ELSEVIER

Computer Communications 22 (1999) 1574–1584

computer communications

www.elsevier.com/locate/comcom

Fast knockout algorithm for self-route concentration

Shuo-Yen Robert Li*, Hui Li, Gar Man Koo

Department of Information Engineering, The Chinese University of Hong Kong, Hong Kong, People's Republic of China

Received 20 January 1999; received in revised form 15 June 1999; accepted 29 June 1999

Abstract

Concentration has been a popular technique for resolving contention in switching. In particular, self-route concentration is useful inside ATM switching fabrics in many ways. This paper presents an efficient algorithm for the construction of self-route concentrators from multi-stage cascades of 2×2 sorters, where the main criterion on the complexity is the number of stages in the cascade. The algorithm extensively generalizes the existing *Fast Knockout* algorithm and mixes in some *k-sorting* technique. The result includes a best known construction of m-to-n self-route concentrator for all m and n within the practical range. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: ATM switching fabric; Self-route concentration; Fast knockout algorithm; k-sorting algorithm

1. Introduction

Concentration is a technique used in many ways for resolving contention in switching. One type of self-route concentrator in an ATM switching fabric [1–5] is by multi-stage networking of parallel 2×2 sorters and routers, where outputs of one stage lead to inputs of a later stage. In particular, an m-to-n self-route concentrator sorts the m inputs to the extent that the largest n among them can be separated from the remaining $m-n$.

The building block of self-route concentrators is typically a 2×2 device with the internal connection pattern in three possible states: *bar*, *cross* and *undecided*. A 2×2 sorter, depicted by Fig. 1(a) compares the two digital inputs bit by bit until there is a difference between them. Various types of building blocks in self-route switching fabrics can be regarded as special cases of a 2×2 sorter. For example, the 2×2 routing cells, with a 0-bound output and a 1-bound output as depicted in Fig. 1(b), can be regarded as a sorter according to the order of: '0-bound' > 'idle' > '1-bound'.

In a multi-stage packet switching fabric, concentration performed at a certain stage reduces the number of packets that advance to later stages and thereby reduces the blocking probability at later stages. When properly designed, this reduction in blocking at later stages can more than offset the blocking incurred inside concentrators. Therefore concentrators constructed from multi-stage networking of 2×2 sorters are especially suitable for self-route ATM switching.

Unlike software algorithms for sorting, such as "*quicksort*" [6] and hashing [7], concentration algorithms are hardware algorithms that do not have a centralized flow control and require no external storage. A great deal of literature has been devoted to sorting networks and concentration architectures, but relatively little is known about the complexity until recently. The time complexity of a concentration network is the packet delay through the concentration network, which can be measured by the number of stages, also called the *depth*. The space complexity can be measured either by the depth or by the number of 2×2 sorters, which is also called the *cost*. In Ref. [8], a concentrator of O(N) cost has been constructed, where the coefficient of the linear order is formidably large.

This paper is aimed at the construction of efficient m-to-n concentrators with parameters m and n within the practical range rather than the asymptotic order of the complexity. We shall adopt the depth as the main measure of complexity for the following reasons.

Constructions in this paper will typically incur low cost with respect to the depth.

Since there can be at most $m/2$ 2×2 sorters at each stage, the depth is proportionally an upper bound of the cost. It is not proportional to the cost itself since some of the outputs from a stage may skip the next stage. However, in many applications, inputs to each stage need to be synchronized in order to facilitate the design logistics. Thus, when a stage is skipped, the introduction of a delay element may be needed in order to maintain the synchronization. When the extra

* Corresponding author. Fax: + 86-85226035032.
E-mail address: bobli@ie.cuhk.edu.hk (S.-Y.R. Li)

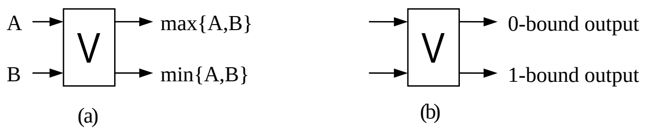

Fig. 1. The 2×2 sorter.

cost of delay elements is added on top of 2×2 sorters, the two measures of the complexity become more proportional.

Even when synchronization is not required, often the VLSI layout aligns 2×2 sorters stage by stage. When the empty space introduced by such alignment is taken into consideration, the total space in the layout of components would be very much proportional to the depth.

Section 2 reviews the *Fast Knockout* (FKO) technique introduced in Ref. [9], wherein the concentration algorithm is aimed at the case when the number of inputs is a power of 2. In principle, any number of inputs can be rounded up to the next power of 2 for invoking this algorithm. However, the inefficiency introduced by the rounding is frequently substantial. Section 3 presents a conceptual generalization of the FKO algorithm in a more efficient way than just rounding. The generalized FKO algorithm requires a sub-algorithm for "*pairing*". Examples of such a sub-algorithm are presented in Section 4. The generalized algorithm is refined in Section 5 for further improvement in efficiency.

The nature of knockout-style concentration not only performs concentration but also automatically sorts the outputs. Striving for ultimate efficiency, Section 6 adapts FKO into an algorithm for concentration without necessarily sorting the outputs. The adapted algorithm, together with the k-sortout algorithm [9], generates the best known self-route m-to-n concentration for all m and n within the practical range. The performance is compared between the new algorithm and the previously arts. Finally, Section 7 concludes the results and suggests possible directions for further research.

2. Fast knockout

A knockout tournament, as in human athletic competitions, is a binary tree of one-against-one matches for selecting the champion. Knockout tournaments are sometimes compounded for selecting a plurality of top winners. Thus there are n knockout tournaments; all losers of the kth tournament, where $k < n$, enter the $(k+1)$st tournament while the winner gets the kth place among n winners. The n tournaments may be scheduled as partially overlapping in time. We shall refer to this type of comparison processes as *Conventional Knockout* (CKO) tournaments. It has first been adopted by Yeh et al. [5] in switching design.

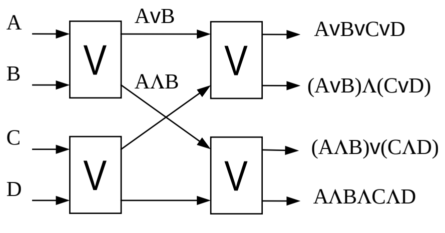

Fig. 2. Boolean outputs of a 2-stage network of 2×2 sorters.

As two packets enter a 2×2 sorter, the numerical contents of their headers are compared against each other. When two numbers, instead of two human athletes, are being compared, they obey laws in Boolean algebra. In Fig. 2, the notation ' $\vee$ ' and ' $\wedge$ ' stand for the Boolean operations of maximum and minimum, respectively.

Outputs from a large network can be very complex Boolean expressions in terms of variables representing the inputs. It is not clear in general how such Boolean expressions can be best adapted for concentration. The CKO algorithm, as well as other concentration algorithms, has to somehow deduce from the Boolean expressions the information more directly related to concentration. The more loss of information contents the logical deduction incurs, the less efficient is the resulting algorithm.

In Ref. [9], CKO tournaments are interpreted as reducing the Boolean information of a partially sorted set to a *state vector* defined below.

Definition 2.1. In a partially sorted set of numbers, define an *i-element* as an element that is known to be smaller than at least i other elements in the set.

Definition 2.2. Fix a number n. If a partially sorted set S of numbers is the disjoint union $S_0 \cup S_1 \cup \ldots \cup S_{n-1} \cup S_n$, where each S_i consists of i-elements of S, then S is said to have the state vector $(|S_0|, |S_1|, \ldots, |S_{n-1}|)$, an n-dimensional integer vector.

A digital signal, or a packet, progressing through an m-to-n self-route concentrator is treated as a binary number. Through each stage of routing cells, new information on the ordering among the m numbers is gathered. The CKO algorithm for concentration starts with the initial state vector $(m,0,0,\ldots,0)$ and terminates at $(1,1,1,\ldots,1)$. This terminating state vector shows that the CKO algorithm performs not only the m-to-n concentration but also the linear sorting among the n winners. Therefore, we shall regard CKO as an algorithm for *concentration/sorting*, as opposed to pure concentration to be considered in Section 6.

The FKO algorithm, to be described in the sequel, is designed to take advantage of the transitive law that governs the ordering on numbers. Let X be an x-element in a set and Y a y-element in a disjoint set. When the two elements X and Y are replaced by $X \vee Y$ and $X \wedge Y$, the latter would be an $(x+y+1)$-element in the union set. Translating into knockout tournaments, this means that the loser in the match between X and Y can go directly into the $(x+y+1)$st tournament. Take the example of the network in Fig. 2, the FKO algorithm would treat the lowest output as a 3-element rather than just a 2-element. This advantage

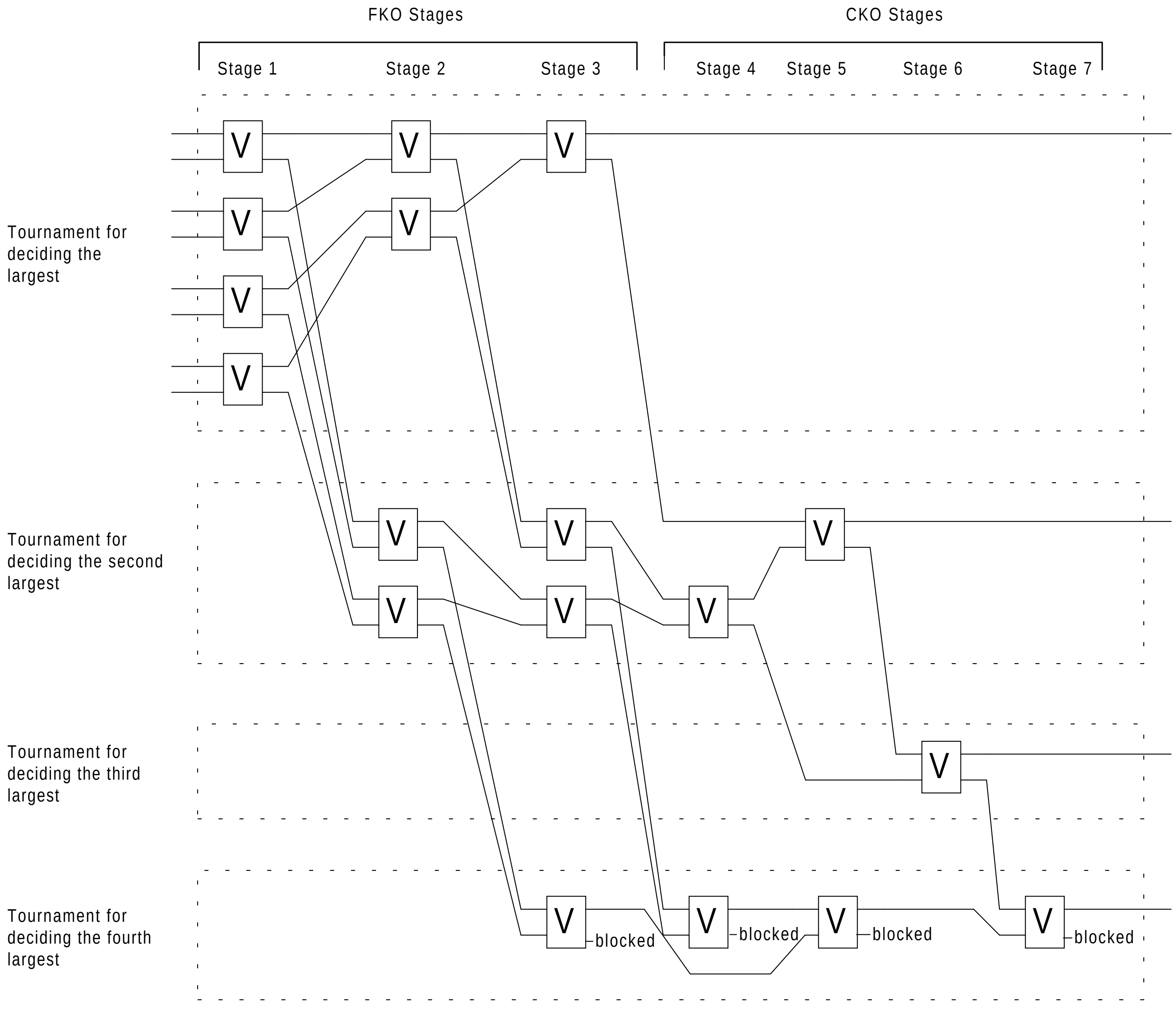
FKO Stages
CKO Stages
Stage 1
Stage 2
Stage 3
Stage 4
Stage 5
Stage 6
Stage 7
Tournament for deciding the largest
Tournament for deciding the second largest
Tournament for deciding the third largest
Tournament for deciding the fourth largest
V
blocked

Fig. 3. The four knock tournaments inside the 8-to-4 FKO concentrator/sorter.

speeds up the knockout process. For this purpose, FKO deduces from the Boolean information of a partially sorted set a state vector of a slightly more complex form than CKO.

Definition 2.3. The disjoint union of z sets with the common state vector $(x_0, x_1, x_2, \ldots x_{n-1})$ is said to have the state vector $z \times (x_0, x_1, x_2, \ldots x_{n-1})$.

Algorithm 2.4 (FKO for 2^k-to-n concentration/sorting [9]). Start with the state vector $2^k \times (1, 0, 0, \ldots, 0)$. Apply the *FKO stage* for k times and follow through with *CKO stages* until the state vector becomes (1,1,1,…,1), where FKO and CKO stages are defined as follows:

FKO stage Let S and T be two disjoint sets sharing the same state vector $(x_0, x_1, x_2, \ldots, x_{n-1})$ and $0 \le i < n$. Pair every i-element of S with an i-element of T for comparison. The loser in the comparison becomes a $(2i+1)$-element of $S \cup T$. Through these comparisons, the state vector of $S \cup T$ changes from $2 \times (x_0, x_1, x_2, \ldots, x_{n-1})$ to $(x_0, x_0 + x_1, x_2, x_1 + x_3, x_4, x_2 + x_5, \ldots, x_{2i}, x_i + x_{2i+1}, \ldots)$. Similarly, when there are $2z$ disjoint sets sharing the same state vector, the state vector of their union changes in a single stage by:

$$2z \times (x_0, x_2, \ldots, x_{n-1}) \rightarrow z \times (x_0, x_0 + x_1, x_2, x_1 + x_3, x_4, x_2 + x_5, \ldots, x_{2i}, x_i + x_{2i+1}, \ldots)$$

CKO stage Let a set in the state $(x_0, x_1, x_2, \ldots x_{n-1})$ be given. For $0 \le i < n$, form $\lfloor x_i/2 \rfloor$ pairs of i-elements for parallel comparisons. The state vector of the set

Table 1
Evolution of the state vector in the 8-to-4 concentration/sorting by CKO and FKO

Stage	State vector	
	CKO tournaments	FKO algorithm
0	(8,0,0,0)	$8\times(1,0,0,0)$
1	(4,4,0,0)	$4\times(1,1,0,0)$
2	(2,4,2,0)	$2\times(1,2,0,1)$
3	(1,3,3,1)	$1\times(1,3,0,3)$
4	(1,2,3,2)	$1\times(1,2,1,2)$
5	(1,1,3,2)	$1\times(1,1,2,1)$
6	(1,1,2,2)	$1\times(1,1,1,2)$
7	(1,1,1,2)	$1\times(1,1,1,1)$
8	(1,1,1,1)	

changes by:

$$(x_0,x_1,x_2,\ldots x_{n-1})\rightarrow\left(\left\lceil\frac{x_0}{2}\right\rceil,\left\lfloor\frac{x_0}{2}\right\rfloor+\left\lceil\frac{x_1}{2}\right\rceil,\ldots,\left\lfloor\frac{x_{i-1}}{2}\right\rfloor+\left\lceil\frac{x_i}{2}\right\rceil,\ldots,\left\lfloor\frac{x_{n-2}}{2}\right\rfloor+\left\lceil\frac{x_{n-1}}{2}\right\rceil\right)$$

through this stage.

Fig. 3 lays out the four knockout tournaments in the 8-to-4 FKO concentrator/sorter. Meanwhile, the evolution processes of the state vector in the 8-to-4 concentration by CKO tournaments and by the FKO algorithm, respectively, are contrasted in Table 1.

Since algorithms from the knockout approach are for the combined purpose of concentration/sorting, they are not very efficient for just the concentration per se when the concentration ratio is small. For example, a better 8-to-4 concentrator/sorter than both FKO and CKO is the classical *Batcher* sorter [10], which requires only six stages. Despite this, all illustrative examples throughout this paper use small parameters m and n for the convenience of presentation.

3. Generalized fast knockout algorithm

While taking advantage of the transitive law, Algorithm 2.4 has managed to keep the state vector in a simple form. Before the kth stage, the state vector is in the form of $2^j\times(x_0,x_1,x_2,\ldots,x_{n-1}), j>0$. Thereafter the state vector is in the simple form of $(x_0,x_1,x_2,\ldots,x_{n-1})$ when CKO stages take over. These are the only two types of state vectors throughout.

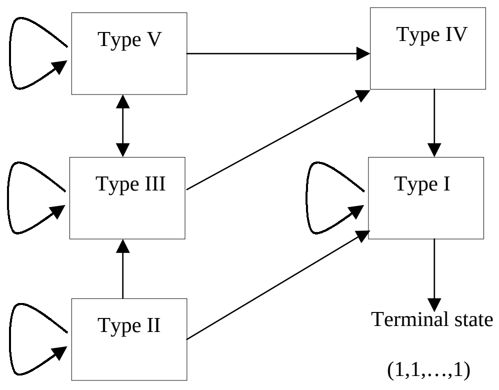

Fig. 4. Transition diagram of the state vector in the generalized FKO algorithm.

In principle, any number of inputs can be rounded up to the next power of 2 in order to apply Algorithm 2.4. However, the inefficiency introduced by the rounding is frequently substantial. To generalize Algorithm 2.4 to m-to-n concentration/sorting, where m is not necessarily a power of 2, the goal again is to try to take full advantage of the transitive law that governs the ordering among numbers. Take the 12-to-5 example for illustration. The initial state vector is $12\times(1,0,0,0,0)$. Two FKO stages apply naturally and turn the state vector first into $6\times(1,1,0,0,0)$ and then into $3\times(1,2,0,1,0)$. Since the multiplier 3 is an odd number, there is no natural way to apply a third FKO stage. We could replace the state vector $3\times(1,2,0,1,0)$ with $(3,6,0,3,0)$ to suit a CKO stage, but that would surrender the knowledge of the disjointness among three partially sorted sets. A better alternative is to regard $3\times(1,2,0,1,0)$ as a new compound form of $2\times(1,2,0,1,0)+(1,2,0,1,0)$, where ‘ + ’, as well as ‘ × ’, indicates disjoint union. Then, we can apply an FKO stage to the part represented by $2\times(1,2,0,1,0)$ and a CKO stage to the residual part. In this way, we take advantage of the disjointness between two of the three partially sorted sets and maintain the disjointness of their union from the third set.

Table 2
Types of state vectors and their corresponding algorithmic actions

Type of state vector	Algorithmic action
Type I: $(x_0,x_1,x_2,\ldots,x_{n-1})$ except for (1,1,1,…,1)	Apply a CKO stage.
Type II: $2k\times(x_0,x_1,x_2,\ldots,x_{n-1})$	Apply an FKO stage.
Type III: $2k\times(x_0,x_1,x_2,\ldots,x_{n-1})+(x'_0,x'_1,x'_2,\ldots,x'_{n-1})$	Apply FKO to the part represented by the first term and CKO to the remainder separately.
Type IV: $(x_0,x_1,x_2,\ldots,x_{n-1})+(x'_0,x'_1,x'_2,\ldots,x'_{n-1})$	Merge the two disjoint sets represented by the two terms with a sub-algorithm for *pairing*.
Type V: $2k\times(x_0,x_1,x_2,\ldots,x_{n-1})+(x'_0,x'_1,x'_2,\ldots,x'_{n-1})+(x''_0,x''_1,x''_2,\ldots,x''_{n-1})$	Apply FKO to the part represented by the first term; merge the parts represented by the second and third terms with a sub-algorithm for *pairing*.

As the algorithmic action gets more complicated, more compound forms of the state vector may result. The algorithmic action, must at all times contain the state vector within manageable form so that a computer can generate the hardware design within a reasonable time.

Algorithm 3.1 (FKO for *m*-to-*n* concentration/sorting). Start with the state vector $m \times (1, 0, 0, \ldots, 0)$. At every stage, perform the algorithmic action as prescribed in Table 2 depending upon the type of the state vector.

The finite-state diagram in Fig. 4 describes the transition of the type of the state vector through one stage of comparisons.

We now illustrate this generalized algorithm by the 12-to-5 example. After the initial two FKO stages, the state vector becomes $2 \times (1, 2, 0, 1, 0) + (1, 2, 0, 1, 0)$, which is of Type III. After another stage of comparisons, the state vector becomes $(1, 3, 0, 3, 0) + (1, 1, 1, 1, 0)$, which belongs to Type IV. Fig. 5 depicts the three initial stages in the 12-to-5 FKO, and Table 3 gives the evolution of the state vector through them.

The competing numbers that have survived through three stages of comparisons are divided into two disjoint sets with state vectors (1,3,0,3,0) and (1,1,1,1,0), respectively. The

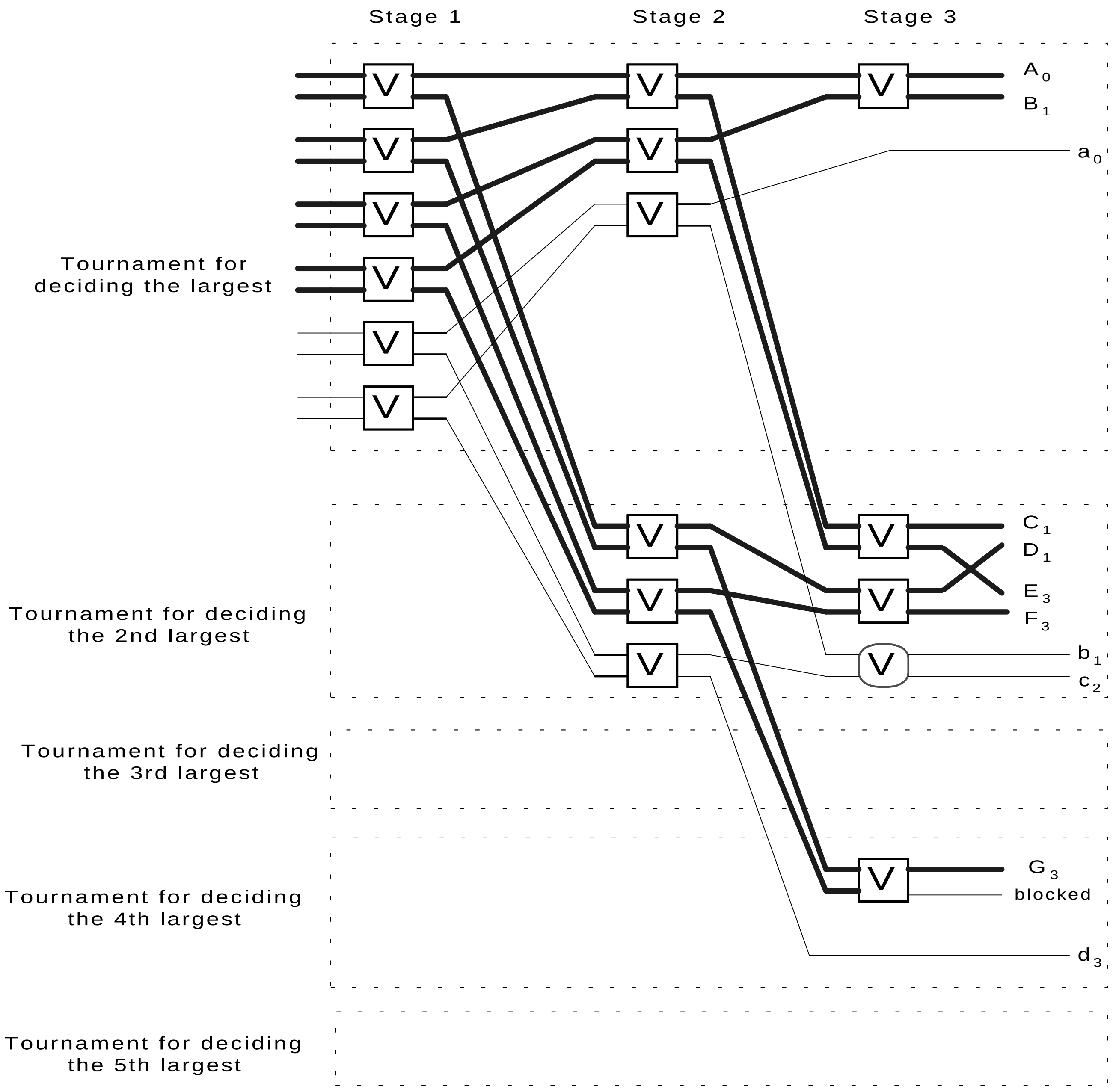
Stage 1
Stage 2
Stage 3
Tournament for deciding the largest
Tournament for deciding the 2nd largest
Tournament for deciding the 3rd largest
Tournament for deciding the 4th largest
Tournament for deciding the 5th largest
A_0
B_1
a_0
C_1
D_1
E_3
F_3
b_1
c_2
G_3
blocked
d_3

Fig. 5. Three initial stages by the 12-to-5 FKO algorithm.

Table 3
The evolution of the state vector through the initial three stages by the 12-to-5 FKO algorithm

Stage	Resulting state vector
0	$12 \times (1, 0, 0, 0, 0)$
1	$6 \times (1, 1, 0, 0, 0)$
2	$3 \times (1, 2, 0, 1, 0) =$
	$2 \times (1, 2, 0, 1, 0) + (1, 2, 0, 1, 0)$
3	$(1, 3, 0, 3, 0) + (1, 1, 1, 1, 0)$

next stage needs to merge the two sets together in a way that takes the full advantage of their disjointness. An algorithm for this purpose will be called a sub-algorithm for *pairing*, examples of which are presented in the next section.

4. Sub-algorithms for pairing

As stated in Table 2, a sub-algorithm for pairing should prescribe a whole stage of comparisons when the state vector is of Type IV. It is also invoked to prescribe part of a stage when the state vector is of Type V.

The initial 12 inputs in Fig. 5 can be divided into a group of 8 elements and another of 4 such that every comparison in the three stages is within one group or the other. Outputs of stage 3 are thus in two disjoint sets, one consisting of elements labeled $A_0, B_1, C_1, D_1, E_3, F_3$, and G_3 in Fig. 5 and the other of a_0, b_1, c_2, and d_3. A generic subscript j in the label of an element indicates a j-element, and the alphabetical order in labeling follows the increasing order of the subscript. This labeling suggests the following simple but effective pairing algorithm.

Algorithm 4.1 (The Left-first sub-algorithm for pairing). Apply the recursive step below until one of the two sets is exhausted. Then, apply CKO to the residual elements.

Recursive step Remove an x-element from one of the sets with the smallest x and a y-element from the other set with the smallest possible y. Make a comparison between the two removed elements.

In the recursive step of this algorithm, two elements from two disjoint sets are compared. This takes advantage of the transitive law in the FKO style. An immediate variation of the Left-first paring sub-algorithm is as follows.

Algorithm 4.2 (The Match-first sub-algorithm for pairing). Let A and B be two disjoint sets with state vectors $(a_0, a_1, a_2, \ldots, a_{n-1})$ and $(b_0, b_1, b_2, \ldots, b_{n-1})$,, respectively. Apply the following recursive step until there is no index j such that both a_j and b_j are nonzero. Then perform the Left-first pairing sub-algorithm to residual sets.

Recursive step Let j be an index such that both a_j and b_j are nonzero. Take a j-element out of each set for comparison.

Table 4 below tabulates the evolution of the state vector in 12-to-6 FKO concentration/sorting with the two pairing sub-algorithms, respectively.

Either of the two sub-algorithms in the above may be invoked by Algorithm 3.1. Within the practical range of $4 \leq m \leq 3000$ and $2 \leq n \leq 32$, there are numerous cases where Match-first is superior to Left-first in terms of the depth of the network, while there are just 30 cases where Left-first is better.

Moreover, there are cases when neither sub-algorithm is optimal, e.g. when $20 \leq m \leq 21$ and $7 \leq n \leq 10$. This fact suggests the needs of further fine-tuning of the FKO algorithm, which is to be done in the next section. Below we present another sub-algorithm for pairing, of which the core subroutine (Algorithm 4.4) will be useful in the next section.

Algorithm 4.3 (Another sub-algorithm for pairing). Let A and B be independent and partially sorted sets. Assume without loss of generality that A contains at least as many elements as B. Apply Algorithm 4.4 recursively until D is depleted with the sets $C = A$ and $D = B$ initially. Then, apply CKO comparisons to the residual set of A.

Algorithm 4.4 (Subroutine for selecting an element from each of two given sets). Let A and B be disjoint partially sorted sets. Let C and D be subsets of A and B, respectively. Select an i-element of C and a j-element of D for comparison according to the following criteria for selecting the pair (i, j) :

Main criterion: To minimize $|i - j|$.
Tiebreaker: To minimize $\min\{i, j\}$.

Example. In Algorithm 4.3, let A and B have the state vectors (1,4,0,6) and (1,1,2,1), respectively. Then Algorithm 4.4 is invoked five times with the selected pairs (i, j) at each iteration being (0,0), (1,1), (3,3), (1,2), and (1,2).

Table 4
The evolution of the state vector in the 12-to-6 FKO with Left- and Match-first pairing

Stage	Left-first pairing	Match-first pairing
0	$12 \times (1, 0, 0, 0, 0, 0)$	
1	$6 \times (1, 1, 0, 0, 0, 0)$	
2	$3 \times (1, 2, 0, 1, 0, 0) = 2 \times (1, 2, 0, 1, 0, 0) + (1, 2, 0, 1, 0, 0)$	
3	$(1, 3, 0, 3, 0, 0) + (1, 1, 1, 1, 0, 0)$	
4	(1,4,0,3,2,1)	(1,4,0,3,2,0)
5	(1,2,2,2,2,2)	(1,2,2,2,2,1)
6	(1,1,2,2,2,2)	
7	(1,1,1,2,2,2)	
8	(1,1,1,1,2,2)	
9	(1,1,1,1,1,2)	
10	(1,1,1,1,1,1)	

5. Optimization in the FKO algorithm for concentration/sorting

The next algorithm is a more sophisticated version of Algorithm 4.4.

Algorithm 5.1 (Subroutine for selecting an element from each of two given sets). Let A and B be independent and partially sorted sets. Let C and D be subsets of A and B, respectively. Remove an i-element of C and a j-element of D according to the following criteria for selecting the pair (i,j) :

Main criterion: To minimize $|i-j|$.
Secondary criterion: To maximize $\min\{i+j+1,n\}-i$.
Tiebreaker: There may be multiple optimal choices for the pair (i,j) by the main and secondary criteria. Among them, if there is at least one with $i<\lceil n/2\rceil$ and $j<\lceil n/2\rceil$, then choose the one among such pairs that minimizes $2\times\lceil n/2\rceil-i-j$, else, choose the one with the smallest $\min\{i,j\}$.

Example. Let $n=7$. Let the state vectors of A and B be (0,3,0,5,0,0,0) and (0,0,2,0,0,0,0), respectively. The smallest possible value of $|i-j|$ is 1, which is achieved by $(i,j)=(1,2)$ or (3, 2). For both of these pairs, the corresponding value of the $\min\{i+j+1,n\}-i$ is 3. Since both pairs satisfy the inequalities $i<\lceil n/2\rceil$ and $j<\lceil n/2\rceil$, the tiebreaker calculates their corresponding values of $2\times\lceil n/2\rceil-i-j$ to be 5 and 3, respectively, and selects the pair $(i,j)=(3,2)$ accordingly.

The next algorithm fine-tunes the FKO algorithm for concentration/sorting by calling Algorithms 4.4 and 5.1 as subroutines.

Algorithm 5.2 (FKO for m-to-n concentration/sorting). Initially a set with the state vector $m\times(1,0,0,\ldots,0)$ is given. At every stage, perform the algorithmic action as prescribed in Table 2. If the state vector is of the type I, II, III, or V with Algorithm 4.3 applied to the case of type V. If the state vector is of type IV, say, $(a_0,a_1,\ldots,a_{n-1})+(b_0,b_1,\ldots,b_{n-1})$, perform the following four-step procedure.

Step 1 Let A and B be two independent and partially sorted sets with the state vectors $(a_0,a_1,\ldots,a_{n-1})$ and $(b_0,b_1,\ldots,b_{n-1})$, respectively. Let A' and B' denote their respective subsets after the deletion of all $(n-1)$-elements. Without loss of generality we shall assume that $a_0+a_1+\ldots+a_{n-2}\geq b_0+b_1+\ldots+b_{n-2}$.
Step 2 Starting with $C=A'$ and $D=B'$, apply Algorithm 5.1 recursively until D is depleted. Denote by R the remaining subset of A. Denote by r_j the number j-elements of A that are contained in R. In particular, $r_{n-1}=a_{n-1}$.

Step 3 Denote the vector $\rho=(r_0,r_1,\ldots,r_{n-1})$. A vector $\rho'=(r_0+c_0,r_1+c_1,\ldots,r_{n-1}+c_{n-1})$ is said to be in the neighborhood of ρ if $c_0,c_1,\ldots,c_{n-1}$ are integers subject to the following restrictions.

- $c_j=0,1$, or -1 for every j, and, if $r_j=0$, then $c_j=0$.
- $c_0+c_1\cdots+c_j=0,1$ or -1 for every j, and $c_0+c_1\ldots+c_{n-1}=0$.
- $0\leq r_j+c_j\leq a_j$ for every j.In particular, the vector ρ itself is also in this neighborhood.

Step 4 For every vector ρ' in the neighborhood of ρ, perform the following sub-steps.

Let R' denote a subset of A with the state vector ρ'. If R' is not an empty set and b_{n-1} is an odd number, choose an i-element of R' with the maximum possible i and pair this element with an $(n-1)$-element of B.
Starting with $C=A'\backslash R'$ and $D=B'$, apply Algorithm 4.4 recursively until D is depleted. This completely pairs off all elements in $A'\backslash R'$ and B'.
Remove all paired elements do far from A and B. Apply CKO to the residual sets of A and B separately. This completes the prescription of a stage of comparisons. This stage results in a set with a state vector of the type I.
Apply CKO stages to this set until the state vector becomes (1,1,…,1). Note that the number of CKO stages in this process may depend upon the choice of the vector ρ'.

Among all possible choices of the neighboring vector ρ', choose one that incurs the fewest CKO stages in the final sub-step. Ignore the results of all actions that have been performed for all other choices of ρ'.

The finite-state diagram in Fig. 4 still applies except that the transition from ‘Type IV’ through ‘Type I’ to the terminal state may require multiple runs before the optimal one is identified.

Example. Table 5 gives the evolution of state vector through four initial stages in the 21-to-7 concentration by Algorithm 5.2. The initial four stages lead to a set $A\cup B$, where A and B independently have state vectors (1,4,0,6,0,0,0) and (1,1,2,1,0,0,0), respectively. Since the state vector of $A\cup B$ is of the type IV, the 4-step procedure in Algorithm 5.2 applies. Step 2 in the procedure recursively chooses pairs of $(i,j)=(0,0)$, (1,1), (3,3), (3,2), and (3,2). We then have $\rho=(0,3,0,3,0,0,0)$ with a neighborhood that also includes vectors (0,4,0,2,0,0,0) and (0,2,0,4,0,0,0). Step 4 in the algorithm compares the three vectors in the neighborhood (see the contrast in Table 6.) Thus the sub-steps prescribed for $\rho'=(0,4,0,2,0,0,0)$ are adopted.

Table 5
The state vector through four initial stages

Stage	State vector
0	$21 \times (1,0,0,0,0,0,0)$
1	$10 \times (1,1,0,0,0,0,0) + (1,0,0,0,0,0,0)$
2	$5 \times (1,2,0,1,0,0,0) + (1,0,0,0,0,0,0)$
3	$2 \times (1,3,0,3,0,0,0) + (1,2,1,1,0,0,0)$
4	$(1,4,0,6,0,0,0) + (1,1,2,1,0,0,0)$

6. Algorithm for pure concentration and performance comparison

An m-to-n concentrator built by any algorithm in the knockout style is actually a concentrator/sorter. Conceivably, if the construction can be adapted into a *pure* concentrator, i.e. without necessarily sorting the outputs, then better efficiency for just the concentration per se might be attained. Fig. 6 depicts a mechanism for the conversion of two $m/2$-to-n concentrator/sorters into an m-to-n pure concentrator. This mechanism is based upon the same principle of "k-sorting" [11], which generalizes the algorithm in [10].

- The concatenation of an increasing sequence with a decreasing sequence forms a *unimodal circle*. When two antipodal terms on the unimodal circle are compared, the winner (*resp*. loser) belongs to the larger (*resp*. smaller) half of the circle.

In fact, each half of the output sequence of the network in Fig. 6 forms a unimodal sequence by itself.

Algorithm 6.1 (FKO algorithm for m-to-n concentration). Assume that $m \geq 2n$. Apply Algorithm 5.2 to both $\lceil m/2 \rceil$-to-n and $\lfloor m/2 \rfloor$-to-n concentration/sorting in the construction of Fig. 6.

The remainder of this section compares the performance of Algorithm 6.1 against previously known algorithms. We shall consider the m-to-n concentration only for $m \geq 2n$, since it is equivalent to m-to-$(m-n)$ concentration. Moreover, m-to-n concentration is a special case of both $(m+1)$-to-n concentration and $(m+1)$-to-$(n+1)$ concentration, hence its complexity is bound by both these two regardless of the algorithm. This in particular allows the rounding up of the parameter m to the next power of 2.

Table 6
Comparison among vectors in the neighborhood of the vector ρ

Vector ρ'	(0,3,0,3,0,0,0)	(0,4,0,2,0,0,0)	(0,2,0,4,0,0,0)
State vector after stage 5	(1,4,3,4,1,0,2)	(1,4,4,2,1,1,2)	(1,4,2,4,3,0,1)
Number of subsequent CKO stages	11	10	11

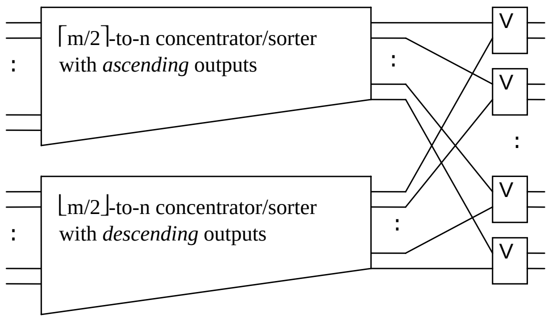
⌈m/2⌉-to-n concentrator/sorter
with *ascending* outputs
⌊m/2⌋-to-n concentrator/sorter
with *descending* outputs
V
V
V
V

Fig. 6. Adaptation of concentrator/sorters into a pure concentrator.

Table 7 tabulates the depth of m-to-n concentration by four algorithms for $2 \leq n \leq 16$ and $2n \leq m \leq 512$. The first number in each table entry is the depth by Algorithm 6.1. The second number is by the previously known FKO algorithm [9], which is equivalent to adapting Algorithm 2.4 by the mechanism in Fig. 6. The third is by sorters [6] of half sequences in Fig. 6.

Finally, the fourth number is by the k-sortout with the following detail. First, round up m and n to the nearest powers of 2, say, M and N. The k-sortout algorithm [9] with the parameter $k = N$ constructing two independent $M/2$-to-N concentrator/sorters before a final stage as in Fig. 6. Through truncation, the $M/2$-to-N concentrator/sorters may be regarded as $M/2$-to-n concentrator/sorters. Hence an M-to-n concentrator can be constructed at the same depth as an M-to-N concentrator.

We do not include CKO among the compared algorithm so that the table size does not become too large. Besides, Algorithm 6.1 by its nature can only be better than CKO.

Bold numbers in the table indicate best known depths of concentrators from multi-stage cascades of 2×2 sorters. Shaded entries indicate that Algorithm 6.1 uniquely constructs the best known concentrators. Algorithm 6.1 and the k-sortout algorithm together generate a best known m-to-n concentrator for every pair (m,n) within the practical range.

7. Epilogue

One type of self-route concentrators in an ATM switching fabric is by multi-stage cascade of parallel 2×2 sorters, where outputs of one stage lead to inputs of a later stage. The 2×2 sorter includes the 2×2 routing cells as a special case. We have generalized the FKO technique, originally for 2^k-to-n concentration, to m-to-n concentration in a way that strives for the efficiency. The measure of complexity is by the number of stages in the cascade. The generalized algorithm, together with the existing technique of *k-sortout*, generates a best-known m-to-n concentrator for every pair (m,n) within the practical range.

All concentrators considered in this paper perform

Table 7
Complexity of four concentration algorithms

n / m	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
4	**2,2**														
	3,**2**														
5	**4,4**														
	4,4														
6-7	**4,4**	4,4													
	4,4	4,4													
8	**4,4**	4,4	**4,4**												
	4,4	4,4	**4,4**												
9	**5**,6	6,7	7,8												
	6,6	6,7	**6**,7												
10	**5**,6	6,7	7,8	7,9											
	6,6	6,7	**6**,7	**6**,7											
11	**6,6**	7,7	8,8	8,9											
	6,6	6,7	**6**,7	**6**,7											
12	**6,6**	7,7	8,8	8,9	8.10										
	6,6	6,7	**6**,7	**6**,7	6,7										
13	**6,6**	**7,7**	8,8	9,9	10,10										
	7,**6**	**7,7**	**7,7**	**7,7**	**7,7**										
14-15	**6,6**	**7,7**	8,8	9,9	10,10	10.10									
	7,**6**	**7,7**	**7,7**	**7,7**	**7,7**	**7.7**									
16	**6,6**	**7,7**	8,8	9,9	10,10	10.10	10,10								
	7,**6**	**7,7**	**7,7**	**7,7**	**7,7**	**7.7**	**7,7**								
17	**7,7**	**8,8**	9,10	10,12	11,13	12.14	13,15								
	8,8	**8**.10	**8**,10	**8**,11	**8**.11	**8**,11	**8**,11								
18	**7,7**	**8,8**	9,10	10,12	11,13	12.14	13,15	13,16							
	8,8	**8**.10	**8**,10	**8**,11	**8**.11	**8**,11	**8**,11	**8**,11							
19	**7,7**	**8,8**	9,10	10,12	11,13	12.14	13,15	14,16							
	8,8	**8**.10	**8**,10	**8**,11	**8**.11	**8**,11	**8**,11	**8**,11							
20	**7,7**	**8,8**	9,10	10,12	11,13	12.14	13,15	14,16	14.17						
	8,8	**8**.10	**8**,10	**8**,11	**8**.11	**8**,11	**8**,11	**8**,11	**8**,11						
21	**7,7**	**8,8**	**9**,10	10,12	11,13	12.14	13,15	14,16	15.17						
	9,8	9.10	**9**,10	**9**,11	9.11	**9**,11	**9**,11	**9**,11	**9**,11						
22	**7,7**	**8,8**	**9**,10	10,12	11,13	12.14	13,15	14,16	15.17	15,18					
	9,8	9.10	**9**,10	**9**,11	9.11	**9**,11	**9**,11	**9**,11	**9**,11	**9**,11					
23	**7,7**	**8,8**	**9**,10	10,12	11,13	12.14	13,15	14,16	15.17	16,18					
	9,8	9.10	**9**,10	**9**,11	9.11	**9**,11	**9**,11	**9**,11	**9**,11	**9**,11					
24	**7,7**	**8,8**	**9**,10	10,12	11,13	12.14	13,15	14,16	15,17	16,18	16,19				
	9,8	9.10	**9**,10	**9**,11	9.11	**9**,11	**9**,11	**9**,11	**9**,11	**9**,11	**9**,11				
25	**7,7**	**8,8**	**9**,10	11,12	12,13	13.14	14,15	16,16	16.17	17,18	18,19				
	10,8	10,10	10,10	**10**,11	**10**,11	**10**.11	**10**,11	**10**,11	**10**.11	**10**,11	**10**,11				
26	**7,7**	**8,8**	**9**,10	11,12	12,13	13.14	14,15	16,16	16.17	17,18	18,19	18,20			
	10,8	10,10	10,10	**10**,11	**10**,11	**10**.11	**10**,11	**10**,11	**10**.11	**10**,11	**10**,11	**10**,11			
27	**7,7**	**8,8**	**10,10**	12,12	13,13	14.14	15,15	16,16	17.17	18,18	19,19	20,20			
	10.8	10,10	**10,10**	**10**.11	**10**,11	**10** 11	**10**,11	**10**,11	**10**.11	**10**,11	**10**,11	**10**,11			
28	**7,7**	**8,8**	**10,10**	12,12	13,13	14.14	15,15	16,16	17.17	18,18	19,19	20,20	20.21		
	10,8	10,10	**10,10**	**10**,11	**10**,11	**10** 11	**10**,11	**10**,11	**10**.11	**10**,11	**10**,11	**10**,11	**10**.11		
29	**7,7**	**8,8**	**10,10**	12,12	13,13	14.14	15,15	16,16	17.17	18,18	19,19	20,20	21.21		
	10,8	10,10	**10,10**	**10**,11	**10**,11	**10**.11	**10**,11	**10**,11	**10**.11	**10**,11	**10**,11	**10**,11	**10**.11		
30-31	**7,7**	**8,8**	**10,10**	12,12	13,13	14.14	15,15	16,16	17.17	18,18	19,19	20,20	21.21	21,21	
	10,8	10,10	**10,10**	**10**,11	**10**,11	**10**.11	**10**,11	**10**,11	**10**.11	**10**,11	**10**,11	**10**,11	**10**.11	**10**,11	
32	**7,7**	**8,8**	**10,10**	12,12	13,13	14.14	15,15	16,16	17,17	18,18	19,19	20,20	21.21	21,21	21,21
	10.8	10,10	**10,10**	**10**,11	**10**,11	**10**.11	**10**,11	**10**,11	**10**.11	**10**,11	**10**,11	**10**,11	**10**.11	**10**,11	**10**,11
33-34	**8**,9	**9**.10	**11**,12	**13**,14	**14**,16	**15**.18	16,20	17,21	18.23	19,25	20,26	21,27	22.28	23,29	24,30
	16,10	16,13	16,13	16,15	16,15	16 **15**	16,**15**	**16,16**	**16.16**	**16,16**	**16,16**	**16,16**	**16.16**	**16,16**	**16,16**
35-36	**8**,9	**10,10**	**11**,12	**13**,14	**14**,16	**15**.18	16,20	17,21	18.23	19,25	20,26	21,27	22.28	23,29	24,30
	16,10	16,13	16,13	16,15	16,15	16.**15**	16,**15**	**16,16**	**16.16**	**16,16**	**16,16**	**16,16**	**16.16**	**16,16**	**16,16**
37-38	**8**,9	**10,10**	**11**,12	**13**,14	**15**,16	**15**.18	16,20	17,21	18.23	19,25	20,26	21,27	22.28	23,29	24,30
	16,10	16,13	16,13	16,15	16,**15**	16.**15**	16,**15**	**16,16**	**16.16**	**16,16**	**16,16**	**16,16**	**16.16**	**16,16**	**16,16**
39-42	**8**,9	**10,10**	**11**,12	**13**,14	**15**,16	16 18	17,20	18,21	19.23	20,25	21,26	22,27	23.28	24,29	25,30
	16,10	16,13	16,13	16,15	16,**15**	16 **15**	16,**15**	**16,16**	**16.16**	**16,16**	**16,16**	**16,16**	**16.16**	**16,16**	**16,16**
43-48	**8**,9	**10,10**	**12,12**	**14,14**	16,16	17.18	18,20	19,21	20.23	21,25	22,26	23,27	24.28	25,29	26,30
	16,10	16,13	16,13	16,15	16,**15**	16.**15**	16,**15**	**16,16**	**16.16**	**16,16**	**16,16**	**16,16**	**16.16**	**16,16**	**16,16**
49-50	**9,9**	**10,10**	**12,12**	**14,14**	16,16	18.18	19,20	20,21	21.23	22,25	23,26	24,27	25.28	26,29	27,30
	16,10	16,13	16,13	16,15	16,**15**	16.**15**	16,**15**	**16,16**	**16.16**	**16,16**	**16,16**	**16,16**	**16.16**	**16,16**	**16,16**

Table 7 *Continued.*

51-52	**9,9**	**10,10**	**12,12**	**14,14**	16,16	18.18	19,20	20,21	22.23	23,25	24,26	25,27	26.28	27,29	28,30
	16,10	16,13	16,13	16,15	16,**15**	16.**15**	16,**15**	**16,16**	**16.16**	**16,16**	**16,16**	**16,16**	**16.16**	**16,16**	**16,16**

Table 7 (continued)

n m	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
53-54	**9,9**	**10,10**	**12,12**	**14,14**	16,16	18.18	20,20	21,21	22.23	23,25	24,26	25,27	26.28	27,29	28,30
	16,10	16,13	16,13	16,15	16,**15**	16.**15**	16,**15**	**16,16**	**16.16**	**16,16**	**16,16**	**16,16**	**16.16**	**16,16**	**16,16**
55-56	**9,9**	**10,10**	**12,12**	**14,14**	16,16	18.18	20,20	21,21	22.23	24,25	25,26	26,27	27.28	28,29	29,30
	16,10	16,13	16,13	16,15	16,**15**	16.**15**	16,**15**	**16,16**	**16.16**	**16,16**	**16,16**	**16,16**	**16.16**	**16,16**	**16,16**
57-58	**9,9**	**10,10**	**12,12**	**14,14**	16,16	18.18	20,20	21,21	22.23	24,25	26,26	27,27	28.28	29,29	30,30
	16,10	16,13	16,13	16,15	16,**15**	16.**15**	16,**15**	**16,16**	**16.16**	**16,16**	**16,16**	**16,16**	**16.16**	**16,16**	**16,16**
59-64	**9,9**	**10,10**	**12,12**	**14,14**	16.16	18.18	20,20	21,21	23.23	25,25	26,26	27,27	28.28	29,29	30,30
	16,10	16,13	16,13	16,15	16,**15**	16.**15**	16,**15**	**16,16**	**16.16**	**16,16**	**16,16**	**16,16**	**16.16**	**16,16**	**16,16**
65-66	**9,10**	**11,12**	**13,14**	**15**.16	**17,18**	**19**.20	21,22	22,24	24.26	26,28	27,30	28,32	29.34	30,36	31,38
	22,12	22,16	22,16	22,19	22,19	22.**19**	**22,19**	22,**21**	22.**21**	22,**21**	**22,21**	22,**21**	22.**21**	22,**21**	**22,21**
67-70	**9,10**	**11,12**	**13,14**	**15**.16	**17,18**	**19**.20	21,22	23,24	25.26	27,28	28,30	29,32	30.34	31,36	32,38
	22,12	22,16	22,16	22,19	22,19	22.**19**	**22,19**	22,**21**	22.**21**	22,**21**	**22,21**	22,**21**	22.**21**	22,**21**	**22,21**
71-74	**9,10**	**11,12**	**13,14**	**15**.16	**17,18**	**19**.20	21,22	23,24	25.26	27,28	29,30	31,32	32.34	33,36	34,38
	22,12	22,16	22,16	22,19	22,19	22.**19**	**22,19**	22,**21**	22.**21**	22,**21**	**22,21**	22,**21**	22.**21**	22,**21**	**22,21**
75-80	**9,10**	**11,12**	**14,14**	**16,16**	**18,18**	**19**.20	21,22	23,24	25.26	27,28	29,30	31,32	33.34	35,36	36,38
	22,12	22,16	22,16	22,19	22,19	22.**19**	**22,19**	22,**21**	22.**21**	22,**21**	**22,21**	22,**21**	22.**21**	22,**21**	**22,21**
81-82	**10,10**	**12,12**	**14,14**	**16,16**	**18,18**	20.20	22,22	24,24	26.26	28,28	30,30	32,32	34.34	36,36	37,38
	22,12	22,16	22,16	22,19	22,19	22.**19**	**22,19**	22,**21**	22.**21**	22,**21**	**22,21**	22,**21**	22.**21**	22,**21**	**22,21**
83-	**10,10**	**12,12**	**14,14**	**16,16**	**18,18**	20.20	22,22	24,24	26.26	28,28	30,30	32,32	34.34	36,36	38,38
128	22,12	22,16	22,16	22,19	22,19	22.**19**	**22,19**	22,**21**	22.**21**	22,**21**	**22,21**	22,**21**	22.**21**	22,**21**	**22,21**
129-	**10,11**	**12,13**	**14**.15	**16**.17	**18,19**	**20**.21	**22,23**	**24,25**	**26**.27	28,29	30,31	32,33	34.35	36,37	38,39
130	29,14	29,19	29,19	29,23	29,23	29.23	29,23	29,26	29.**26**	29,**26**	29,**26**	29,**26**	29.**26**	29,**26**	29,**26**
131-	**11,11**	**13,13**	**15,15**	**17,17**	**19,19**	**21.21**	**23,23**	**25,25**	27.27	29,29	31,31	33,33	35.35	37,37	39,39
256	29,14	29,19	29,19	29,23	29,23	29.23	29,**23**	29,26	29.**26**	29,**26**	29,**26**	29,**26**	29.**26**	29,**26**	29,**26**
257-	**12,12**	**14,14**	**16,16**	**18,18**	**20,20**	**22 22**	**24,24**	**26,26**	**28.28**	**30,30**	32,32	34,34	36.36	38,38	40,40
512	37,16	37,22	37,22	37,27	37,27	37.27	37,27	37,31	37.31	37,31	37,**31**	37,**31**	37.**31**	37,**31**	37,**31**

deterministic comparisons. However, concentration itself is often statistical in nature. Loss is incurred in concentration when too many inputs are active. Hence, concentration parameters are engineered so that the loss is contained within tolerance. One direction of further research is the application of FKO to statistical concentration. Another direction would be to further improve the FKO technique, either for pure concentration or for the combined concentration/sorting. Conceivably, there could be a way to do *m*-to-*n* FKO with a large variety of *types* of the state vector instead of just the five types used in Algorithm 5.2. Meanwhile, it would be interesting to see new ways for adapting concentrator/sorters into concentrators besides the mechanism illustrated in Fig. 6.

Acknowledgements

Research by Shuo-Yen Robert Li was supported in part by RGC Earmarked Grant 480/95E.

References

[1] D.X. Chen, J.M. Mark, SCOQ: a fast packet switch with shared concentration and output queueing, IEEE/ACM Trans. Networking 1 (1) (1993) 142–151.

[2] A. Huang, S. Knauer, Starlite: a wideband digital switch, Proceedings of GLOBECOM, 1984, pp. 121–125.

[3] J.N. Giacopelli, J.J. Hickey, M.S. Marcus, W.D. Sincoskie, M. Littlewood, Sunshine: a high-performance self-routing broadband packet switch architecture, IEEE J. Selected Areas Commun. 9 (8) (1991) 1289–1298.

[4] W. Wang, F.A. Tobagi, The CHRISTMAS-tree switch: an output queueing space-division fast packet switch base on interleaving distribution and concentration function, Proc. IEEE INFOCOM 1 (1991) 163–170.

[5] Y.S. Yeh, M.G. Hluchyj, A.S. Acampora, The knockout switch: a simple, modular architecture for high performance packet switching, IEEE J. Selected Areas Commun. 5 (8) (1987) 1274–1283.

[6] D.E. Knuth, The Art of Computer Programming, 3, Addison-Wesley, Reading, MA, 1973.

[7] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, MIT Press, Cambridge, MA, 1994.

[8] N. Pippenger, Self-routing superconcentrators, J. Comput. Syst. Sci. 52 (1996) 53–60.

[9] S.-Y.R. Li, C.M. Lau, Concentrators in ATM switching, Comput. Syst. Sci. Engng 11 (6) (1996) 335–342.

[10] K.E. Batcher, Sorting networks and their applications, Proceedings of AFIP Spring Joint Computer Conference 32 (1968) 307–314.

[11] S.-Y.R. Li, Partial sorting and concentration by parallel networks, Proceedings of the International Workshop on Discrete Mathematics and Algorithms, 1994, pp. 27–43.

Shuo-Yen Robert Li received the BS degree from National Taiwan University in 1970 and the PhD degree from University of California, Berkeley in 1974. He then taught applied mathematics at MIT for two years. From 1976–79, he taught mathematics, statistics and computer science in the University of Illinois at Chicago and eventually resigned as a tenured Associate Professor. From 1979 to 1989, he was a technical member at Bell Labs and Bellcore working on switching systems and theoretic research. Since 1989, he has been Professor of Information Engineering at the Chinese University of Hong Kong. Publications of Prof. Li has appeared in journals in the areas of algorithms/complexity, computers, communications, commutative algebra, control, discrete mathematics, game theory, information theory, statistics, and VLSI. He holds five and half US patents.

Hui Li was born in GuangDong, P.R. China, in 1964. He received the Bachelor and Master of Engineering degrees both from TsingHua University (Beijing) in 1986 and 1989 respectively.

From 1989 to 1993 he worked at the China National Computer Software and Technology Corporation. Then, He was a Lecturer with the ShenZhen University. Since 1997, he is studying for his PhD degree with the Department of Information Engineering at the Chinese University of Hong Kong. His current interesting is broadband switching theory and system.

Gar-Man Simon Koo was born in Hong Kong, China. He received the BEng (Hon) degree in Information Engineering from the Chinese University of Hong Kong in 1997 and the MSEE degree from the Polytechnic University, New York, in 1998. He was with the Video Networking Laboratory, Polytechnic University, where he participated in projects sponsored by MCI on multicasting protocol development and network analysis.

He is currently working towards the PhD degree in Information Engineering at the Chinese University of Hong Kong, affiliated to the Telephone Laboratory. His research interests have been in switching theory, protocol development, communication networks analysis and optimization.