4.1.8** *The amortized contention of layer  $l$  with concurrency  $n$  is at most*

$$\text{cont}(l, n) \leq \frac{qn}{w} + q(k+1).$$

As an application of the previous discussion, we will compute the contention of the butterfly network that was described in Section 4.1.2.

**Lemma 4.1.9** *The amortized contention of the butterfly network  $\mathcal{D}(w)$  with concurrency  $n$  is at most*

$$\text{cont}(\mathcal{D}(w), n) \leq \frac{2n \lg w}{w} + \lg^2 w + 3 \lg w.$$

**Proof:** Denote by  $l(w)$  the final layer of  $\mathcal{D}(w)$ . From the inductive construction of the network  $\mathcal{D}(w)$  (see Section 4.1.2) we have that a token first traverses one of the two  $\mathcal{D}_0(w/2)$  or  $\mathcal{D}_1(w/2)$  and then the layer  $l(w)$ . The concurrency of each  $\mathcal{D}_0(w/2)$  or  $\mathcal{D}_1(w/2)$  is equal to  $n/2$ , since each input wire accepts at most  $n/w$  processes. Furthermore, the concurrency of layer  $l(w)$  is  $n$ , since all processes traverse this layer. Therefore, the amortized contention incurred by any token is equal to

$$\text{cont}(\mathcal{D}(w), n) = \text{cont}\left(\mathcal{D}\left(\frac{w}{2}\right), \frac{n}{2}\right) + \text{cont}(l(w), n).$$

By Lemma 4.1.4, the output sequence of layer  $l$  has the  $\lg w$ -smooth property. Since layer  $l$  is made from  $(2, 2)$ -balancers, and the total output width of the layer is  $w$ , we have from Corollary 4.1.8 that

$$\text{cont}(l(w), n) \leq 2\frac{n}{w} + 2(\lg w + 1) = 2\left(\frac{n}{w} + \lg w + 1\right).$$

For the base case  $w = 2$ , the network  $\mathcal{D}(2)$  is just a  $(2, 2)$ -balancer with concurrency  $2n/w$ , which trivially has amortized contention at most  $2n/w$ .

Let  $k = \lg w$ . We have

$$\begin{aligned}
\text{cont}(\mathcal{D}(w), n) &= \text{cont}\left(\mathcal{D}\left(\frac{w}{2}\right), \frac{n}{2}\right) + \text{cont}(l(w), n) \\
&\leq \text{cont}\left(\mathcal{D}\left(\frac{w}{2}\right), \frac{n}{2}\right) + 2\left(\frac{n}{w} + k + 1\right) \\
&= \text{cont}\left(\mathcal{D}\left(\frac{w}{2^2}\right), \frac{n}{2^2}\right) + \text{cont}\left(l\left(\frac{w}{2}\right), \frac{n}{2}\right) + 2\left(\frac{n}{w} + k + 1\right) \\
&\leq \text{cont}\left(\mathcal{D}\left(\frac{w}{2^2}\right), \frac{n}{2^2}\right) + 2\left(\frac{n/2}{w/2} + (k-1) + 1\right) + 2\left(\frac{n}{w} + k + 1\right) \\
&= \text{cont}\left(\mathcal{D}\left(\frac{w}{2^2}\right), \frac{n}{2^2}\right) + 2\left(2\frac{n}{w} + 2k - 1 + 2\right) \\
&= \dots \\
&\leq \text{cont}\left(\mathcal{D}\left(\frac{w}{2^j}\right), \frac{n}{2^j}\right) + 2\left(j\frac{n}{w} + jk - \sum_{i=1}^{j-1} i + j\right) \\
&= \text{cont}\left(\mathcal{D}\left(\frac{w}{2^j}\right), \frac{n}{2^j}\right) + 2\left(j\frac{n}{w} + jk - \frac{j^2 - 3j}{2}\right) \\
&= \dots \\
&\leq \text{cont}\left(\mathcal{D}\left(\frac{w}{2^{k-1}}\right), \frac{n}{2^{k-1}}\right) \\
&\quad + 2\left((k-1)\frac{n}{w} + (k-1)k - \frac{(k-1)^2 - 3(k-1)}{2}\right) \\
&\leq 2\frac{n}{w} + 2\left((k-1)\frac{n}{w} + (k^2 - k) - \frac{(k^2 - 2k + 1) - 3(k-1)}{2}\right) \\
&\quad (\text{since } \text{cont}(\mathcal{D}(2), \frac{2n}{w}) \leq \frac{2n}{w}) \\
&= 2k\frac{n}{w} + k^2 + 3k - 4 \\
&\leq 2k\frac{n}{w} + k^2 + 3k,
\end{aligned}$$

as needed. ■

## 4.2 The Counting Network $\mathcal{C}(w, t)$

In this Section, we present the construction of a counting network  $\mathcal{C}(w, t)$ , where  $w = 2^k$ ,  $t = p2^l$ ,  $p \geq 1$  and  $1 \leq k \leq l$ .

The construction of network  $\mathcal{C}(w, t)$  is inductive, and has a similar structure as the bitonic network construction of Section 1.5.4. In the construction, we use two smaller counting networks and then we merge their outputs using the difference merging network of Section 4.3, as shown in Figure 4.3. As we will show in Section 4.3, the depth of the difference merging network depends only on the difference of the sums of the output sequences of the small counting networks. We can bound this difference by using a layer of balancers in the



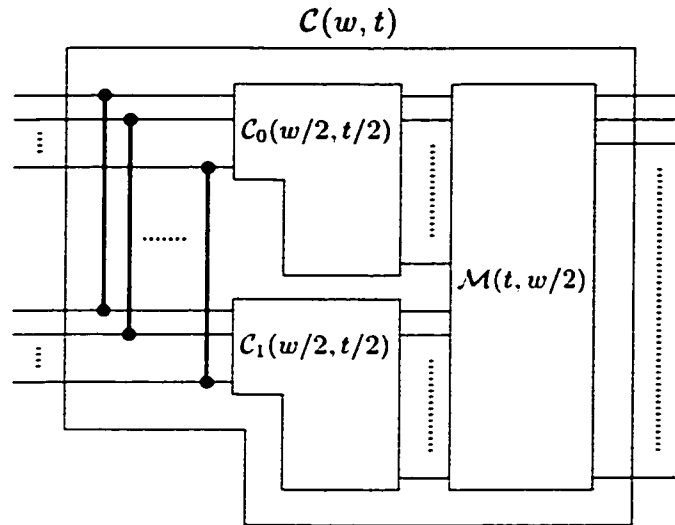


Figure 4.3: The counting network  $\mathcal{C}(w, t)$

inputs of the small counting networks (see Figure 4.3), so that the difference is at most  $w/2$ . Subsequently, the depth of the merging network  $\mathcal{M}$  will depend only on the input width  $w$ , and therefore, the depth of the whole counting network will depend only on  $w$ .

In detail, let  $X^{(w)}$  and  $Y^{(t)}$  denote the input and output sequences, respectively, of  $\mathcal{C}(w, t)$ . The construction of  $\mathcal{C}(w, t)$  is by induction on  $w$ . For the base case  $w = 2$  the network  $\mathcal{C}(2, t)$  is just a  $(2, t)$ -balancer. For the inductive case, we assume that we are given the networks  $\mathcal{C}(w', t')$ , for all  $2 \leq w' < w$  and any  $t'$ . The network  $\mathcal{C}(w, t)$  is constructed as follows (see Figure 4.3). We take  $w/2$   $(2, 2)$ -balancers  $b_0, \dots, b_{w/2-1}$ . The first and second input wires of balancer  $b_i$  are connected to  $x_i$  and  $x_{i+w/2}$ , respectively, for all  $0 \leq i < w/2$ . Next, we take two copies of  $\mathcal{C}(w/2, t/2)$ , given by the induction hypothesis, denoted as  $\mathcal{C}_0(w/2, t/2)$  and  $\mathcal{C}_1(w/2, t/2)$ . Denote by  $E^{(w/2)}$  and  $G^{(t/2)}$  the respective input and output sequences of network  $\mathcal{C}_0(w/2, t/2)$ , and by  $F^{(w/2)}$  and  $H^{(t/2)}$  the respective input and output sequences of network  $\mathcal{C}_1(w/2, t/2)$ . The first output wire of balancer  $b_i$  is connected to the  $i$ th input wire  $e_i$  of  $\mathcal{C}_0(w/2, t/2)$  and the second output wire to the  $i$ th input wire  $f_i$  of network  $\mathcal{C}_1(w/2, t/2)$ , for all  $0 \leq i < w/2$ . Next, we take a copy of the difference merging network  $\mathcal{M}(t, w/2)$  described below in Section 4.3. The first input sequence of network  $\mathcal{M}(t, w/2)$  is the output sequence  $G^{(t/2)}$  of  $\mathcal{C}_0(w/2, t/2)$ , and the second input sequence is the output sequence  $H^{(t/2)}$  of  $\mathcal{C}_1(w/2, t/2)$ . The output sequence of  $\mathcal{M}(t, w/2)$  is the output sequence  $Y^{(t)}$  of network  $\mathcal{C}(w, t)$ . This completes the construction.

We continue by showing the correctness of  $\mathcal{C}(w, t)$ .

**Theorem 4.2.1** *The network  $\mathcal{C}(w, t)$  is a counting network.*

**Proof:** We will show that for any input sequence  $X^{(w)}$  to network  $\mathcal{C}(w, t)$  the output sequence  $Y^{(t)}$  satisfies the step property.

As described earlier, the construction of  $\mathcal{C}(w, t)$  is by induction on  $w$ . For the base case  $w = 2$ , the network  $\mathcal{C}(2, t)$  is just a  $(2, t)$ -balancer, whose correctness is guaranteed by its definition. Therefore, we only need to show that the network  $\mathcal{C}(w, t)$  is a counting network for  $w > 2$ .

For  $w > 2$  the network  $\mathcal{C}(w, t)$  consists of a layer of balancers, followed by the networks  $\mathcal{C}_0(w/2, t/2)$  and  $\mathcal{C}_1(w/2, t/2)$ . By the induction hypothesis, we have that the respective outputs  $G^{(t/2)}$  and  $H^{(t/2)}$  of  $\mathcal{C}_0(w/2, t/2)$  and  $\mathcal{C}_1(w/2, t/2)$ , satisfy the step property. These sequences are fed to the inputs of the network  $\mathcal{M}(t, w/2)$  whose output is the sequence  $Y^{(t)}$ . Since by Proposition 4.3.2 the network  $\mathcal{M}(t, w/2)$  is a difference merging network, the sequence  $Y^{(t)}$  will have the step property if  $0 \leq \Sigma(G^{(t/2)}) - \Sigma(H^{(t/2)}) \leq w/2$ .

We only need to show that  $0 \leq \Sigma(G^{(t/2)}) - \Sigma(H^{(t/2)}) \leq w/2$ . By the sum preservation property of networks  $\mathcal{C}_0(w/2, t/2)$  and  $\mathcal{C}_1(w/2, t/2)$ , we have for their respective input sequences  $E^{(w/2)}$  and  $F^{(w/2)}$

$$\begin{aligned}\Sigma(E^{(w/2)}) &= \Sigma(G^{(t/2)}), \text{ and} \\ \Sigma(F^{(w/2)}) &= \Sigma(H^{(t/2)}),\end{aligned}$$

and thus we only need to show that  $0 \leq \Sigma(E^{(w/2)}) - \Sigma(F^{(w/2)}) \leq w/2$ .

The sequences  $E^{(w/2)}$  and  $F^{(w/2)}$  are connected to the outputs of the  $(2, 2)$ -balancers  $b_0, \dots, b_{w/2-1}$  so that the first output wire of  $b_i$  is connected to  $e_i$  and the second to  $f_i$ , for all  $0 \leq i \leq w/2 - 1$ . Since the outputs of balancer  $b_i$  have the step property for any input sequence  $X^{(w)}$ , we have that  $0 \leq e_i - f_i \leq 1$ , for all  $0 \leq i < w/2$ . By summing these inequalities for all the  $w/2$  balancers we have

$$\begin{aligned}0 &\leq (e_0 + \dots + e_{w/2-1}) - (f_0 + \dots + f_{w/2-1}) \leq \frac{w}{2} \implies \\ 0 &\leq \Sigma(E^{(w/2)}) - \Sigma(F^{(w/2)}) \leq \frac{w}{2}.\end{aligned}$$

Subsequently, the sequence  $Y^{(t)}$  has the step property, as needed. ■

Next, we calculate the depth of  $\mathcal{C}(w, t)$ . We show that the depth of this network depends only on the input width  $w$ . This is because, as we will show in the next section, the depth of the difference merging network  $\mathcal{M}(t, w/2)$  depends only on the difference between the sums of the two sequences it merges, which is at most  $w/2$ , and thus the depth depends only on  $w$ .

**Theorem 4.2.2**

$$\text{depth}(\mathcal{C}(w, t)) = \frac{\lg^2 w + \lg w}{2}.$$

**Proof:** As described earlier, the construction of  $\mathcal{C}(w, t)$  is by induction on  $w$ . For the base case  $w = 2$ , the network  $\mathcal{C}(2, t)$  consists of a single  $(2, t)$ -balancer and thus its depth is 1. For the general case  $w > 2$ , the network  $\mathcal{C}(w, t)$  consists of a layer of  $(2, 2)$ -balancers with depth 1, followed by two “parallel” copies of network  $\mathcal{C}(w/2, t/2)$ , followed by the network

$\mathcal{M}(t, w/2)$  whose depth is given in Proposition 4.3.3. Therefore, we have the recurrence:

$$\begin{aligned}
\text{depth}(\mathcal{C}(w, t)) &= 1 + \text{depth}\left(\mathcal{C}\left(\frac{w}{2}, \frac{t}{2}\right)\right) + \text{depth}\left(\mathcal{M}\left(t, \frac{w}{2}\right)\right) \\
&= 1 + \text{depth}\left(\mathcal{C}\left(\frac{w}{2}, \frac{t}{2}\right)\right) + \lg \frac{w}{2} \\
&\quad (\text{since by Proposition 4.3.3 } \text{depth}(\mathcal{M}(t, \frac{w}{2})) = \lg \frac{w}{2}) \\
&= 1 + \left(1 + \text{depth}\left(\mathcal{C}\left(\frac{w}{4}, \frac{t}{4}\right)\right) + \text{depth}\left(\mathcal{M}\left(\frac{t}{2}, \frac{w}{4}\right)\right)\right) + \lg \frac{w}{2} \\
&= 1 + \left(1 + \text{depth}\left(\mathcal{C}\left(\frac{w}{4}, \frac{t}{4}\right)\right) + \lg \frac{w}{4}\right) + \lg \frac{w}{2} \\
&\quad (\text{since by Proposition 4.3.3 } \text{depth}(\mathcal{M}(\frac{t}{2}, \frac{w}{4})) = \lg \frac{w}{4}) \\
&= 2 + \text{depth}\left(\mathcal{C}\left(\frac{w}{2^2}, \frac{t}{2^2}\right)\right) + \sum_{i=1}^2 \lg \frac{w}{2^i} \\
&= \dots \\
&= k + \text{depth}\left(\mathcal{C}\left(\frac{w}{2^k}, \frac{t}{2^k}\right)\right) + \sum_{i=1}^k \lg \frac{w}{2^i} \\
&= k + \text{depth}\left(\mathcal{C}\left(\frac{w}{2^k}, \frac{t}{2^k}\right)\right) + k \lg w - \sum_{i=1}^k \lg 2^i \\
&= k + \text{depth}\left(\mathcal{C}\left(\frac{w}{2^k}, \frac{t}{2^k}\right)\right) + k \lg w - \sum_{i=1}^k i \\
&= k + \text{depth}\left(\mathcal{C}\left(\frac{w}{2^k}, \frac{t}{2^k}\right)\right) + k \lg w - \frac{k^2}{2} - \frac{k}{2} \\
&= \dots \\
&= \lg w - 1 + \text{depth}\left(\mathcal{C}\left(\frac{w}{2^{\lg w - 1}}, \frac{t}{2^{\lg w - 1}}\right)\right) + (\lg w - 1) \lg w \\
&\quad - \frac{(\lg w - 1)^2}{2} - \frac{\lg w - 1}{2} \\
&= \lg w - 1 + \text{depth}\left(\mathcal{C}\left(2, \frac{2t}{w}\right)\right) + \lg^2 w - \lg w \\
&\quad - \frac{\lg^2 w - 2 \lg w + 1}{2} - \frac{\lg w - 1}{2} \\
&= -1 + 1 + \lg^2 w - \frac{\lg^2 w - \lg w}{2} \\
&\quad (\text{since } \text{depth}(\mathcal{C}(2, \frac{2t}{w})) = 1) \\
&= \frac{\lg^2 w + \lg w}{2},
\end{aligned}$$

as needed. ■

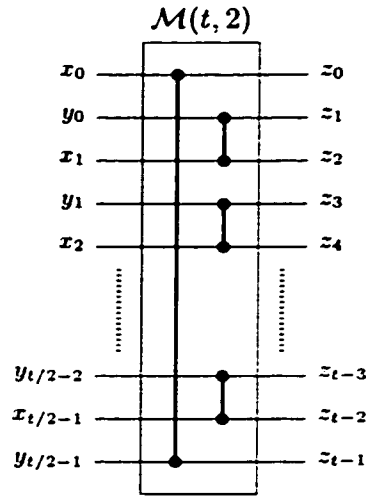


Figure 4.4: The difference merging network  $\mathcal{M}(t, 2)$

### 4.3 The Difference Merging Network $\mathcal{M}(t, \delta)$

In this section we present the construction of a difference merging network  $\mathcal{M}(t, \delta)$ , where  $t = p2^l$ ,  $\delta = 2^k$ ,  $p \geq 1$  and  $1 \leq k < l$ . Let  $X^{(t/2)}$  and  $Y^{(t/2)}$  denote the first and second input sequences, respectively, of  $\mathcal{M}(t, \delta)$  and let  $Z^{(t)}$  denote its output sequence.

The construction is by induction on  $\delta$ . For the base case  $\delta = 2$ , the network  $\mathcal{M}(t, 2)$  consists of a single layer of  $t/2$   $(2, 2)$ -balancers  $b_0, \dots, b_{t/2-1}$  (see Figure 4.4). For  $1 \leq i < t/2$ , the first and second input wires of balancer  $b_i$  are connected to  $y_{i-1}$  and  $x_i$ , respectively, and the first and second output wires are connected to  $z_{2i-1}$  and  $z_{2i}$ , respectively. For balancer  $b_0$ , the first and second input wires are connected to  $x_0$  and  $y_{t/2-1}$ , respectively, the first and second output wires are connected to  $z_0$  and  $z_{t-1}$ , respectively.

For the inductive case  $\delta > 2$ , we assume that we have constructed the networks  $\mathcal{M}(t', \delta')$ , for all  $2 \leq \delta' < \delta$  and any  $t'$ . The network  $\mathcal{M}(t, \delta)$  is constructed as follows (see Figure 4.5). We take two copies of the network  $\mathcal{M}(t/2, \delta/2)$  denoted as  $\mathcal{M}_0(t/2, \delta/2)$  and  $\mathcal{M}_1(t/2, \delta/2)$ , that are given by the induction hypothesis. The first input sequence of  $\mathcal{M}_0(t/2, \delta/2)$ , is the even subsequence of  $X^{(t/2)}$ , namely  $X_e^{(t/4)}$ , and the second input sequence is the even subsequence of  $Y^{(t/2)}$ , namely  $Y_e^{(t/4)}$ . The first input sequence of  $\mathcal{M}_1$  is the odd subsequence of  $X^{(t/2)}$ , namely  $X_o^{(t/4)}$ , and the second input sequence is the odd subsequence of  $Y^{(t/2)}$ , namely  $Y_o^{(t/4)}$ . Let  $G^{(t/2)}$  and  $H^{(t/2)}$  denote the output sequences of networks  $\mathcal{M}_0(t/2, \delta/2)$  and  $\mathcal{M}_1(t/2, \delta/2)$ , respectively. Next, we take a copy of the network  $\mathcal{M}(t, 2)$  which is given by the induction base. The first input sequence of  $\mathcal{M}(t, 2)$  is the output sequence

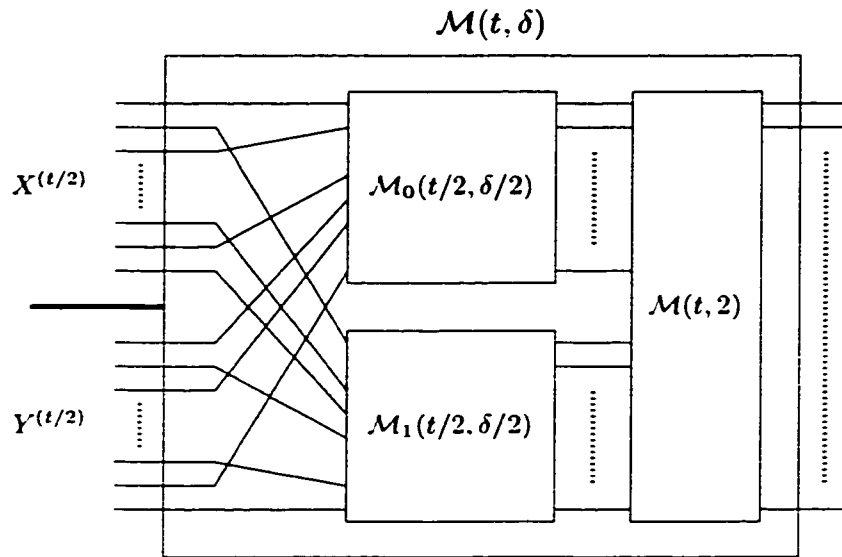


Figure 4.5: The difference merging network  $\mathcal{M}(t, \delta)$

$G^{(t/2)}$  of  $\mathcal{M}_0(t/2, \delta/2)$  and the second input sequence is the output sequence  $H^{(t/2)}$  of  $\mathcal{M}_1(t/2, \delta/2)$ . Finally, the output sequence of  $\mathcal{M}(t, 2)$  is the output sequence  $Z^{(t)}$  of  $\mathcal{M}(t, \delta)$ . This completes the description of the construction.

We continue by showing the correctness of  $\mathcal{M}(t, \delta)$ . We start by showing the correctness of  $\mathcal{M}(t, 2)$ .

**Proposition 4.3.1** *The network  $\mathcal{M}(t, 2)$  is a difference merging network.*

**Proof:** Let's assume that each of the input sequences  $X^{(t/2)}$  and  $Y^{(t/2)}$  of network  $\mathcal{M}(t, 2)$  satisfies the step property and  $0 \leq \Sigma(X^{(t/2)}) - \Sigma(Y^{(t/2)}) \leq 2$ . We will show that the output sequence  $Z^{(t)}$  satisfies the step property.

Denote by  $a$  and  $b$  the highest value of the elements of sequence  $X^{(t/2)}$  and  $Y^{(t/2)}$ , respectively. Let  $k$  and  $l$  be the step points of sequences  $X^{(t/2)}$  and  $Y^{(t/2)}$ , respectively. By Lemma 4.1.1, since  $t/2 \geq 2$ , we have  $0 \leq a - b \leq 1$ . Next, we continue by considering separately the cases  $a = b$  and  $a = b + 1$ .

First, consider the case  $a = b$ . By Lemma 4.1.2, we have  $0 \leq k - l \leq 2$ . There are three cases:  $k = l$ ,  $k = l + 1$  and  $k = l + 2$ . For each of these cases, we see in Table 4.1 the input and output values of all the balancers  $b_i$ , for  $0 \leq i < t/2$ , for input and output wires 0 and 1. (In the table we assume that  $1 < k$  and  $l < t/2$ . The analysis for  $k = 1$  or  $l = t/2$  is similar.) Consider now the case  $k = l$ . According to the table, since the outputs of the balancers form the output sequence  $Z^{(t)}$ , the first  $2k$  elements of sequence  $Z^{(t)}$  will have

	connections		$k = l$		$k = l + 1$		$k = l + 2$	
Balancer $b_i$	input wire		input value		input value		input value	
	0	1	0	1	0	1	0	1
$i = 0$	$x_0$	$y_{t/2-1}$	$a$	$a - 1$	$a$	$a - 1$	$a$	$a - 1$
$1 \leq i < k - 1$	$y_{i-1}$	$x_i$	$a$	$a$	$a$	$a$	$a$	$a$
$i = k - 1$	$y_{k-2}$	$x_{k-1}$	$a$	$a$	$a$	$a$	$a - 1$	$a$
$i = k$	$y_{k-1}$	$x_k$	$a$	$a - 1$	$a - 1$	$a - 1$	$a - 1$	$a - 1$
$k + 1 \leq i < t/2$	$y_{i-1}$	$x_i$	$a - 1$	$a - 1$	$a - 1$	$a - 1$	$a - 1$	$a - 1$
Balancer $b_i$	output wire		output value		output value		output value	
	0	1	0	1	0	1	0	1
$i = 0$	$z_0$	$z_{t-1}$	$a$	$a - 1$	$a$	$a - 1$	$a$	$a - 1$
$1 \leq i < k - 1$	$z_{2i-1}$	$z_{2i}$	$a$	$a$	$a$	$a$	$a$	$a$
$i = k - 1$	$z_{2k-3}$	$z_{2k-2}$	$a$	$a$	$a$	$a$	$a$	$a - 1$
$i = k$	$z_{2k-1}$	$z_{2k}$	$a$	$a - 1$	$a - 1$	$a - 1$	$a - 1$	$a - 1$
$k + 1 \leq i < t/2$	$z_{2i-1}$	$z_{2i}$	$a - 1$	$a - 1$	$a - 1$	$a - 1$	$a - 1$	$a - 1$

Table 4.1: The case  $a = b$ 

	connections		values	
Balancer $b_i$	input wire		input	
	0	1	0	1
$i = 0$	$x_0$	$y_{t/2-1}$	$a$	$a - 2$
$1 \leq i < t/2$	$y_{i-1}$	$x_i$	$a - 1$	$a - 1$
Balancer $b_i$	output wire		output	
	0	1	0	1
$i = 0$	$z_0$	$z_{t-1}$	$a - 1$	$a - 1$
$1 \leq i < t/2$	$z_{2i-1}$	$z_{2i}$	$a - 1$	$a - 1$

Table 4.2: The case  $a = b + 1$ ,  $k = 1$ ,  $l = t/2 - 1$ 

value  $a$  and the rest will have value  $a - 1$ . Therefore, the sequence  $Z^{(t)}$  satisfies the step property. In a similar way, we can show for the rest of the cases that the output sequence  $Z^{(t)}$  satisfies the step property.

Next, consider the case  $a = b + 1$ . By Lemma 4.1.2, either  $l = t/2$  and  $1 \leq k \leq 2$ , or  $l = t/2 - 1$  and  $k = 1$ . If  $l = t/2$  there are two possibilities:  $k = 1$  or  $k = 2$ . By doing an analysis similar to the case where  $a = b$  and  $k = l + 1$  or  $k = l + 2$ , described above, we have that the output sequence  $Z^{(t)}$  satisfies the step property. If  $l = t/2 - 1$  and  $k = 1$ , the inputs and outputs of balancers  $b_i$ ,  $0 \leq i < t/2$ , are shown in Table 4.2. According to the table, the outputs of all the balancers have values equal to  $a - 1$ , and subsequently the output sequence  $Z^{(t)}$  satisfies the step property.

Therefore, in all cases the output sequence  $Z^{(t)}$  satisfies the step property, as needed. ■

We continue by showing the correctness of network  $\mathcal{M}(t, \delta)$ , for any  $\delta$ .

**Proposition 4.3.2** *The network  $\mathcal{M}(w, \delta)$  is a difference merging network.*

**Proof:** Let's assume that each of the input sequences  $X^{(t/2)}$  and  $Y^{(t/2)}$  of network  $\mathcal{M}(t, \delta)$  satisfies the step property and  $0 \leq \Sigma(X^{(t/2)}) - \Sigma(Y^{(t/2)}) \leq \delta$ . We will show that the output sequence  $Z^{(t)}$  satisfies the step property.

As described earlier, the construction of  $\mathcal{M}(t, \delta)$  is by induction on  $\delta$ . By Proposition 4.3.1, for the base case  $\delta = 2$  the network  $\mathcal{M}(t, 2)$  is a difference merging network. Therefore, we only need to show that the network  $\mathcal{M}(t, \delta)$  is a merging network for any  $\delta > 2$ .

For  $\delta > 2$ , the network  $\mathcal{M}(t, \delta)$  consists of the networks  $\mathcal{M}_0(t/2, \delta/2)$  and  $\mathcal{M}_1(t/2, \delta/2)$ , whose correctness is guaranteed by the induction hypothesis, and from the network  $\mathcal{M}(t, 2)$  whose correctness was given in Proposition 4.3.1. The respective outputs of  $\mathcal{M}_0(t/2, \delta/2)$  and  $\mathcal{M}_1(t/2, \delta/2)$ , denoted  $G^{(t/2)}$  and  $H^{(t/2)}$ , are fed to the inputs of the network  $\mathcal{M}(t, 2)$ . Since  $Z^{(t)}$  is the output sequence of  $\mathcal{M}(t, 2)$  and this network is a difference merging network, the sequence  $Z^{(t)}$  will have the step property if each of  $G^{(t/2)}$  and  $H^{(t/2)}$  satisfy the step property and  $0 \leq \Sigma(G^{(t/2)}) - \Sigma(H^{(t/2)}) \leq 2$ .

First we show that the sequence  $G^{(t/2)}$ , satisfies the step property. The input sequences of network  $\mathcal{M}_0(t/2, \delta/2)$  are the even subsequences of  $X^{(t/2)}$  and  $Y^{(t/2)}$ , namely  $X_e^{(t/4)}$  and  $Y_e^{(t/4)}$ . Since network  $\mathcal{M}_0(t/2, \delta/2)$  is a difference merging network, the sequence  $G^{(t/2)}$  has the step property if each of  $X_e^{(t/4)}$  and  $Y_e^{(t/4)}$  have the step property and  $0 \leq \Sigma(X_e^{(t/4)}) - \Sigma(Y_e^{(t/4)}) \leq \delta/2$ . Since we assumed that  $X^{(t/2)}$  and  $Y^{(t/2)}$  satisfy the step property, we have by Observation 1.4.1 that each of the subsequences  $X_e^{(t/4)}$  and  $Y_e^{(t/4)}$  have the step property. Furthermore, since we assumed that  $0 \leq \Sigma(X^{(t/2)}) - \Sigma(Y^{(t/2)}) \leq \delta$ , we have by Lemma 4.1.3 that  $0 \leq \Sigma(X_e^{(t/4)}) - \Sigma(Y_e^{(t/4)}) \leq \delta/2$ . Subsequently, the sequence  $G^{(t/2)}$  satisfies the step property. In a similar way, we can show that the sequence  $H^{(t/2)}$  satisfies the step property too.

Now we show that  $0 \leq \Sigma(G^{(t/2)}) - \Sigma(H^{(t/2)}) \leq 2$ . By the sum preservation property of networks  $\mathcal{M}_0(t/2, \delta/2)$  and  $\mathcal{M}_1(t/2, \delta/2)$  we have

$$\begin{aligned}\Sigma(G^{(t/2)}) &= \Sigma(X_e^{(t/4)}) + \Sigma(Y_e^{(t/4)}), \text{ and} \\ \Sigma(H^{(t/2)}) &= \Sigma(X_o^{(t/4)}) + \Sigma(Y_o^{(t/4)}).\end{aligned}$$



By Observation 4.1.1 we have

$$\begin{aligned} 0 &\leq \Sigma(X_e^{(t/4)}) - \Sigma(X_o^{(t/4)}) \leq 1, \text{ and} \\ 0 &\leq \Sigma(Y_e^{(t/4)}) - \Sigma(Y_o^{(t/4)}) \leq 1. \end{aligned}$$

Adding these two inequalities we get

$$\begin{aligned} 0 &\leq (\Sigma(X_e^{(t/4)}) + \Sigma(Y_e^{(t/4)})) - (\Sigma(X_o^{(t/4)}) + \Sigma(Y_o^{(t/4)})) \leq 2 \implies \\ 0 &\leq \Sigma(G^{(t/2)}) - \Sigma(H^{(t/2)}) \leq 2. \end{aligned}$$

Since each of  $G^{(t/2)}$  and  $H^{(t/2)}$  satisfy the step property and  $0 \leq \Sigma(G^{(t/2)}) - \Sigma(H^{(t/2)}) \leq 2$ , the output sequence  $Z^{(t)}$  satisfies the step property, as needed. ■

We continue by calculating the depth of network  $\mathcal{M}(t, \delta)$ . We show that the depth depends only on the difference  $\delta$ .

**Proposition 4.3.3**  $\text{depth}(\mathcal{M}(t, \delta)) = \lg \delta$ .

**Proof:** As described above, the construction of  $\mathcal{M}(t, \delta)$  is by induction on  $\delta$ . For the base case  $\delta = 2$ , the network  $\mathcal{M}(t, 2)$  consists of a single layer of balancers and thus its depth is 1. For the general case  $\delta > 2$ , the network  $\mathcal{M}(t, \delta)$  consists of two “parallel” copies of network  $\mathcal{M}(t/2, \delta/2)$ , connected in series with the network  $\mathcal{M}(t, 2)$ . Therefore, we have the

recurrence:

$$\begin{aligned}
\text{depth}(\mathcal{M}(t, \delta)) &= \text{depth}\left(\mathcal{M}\left(\frac{t}{2}, \frac{\delta}{2}\right)\right) + \text{depth}(\mathcal{M}(w, 2)) \\
&= \text{depth}\left(\mathcal{M}\left(\frac{t}{2}, \frac{\delta}{2}\right)\right) + 1 \\
&\quad (\text{since } \text{depth}(\mathcal{M}(t, 2)) = 1) \\
&= \text{depth}\left(\mathcal{M}\left(\frac{t}{4}, \frac{\delta}{4}\right)\right) + 1 + 1 \\
&= \text{depth}\left(\mathcal{M}\left(\frac{t}{2^2}, \frac{\delta}{2^2}\right)\right) + 2 \\
&= \dots \\
&= \text{depth}\left(\mathcal{M}\left(\frac{t}{2^k}, \frac{\delta}{2^k}\right)\right) + k \\
&= \dots \\
&= \text{depth}\left(\mathcal{M}\left(\frac{t}{2^{\lg \delta - 1}}, \frac{\delta}{2^{\lg \delta - 1}}\right)\right) + \lg \delta - 1 \\
&= \text{depth}\left(\mathcal{M}\left(\frac{2t}{\delta}, 2\right)\right) + \lg \delta - 1 \\
&= 1 + \lg \delta - 1 \\
&\quad (\text{since } \text{depth}(\mathcal{M}(\frac{2t}{\delta}, 2)) = 1) \\
&= \lg \delta,
\end{aligned}$$

as needed. ■

#### 4.4 Contention Analysis of $\mathcal{C}(w, t)$

In this section we compute the amortized contention of the counting network  $\mathcal{C}(w, t)$ , described in Section 4.1.3. When we unfold the construction of the network  $\mathcal{C}(w, t)$  we see that it consists of three parts  $A$ ,  $B$  and  $C$ , as shown in Figure 4.1. Part  $A$  has input and output width  $w$  and it consists of  $\lg w - 1$  layers of  $(2, 2)$ -balancers. Part  $B$  has input width  $w$ , output width  $t$ , and is a single layer of  $(2, 2t/w)$ -balancers. Finally, part  $C$  has input and output width  $t$  and it consists of  $\text{depth}(\mathcal{C}(w, t)) - \lg w = O(\lg^2 w)$  layers of  $(2, 2)$ -balancers.

Let

$$s = \left\lfloor \frac{w \lg w}{t} \right\rfloor + 2.$$

We first show that the output sequence of part  $B$  has the  $s$ -smooth property.

**Lemma 4.4.1** *The output sequence of part  $B$  has the  $s$ -smooth property.*

**Proof:** To show this, replace temporarily each  $(2, 2t/w)$ -balancer of part  $B$  with a  $(2, 2)$ -balancer. Now, we can easily see that the combination of parts  $A$  and  $B$  gives us a network that is isomorphic to the butterfly network  $\mathcal{D}(w)$ , described in Section 4.1.2. By Lemma 4.1.4, the output sequence of part  $B$  has the  $\log w$ -smooth property. Denote by  $b_0, \dots, b_{w/2-1}$  the  $(2, 2)$ -balancers of part  $B$ . Let  $X_i^{(2)}$  denote the output sequence of balancer  $b_i$ . Since the output of part  $B$  has the  $\log w$ -smooth property, we have that  $|\Sigma(X_i^{(2)}) - \Sigma(X_j^{(2)})| \leq 2 \lg w$ , for any  $0 \leq i, j < w/2$  (the factor 2 in  $\lg w$  comes from the fact each balancer has two output wires).

Now, restore the old  $(2, 2t/w)$ -balancers to part  $B$ . Denote by  $\hat{b}_0, \dots, \hat{b}_{w/2-1}$  these balancers and by  $\hat{X}_0^{(2t/w)}, \dots, \hat{X}_{w/2-1}^{(2t/w)}$  their respective output sequences. For any balancer  $\hat{b}_i$ , the only change from  $b_i$  is the number of output wires. Therefore, the total sum of tokens that leave from any balancer in both cases is the same. That is,  $\Sigma(X_i^{(2)}) = \Sigma(\hat{X}_i^{(2t/w)})$ . Subsequently,  $|\Sigma(\hat{X}_i^{(2t/w)}) - \Sigma(\hat{X}_j^{(2t/w)})| \leq 2 \lg w$  for any two balancers  $\hat{b}_i$  and  $\hat{b}_j$ . By Lemma 4.1.1, the maximum values on any two output wires, one wire from balancer  $\hat{b}_i$  and the other from balancer  $\hat{b}_j$ , will differ by at most  $\lfloor (2 \lg w)/(2t/w) \rfloor + 1$ . Therefore, the maximum difference between any two output wires is  $\lfloor w \lg w/t \rfloor + 2 = s$ , and subsequently the output sequence of part  $B$  has the  $s$ -smooth property, as needed. ■

Next, we compute the contention of the network  $\mathcal{C}(w, t)$ .

**Theorem 4.4.2** *The contention of network  $\mathcal{C}(w, t)$  with concurrency  $n$  is at most*

$$\text{cont}(\mathcal{C}(w, t), n) \leq \frac{2n \lg w}{w} + \frac{n \lg^2 w}{t} + \frac{w \lg^3 w}{t} + 4 \lg^2 w + 3 \lg w.$$

**Proof:** Let  $AB$  denote the combined network of parts  $A$  and  $B$ . By the construction of  $\mathcal{C}(w, t)$  we have

$$\text{cont}(\mathcal{C}(w, t), n) = \text{cont}(AB, n) + \text{cont}(C, n).$$

As we did in the proof of Lemma 4.4.1, we first replace the balancers of part  $B$  with  $(2, 2)$ -balancers. The resulting  $AB$  network is isomorphic to the butterfly network  $\mathcal{D}(w)$ . Subsequently, by Lemma 4.1.9,

$$\text{cont}(AB, n) \leq \frac{2n \lg w}{w} + \lg^2 w + 3 \lg w.$$

This contention remains the same when we restore the original  $(2, 2t/w)$ -balancers to part  $B$  (since until the layer of  $B$ , and in the layer of  $B$ , a token can't see any difference in the number of stalls it incurs).

Now, consider part  $C$ . Trivially, the concurrency for every layer of  $C$  is  $n$ . By Lemma 4.4.1 the input sequence to part  $C$  has the  $s$ -smooth property. It is easy to observe that, since the input sequence of part  $C$  has the  $s$ -smooth property, the output sequence of each layer of part  $C$  will also have the  $s$ -smooth property. Therefore, by corollary 4.1.8, every layer of  $C$  has amortized contention at most  $2n/t + 2(s+1)$ . By Theorem 4.2.2, the number of layers of part  $C$  is

$$\text{depth}(C(w, t)) - \lg w = \frac{\lg^2 w - \lg w}{2}.$$

The total contention of part  $C$  is equal to the contention of a layer multiplied by the number of layers. Subsequently, we have

$$\begin{aligned} \text{cont}(C, n) &\leq \left(2\frac{n}{t} + 2(s+1)\right) \frac{\lg^2 w - \lg w}{2} \\ &\leq \frac{n \lg^2 w}{t} + s \lg^2 w + \lg^2 w \\ &\leq \frac{n \lg^2 w}{t} + \frac{w \lg^3 w}{t} + 3 \lg^2 w \end{aligned}$$

Adding the contentions from parts  $AB$  and  $C$  we have,

$$\begin{aligned} \text{cont}(C(w, t), n) &= \text{cont}(AB, n) + \text{cont}(C, n) \\ &\leq \frac{2n \lg w}{w} + \lg^2 w + 3 \lg w + \frac{n \lg^2 w}{t} + \frac{w \lg^3 w}{t} + 3 \lg^2 w \\ &= \frac{2n \lg w}{w} + \frac{n \lg^2 w}{t} + \frac{w \lg^3 w}{t} + 4 \lg^2 w + 3 \lg w. \end{aligned}$$

as needed. ■

## 4.5 Related Work

There are only two other known counting network constructions with different input and output width. The first construction is given by Shavit and Zemach [49] and is has the form of a binary tree with 1 input wire and  $w$  output wires, and depth  $\lg w$ , built from  $(1, 2)$ -balancers. The second construction is given by Aiello *et al.* [4] and it has input width  $w$ , output width  $w \lg w$ , depth  $O(\lg w)$ , and is build from  $(2, 2)$ -balancers and  $(1, 2)$ -balancers. However, this construction uses as a building block the AKS sorting network and it is of no practical use.

## 4.6 Discussion

We presented a counting network construction with  $w$  input wires and  $t$  output wires, where  $w = 2^k$ ,  $t = p2^l$ , and  $w \leq t$ . This is one of a very few constructions known whose output width is *not* a power of two.

Several interesting questions remain. Is it possible to extend our construction to arbitrary input and output widths, other than multiples of a power of two? It follows from impossibility results in [2, 21] that appropriate balancer sizes would have to be used for such extension. Using such larger balancers is often expected to cause a reduction in depth, as in the construction of Chapter 3. What would be a trade-off between depth and contention in this situation?

## Chapter 5

# Introduction to Greedy Hot-Potato Routing in the Mesh

### 5.1 Greedy Hot-Potato Routing

Routing is essential in parallel multiprocessors: the parallel processors need to exchange messages while they solve a computational problem. The messages are sent in *packets*, where each packet has a header, with necessary routing information, and a message body. The packets have to be routed from their source to their destination processor through an underlying network that connects the processors. It is desirable that the packets arrive at their destinations as fast as possible.

One of the simplest networks for parallel multiprocessors is the *2-dimensional mesh network*, which is the network we consider here. The mesh network is an  $n \times n$  array of nodes, where each node corresponds to a processor. As shown in Figure 5.1, each node is connected to its adjacent nodes by a link (except of the nodes at the edges). We model the network so that the time is discrete and the nodes are *synchronized*: at each time step each node receives packets from its adjacent nodes, then makes routing decisions, and then forwards packets to its adjacent nodes according to the routing decisions. At each time step, a node is allowed to send at most one packet per link.

In traditional store-and-forward routing algorithms, the nodes of the network have buffers. Usually, the packets follow prespecified paths and a packet may wait in the buffer of a node until the link it wishes to follow is free.

Here, we study another kind of routing algorithms that we call *hot-potato routing algorithms*. In hot-potato routing the nodes have no buffers to store messages in transit: any

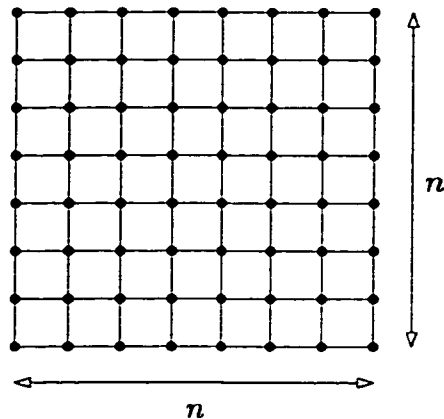


Figure 5.1: The  $n \times n$  mesh network

packet that arrives at a node other than its destination must immediately be forwarded to another adjacent node. Namely, a packet is treated by the nodes like a “hot potato,” which is too hot to keep. Hot-potato routing was first proposed by Baran [8].

Hot-potato routing algorithms are interesting because they have been observed to work well in practice: hot-potato routing algorithms have been used in parallel machines such as the HEP multiprocessor [51], the Connection machine [33], and the Caltech Mosaic C [47], as well as high speed communication networks [45]. Hot-potato routing algorithms are well-suited for optical networks [1, 29, 45, 53, 54] because it is difficult to buffer optical messages.

A special kind of hot-potato routing algorithms, that we consider here, are the so called *greedy* algorithms [8, 12]. In greedy hot-potato routing, the packets simply try to get closer to the destinations. In particular, at any time step in a node, a packet always tries to follow any link that brings it closer to its destination. In case the packet cannot follow any such a link, because other advancing packets will occupy these links, then it is forced to follow some other link that takes it further away from its destination, in which case we say that the packet is *deflected*.

Greedy hot-potato algorithms are particularly attractive because they tend to be simple, admitting efficient hardware implementations. Greedy algorithms are also adaptive, namely, when contention is low, packets follow the shortest routes to their destinations. Furthermore, it has been observed that greedy hot-potato algorithms perform extremely well in practice [45] (their performance is close to the traditional store-and-forward algorithms with buffers).

## 5.2 Contributions

Although greedy hot-potato algorithms have been observed to perform very well in practice, there has been no adequate formal analysis that could explain this good behavior. Ben-Dor *et al.* [12, Section 1.1] remark:

Although fairly simple greedy hot-potato algorithms perform very well in practice, they resist formal analysis attacks.

Here, we give new greedy hot-potato algorithms, with improved theoretical bounds, which give the first formal explanation of the good behavior of greedy algorithms.

We analyze our algorithms for a specific class of routing problems known as *batch routing problems*. In a batch routing problem every node in the mesh is the source of exactly one packet at time zero. The question that we want to answer for batch problems is: how much time is needed until all the  $n^2$  packets reach their destinations?

There are three interesting batch routing problems, which are distinguished by the distribution of the packet destinations.

- **Permutation Problem:** every node is the source and destination of exactly one packet.
- **Random Destinations Problem:** every packet chooses its destination uniformly and at random at time zero.
- **General Problem:** the destinations are chosen arbitrarily. General batch routing problems are sometimes called “many-to-one” in the sense that a node may be the destination of multiple packets.

In Chapter 6 we present a new greedy hot-potato algorithm that is tuned for the permutation and random destinations problems. The time achieved by this algorithm is  $O(n \log n)$ , and it is an improvement from the previously known bound of  $O(n^2)$ .

In Chapter 7 we modify the previous algorithm so that it solves general batch problems. As it is explained in that chapter, for any instance  $I$  of the general batch problem there is a trivial lower bound  $LB_I$ . If  $LB_I$  is at least  $\Omega(n)$ , then our algorithm solves this batch problem in time  $O(LB_I \cdot \log^3 n)$ . This is the first hot-potato algorithm (greedy or non-greedy), and one of the few known routing algorithms (with or without buffers), which is competitive to the lower bound.

Before we proceed in presenting the above results, we present in Section 5.3 some necessary preliminaries which are common to the subsequent chapters.



## 5.3 Preliminaries

### 5.3.1 Mesh

In the  $n \times n$  mesh network, each node has coordinates  $(x, y)$ , for  $0 \leq x, y < n$ , where  $x$  is a column and  $y$  a row. The lower-left corner has coordinates  $(0, 0)$  and the upper-right corner has coordinates  $(n - 1, n - 1)$ .

Each node (except at the edge of the mesh) is connected to its neighbors by four links, denoted *up*, *down*, *left* and *right*. The links are bidirectional, namely, at each time step and at each link direction at most one packet is allowed.

The *distance* between nodes  $v = (x, y)$  and  $v' = (x', y')$ , denoted  $dist(v, v')$  is the quantity

$$dist(v, v') = |x - x'| + |y - y'|.$$

This distance measures how long it takes for an undeflected packet to travel from node  $v$  to  $v'$ . This distance is sometimes called the *Manhattan* metric or  $L_2$  norm.

The algorithms we will describe in the next chapters also apply to the  $n \times n$  torus, which is the same with the mesh with the only difference that each node  $(i, n - 1)$  has a link to the node  $(i, 0)$ , and each node  $(n - 1, i)$  has a link to the node  $(0, i)$ , for all  $0 \leq i < n$ . For brevity, we will focus here on the mesh, postponing discussion about the torus to the corresponding discussion sections.

### 5.3.2 Deflections

The analysis of our algorithms is based on estimating how many times a packet is deflected in the network. Knowing the number of deflections, we can easily compute the total number of time steps that the packet is in the network, as the following lemma shows.

**Lemma 5.3.1** *If a packet  $\pi$  is deflected in total  $x$  times, then it will reach its destination in at most  $2x + 2n - 2$  time steps.*

**Proof:** Initially, the distance from  $\pi$  to its destination is no more than  $2n - 2$ . Each time  $\pi$  is deflected, the distance increases by one, and each time it follows a link closer to its destination the distance decreases by one. The total number of time steps is equal to the total number of links that the packet follows. ■

### 5.3.3 Equations

We make use of the following inequalities.

For all  $n, t$ , such that  $n \geq 1$  and  $|t| \leq n$ ,

$$e^t \left(1 - \frac{t^2}{n}\right) \leq \left(1 + \frac{t}{n}\right)^n \leq e^t. \quad (5.1)$$

For all  $p, k$ , such that  $0 < p < 1$  and  $k \geq 1$ ,

$$1 - p \leq \left(1 - \frac{p}{k}\right)^k. \quad (5.2)$$

## Chapter 6

# An One-Bend Greedy Hot-Potato Algorithm

We present a new randomized greedy hot-potato routing algorithm for the  $n \times n$  mesh.<sup>1</sup> Like some earlier algorithms [12, 30], we assign priorities to packets. Each packet is divided into an immutable message part, and a mutable header containing the packet's priority (three bits suffice).

A novel aspect of our algorithm is the way it exploits randomization to adjust priorities. Each time a packet is deflected, there is a small probability it will attempt a *home run*: it increases its priority and attempts to travel by a one-bend path directly to its destination (see Figure 6.2). As we show in the analysis, when a packet attempts a home run it has a good chance to reach its destination, without interruptions from other high priority packets.

Both the home run technique and its analysis are novel. For any permutation or random destinations batch problem, the analysis ensures that each packet reaches its destination in asymptotically optimal expected  $O(n)$  steps, and all packets reach their destinations in  $O(n \log n)$  steps with high probability  $(1 - 1/n)$ , an improvement over the previously-known deterministic upper bound of  $O(n^2)$ , for greedy algorithms [12].

Our algorithm is not restricted to the permutation and the random destinations problems, it also considers general batch problems. In particular, the time complexity of our algorithm depends on the maximum number  $m$  of packets that have destination in any row or column. For any general batch problem instance, our algorithm routes all the packets to their destinations in time  $O(m \log n)$ . If, for example, the distribution of the destinations is uniform (as in the permutation or random destinations problem) then  $m = O(n)$  and the

---

<sup>1</sup>The work of this chapter was first presented in [20].

time to our algorithm is  $O(n \log n)$ . If in a general batch problem instance the distribution of the packet destinations to the nodes of the network is uniform, then our algorithm solves this problem instance in time which is  $\log n$ -competitive to the lower bound. However, there are some “pathological” problem instances for which the time of our algorithm is far from the optimal, and for these problems the algorithm of Chapter 7 is needed.

We continue as follows. In Section 6.1 we give some necessary preliminaries. In Section 6.2 we describe our algorithm. In Section 6.3 we analyze the timing behavior of our algorithm. In Section 6.4 we describe how our algorithm can be applied to solve specific batch routing problems. Finally, we conclude in Section 6.6 with a discussion and open problems.

## 6.1 Preliminaries

We say that a packet is *restricted* if it is on the same row or column as its destination node. A *good link* for a packet is one that brings it closer to its destination, and a *bad link* is one that does not. Good *row links* and *column links* are defined in the obvious way.

A packet is deflected if it is forwarded along a bad link. A node that receives a packet can tell whether the packet was just deflected by comparing the packet’s destination and the link the packet came from.

## 6.2 The Algorithm

In our algorithm the packets use *priorities*: each node routes higher-priority packets before routing lower-priority packets. As a result, a packet is deflected only if its good links are already taken by advancing packets of greater or equal priority.

We start with a highly simplified, informal overview of our algorithm. Initially, all packets are routed greedily with equal (low) priority. Each time a packet is deflected, however, in the node that the deflected packet is received, there is a small probability that the deflected packet will change its state and will become *excited*, causing its priority to jump higher. When a packet becomes *excited*, it tries to take one of the two shortest “one-bend” paths to its destination, a strategy we call a *home run* (see Figure 6.2). The home run succeeds if the packet arrives at its destination without interruptions by other high priority packets. As the analysis will show, an essential aspect of our algorithm is that any packet that attempts a home run will succeed with constant probability (that is, probability independent of  $n$ ). We exploit this property to analyze both the expected and “with high probability” behavior of the algorithm.

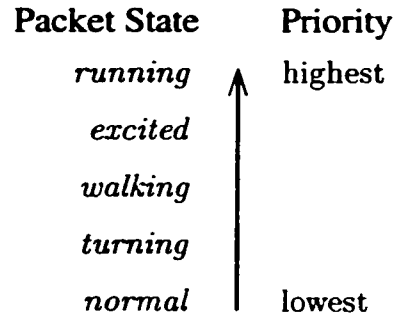


Figure 6.1: Packet States and Priority

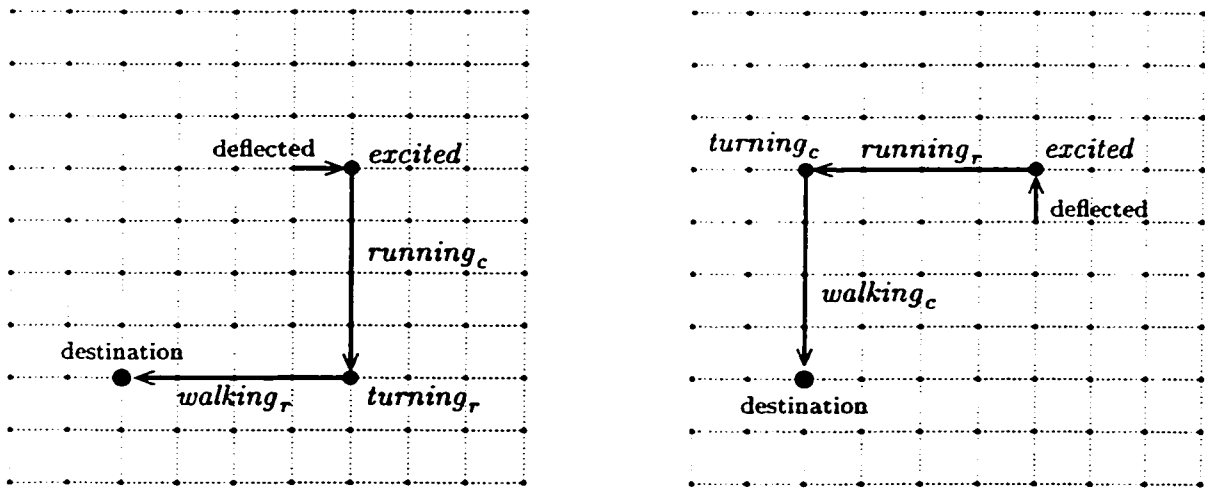


Figure 6.2: A Home Run: column-first and row-first traversals

In detail, the algorithm assigns every packet one or more *preferred* links, and tries to forward each packet along a preferred link. A preferred link is always a good link (taking the packet closer to its destination), but not every good link is preferred. Consider a packet  $\pi$ . At any time step, the packet  $\pi$  occupies only one of the following states, where each state corresponds to a priority (see Figures 6.1 and 6.2).

- *normal*: The packet  $\pi$  starts out in the *normal* state. In this state, the good links are preferred: each node forwards a normal packet along one of its good links, unless those links are already occupied by other advancing packets. Initially, all the packets are in the *normal* state.
- *excited*: Let's assume that the *normal* packet  $\pi$  was deflected at the previous time step so that at the current time step appears in a node  $v$ , one link further away from its destination (since the packet was deflected, it followed a bad link). At node  $v$ , the packet  $\pi$  enters the *excited* state with probability  $p$  (given below), and otherwise it remains *normal* with probability  $1 - p$ . When a packet becomes *excited*, it tries to follow a one-bend path to its destination (a home run). It flips an equal probability coin to choose whether to traverse the row or the column first, and this choice determines its preferred link.
- *running*: If the *excited* packet  $\pi$  succeeds in following its preferred link, then the next time step it appears in a node one link closer to its destination. In this node the packet changes its state once more and it becomes a *running* packet, and it will remain a running packet for the first part of the home run (until the bend). In particular, if the packet  $\pi$ , when it started its home run, has chosen to traverse the column first, then it is in the *running<sub>c</sub>* state on the column part of its one-bend path. In this state, the good column link is preferred. If the packet  $\pi$ , when it started its home run, has chosen to traverse the row first, then the packet is in the *running<sub>r</sub>* state on the row, and the good row link is preferred. (Notice, that a packet stays in the *excited* state for at most one time step.)
- *turning*: When the *running* packet  $\pi$ , reaches its destination row or column, in the node that it turns, the packet changes its state and it becomes a *turning* packet. In particular, if packet  $\pi$ , when it started its home run, has chosen to traverse the column first, then it enters the *turning<sub>r</sub>* state when it reaches its destination row, and its preferred link is along the row. If the packet  $\pi$ , when it started its home run, has chosen to traverse the row first, then it enters the *turning<sub>c</sub>* state, and its preferred

link is along the column. The packet stays in the *turning* state for at most one time step.

- *walking*: In the next node after the bend of the home run, the *turning* packet  $\pi$  changes its state again and it becomes a *walking* packet. In particular, if packet  $\pi$ , when it started its home run, has chosen to traverse the column first, then it is in the *walking<sub>r</sub>* state as it traverses the destination row, and the preferred link is the good row link. If the packet  $\pi$ , when it started its home run, has chosen to traverse the row first, then it is in the *walking<sub>c</sub>* state as it traverses the column, and the preferred link is the good column link.

If a restricted packet becomes *excited*, it enters the *walking<sub>r</sub>* or *walking<sub>c</sub>* state directly. We use *running* to denote either the *running<sub>c</sub>* or *running<sub>r</sub>* states when the distinction is unimportant, and similarly for the other dual states.

If, at any time step in a node  $v$ , a packet is unable to follow its one-bend path, because its preferred links are taken by packets of the same or higher priority packets which are advancing, it re-enters immediately (in the same time step) the *normal* state. In this case, node  $v$  attempts to forward the packet as a *normal* packet by sending to any available good link (which is not occupied by advancing packets).

At each step, a node greedily routes up to four arriving packets. If the node receives a packet which was deflected in the previous time step, then the deflected packet is marked *excited* with probability  $p$ . The node then routes each of the packets in priority order. If the packet can take its preferred link, it is forwarded along that link. Otherwise, the packet's priority is either reduced to *normal*, or if it is already *normal*, it is forwarded along any unoccupied link. This algorithm is greedy because a packet fails to follow any of its good links only if other packets (that move forward) are traversing these links.

Two packets with the same preferred link are said to *conflict*. Conflicts between packets in the same states are resolved arbitrarily. In conflicts between packets of different priorities, the higher priority packets always win. Note that some states never conflict: for example, a packet in the *running<sub>r</sub>* state will never conflict with a packet in the *running<sub>c</sub>* state. For a packet which attempts a home run, we say that it travels *uninterrupted* to its destination, if during the home run it wins over all the conflicts that it encounters.

### 6.3 Time Analysis

In this section, we give the time analysis of our algorithm.

Every randomized algorithm needs an adversary who attempts to frustrate the algorithm's goals. In our network model, the adversary can control any of the following outcomes in the algorithm's execution:

- The initial distribution of the destinations.
- The resolution of conflicts between packets with the same priorities.
- The choice of the link that a *normal* packet will follow (at any step, there may be more than one alternative links).

In our algorithm, we assume that the adversary has the full power to control any of the above outcomes in the most malicious way, in order to disrupt the fast delivery of the packets. Therefore, the analysis of our algorithm corresponds to the worst case scenario, produced by any adversary.

A key term in the analysis of our algorithm is the maximum number of packets addressed to any single row or column. More precisely,

**Definition 6.3.1** *Given a batch routing problem, let  $m_r$  and  $m_c$  be respectively the maximum number of packets targeted to any row or column. Define*

$$m := \max(n, \min(m_r, m_c)).$$

Notice that  $n \leq m \leq n^2$ . Many of our complexity results are expressed in terms of  $m$  (and  $n$ ). The value  $m$  is a rough reflection of a problem's inherent difficulty: high values of  $m$  imply high levels of congestion in the rows or columns, and vice-versa.

As will be explained in Section 6.4, the parameter  $m$  is related to the permutation and random destinations batch problems (actually, in these problems  $m = n$ ). Moreover, the parameter  $m$  is related to the general batch problem, and we show that our algorithm can be used to solve efficiently some instances of the general batch problem.

For any batch problem, our algorithm guarantees that all packets reach their destination nodes in at most  $O(m \ln n)$  steps, with high probability. Even though the complexity analysis depends on  $m$ , the value  $m$  is not known to the algorithm.

Our algorithm is parameterized by  $p$ , the probability that a deflected packet becomes *excited*. Different values of  $p$  yield different behaviors. In Section 6.3.1, we compute the probability for a packet to succeed in its *home run*, leaving  $p$  unbound. In Section 6.3.2, we compute the expected time needed for a single packet to reach its destination. With a proper choice of  $p$ , we can achieve an  $O(n)$  expected time. In Section 6.3.3, we turn our attention to the time needed to solve a batch problem. With a proper (time-dependent) choice of  $p$ , we can achieve  $O(m \ln n)$  time with high probability.



### 6.3.1 Succeeding in a Home Run

We now analyze the probability that a packet  $\pi$  will successfully complete a home run by reaching its destination. When analyzing the behavior of  $\pi$ , we give full power to the adversary. In particular, each time the packet  $\pi$  becomes *excited*, the adversary is allowed to place the other packets at nodes in the mesh, and to choose their destinations, subject only to the constraint that no more than  $m_r$  or  $m_c$  packets can have destinations in the same row or column as  $\pi$ . The probability that an *excited* packet will complete a home run against such an adversary is a lower bound for that probability in any actual execution.

Without loss of generality, let's assume that the packet's destination is down and left from the packet's current position. Furthermore, without loss of generality, we assume that in the home run the packet chooses to follow the column first. The case where the packet is restricted (already on its destination row or column) is considered below as a special case.

Recall that a deflected packet becomes *excited* with probability  $p$ . We will derive lower bounds on the probability for "good things" happening in terms of  $p$ , and then use these bounds to motivate our choice for  $p$ .

**Lemma 6.3.1** *The probability that a particular node contains no excited packet is at least  $p' := (1 - p)^4$ .*

**Proof:** A packet becomes *excited* only if it was deflected in the preceding step, with probability  $p$ . It will fail to become *excited* with probability at least  $1 - p$ . Since a node contains at most four packets, all four will fail to become *excited* with probability at least  $(1 - p)^4$ . ■

**Lemma 6.3.2** *An excited packet follows its preferred link and enters the running state with probability at least  $(1 - p)^{4n}$ .*

**Proof:** Consider an *excited* packet  $\pi$  at node  $(x, y)$  at time  $t$ . Our priority assignment guarantees that an *excited* packet can be interrupted only by another conflicting *excited* or *running* packet.

- By Lemma 6.3.1, the probability that node  $(x, y)$  has no *excited* packet at time  $t$  is at least  $p'$ . Therefore, with at least this probability there is no other conflicting *excited* packet.
- Recall that we assume (without loss of generality) that the packet's  $\pi$  preferred link is in the down direction. In order for a conflicting packet to be in the *running<sub>c</sub>* state

at time  $t$  and at node  $(x, y)$ , it must have become *excited* at some node  $(x, y + d)$  at time  $t - d$ , where  $1 \leq d \leq n - y - 1$ . There are at most  $n - 1$  such possible nodes. By Lemma 6.3.1, the probability that no packet became *excited* at any of those nodes and times is at least  $p'^{n-1}$ . Therefore, with at least this probability there is no conflicting *running<sub>c</sub>* packet.

In total, the probability the packet  $\pi$  will enter the *running* state is at least

$$p' \cdot p'^{n-1} = (1 - p)^{4n}.$$

■

**Lemma 6.3.3** *A packet in the running state will always proceed along its preferred link.*

**Proof:** Our priority rules ensure that a packet in the *running* state can be interrupted only by another packet in the *running* state. Any *running* packet whose preferred link is (say) down, must have arrived from the up link. At most one packet could have arrived from the up link, so at most one *running* packet prefers to exit on the down link, and no conflict can occur. ■

**Lemma 6.3.4** *A packet in the turning state successfully enters the walking state with probability at least  $(1 - p)^{4n+m_r}$ .*

**Proof:** Consider a *turning* packet  $\pi$  at node  $(x, y)$  at time  $t$ . Let's assume (without loss of generality) the packet arrived from the up link, and prefers the left link. Our priority rules ensure that this packet can be interrupted only by either: an *excited* packet, or a *running<sub>r</sub>* packet arriving from the right link, or a *walking<sub>r</sub>* packet arriving from the right link, or by a *turning<sub>r</sub>* packet arriving from the down link.

- By Lemma 6.3.1, there is no *excited* packet at node  $(x, y)$  at time  $t$  with probability at least  $p'$ .
- A conflicting packet in the *running<sub>r</sub>* state must have become *excited* at node  $(x + d, y)$  at time  $t - d$  for it to be in the *running<sub>r</sub>* state at node  $(x, y)$  and time  $t$ , for  $1 \leq d \leq n - x - 1$ . As in the proof of Lemma 6.3.2 the adversary will fail to excite all such packets with probability at least  $p'^{n-1}$ .
- Suppose now that some packet  $\sigma$  conflicts with  $\pi$  at time  $t$ , while  $\sigma$  is in either the *turning<sub>r</sub>* or *walking<sub>r</sub>* states. Packet  $\sigma$  must have been deflected at some time  $t - d - 1$ .

then became *excited* at time  $t - d$ , and traversed  $d$  links to collide with  $\pi$ , all without being deflected. After the packet  $\sigma$  becomes *excited*, it is always in “phase” with packet  $\pi$ , namely, at each time step they are at the same distance from node  $(x, y)$ , in order to collide at this node.

The key observation is that if  $\sigma$  failed to become *excited* at a deflection that could bring it in “phase” with  $\pi$ , then  $\sigma$  will never catch up to  $\pi$  in time to interrupt  $\pi$  in node  $(x, y)$ . There is at most one deflection that could bring  $\sigma$  in “phase” with  $\pi$ . From the adversary’s point of view,  $\sigma$  had only one chance to become *excited* in a way that can threaten  $\pi$ , and this chance is given with probability  $p$ . Subsequently,  $\sigma$  is not a threat to  $\pi$  with probability at least  $1 - p$ . Because  $\pi$  is already on its destination row, the number of packets like  $\sigma$  that could be a possible threat to  $\pi$  are at most  $m_r - 1$ , the number of the rest packets that have destinations in the same row. Therefore, with probability at least

$$(1 - p)^{m_r - 1} \geq (1 - p)^{m_r},$$

there is no conflicting packet in either the *turning<sub>r</sub>* or *walking<sub>r</sub>* state at node  $(x, y)$  at time  $t$ .

In total, the probability that the *turning* packet  $\pi$  will not be interrupted, and successfully become *walking* in the next step, is at least

$$p' \cdot p^{m-1} \cdot (1 - p)^{m_r} = (1 - p)^{4n+m_r}.$$

■

The symmetric lemma holds for  $m_c$ .

**Lemma 6.3.5** *A packet in the walking state arrives at its destination with probability at least  $(1 - p)^{4n}$ .*

**Proof:** Consider a *walking* packet  $\pi$  in node  $v$  at time  $t$ . Let’s assume (without loss of generality) that the packet’s preferred link is the left link. Our priority rules imply that the packet can be interrupted (and thus deflected) only by a packet in the *walking*, *running*, or *excited* states.

- Since packet  $\pi$  arrives at node  $v$  from the right link and prefers to exit on the left, a conflicting packet in the *running* or *walking* state must also have come from the right link and prefers to exit on the left, which cannot happen. Therefore, the packet will not conflict with another *running* or *walking* packet.

- By Lemma 6.3.2, there is no *excited* packet at node  $v$  at time  $t$  with probability at least  $p'$ .

Therefore, packet  $\pi$  will not be interrupted in node  $v$  with probability at least  $p'$ . The packet will remain in the *walking* state for at most  $n-2$  nodes before reaching its destination node. Therefore, packet  $\pi$  will not be interrupted in any of these nodes, and it will complete a home run successfully, with probability at least

$$p'^{m-2} > (1-p)^{4n}.$$

■

**Lemma 6.3.6** *If an excited packet chooses to follow its good column link, it will complete a home run to its destination with probability at least  $(1-p)^{12n+m_r}$ .*

**Proof:** The result follows from combining Lemmas 6.3.2, 6.3.3, 6.3.4, and 6.3.5:

$$(1-p)^{4n} \cdot 1 \cdot (1-p)^{4n+m_r} \cdot (1-p)^{4n} = (1-p)^{12n+m_r}.$$

■

The symmetric lemma holds for  $m_c$ .

If a packet is restricted (already in its destination row or column) when it becomes *excited*, it immediately tries to enter the next step the *walking* state. The proof of the next Lemma follows from the proof of Lemma 6.3.2.

**Lemma 6.3.7** *If a restricted packet becomes excited, it succeeds in following the good column (or row) link and entering the walking state with probability at least  $(1-p)^{4n}$ .*

Finally, we can compute the probability that a packet succeeds in a home run.

**Theorem 6.3.8** *An excited packet will complete its home run with probability at least  $\frac{1}{2}(1-p)^{12n+m}$ .*

**Proof:** For unrestricted packets, the proof follows directly from Lemma 6.3.6, and the fact that the packet chooses each alternative home run (row-first or column-first) with probability 1/2. For restricted packets, it follows from Lemmas 6.3.5 and 6.3.7. ■

### 6.3.2 Expected Case Analysis

In this section, we analyze the expected number of steps for a specific packet  $\pi$  to arrive at its destination. Here, we will assume that the expected number of packets with the same destination row or column is  $n$ , and thus  $m = n$ . This assumption applies, for example, to the permutation problem. (In the random destinations problem we have  $m = O(n)$  with high probability.)

We will prove that the expected time needed for a single packet to reach its destination is  $O(n)$ . This time is optimal, since there are batch problem instances where a packet may originate at distance  $\Omega(n)$  from its destination.

We now consider the behavior of the algorithm when the probability  $p$ , of becoming *excited* after a deflection, is

$$p := \frac{1}{13n}.$$

**Lemma 6.3.9** *An excited packet will complete its home run with probability at least  $1/4e$ .*

**Proof:** By Theorem 6.3.8, the probability of completing a home run is at least  $\frac{1}{2}(1 - p)^{12n+m}$ . With  $p = 1/13n$  and  $m = n$ , and by applying Equation 5.1, we have

$$\begin{aligned} \frac{1}{2}(1 - p)^{12n+m} &= \frac{1}{2} \left(1 - \frac{1}{13n}\right)^{13n} \\ &\geq \frac{1}{2} \frac{1}{e} \left(1 - \frac{1}{13n}\right) \\ &\geq \frac{1}{2} \frac{1}{2e}. \end{aligned}$$

■

**Lemma 6.3.10** *Each time a packet is deflected, it becomes excited and then completes a home run to its destination with probability at least  $1/52en$ .*

**Proof:** Let  $\pi$  be a deflected packet that gets *excited* with probability  $p$ . According to Lemma 6.3.9, after packet  $\pi$  becomes *excited*, it will complete a home run with probability at least  $1/4e$ . Therefore, the probability for completing a home run after a deflection is at least

$$p \cdot \frac{1}{4e} = \frac{1}{52en}.$$

■

Here, we prove the main result for the expected case analysis.

**Theorem 6.3.11** *The expected number of steps for a packet to arrive at its destination is bounded from above by  $O(n)$ .*

**Proof:** Let  $q = 1/52en$  be the lower bound on the probability to do a successful home run after a deflection, such as calculated in Lemma 6.3.10. Each deflection can be thought of as a Bernoulli trial with probability of success at least  $q$ , for reaching the destination. Therefore, the expected number of deflections until the packet reaches the destination is bounded from above by  $1/q = 52en$ . Subsequently, by Lemma 5.3.1. we have that the expected number of time steps until the packet reaches its destination is bounded from above by  $2(52en) + 2n - 2 = O(n)$ . ■

### 6.3.3 Analysis “with High Probability”

In this section we show that with a proper choice of  $p$ , all packets will be delivered within time  $O(m \ln n)$  with high probability (meaning with probability at least  $1 - 1/n$ ). The challenge here is that we assume that nodes do not know  $m$ , the instance-specific measure of congestion. The key technique here is to allow  $p$  to vary with time, ensuring that  $p$  lies within the “right” range sufficiently long to guarantee timely delivery. If the value of  $m$  were known to the algorithm,  $p$  could be constant (and approximately  $1/m$ ).

We will use the following constants.

$$\begin{aligned} c &= 48e & c' &= 13 \cdot 3c \\ t_0 &= c'm \ln n + 2n & t_1 &= 3c'm \ln n \end{aligned}$$

Let the probability  $p$  that a deflected normal packet becomes *excited* be the following function of time:

$$p(t) := \frac{c \ln t}{t}.$$

Notice that when  $t$  lies in the range  $t_0$  to  $t_1$ , the value of  $p(t)$  is approximately  $1/m$ , the desired value.

**Lemma 6.3.12** *If a packet  $\pi$  becomes excited at time  $t \geq t_0$ , then the probability of completing a home run is at least  $1/4e$ .*

**Proof:** Any packet conflicting with  $\pi$  must have started its home run attempt at most  $2n$  steps before  $\pi$ . The probability that such a packet became *excited* was at most

$$p := p(t_0 - 2n) = \frac{c \ln(c'm \ln n)}{c'm \ln n}.$$

By Theorem 6.3.8, the probability of a home run is at least  $\frac{1}{2}(1-p)^{12n+m}$ . Since  $n \leq m \leq n^2$ , by taking  $n$  to be sufficiently large, such that  $4c' \ln n \leq n$ , and by applying Equation 5.1 we get

$$\begin{aligned}
\frac{1}{2}(1-p)^{12n+m} &\geq \frac{1}{2} \left(1 - \frac{c \ln(c'm \ln n)}{c'm \ln n}\right)^{13m} \\
&> \frac{1}{2} \left(1 - \frac{3c \ln n}{c'm \ln n}\right)^{13m} \\
&\geq \frac{1}{2} \left(1 - \frac{1}{13m}\right)^{13m} \\
&\geq \frac{1}{2} \frac{1}{e} \left(1 - \frac{1}{13m}\right) \\
&\geq \frac{1}{2} \frac{1}{2e}.
\end{aligned}$$

■

**Lemma 6.3.13** *Each time  $t$  (with  $t_0 \leq t < t_1$ ) a packet is deflected, it will complete a home run with probability at least*

$$\frac{c}{12ec'm}.$$

**Proof:** Whenever a packet is deflected with probability at least  $p(t_1)$  it becomes *excited*. Since  $n \leq m$ , the probability of getting *excited* is at least

$$\begin{aligned}
p(t_1) &= \frac{c \ln(3c'm \ln n)}{3c'm \ln n} \\
&> \frac{c \ln n}{3c'm \ln n} \\
&= \frac{c}{3c'm}.
\end{aligned}$$

The probability of completing a home run when getting *excited* is, according to Lemma 6.3.12, at least  $1/4e$ . Therefore, the probability of attempting a home run, and completing it, when being deflected is at least

$$\begin{aligned}
p(t_1) \cdot \frac{1}{4e} &= \frac{c}{3c'm} \cdot \frac{1}{4e} \\
&= \frac{c}{12ec'm}.
\end{aligned}$$

■

**Lemma 6.3.14** *With probability at least  $1 - 1/n^3$ , a packet will reach its destination in  $t_1$  steps.*

**Proof:** By Lemma 6.3.13, each time that a packet  $\pi$  is deflected in the time interval  $t_0 \leq t < t_1$ , packet  $\pi$  will complete a home run with probability at least

$$q := \frac{c}{12ec'm}.$$

Since, we always consider the most powerful adversary, which produces the worst case scenario for interrupting the home run of packet  $\pi$ , successive probabilities of succeeding in a home run after a deflection are independent. By Lemma 5.3.1, the number of deflections that can fit in the time interval  $t_0 \leq t < t_1$  is at least

$$\begin{aligned} x &:= \frac{t_1 - t_0 - 2n}{2} \\ &= \frac{(3c' - c)m \ln n - 4n}{2} \\ &= c'm \ln n - 2n. \end{aligned}$$

Since  $n \leq m$ ,

$$\begin{aligned} x &= c'm \ln n - 2n \\ &\geq c'm \ln n - \frac{1}{4}c'm \ln n \\ &= \frac{3}{4}c'm \ln n. \end{aligned}$$

Packet  $\pi$  will fail to reach its destination after  $x$  deflections with probability at most  $(1-q)^x$ . By Equation 5.1, we get

$$\begin{aligned} (1-q)^x &\leq \left(1 - \frac{c}{12ec'm}\right)^{\frac{3}{4}c'm \ln n} \\ &\leq \left(1 - \frac{3}{4c'm}\right)^{\frac{3}{4}c'm \ln n} \\ &\leq e^{-3 \ln n} \\ &= \frac{1}{n^3}. \end{aligned}$$

■

Finally, we obtain the desired result for all packets.

**Theorem 6.3.15** *With high probability (at least  $1-1/n$ ), all packets reach their destination nodes in at most  $O(m \ln n)$  steps.*



**Proof:** By Lemma 6.3.14, a packet will arrive at its destination in  $t_1$  steps with probability at least  $1 - 1/n^3$ .

Since we have made a worst case analysis for each packet by assuming that the adversary produces the worst case scenario for a packet to reach its destination, we can safely assume that the packets are independent of each other in the analysis.

Therefore, the probability that all packets (at most  $n^2$ ) will arrive at their destinations within  $t_1$  steps is at least (by applying Equation 5.2)

$$\left(1 - \frac{1}{n^3}\right)^{n^2} = \left(1 - \frac{1/n}{n^2}\right)^{n^2} \geq 1 - \frac{1}{n}.$$

■

## 6.4 Applications

For the batch permutation problem,  $m = n$ , and for the random destinations problem, it can be easily shown that  $m = O(n)$ , with high probability. From Theorem 6.3.15, we have the following corollary.

**Corollary 6.4.1** *For the batch permutation or random destinations problem, with high probability, all packets reach their destination nodes in at most  $O(n \ln n)$  steps.*

We can also apply our algorithm for general batch problem instances. For example, consider the following *rectangle routing problem*. There are  $n^2$  packets whose destinations are distributed uniformly within a  $w \times h$  rectangle, and all packets originate outside the rectangle. Uniform distribution means that every node within the rectangle is the destination of  $\Theta(n^2/wh)$  packets. Let's assume (without loss of generality) that  $w \geq h$ . In this case, the parameter  $m$  is

$$m = h \cdot \frac{n^2}{wh} = \frac{n^2}{w}.$$

Theorem 6.3.15 says that our algorithm finishes in

$$O\left(\frac{n^2}{w} \cdot \ln n\right)$$

steps, with high probability.

Mansour and Patt-Shamir [42] have noted that there is a trivial lower bound for problems of this kind:  $\Omega(d_{max} + W)$ , where  $d_{max}$  is the maximum initial distance any packet must traverse, and  $W$  is the *network bandwidth lower bound*. The bandwidth lower bound  $W$

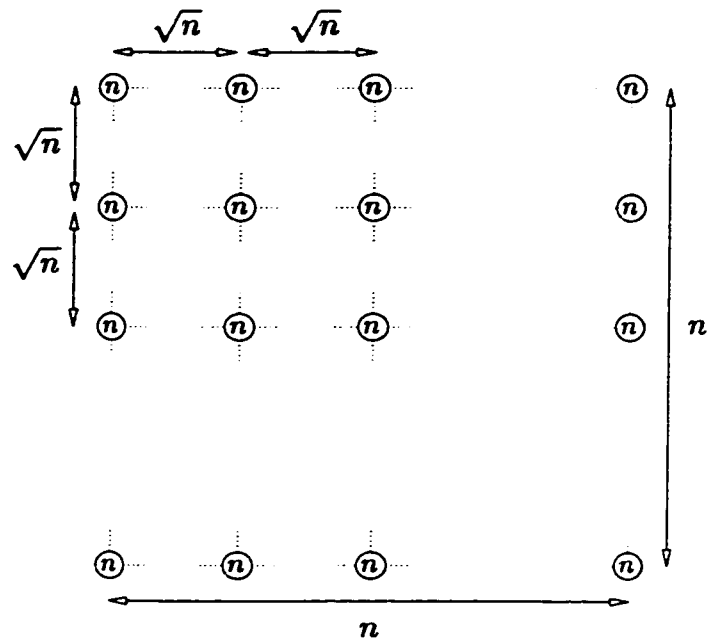


Figure 6.3: A “pathological” batch problem

has to do with regions of nodes (see Figure 7.1). Let  $S$  be any region in the network. Let  $z$  be the number of incoming links in the region’s perimeter, and let  $k$  be the number of packets with destinations in this region, where all these packets originate from outside the region. At each time step, at most  $z$  packets can enter the region  $S$ . Therefore at least  $k/z$  time steps are needed for all the packets to reach their destinations, this is the bandwidth lower bound for  $S$ . Taking the maximum bandwidth lower bound over all possible regions in the network we obtain the bandwidth lower bound  $W$  for any instance of a general batch problem.

For our  $w \times h$  size rectangle routing problem,

$$W = \frac{n^2}{2w + 2h} = \Theta\left(\frac{n^2}{w}\right).$$

Thus, with high probability our algorithm is  $O(\ln n)$ -competitive with the trivial lower bound for this class of problems. Therefore, our algorithm is away by only a  $O(\ln n)$  factor from the optimal for these problems.

However, there are general batch problem instances for which our algorithm finishes in time which worse than the trivial lower bound. For example, consider the “pathological” batch problem of Figure 6.3. In this problem the destinations of the packets are distributed as follows: in the top row only the leftmost node is the destination for  $n$  packets, then the

next node is the row at distance  $\sqrt{n}$  is the destination for  $n$  packets, and so on, until the last node in the row (the rest nodes of the row aren't destinations). The destinations are distributed in the same way for the next (lower) row at distance  $\sqrt{n}$ , and so on until the lowest row (all the rest rows contain no destinations).

In this “pathological” problem  $m = n\sqrt{n}$ , since there are intersecting rows and columns with that many destinations. The time performance of our algorithm for this problem is  $O(n\sqrt{n} \cdot \ln n)$ . It is easy to see that the bandwidth lower bound for this problem is  $W = \Theta(n)$ . Our algorithm is far from the lower bound by a factor of  $\sqrt{n} \cdot \ln n$ . The reason of this divergence from the lower bound is that when the packets attempt their home runs they meet in the highly congested rows and columns. In order to lighten up the congestion, when a packet attempts a home run it should be allowed to go to random intermediate nodes, and thus follow a multi-bend home run path. This is what the algorithm of Chapter 7 does, and that algorithm is  $O(\ln^3 n)$ -competitive with the trivial lower bound.

## 6.5 Related Work

There are several results for the mesh and torus networks for greedy and non-greedy hot-potato routing algorithms.

For greedy hot-potato routing, Ben-Dor *et al.* [12] give a potential function analysis and they provide a simple algorithm for the 2-dimensional  $n \times n$  mesh with  $O(n\sqrt{k})$  steps, where  $k$  is the total number of packets to be routed. They generalized their techniques for the  $d$ -dimensional mesh to obtain  $O(e^d n^{d-1} k^{1/d})$  steps. Borodin *et al.* [13] present a complicated deterministic greedy hot-potato routing algorithm for the  $d$ -dimensional mesh and the 2-dimensional torus where any packet  $p$  finishes in at most  $\text{dist}(p) + 2(k-1)$  steps, where  $\text{dist}(p)$  is the initial distance of  $p$  from its destination (they also present a simple non-greedy algorithm with similar results). For the 2-dimensional mesh and torus this algorithm preserves the  $O(n^{1.5})$  bound given by Bar-Noy *et al.* [7]. For the 2-dimensional case a similar result was independently obtained by Ben-Aroya *et al.* [10]. For a single destination or a small set of destinations Ben-Aroya *et al.* [11] present a randomized algorithm on the  $d$ -dimensional mesh that finishes in  $O(k/d)$  steps, with high probability.

For non-greedy hot-potato routing, Feige and Raghavan [27] present an algorithm for the  $n \times n$  torus that routes any random destinations problem in  $2n + O(\ln n)$  steps with high probability. They also give an alternative algorithm that routes any permutation problem in  $9n$  steps with high probability. Newman and Schuster [46] give a deterministic algorithm for permutation routing on the  $n \times n$  mesh that finishes in  $7n + o(n)$  steps and is based on

sorting. This result was improved by Kaufmann *et al.* [37] to  $3.5n + o(n)$  steps. Kaklamanis *et al.* [35] present an algorithm that routes most of the permutations in the  $d$ -dimensional torus in  $dn/2 + O(\ln^2 n)$  steps and in the 2-dimensional mesh in  $2n + O(\ln^2 n)$  steps. Bar-Noy *et al.* [7] present a simple deterministic algorithm for the  $n \times n$  mesh and torus that routes any permutation problem in  $O(n^{1.5})$  steps. Specifically, their algorithm routes any batch problem in  $O(n\sqrt{m_c})$  steps where  $m_c$  is the maximum number of packets destined to any column. They also give a more complicated algorithm that runs in  $O(n^{1+\epsilon})$  steps for every constant  $\epsilon \geq 0$ . Spirakis and Triantafillou [52] describe a routing algorithm for the random destinations problem on the two-dimensional mesh. In this algorithm, the average time to deliver all packets, where the average is taken over all possible destination assignments, is  $O(n \log n)$ . Broder and Upfal [14] give a dynamic analysis of a non-greedy algorithm for the torus.

## 6.6 Discussion

Our results apply almost verbatim to the  $n \times n$  torus. The only difference is that distances are smaller on a torus: any two nodes are at most  $n$  links apart instead of  $2n$ . As a result, some of the constants are smaller for the torus, reducing the complexity measures by a constant factor.

One important open problem is how to analyze dynamic problems, where packets are inserted into the network at a steady rate (not just at time zero). We think that techniques similar to those proposed by Broder and Upfal [14] are promising.

## Chapter 7

# A Multi-Bend Greedy Hot-Potato Algorithm

We present a new randomized greedy hot-potato routing algorithm for the  $n \times n$  mesh, which solves efficiently general batch problems.<sup>1</sup>

As discussed in Section 6.4, Mansour and Patt-Shamir [42] have noted that there is a trivial lower bound for problems of this kind. If a packet's source and destination are separated by distance  $d$ , then no routing algorithm can deliver that packet in fewer than  $d$  steps. The maximum such distance a packet must traverse in a routing problem  $I$  is called the *distance* lower bound, denoted  $D_I$ . Consider now the case where  $k$  packets have their destinations inside some region of the network and these packets originate from outside the region (see Figure 7.1). All these packets must enter the region. If the region has  $z$  incoming links in its perimeter, then at each time step at most  $z$  packets can enter the region, and thus, no routing algorithm can deliver those  $k$  packets in fewer than  $k/z$  steps. The maximum value of this ratio, taken over all the regions in the network, for a problem instance  $I$  yields the *bandwidth* lower bound, denoted  $W_I$ . The *lower bound* for problem instance  $I$ , which we denote by  $LB_I$ , is just  $\Omega(D_I + W_I)$ .

A family of routing problems is *hard* if the trivial lower bound for each of its members is  $\Omega(n)$ . Our hot-potato routing algorithm solves any hard batch routing problem  $I$  with high probability (at least  $1 - \frac{1}{n}$ ) in time  $O(LB_I \cdot \log^3 n)$ . This algorithm is the first hot-potato algorithm (greedy or non-greedy) whose performance, with high probability, lies within a polylogarithmic factor of optimal for a non-trivial class of batch routing problems. Furthermore, it is one of the few known routing algorithms (with or without buffers) that

---

<sup>1</sup>The work of this chapter was first presented in [19].

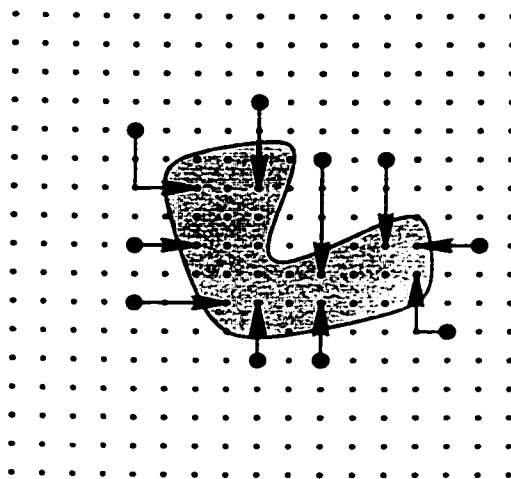


Figure 7.1: Packets with destinations in a region

solves problems of this kind.

Our algorithm is *distributed*: each node makes routing decisions based on its local state, independently of the other nodes. Moreover, nodes know nothing about the initial distribution of destinations (including the values of  $D_I$ ,  $W_I$ , and  $LB_I$ ).

At the heart of our algorithm is a new technique based on *multi-bend* paths, a departure from the paths using a constant number of bends used in most other hot-potato algorithms. Each time a packet is deflected (unable to advance toward its destination), it may, with a certain probability, become *excited*, increasing its priority over non-excited packets. An excited packet attempts to converge on its target by choosing a logarithmic number of random intermediate destinations (see Figures 7.2 and 7.4) in a sequence of squares of decreasing size. As we will show in the analysis, a packet during its multi-bend path has a good chance not to be interrupted by other high-priority packets, and therefore, to successfully reach its destination.

We proceed as follows. In Section 7.1 we give some necessary preliminaries. We present our algorithm in Section 7.2 and in Section 7.3 we give its time analysis. We describe the performance of our algorithm in terms of the trivial lower bound in Section 7.4. In Section 7.5 we present related work. We conclude in Section 7.6 with a discussion and open problems.

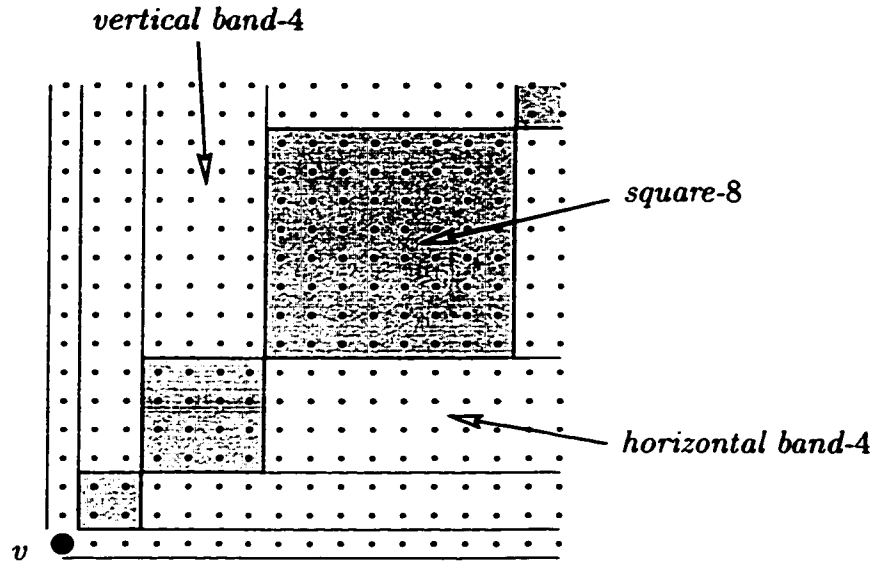


Figure 7.2: The squares and bands of a node

## 7.1 Preliminaries

In the  $n \times n$  mesh, we denote a rectangle with lower leftmost node  $v$  and upper rightmost node  $v'$  as  $[v, v']$ . When necessary we distinguish between the binary logarithm  $\lg$  and the natural logarithm  $\ln$ .

Take a node  $v = (x, y)$  and a number  $z = 2^k$ , where  $k = 0, \dots, \lg n - 1$ . Consider the sub-mesh that is up and right from  $v$ . The *square- $z$*  of  $v$  is the  $z \times z$  square whose lower leftmost node is

$$v' = (x + z - 1, y + z - 1),$$

as shown in Figure 7.2. If the *square- $z$*  does not fit entirely into the mesh, it is truncated at the mesh boundary. Note that *square-1* is the node  $v$  itself. The *horizontal band- $z$*  of  $v$  is the rectangle

$$[(x + 2z - 1, y + z - 1), (n - 1, y + 2z - 1)].$$

The *vertical band- $z$*  of  $v$  is the rectangle

$$[(x + z - 1, y + 2z - 1), (x + 2z - 1, n - 1)].$$

Note that all *square- $z$* , *horizontal band- $z$* , *vertical band- $z$* , for  $z = 2^k$  and  $k = 0, \dots, \lg n - 1$ , partition the sub-mesh that is up and right from  $v$  (say, the rectangle  $[v, (n - 1, n - 1)]$ ). That is, every node in that sub-mesh can be assigned to exactly one *square* or *band*. By

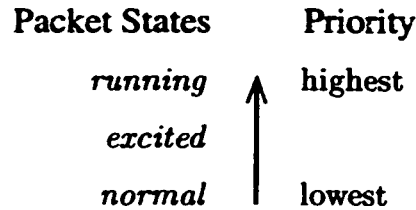


Figure 7.3: Packet states and priority

symmetry, we define the squares and bands of  $v$  in the other three sub-meshes. Similarly, we define the squares and bands for any node in the network.

## 7.2 Algorithm

Our algorithm is greedy: in a node, a packet always tries to follow any link that brings it closer to its destination. When two or more packets are competing in a node for the same link we say that there is a *conflict*.

In our algorithm there are three states for a packet: *normal*, *excited*, *running*. A packet is in only one of these states. Each packet state corresponds to a priority and the *running* packets have the highest priority and the *normal* packets have the lowest (see Figure 7.3). The priorities determine how the conflicts are resolved in the nodes, namely, the higher priority packets win over the lower priority packets. Conflicts between packets of the same priority are resolved in an arbitrary way (except for the packets in the *running* state for which we describe below how the conflicts are resolved). For implementation purposes, each packet can be divided into an immutable message part, and a mutable header containing the packet's priority (two bits suffice).

Initially, all the packets are in the *normal* state. A packet in the *normal* state simply tries to follow any of the available links that brings it closer to its destination. A *normal* packet is deflected whenever other *normal* or higher priority packets (which move forward to their destinations) have already taken all the links that could bring it closer to the destination.

Consider now a *normal* packet  $\pi$  which in the previous time step was deflected from some node at distance  $d - 1$  from its destination, so that in the current time step it appears at node  $v$  at distance  $d$  from its destination (see Figure 7.4). The deflected packet  $\pi$  becomes *excited* at node  $v$  with probability  $p$ , and otherwise, it remains *normal* with probability



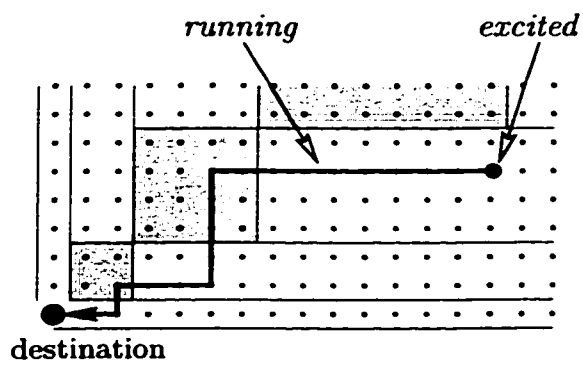
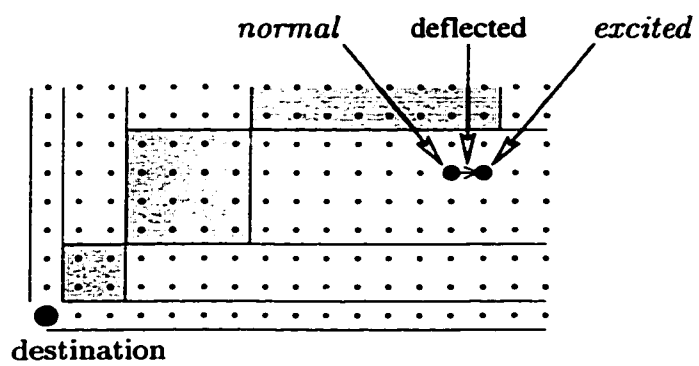


Figure 7.4: Up: a deflected packet becomes *excited*. Down: a successful *running* path

$1 - p$ . The probability  $p$  changes over time and it is given by the function

$$p(t) = \frac{c \ln t}{t},$$

where  $c$  is a constant we will specify later. To avoid notational clutter, we use  $p$  to denote  $p(t)$ , when  $t$  is clear from context. Similarly, any packet becomes *excited* with probability  $p$  whenever it is deflected.

Let's assume now that packet  $\pi$  becomes *excited* in node  $v$ , and that, without loss of generality, packet  $\pi$  has its destination node lower and left from  $v$ . The algorithm assigns each *excited* packet a *preferred link* that brings it closer to its destination. If the node  $v$  is in a *horizontal band- $z$*  of packet  $\pi$ 's destination node, the packet  $\pi$  prefers to take the left link first (see Figure 7.4). If the node  $v$  is in a *vertical band- $z$*  or in a *square- $z$*  of packet  $\pi$ 's destination node, the *excited* packet  $\pi$  prefers to take the down link first.

The *excited* packet  $\pi$  will try to follow its preferred link. There are only two cases that the *excited* packet  $\pi$  will not be able to follow its preferred link. The first case is when the *excited* packet  $\pi$  conflicts in node  $v$  with other excited packets that wish to follow the same preferred link. Such a conflict is resolved in an arbitrary way and packet  $\pi$  may lose. The second case is when packet  $\pi$  conflicts with a *running* packet, which has higher priority. The *running* packets have also a preferred link. If the *excited* packet  $\pi$  conflicts with a *running* packet for the same preferred link then packet  $\pi$  always loses. In both cases, if packet  $\pi$  loses then it loses its high priority and it enters immediately (the same time step) the *normal* state, and it will be treated in node  $v$  as a *normal* packet.

If the *excited* packet  $\pi$  succeeds in taking its preferred link, then in the new node, one link closer to its destination, the packet  $\pi$  changes its state again and it becomes a *running* packet. (Notice that packet  $\pi$  stays in the *excited* state for at most one time step.) Similar to the *excited* packets, any *running* packet has a preferred link that brings it closer to its destination. As long as the *running* packet succeeds in following its preferred link it remains in the *running* state until it reaches its destination, in which case the packet  $\pi$  is absorbed. If in some node the *running* packet is unable to follow its preferred link, because of conflicts with other *running* packets with the same preferred link, then it enters immediately the *normal* state. The preferred links for a *running* packet are chosen as follows (see Figure 7.4).

- If the *running* packet  $\pi$  was *excited* in a node  $v$  in the *horizontal band- $z$*  of its destination, then it will go directly to a random node (1 of  $z$ ) on the same row in *square- $z$*  by following the left links repeatedly. This is the case of Figure 7.4. (From *horizontal band-1*, the packet tries to go directly to its destination.)

- If the *running* packet  $\pi$  was *excited* in a *vertical band- $z$*  it will go to a random node in *square- $z/2$*  by following a one-bend path. The first part of a one-bend path has direction down and the second part has direction left. (From *vertical band-1*, the packet tries to go directly to its destination.)
- If the node  $v$  is in a *square- $z$*  of packet  $\pi$ 's destination node, it will go to a random node in *square- $z/2$*  by following a one-bend path, first down then left.

This procedure is repeated for squares  $z/2, z/4, \dots$  until the *running* packet reaches its destination node (that is,  $z = 1$ ).

A *running* packet *turns* whenever the link it exits a node is not opposite to the link the packet has entered the node. Note that a *running* packet can conflict in a node with another *running* packet, with the same preferred link, only when it turns. In such a case we specify that a turning packet loses to a non-turning packet. There are no conflicts between non-turning packets, and conflicts between turning packets are resolved in an arbitrary way. For convenience, let us call a *running* packet before its first turn a *runningA* packet; a *running* packet on or after its first turn is called a *runningB* packet.

### 7.3 Time Analysis

In this section we give the time analysis of our algorithm. We assume the same powerful adversary described in Section 6.3.

A key term in the analysis is the parameter  $m$  which for any batch problem instance has to do with the distribution of the destinations in square regions of the network.

**Definition 7.3.1** *Let  $S$  be a square subregion of the mesh, and let  $dest(S)$  be the number of packets with destinations in  $S$ . The parameter  $m$  is defined as the maximum number so that no  $k \times k$  square has more than  $mk$  destination packets. Explicitly,*

$$m := \max_{k=1}^n \frac{1}{k} \cdot \max_S dest(S)$$

as  $S$  ranges over all  $k \times k$  square subsets of the mesh.

The parameter  $m$  is related to the bandwidth lower bound and this relationship is explored in Section 7.4. Although our complexity analysis depends on  $m$ , the value of  $m$  is not known to the routing algorithm.

We immediately obtain the following lemma.

**Lemma 7.3.1**  $n \leq m \leq n^2$ .

**Proof:** There are  $n^2$  packets to route, so the maximum possible value of  $\text{dest}(S)$  is  $n^2$ . Taking  $S$  to be the entire mesh,

$$m \geq \frac{\text{dest}(S)}{n} = n.$$

Taking  $S$  to be a single node,

$$m \leq n^2.$$

■

We continue in this section as follows. First we prove in Sections 7.3.1 and 7.3.2 that the probability there will be an *excited* or *running* packet at a given node and time is small. In Section 7.3.3 we will see that a deflected packet has a good chance to be *excited* and *running* to its destination without any interruptions. Finally, in Section 7.3.4 we prove that with high probability each packet will have arrived at its destination after the promised time.

### 7.3.1 The Excited State

To avoid notational clutter in this section and in Sections 7.3.2 and 7.3.3, we will assume that all events are happening after time  $t_0$ , where the exact value of  $t_0$  will be specified in Section 7.3.4. Furthermore, we will assume that the probability of becoming excited at time  $t_0 - 2n$  is  $p$ , and thus, after time  $t_0 - 2n$  a packet becomes *excited* with probability at most  $p$  (since this probability decreases with time).

**Lemma 7.3.2** *The probability that a particular node contains no excited packet is at least  $(1 - p)^4$ .*

**Proof:** A deflected packet becomes *excited* with probability at most  $p$  only if it was deflected in the preceding time step. It will fail to become *excited* with probability at least  $1 - p$ . Since at any time step a node contains at most four packets, all four will fail to become *excited* with probability at least  $(1 - p)^4$ . ■

### 7.3.2 The Running State

**Lemma 7.3.3** *The probability that at node  $v$  and time  $t$  there is no running packet is at least  $(1 - p)^{8n-8}$ .*

**Proof:** Let's assume that at a node  $v = (x, y)$  at time  $t$  there is a *runningA* packet  $\pi$ . Packet  $\pi$  must have become *excited* at some node  $v'$  in the same row or column at time  $t' = t - \text{dist}(v, v')$ , for  $t - t' = 1, \dots, n - 1$ . From Lemma 7.3.2, node  $v'$  will not contain any *excited* packet at time  $t'$  with probability at least  $(1 - p(t'))^4 \geq (1 - p)^4$ . All the nodes  $v'$  (there are at most  $2n - 2$ ) will contain no *excited* packets at the corresponding time steps  $t'$  with probability at least

$$(1 - p)^{4(2n-2)} = (1 - p)^{8n-8}.$$

■

A *runningB* packet  $\pi$  can always be considered as being on a one-bend path from *square-z* to *square-z/2*; we say that the packet  $\pi$  is a *runningB(z)* packet.

**Lemma 7.3.4** *A runningB(z) packet  $\pi$  chooses a particular row or column with probability at most  $2/z$ .*

**Proof:** If the *runningB(z)* packet  $\pi$  is on the first part of the one-bend path (thus the preferred link is a column link), then it has chosen randomly one of  $z$  columns. If the *runningB(z)* packet  $\pi$  is on the second part of the one-bend path (thus the preferred link is a row link), then it has chosen randomly one of  $z/2$  rows. ■

For the rest of this section, we will assume that the packets have destinations down and left.

**Lemma 7.3.5** *A runningB(z) packet  $\pi$  at node  $v = (x, y)$  has its destination inside the square*

$$S := [(x - 2z + 2, y - 2z + 2), (x - \frac{z}{2} + 1, y - \frac{z}{2} + 1)].$$

**Proof:** The current position  $v$  of packet  $\pi$  must be somewhere in the square  $S' = [v', v'']$ , with  $v'$  being the lower left corner of *square-z/2* and  $v''$  being the upper right corner of *square-z* of packet  $\pi$ 's destination. If  $v$  coincides with node  $v'$  then the destination has coordinates

$$(x - \frac{z}{2} + 1, y - \frac{z}{2} + 1).$$

If  $v$  coincides with node  $v''$  then the destination has coordinates

$$(x - 2z + 2, y - 2z + 2).$$

Subsequently, the destination of packet  $\pi$  is inside the square  $S$ . ■

Let  $\pi$  be a packet in the *runningB(z)* state. We say that  $\pi$  *might arrive* at node  $v$  if there is some execution in which  $\pi$  arrives at  $v$  in the *runningB(z)* state.

**Lemma 7.3.6** *The number of runningB(z) packets that might arrive at node v is at most  $3/2 \cdot mz$ .*

**Proof:** With Lemma 7.3.5 only packets with destination inside square  $S$  can be *runningB(z)* packets at node  $v$ . The size of this square is smaller than

$$\frac{3z}{2} \times \frac{3z}{2}.$$

By Definition 7.3.1 the square  $S$  cannot have more than  $m \cdot 3z/2$  destination packets. ■

**Lemma 7.3.7** *A node v contains no runningB(z) packet at time t with probability at least*

$$\left(1 - \frac{2p}{z}\right)^{\frac{3}{2} \cdot mz}.$$

**Proof:** Consider a *runningB(z)* packet  $\pi$  that is at a node  $v$  at time  $t$ . Packet  $\pi$  has at most one chance to get *excited* and appear in  $v$  at time  $t$ . This chance is given to packet  $\pi$  at some time  $t' < t$  and at a node  $v'$  with distance  $t - t'$  from  $v$ . Packet  $\pi$  must have been deflected at time  $t' - 1$  and at time  $t'$  in node  $v'$  it becomes *excited* and follows a *running* path to node  $v$ . If packet  $\pi$  loses this chance and becomes *excited* in a subsequent deflection then it will fail to arrive at node  $v$  at time  $t$  and it will arrive there some time after  $t$ , since any subsequent deflection takes packet  $\pi$  further from node  $v$  by a link.

The probability of  $\pi$  getting *excited* at  $v'$  is at most  $p(t') < p(t)$ . If the *runningB(z)* packet  $\pi$  enters node  $v$  when it is in the column part of its path from *square-z* to *square-z/2*, then from Lemma 7.3.4 it chooses the column of node  $v$  with probability at most  $2/z$ , and similarly for the row part. Subsequently, the probability that packet  $\pi$  appears in  $v$  at time  $t$  is at most  $2p/z$  and the probability that it doesn't appear in  $v$  at all is at least

$$1 - \frac{2p}{z}.$$

According to Lemma 7.3.6, the number of possible packets like  $\pi$  is at most  $3/2 \cdot mz$ . Therefore, none of them will be in node  $v$  and time  $t$  with probability at least

$$\left(1 - \frac{2p}{z}\right)^{\frac{3}{2} \cdot mz},$$

as needed. ■

**Lemma 7.3.8** *The probability that at a node  $v$  there is no runningB packet at time  $t$  is at least  $(1 - p)^{3m(\lg n - 1)}$ .*

**Proof:** A *runningB* packet can be a *runningB(z)* packet, with  $z = 2^k$  and  $k$  is one of  $1, \dots, \lg n - 1$ . Considering all the values of  $z$ , by Lemma 7.3.7 and applying Equation 5.2, we have that there will be no *runningB* packet at node  $v$  and time  $t$  with probability at least

$$\begin{aligned} \prod_{k=1}^{\lg n - 1} \left(1 - \frac{2p}{2^k}\right)^{\frac{3}{2} \cdot m \cdot 2^k} &= \prod_{k=1}^{\lg n - 1} \left(1 - \frac{p}{2^{k-1}}\right)^{3m \cdot 2^{k-1}} \\ &\geq \prod_{k=1}^{\lg n - 1} (1 - p)^{3m} \\ &= (1 - p)^{3m(\lg n - 1)}. \end{aligned}$$

■

### 7.3.3 One Packet

**Lemma 7.3.9** *At a node  $v$  at time  $t$  there is no excited or running packet with probability at least  $(1 - p)^{12m \lg n}$ .*

**Proof:** The probability that no *excited* or *running* (*runningA* or *runningB*) packet is at node  $v$  and time  $t$  is the product of the probabilities that each of them is not at node  $v$  and time  $t$ .

- By Lemma 7.3.2, there is no *excited* packet at node  $v$  and time  $t$  with probability at least  $(1 - p)^4$ .
- By Lemma 7.3.3, there is no *runningA* packet at node  $v$  and time  $t$  with probability at least  $(1 - p)^{8n-8}$ .
- By Lemma 7.3.8, there is no *runningB* packet with destination down and left at node  $v$  and time  $t$  with probability at least  $(1 - p)^{3m(\lg n - 1)}$ . Since there are four symmetric cases, the probability to have no *runningB* packet at all is at least  $(1 - p)^{12m(\lg n - 1)}$ .

From Lemma 7.3.1 we know that  $m \geq n$ . Therefore we can bound the product (for the sake of simplicity)

$$(1 - p)^4 \cdot (1 - p)^{8n-8} \cdot (1 - p)^{12m(\lg n - 1)} > (1 - p)^{12m \lg n}.$$

■

We say that an *excited* packet travels *uninterrupted* to its destination if it becomes *running* and reaches its destination without encountering any conflicts.

**Theorem 7.3.10** *When a packet becomes excited it will travel uninterrupted to its destination with probability at least  $(1 - p)^{24m \lg^2 n}$ .*

**Proof:** By Lemma 7.3.9 there is no other *excited* or *running* packet at any specific node  $v$  and time  $t$  with probability at least

$$q := (1 - p)^{12m \lg n}.$$

After a packet  $\pi$  becomes *excited* it will become *running* and it will turn at most  $2(\lg n - 1)$  times before arriving at the destination. Each of those turns, and the single transition from *excited* to *running*, will be successful with probability at least  $q$ , since with at least this probability it will not conflict with other packets. Whenever the *running* packet  $\pi$  is not turning, it will successfully take its preferred link since it is the only packet entering from the opposite link. Therefore, an *excited* packet will succeed to reach its destination with probability at least

$$q \cdot q^{2(\lg n - 1)} \geq (1 - p)^{24m \lg^2 n}.$$

■

### 7.3.4 All Packets

Our algorithm satisfies some interesting properties with high probability (at least  $1 - 1/n$ ). We use the following constants and time steps.

$$\begin{aligned} c &= 24e & c' &= 3 \cdot 24c \lg^2 e \\ t_0 &= c'm \ln^3 n + 2n & t_1 &= 3c'm \ln^3 n \end{aligned}$$

Recall that the probability of becoming excited is

$$p(t) = \frac{c \ln t}{t}.$$

If the value of  $m$  were known to the algorithm, then  $p$  would not need to vary with time.

**Lemma 7.3.11** *If packet  $\pi$  becomes excited at time  $t \geq t_0$ , then the probability of reaching its destination is at least  $1/2e$ .*



**Proof:** Any packet conflicting with packet  $\pi$  must have been *excited* at most  $2n$  steps before  $\pi$ . The probability that such a packet became *excited* was at most

$$p = p(t_0 - 2n) = \frac{c \ln(c'm \ln^3 n)}{c'm \ln^3 n}.$$

By Theorem 7.3.10, packet  $\pi$  will reach its destination with probability at least  $(1 - p)^{24m \lg^2 n}$ . Since  $m \leq n^2$  (Lemma 7.3.1), by taking  $n$  to be sufficiently large, such that  $c' \ln^3 n \leq n$  (thus  $c'm \ln^3 n \leq n^3$ ), and by Equation 5.1 we get

$$\begin{aligned} (1 - p)^{24m \lg^2 n} &= \left(1 - \frac{c \ln(c'm \ln^3 n)}{c'm \ln^3 n}\right)^{24m \lg^2 n} \\ &\geq \left(1 - \frac{3c \ln n}{c'm \ln^3 n}\right)^{24m \lg^2 n} \\ &= \left(1 - \frac{1}{24m \lg^2 n}\right)^{24m \lg^2 n} \\ &\geq \frac{1}{e} \left(1 - \frac{1}{24m \lg^2 n}\right) \\ &\geq \frac{1}{2e}. \end{aligned}$$

■

**Lemma 7.3.12** *Each time  $t$  (with  $t_0 \leq t < t_1$ ) a packet is deflected, it will arrive uninterrupted at its destination with probability at least*

$$\frac{c}{6ec'm \ln^2 n}.$$

**Proof:** Whenever a packet is deflected with probability at least  $p(t_1)$  it becomes *excited*. Since  $n \leq m$  (Lemma 7.3.1), we get

$$\begin{aligned} p(t_1) &= \frac{c \ln(3c'm \ln^3 n)}{3c'm \ln^3 n} \\ &> \frac{c \ln n}{3c'm \ln^3 n} \\ &= \frac{c}{3c'm \ln^2 n}. \end{aligned}$$

The probability of a packet arriving at its destination without interruptions when becoming *excited* is according to Lemma 7.3.11 at least  $1/2e$ . Therefore, the probability of the packet arriving at its destination without interruptions when being deflected is at least

$$\begin{aligned} p(t_1) \cdot \frac{1}{2e} &= \frac{c}{3c'm \ln^2 n} \cdot \frac{1}{2e} \\ &= \frac{c}{6ec'm \ln^2 n}. \end{aligned}$$

■

**Lemma 7.3.13** *If a packet  $\pi$  is deflected  $x$  times, then it will reach its destination in at most  $2x + 2n - 2$  steps.*

**Proof:** Initially, the distance of  $\pi$ 's source to its destination is no more than  $2n - 2$ . Each time  $\pi$  is deflected the distance increases by a link, and each time it follows a link closer to its destination the distance decreases. ■

**Lemma 7.3.14** *With probability at least  $1 - 1/n^3$ , a packet will reach its destination in  $t_1$  steps.*

**Proof:** By Lemma 7.3.12, each time that a packet  $\pi$  is deflected in the time interval  $t_0 \leq t < t_1$ , packet  $\pi$  will arrive at its destination without interruptions with probability at least

$$q := \frac{c}{6ec'm \ln^2 n}.$$

Since we always consider the most powerful adversary, which produces the worst case scenario for interrupting the *running* path of packet  $\pi$ , successive probabilities of successfully reaching the destination after a deflection are independent. By Lemma 7.3.13, the number of deflections of packet  $\pi$  that can fit in the time interval  $t_0 \leq t < t_1$  is at least

$$\begin{aligned} x &:= \frac{t_1 - t_0 - 2n}{2} \\ &= \frac{(3c' - c')m \ln^3 n - 4n}{2} \\ &= c'm \ln^3 n - 2n. \end{aligned}$$

Since  $n \leq m$  (Lemma 7.3.1), we obtain

$$\begin{aligned} x &= c'm \ln^3 n - 2n \\ &\geq c'm \ln^3 n - \frac{1}{4}c'm \ln^3 n \\ &= \frac{3}{4}c'm \ln^3 n. \end{aligned}$$

Packet  $\pi$  will fail to reach its destination after  $x$  deflections with probability at most  $(1 - q)^x$ .

By Equation 5.1, we get

$$\begin{aligned} (1 - q)^x &\leq \left(1 - \frac{c}{6ec'm \ln^2 n}\right)^{\frac{3}{4}c'm \ln^3 n} \\ &= \left(1 - \frac{3 \ln n}{\frac{3}{4}c'm \ln^3 n}\right)^{\frac{3}{4}c'm \ln^3 n} \\ &\leq e^{-3 \ln n} \\ &= \frac{1}{n^3}. \end{aligned}$$

■

**Theorem 7.3.15** *For any batch problem instance, with high probability (at least  $1 - 1/n$ ), all packets reach their destination nodes in at most  $O(m \ln^3 n)$  steps.*

**Proof:** By Lemma 7.3.14, a packet will arrive at its destination in  $t_1$  steps with probability at least  $1 - 1/n^3$ .

Since we have made a worst case analysis for each packet by assuming that the adversary produces the worst case scenario for a packet to reach its destination, we can safely assume that the packets are independent of each other in the analysis.

Therefore, the probability that all  $n^2$  packets will arrive at their destinations within  $t_1$  steps is at least (applying Equation 5.2)

$$\left(1 - \frac{1}{n^3}\right)^{n^2} = \left(1 - \frac{1/n}{n^2}\right)^{n^2} \geq 1 - \frac{1}{n}.$$

■

## 7.4 Lower Bound

In this section, we show how our algorithm relates to the trivial lower bound

$$LB_I = \Omega(D_I + W_I).$$

Recall that for any batch problem instance  $I$  the bandwidth lower bound  $W_I$  for a region  $S$  is defined by taking the number of packets  $s_1$  with origins outside  $S$  and destinations inside  $S$ , divided by the *perimeter* of  $S$ , the number of links leading into  $S$ .

**Definition 7.4.1** *For any instance  $I$  of a routing problem, and any region  $S$  of the mesh, let  $dest(S)$  be the number of packets in  $I$  with destinations in  $S$  (independently of their origins),  $perim(S)$  the perimeter of  $S$ , and  $M_I(S) = dest(S)/perim(S)$ .*

$$M_I = \max(M_I(S)) \text{ where } S \text{ ranges over all regions } S.$$

It is immediate that  $m/4 \leq M_I$ , since  $m/4$  is the maximum  $M_I$  taken over square regions only (see Definition 7.3.1).

For any region  $S$ , let  $s_0$  be the number of packets in  $I$  with sources within  $S$ , and  $s_1$  be the number of packets with sources outside  $S$ .

$$\begin{aligned}
 M_I(S) &= \frac{s_0 + s_1}{\text{perim}(S)} \\
 &= \frac{s_0}{\text{perim}(S)} + \frac{s_1}{\text{perim}(S)} \\
 &= \frac{s_0}{\text{perim}(S)} + W_I(S) \\
 &\leq n + W_I(S).
 \end{aligned}$$

Subsequently,

$$M_I \leq n + W_I.$$

It follows that  $m/4$  differs from  $W_I$  by an additive term of at most  $n$ .

If the problem  $I$  is hard then  $LB_I$  is  $\Omega(n)$ , so  $W_I = \Omega(n)$  or  $D_I = \Omega(n)$ . If  $W_I = \Omega(n)$ , then

$$\Omega(M_I) \leq \Omega(n + W_I) = \Omega(W_I) \leq \Omega(LB_I).$$

If  $D_I = \Omega(n)$ , then

$$\Omega(M_I) \leq \Omega(n + W_I) = \Omega(D_I + W_I) = \Omega(LB_I).$$

In either case,

$$\Omega(m) \leq \Omega(M_I) \leq \Omega(LB_I).$$

Substituting in Theorem 7.3.15, we obtain our main result.

**Corollary 7.4.1** *For any instance  $I$  of a hard batch problem, with high probability, all packets reach their destination nodes in at most  $O(LB_I \cdot \ln^3 n)$  steps.*

## 7.5 Related Work

For mesh-like networks, there are many hot-potato algorithms tuned for the batch permutation and random destinations routing problems [20, 27, 35, 37, 46, 52] (see also Chapter 6).

In the more general setting of arbitrary many-to-one batch routing problems, there are several known hot-potato algorithms. Using potential function analysis, Ben-Dor *et al.* [12] provide a simple algorithm for the 2-dimensional  $n \times n$  mesh with  $O(n\sqrt{k})$  steps, where  $k$  is the total number of packets to be routed. They generalized their techniques for the

$d$ -dimensional mesh to obtain  $O(e^d n^{d-1} k^{1/d})$  steps. Borodin *et al.* [13] present a hot-potato routing algorithm for the  $d$ -dimensional mesh with  $D_I + 2(k - 1)$  steps, where  $D_I$  is the distance lower bound for any routing problem instance  $I$ . Similarly, Ben-Aroya *et al.* [10] give an algorithm that finishes in  $D_I + 2(k - 1)$  steps in the two-dimensional mesh. For single target problems, Ben-Aroya *et al.* [11] give a randomized algorithm for the  $d$ -dimensional mesh that finishes in  $O(k/d)$  steps, with high probability.

For the problems we consider here, in which there are  $n^2$  packets to be routed, all these algorithms require  $O(n^2)$  steps, which can be achieved by a naive solution. The algorithm presented here is the first that does better.

Solving arbitrary routing problems is difficult even for the traditional store-and-forward routing algorithms. The best store-and-forward algorithm is due to Mansour and Patt-Shamir [42], and performs within a factor of  $\log(D_I)$  of the lower bound  $LB_I$  for any routing problem instance  $I$ . In their algorithm the nodes have buffers of size  $\log(D_I)$ . There was no known similar result for hot-potato algorithms. It was surprising to find that for hard routing problems, in which  $D_I$  is  $\Omega(n)$ , our hot-potato algorithm matches this bound within a polylogarithmic factor even though it uses no buffers.

## 7.6 Discussion

Our algorithm carries over to the torus with a slight (constant-factor) improvement in the time bounds, because all worst-case distances are shorter.

This algorithm raises a number of open problems. The most obvious concerns “easy” batch problems with sub-linear lower bounds. We do not know whether the class of easy routing problems would yield to the same (or similar) algorithm with a more refined complexity analysis, or whether a different algorithm is needed. It would also be interesting to consider whether this kind of multi-bend algorithm could be adapted to networks of dimension higher than two, or to networks with a different topological structures. (Note that the distance and bandwidth lower bounds apply to arbitrary networks.)

## Chapter 8

# Conclusions

In this thesis, we studied two distributed structures: counting networks and the mesh network. Counting networks are used for implementing distributed counters; mesh is a communication network.

So far, counting networks have been used to solve distributed computing problems that required increment operations only, and these operations are realized by accessing the network with tokens. However, several distributed computing problems, like the semaphores, critical regions, and monitors, all rely on applying both increment and decrement operations. In counting networks, decrement operations can be realized with the use of antitokens. In Chapter 2, we show that counting networks can support simultaneously increment and decrement operations. In other words, we show that if a balancing network satisfies the step property with tokens only, then it will still satisfy the step property even when antitokens are introduced. We generalize this result to a wide class of balancing networks, which includes the smoothing networks and threshold networks, and we show that these networks preserve their respective properties, of their output sequences, even when antitokens are introduced.

In the two subsequent chapters, we presented new counting network constructions. It was an open question whether we could construct a counting network with arbitrary input/output width  $w$  that simultaneously has a small depth, and the depth expression doesn't hide any large constants. In Chapter 3, we presented the construction of an arbitrary-width counting network which has depth  $O(\log^2 w)$ , and this depth expression doesn't hide any large constants. Given any factorization of the width  $w$  (to prime factors or not), we build our counting network with balancers of input/output widths smaller or equal to the biggest factor in the factorization. By choosing the right factorization, we can trade-off balancer

widths and the depth of the network.

In Chapter 4, we presented a novel counting network construction which has input width smaller or equal to the output width. In this construction, we can fix the input width of the network and we can increase the output width arbitrarily. The depth of the network depends only on the input width, and while we increase the output width the network depth remains the same. Increasing the output width, the number of balancers in each layer of the network increases. As a consequence, the network contention decreases, since by having more balancers in the network the chance that the tokens meet in the same balancer decreases. Simultaneously, the network latency remains low, since the depth remains the same. By choosing the right output width, we can optimize the latency and contention of the network.

For the mesh network, we presented new greedy hot-potato communication algorithms. Greedy hot-potato routing algorithms have been observed to perform extremely well in practice. However, these algorithms are very hard to analyze, and so far, there had been no formal analysis that could explain their good behavior, observed in practice. In Chapter 6, we presented a new greedy hot-potato algorithm, which we analyzed for batch routing problems. We showed that for the  $n \times n$  mesh this algorithm routes any permutation and random destinations batch problem in time  $O(n \log n)$ , with high probability. This bound is a significant improvement over the previously known bound of  $O(n^2)$ .

In Chapter 7, we studied the general batch problem in the  $n \times n$  mesh, where destinations of packets are chosen arbitrarily. Any such routing problem has a lower bound  $L$ , which is a lower bound on the time needed for all the packets to be routed, for any routing algorithm. In the same chapter we presented a new greedy hot-potato routing algorithm which solves such kinds of problems. If  $L = \Omega(n)$ , the time bound obtained by our algorithm is  $O(L \cdot \log^3 n)$ , with high probability. This is the first hot-potato algorithm which is competitive to the lower bound  $L$ .

## Appendix A

# Counting Networks of Arbitrary Width

We prove Equations 3.1, 3.2, and 3.3 of Section 3.3.2.

Take any two integers  $p, q \geq 2$ . Let  $\hat{p} = \lfloor \sqrt{p} \rfloor$ ,  $\bar{p} = p - \hat{p}^2$ , and similarly define  $\hat{q}$  and  $\bar{q}$  for  $q$ . Let also  $m = \max(p, q)$ ,  $r = \max(\hat{p}, \hat{q})$ , and  $s = \max(\bar{p}, \bar{q})$ .

Obviously, if  $m = p$  then  $r = \hat{p}$ , and if  $m = q$  then  $r = \hat{q}$ . Since  $\hat{p}^2 \leq p$  and  $\hat{q}^2 \leq q$ , we have  $r^2 \leq m$  and thus Equation 3.1 holds.

We continue by showing the inequality:

$$s < 2\sqrt{p} - 1 \tag{A.1}$$

**Proof:** Since  $\hat{p} = \lfloor \sqrt{p} \rfloor > \sqrt{p} - 1$ , we have

$$\begin{aligned} \bar{p} &= p - \hat{p}^2 \\ &< p - (\sqrt{p} - 1)^2 \\ &= 2\sqrt{p} - 1. \end{aligned}$$

and thus  $\bar{p} < 2\sqrt{p} - 1$ . Similarly,  $\bar{q} < 2\sqrt{q} - 1$ .

Since  $\bar{p} < 2\sqrt{p} - 1$  and  $p \leq m$  we have  $\bar{p} < 2\sqrt{m} - 1$ . Similarly,  $\bar{q} < 2\sqrt{m} - 1$ . Since  $s = \max(\bar{p}, \bar{q})$ , we have  $s < 2\sqrt{m} - 1$  as needed.  $\blacksquare$

Next, we show the correctness of Equation 3.2 which can be written as:

$$r \left\lceil \frac{s}{2} \right\rceil \leq m \tag{A.2}$$



**Proof:** By Equation A.1, we have  $s/2 < \sqrt{m} - 1/2$ . Subsequently,  $\lceil s/2 \rceil \leq \lceil \sqrt{m} - 1/2 \rceil$ , and  $r\lceil s/2 \rceil \leq r\lceil \sqrt{m} - 1/2 \rceil$ . We only need to show that  $r\lceil \sqrt{m} - 1/2 \rceil \leq m$ .

First, we examine the case  $\sqrt{m} - r < 1/2$ . We have  $\lceil \sqrt{m} - 1/2 \rceil = r$ , and  $r\lceil \sqrt{m} - 1/2 \rceil = r^2$ . Since  $r \leq \sqrt{m}$ , we have  $r^2 \leq m$ . Therefore,  $r\lceil \sqrt{m} - 1/2 \rceil \leq m$ , as needed.

Next, we examine the case  $\sqrt{m} - r \geq 1/2$ . We have  $\lceil \sqrt{m} - 1/2 \rceil = r + 1$ , and  $r\lceil \sqrt{m} - 1/2 \rceil = r^2 + r$ . Since  $r \leq \sqrt{m} - 1/2$ , we have

$$\begin{aligned} r^2 + r &\leq \left(\sqrt{m} - \frac{1}{2}\right)^2 + \sqrt{m} - \frac{1}{2} \\ &= m - \frac{1}{4} \\ &\leq m. \end{aligned}$$

Therefore,  $r\lceil \sqrt{m} - 1/2 \rceil \leq m$ , as needed. ■

Finally, we show the correctness of Equation 3.3 which can be written as:

$$\left\lfloor \frac{s}{2} \right\rfloor \left\lfloor \frac{s}{2} \right\rfloor \leq m \tag{A.3}$$

**Proof:** By Equation A.2 we only need to show that  $\lfloor s/2 \rfloor \leq r$ . By Equation A.1, we have that  $s/2 < \sqrt{m} - 1/2$ . Therefore,  $\lfloor s/2 \rfloor \leq \lfloor \sqrt{m} - 1/2 \rfloor$ . Since  $\lfloor \sqrt{m} - 1/2 \rfloor \leq r$ , we have  $\lfloor s/2 \rfloor \leq r$ , as needed. ■

## Appendix B

# Irregular Counting Networks

**Lemma 4.1.2** If  $X^{(w)}$  and  $Y^{(w)}$ , where  $w \geq 2$ , are step sequences with highest values  $a$  and  $b$ , respectively, and step points  $k$  and  $l$ , respectively, such that

$$0 \leq \Sigma(X^{(w)}) - \Sigma(Y^{(w)}) \leq 2$$

then either  $a = b$  and

$$0 \leq k - l \leq 2,$$

or  $a = b + 1$  and

$$l = w \text{ and } 1 \leq k \leq 2. \text{ or}$$

$$l = w - 1 \text{ and } k = 1.$$

**Proof:** By Lemma 4.1.1 we have  $0 \leq a - b \leq 1$ . Therefore we have only two cases:  $a = b$  or  $a = b + 1$ .

Consider the case  $a = b$ . We will show that  $0 \leq k - l \leq 2$ . We have

$$\begin{aligned} \Sigma(X^{(w)}) &= ka + (w - k)(a - 1), \text{ and} \\ \Sigma(Y^{(w)}) &= lb + (w - l)(b - 1) \\ &= la + (w - l)(a - 1). \end{aligned}$$

Subsequently,

$$\begin{aligned} \Sigma(X^{(w)}) - \Sigma(Y^{(w)}) &= ka + (w - k)(a - 1) - la - (w - l)(a - 1) \\ &= k - l. \end{aligned}$$

Since  $0 \leq \Sigma(X^{(w)}) - \Sigma(Y^{(w)}) \leq 2$ , we have  $0 \leq k - l \leq 2$ , as needed.

Next, consider the case  $a = b + 1$ . We will show that either  $l = w$  and  $1 \leq k \leq 2$ , or  $l = w - 1$  and  $k = 1$ . We have

$$\begin{aligned}\Sigma(X^{(w)}) &= ka + (w - k)(a - 1), \text{ and} \\ \Sigma(Y^{(w)}) &= lb + (w - l)(b - 1) \\ &= l(a - 1) + (w - l)(a - 2).\end{aligned}$$

Subsequently,

$$\begin{aligned}\Sigma(X^{(w)}) - \Sigma(Y^{(w)}) &= ka + (w - k)(a - 1) - l(a - 1) - (w - l)(a - 2) \\ &= k - l + w.\end{aligned}$$

Since  $0 \leq \Sigma(X^{(w)}) - \Sigma(Y^{(w)}) \leq 2$ , we have  $0 \leq k - l + w \leq 2$ . If  $l = w$  then  $k \leq 2$ , and since  $k$  is a step point we have  $k \geq 1$  and thus  $1 \leq k \leq 2$ , as needed. Otherwise, if  $l < w$  from inequality  $k - l + w \leq 2$ , we get  $k + 1 \leq 2$ . Since  $k \geq 1$ , we must have  $k = 1$ . Furthermore, from inequality  $k - l + w \leq 2$  we get  $l \geq w - 1$ . Since  $l < w$ , we have  $l = w - 1$ , as needed.

■

# Bibliography

- [1] A. S. Acampora and S. I. A. Shah. Multihop lightwave networks: a comparison of store-and-forward and hot-potato routing. In *Proc. IEEE INFOCOM*, pages 10–19, 1991.
- [2] E. Aharonson and H. Attiya. Counting networks with arbitrary fan-out. *Distributed Computing*, 8(4):163–169, 1995.
- [3] W. Aiello, C. Busch, M. Herlihy, M. Mavronicolas, N. Shavit, and D. Touitou. Supporting increment and decrement operations in balancing networks. In *Proceedings of the 16th International Symposium on Theoretical Aspects of Computer Science (STACS'99)*, *Lecture Notes in Computer Science*, Springer-Verlag, volume 1563, pages 377–386, Trier, Germany, 1999. Also, to appear in the Chicago Journal of Theoretical Computer Science.
- [4] W. Aiello, R. Venkatesan, and M. Yung. Coins, weights and contention in balancing networks. In *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing (PODC'94)*, pages 193–205, Los Angeles, August 1994.
- [5] M. Ajtai, J. Komlós, and E. Szemerédi. An  $O(n \log n)$  sorting network. *Combinatorica*, 3:1–19, 1983.
- [6] J. Aspnes, M. Herlihy, and N. Shavit. Counting networks. *Journal of the ACM*, 41(5):1020–1048, September 1994.
- [7] A. Bar-Noy, P. Raghavan, B. Schieber, and H. Tamaki. Fast deflection routing for packets and worms. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Distributed Computing*, pages 75–86, Ithaca, New York, USA, August 1993.
- [8] P. Baran. On distributed communications networks. *IEEE Transactions on Communications*, pages 1–9, 1964.

- [9] K. E. Batcher. Sorting networks and their applications. In *Proc. AFIPS 1968 Spring Jt. Computer Conf.*, volume 32, pages 307–314, Washington, D.C., 1968. Thompson Book,Co.
- [10] I. Ben-Aroya, T. Eilam, and A. Schuster. Greedy hot-potato routing on the two-dimensional mesh. *Distributed Computing*, 9(1):3–19, 1995.
- [11] I. Ben-Aroya, I. Newman, and A. Schuster. Randomized single-target hot-potato routing. *Journal of Algorithms*, 23(1):101–120, April 1997.
- [12] A. Ben-Dor, S. Halevi, and A. Schuster. Potential function analysis of greedy hot-potato routing. *Theory of Computing Systems*, 31(1):41–61, January/February 1998.
- [13] A. Borodin, Y. Rabani, and B. Schieber. Deterministic many-to-many hot potato routing. *IEEE Transactions on Parallel and Distributed Systems*, 8(6):587–596. June 1997.
- [14] A. Broder and E. Upfal. Dynamic deflection routing on arrays. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 348–358, May 1996.
- [15] C. Busch, N. Demetriou, M. Herlihy, and M. Mavronicolas. Threshold counters with increments and decrements. In *Proceedings of the 6th International Colloquium on Structural Information and Communication Complexity (SIROCCO'99)*, pages 47–61, Lacaunau, France, July 1999.
- [16] C. Busch, N. Demetriou, M. Herlihy, and M. Mavronicolas. A combinatorial characterization of properties preserved by antitokens. In *Proceedings of the European Conference on Parallel Computing (Euro-Par 2000)*, Munich, Germany, 2000. To appear.
- [17] C. Busch, N. Hardavellas, and M. Mavronicolas. Contention in counting networks (abstract). In *Proceedings of the 13th annual ACM Symposium on Principles of Distributed Computing (PODC'94)*, page 404, Los Angeles, August 1994.
- [18] C. Busch and M. Herlihy. Sorting and counting networks of small-depth and arbitrary width. In *Proceedings of the 11th ACM Symposium on Parallel Algorithms and Architectures (SPAA '99)*, pages 64–73, Saint-Malo, France, June 1999.
- [19] C. Busch, M. Herlihy, and R. Wattenhofer. Hard-potato routing. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC'00)*, May 2000. To appear.

- [20] C. Busch, M. Herlihy, and R. Wattenhofer. Randomized greedy hot-potato routing. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '00)*, pages 458–466, January 2000.
- [21] C. Busch and M. Mavronicolas. A combinatorial treatment of balancing networks. *Journal of the ACM*, 43(5):794–839, September 1996.
- [22] C. Busch and M. Mavronicolas. Impossibility results for weak threshold networks. *Information Processing Letters*, 63(2):85–90, July 1997.
- [23] C. Busch and M. Mavronicolas. An efficient counting network. In *Proceedings of the 1st Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing (IPPS/SPDP'98)*, pages 380–385, March 1998.
- [24] E. W. Dijkstra. Cooperating sequential processes. In *Programming Languages*, pages 43–112. Academic Press, 1968.
- [25] M. Dowd, Y. Perl, L. Rudolph, and M. Saks. The periodic balanced sorting network. *Journal of the ACM*, 36(4):738–757. October 1989.
- [26] C. Dwork, M. Herlihy, and O. Waarts. Contention in shared memory algorithms. *Journal of the ACM*, 44(6):779–805. November 1997.
- [27] U. Feige and P. Raghavan. Exact analysis of hot-potato routing. In IEEE, editor, *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 553–562, Pittsburgh, PN, October 1992. IEEE Computer Society Press.
- [28] E. W. Felten, A. LaMarca, and R. Ladner. Building counting networks from larger balancers. Technical Report TR 93-04-09, University of Washington, April 1993.
- [29] A. G. Greenberg and J. Goodman. Sharp approximate models of deflection routing. *IEEE Transactions on Communications*, 41(1):210–223, January 1993.
- [30] B. Hajek. Bounds on evacuation time for deflection routing. *Distributed Computing*, 1:1–6, 1991.
- [31] P. B. Hansen. *Operating System Principles*. Prentice-Hall, Englewood Cliffs, NJ, 1973.
- [32] N. Hardavellas, D. Karakos, and M. Mavronicolas. Notes on sorting and counting networks. In *Proceedings of the 7th International Workshop on Distributed Algorithms (WDAG'93)*, volume 725 of *Lecture Notes in Computer Science*, pages 234–248, Lausanne, Switzerland, September 1993. Springer-Verlag.

- [33] W. D. Hillis. *The Connection Machine*. MIT press, 1985.
- [34] C. A. R. Hoare and R. N. Periott. *Operating Systems Techniques*. Academic Press (New York NY), London, 1972.
- [35] Ch. Kaklamanis, D. Krizanc, and S. Rao. Hot-potato routing on processor arrays. In *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 273–282, Velen, Germany, June 30–July 2, 1993. SIGACT and SIGARCH.
- [36] S. Kapidakis and M. Mavronicolas. Distributed. low contention task allocation. In *Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing (SPDP'96)*, pages 358–365, Washington, October 1996.
- [37] M. Kaufmann, H. Lauer, and H. Schroder. Fast deterministic hot-potato routing on meshes. In Springer-Verlag, editor, *Proc. of the 5th International Symposium on Algorithms and Computation (ISAAC)*. *Lecture Notes in Computer Science*, volume 834, pages 333–341, 1994.
- [38] M. Klugerman. *Small-Depth Counting Networks and Related Topics*. PhD thesis, Department of Mathematics, Massachusetts Institute of Technology, September 1994.
- [39] M. Klugerman and C. G. Plaxton. Small-depth counting networks. In *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing (STOC'92)*, pages 417–428, Victoria, B.C., Canada, May 1992.
- [40] D. E. Knuth. *The Art of Computer Programming III: Sorting and Searching*, volume 3. Addison-Wesley, 1973.
- [41] N. Lynch, N. Shavit, A. Shvartsman, and D. Touitou. Counting networks are practically linearizable. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC'96)*, pages 280–289, New York, May 1996.
- [42] Y. Mansour and B. Patt-Shamir. Many-to-one packet routing on grids. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 258–267, 29 May–1 June 1995.
- [43] M. Mavronicolas, M. Merritt, and G. Taubenfeld. Sequentially consistent versus linearizable counting networks. In *Proceedings of the 18th Annual ACM Symposium on Principles of Distributed Computing (PODC'99)*, pages 133–142, Atlanta, Georgia, May 1999.

- [44] M. Mavronicolas, M. Papatriantafyllou, and P. Tsigas. The impact of timing on linearizability in counting networks. In *Proceedings of the 11th International Parallel Processing Symposium (IPPS'97)*, pages 684–688, Los Alamitos, April 1997.
- [45] N. F. Maxemchuk. Comparison of deflection and store and forward techniques in the Manhattan street and shuffle exchange networks. In *Proc. IEEE INFOCOM*, pages 800–809, 1989.
- [46] I. Newman and A. Schuster. Hot-potato algorithms for permutation routing. *IEEE Transactions on Parallel and Distributed Systems*, 6(11):1168–1176, November 1995.
- [47] C. L. Seitz. The caltech mosaic C: An experimental, fine-grain multicomputer. In *4th symp. on Parallel Algorithms and Architectures*, June 1992. Keynote Speech.
- [48] N. Shavit and D. Touitou. Elimination trees and the construction of pools and stacks. *Theory of Computing Systems*, 30(6):545–570, Nov./Dec. 1997.
- [49] N. Shavit and A. Zemach. Diffracting trees. *ACM Transactions on Computer Systems*, 14(4):385–428, November 1996.
- [50] A. Silberschatz and P. B. Galvin. *Operating System Concepts*. Addison Wesley, 4th edition, 1994.
- [51] B. Smith. Architecture and applications of the HEP multiprocessor computer system. In *Proc. Fourth Symp. Real Time Signal Processing IV*, pages 241–248. SPIE, 1981.
- [52] P. Spirakis and V. Triantafyllou. Pure greedy hot-potato routing in the 2-D mesh with random destinations. *Parallel Processing Letters*, 7(3):249–258, September 1997.
- [53] T. Szymanski. An analysis of “hot potato” routing in a fiber optic packet switched hypercube. In *Proc. IEEE INFOCOM*, pages 918–925, 1990.
- [54] Z. Zhang and A. S. Acampora. Performance analysis of multihop lightwave networks with hot potato routing and distance age priorities. In *Proc. IEEE INFOCOM*, pages 1012–1021, 1991.