Fast Fair Arbiter Design in Packet Switches

Feng Wang and Mounir Hamdi
Computer Science Department
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
{fwang, hamdi}@cs.ust.hk

Abstract*[1] -- *All arbiters proposed in the literature suffer from one of the following problems: large time complexity and/or unfairness. The first generation arbiters for switches take into consideration the issue of fairness by using a rotating round robin priority list, but their arbitration time is proportional to the number of inputs which makes them unscalable for a given fixed amount of arbitration time. To reduce the time complexity, Chao [1] proposed a tree arbiter structure which can perform the arbitration in a fast and efficient way, but this framework can not guarantee fairness to all the inputs. When it is fed by adversary traffic, some of the traffic may not get its fair share of the bandwidth. Motivated by solving these two problems, we propose a new algorithm which guarantees fairness and has O(logN) time. In addition, we explore the possibility that our solution of arbiter design can be embedded into the switch crossbar, thus reducing the cost as well as power consumption.

I. INTRODUCTION

High-performance packet switches are becoming more and more important with the continuous growth of Internet traffic. The ideal packet switch is an output-queued switch which has optimal delay-throughput performance under all traffic conditions. However, a direct implementation of OQ switches needs to run N times faster than the line rate for an NxN switch. In practice, we can only afford an architecture called CIOQ (Combined Input/Output Queued) switch. VOQ is the well-known technology used in CIOQ switches rather than FIFO queue to solve the notorious HoL problem. [2] Most switches/routers commercially available nowadays use VOQ technology as their memory strategy.

In VOQ scheduling, bipartite graph matching is the most essential method to perform arbitration [2]. However, finding the maximum matching of a bipartite graph is very costly, especially when we want to employ some centralized algorithms whose time complexity is no less than $O(N^{2.5})$ as far as we know. We can only afford some heuristic algorithms to approximate the maximal matching, such as, iSLIP [4], FIRM [6], and DRRM [7]. In fact, all the heuristic algorithms are designed in a distributed fashion. The reason why we favor distributed algorithms lies in its relatively easy hardware implementation. However, distribution algorithms over shared resources always lead to contention. In all the heuristic algorithms, such as, iSLIP, FIRM, and DRRM, input and output contention resolution plays a very crucial role in leveraging the throughput of routers. We need fast, fair arbiters residing in the input and output side to select one of the input requests efficiently. Arbiter design is always an important topic in the areas of router scheduler and network-on-chip systems [5]. In this paper, we only focus on the design of arbiters for the output contention.

[1] This research is supported under RGC HKUST6200/02E.

II. RELATED WORK

Consider an NxN packet switch. To resolve the output contention, a solution is to use an arbiter for each output to fairly select one among those incoming packets and send back a grant signal to the corresponding input. The arbitration procedure is as follows [1]:

1. During every arbitration cycle, each input submits a one-bit request signal to each output arbiter, indicating whether its packet, if any, is destined for the output.
2. Each output arbiter collects up to N request signals, among which one input which has an active request is granted according to some priority order.
3. A grant signal is sent back to acknowledge the input.

The second step is the most important in the performance of input-output matching. Two criteria must be taken into consideration here: speed and fairness. According to both industry and academia, two major proposals have come into being: Programmable Priority Encoder (PPE) for iSLIP [4] and Ping-Pong Arbitration (PPA) [1]. PPE uses round robin rotation to set its priority list to guarantee fairness. However, it is a centralized switch arbiter and the arbitration time is proportional to the number of inputs. As a result, the switch size or capacity is limited by a given fixed arbitration time. On the other hand, PPA has less time complexity, i.e. $O(logN)$ [1], but its priority list is not determined and unfair, which makes the throughput and performance unpredictable.

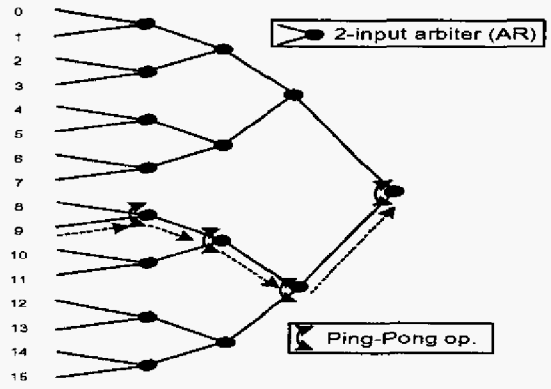

Figure 1: Ping-Pong Arbiter description

The basic idea behind PPA is to divide the *inputs* into groups. Each group has its own arbiter. Further grouping can be applied recursively to all the *group request* signals at the current layer, forming a tree structure, as illustrated in Figure 1. Thus, an arbiter with N inputs can be constructed using multiple small-size *2-input arbiters* (AR) at each layer. Using this strategy, the arbitration time is reduced to *O(logN)*.

An AR contains an internally feedback state that indicates which input of the two is favored. Once an input is granted in one arbitration cycle, the other input will be favored in the next cycle. In other words, the granted request is always chosen between the upper input and the lower input alternatively. The basic intuition behind this ping pong fashion is to make sure that *once an input is favored in this cycle, it will get the lowest priority in the next cycle.*

The most distinctive feature of this tree-structured arbiter is that it distributes the computing of priorities over all the nodes (AR). However, the priority list produced by this mechanism is *not* predictable, although every configuration of the ARs *can* produce some permutation of N numbers, e.g. 1→3→2→4→1→..., when N = 4 for instance. The only strong point of this structure is that it pushes the current granted input into the rear of the priority list in the next round of arbitration, which is one property of the round robin arbiter. Moreover, the ping pong fashion can lead to serious unfairness under some adversary traffic. We use figure 2 to illustrate this situation.

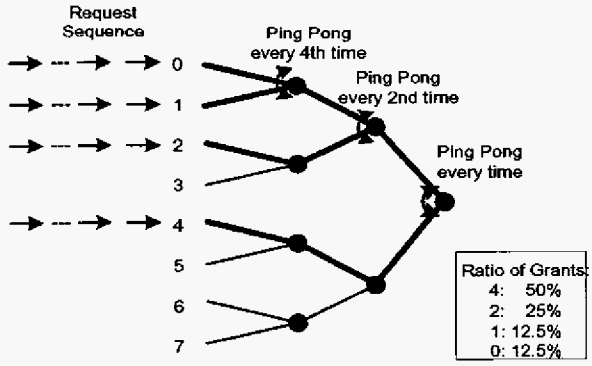

Figure 2: Unfairness of PPA

We can see from figure 2 that input 4 gains half of the whole bandwidth. The unfairness is produced by the careless toggling of the states of the ARs regardless of the traffic weights when doing arbitration. Ping-pong arbiter can not maintain an exact rotating round robin priority list.

We are motivated by designing an exact rotating round robin arbiter which is regarded by PPE to be fair while maintaining the low time complexity property of PPA at the same time. And we will show that we can implant the arbiters into a switch crossbar.

The rest of the paper is organized as following. We describe our Fast Fair Arbiter (FFA) in section III. After that, we analyze the most distinguished features of FFA and discuss the possibility of implanting the arbiter into the crossbar chip. Then we have a conclusion in section V.

III. OUR FAST FAIR ARBITER

We propose a new arbiter which maintains the low time complexity of PPA and can be fair at the same time. In fact, our arbiter can implement any round robin priority list which can be evenly fair or weighted.

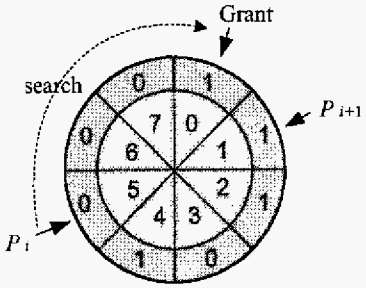

Figure 3: Round Robin Disc

Fairness: First, we need to make clear what the fairness means. Most fair crossbar schedulers are based on round robin arbitration. (See figure 3) The round robin has a rotating priority pointer denoted by P. When doing arbitration, P just rotates clockwise until it hits a '1', then grants for that input. The initial position of P for the next cycle is usually determined by various algorithms, such as iSLIP, RRM, and FIRM.

The round robin arbiter guarantees that none of the input ports are starved, and that all are treated fairly. For example, for the adversary traffic that we illustrated in Figure 2, we can easily calculate that each flow gets just 25% bandwidth.

If we want to get a weighted priority distribution among all the input ports, we can use weighted round robin which can be defined in a straightforward way.

Round robin guarantees fairness. However, a direct implementation of the rotating round robin scheme is very time costly; it is *O(N)*, where N is the number of inputs.

Like [1], we also use a tree structure. Our contribution is that we can produce the exact round robin priority list with a little modification in the small *2-input arbiters* (AR). Let us first describe AR in [1] in more detail. In each node of the tree, we need to maintain a one-bit status, 0 or 1, indicating which one of the input requests has higher priority. We use 0 to denote that the upper input has higher priority and 1 to denote that the lower input has higher priority. The structure of a *2-input arbiter* (AR) is shown in figure 4.

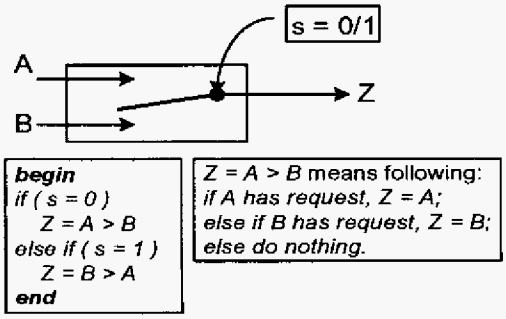

Figure 4: Function of 2-input Arbiter (AR)

Lemma 1: If all the arbiter states are set as '0', it will produce the priority list as {0, 1, 2..., N-1}, as we can see from figure 5.

This is easy to prove. Input 0 will always defeat inputs 1, 2, ... N-1; input 1 will always defeat inputs 2, 3, ... N-1; input 2 will always defeat inputs 3, 4, ... N-1, and so on. Thus, the smaller the input number, the higher the priority. ■

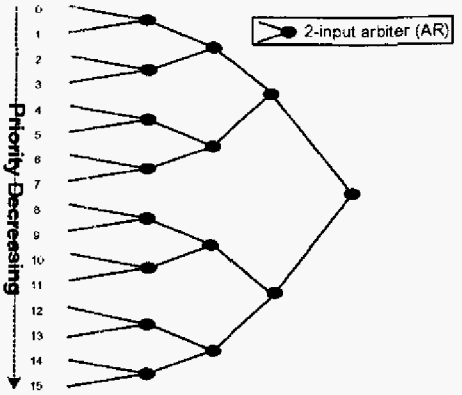

Figure 5: Priority list with all states being '0'

We are now required to produce any rotating priority list, for example, {6, 7, ... N-1, 0, 1, ... 4, 5}, as shown in figure 6 where input 6 has the highest priority, input 7 the second, and so on, and input 5 has the lowest priority.

Lemma 2: In an N-input ($N=2^k$) tree arbiter, for an input I, there's one and only one path which contains k ($=log_2N$) ARs from the *root* to I. If we set the states of the k ARs from *root* to input I as the sequence of the *binary* representation of number I, then input I has the highest priority, no matter how the other arbiter states are set.

For example, if we set the four ARs along the path from root to input 6 of figure 6 as '0110' which is the binary representation of number 6, then input 6 gets the highest priority among all the 16 inputs, no matter how other ARs are set.

Proof: First, notice this is a full binary search tree which can use the bits of the edges (0 or 1) along the paths from root to input to represent the index of that input. Trace from *root* to input I. (You may imagine I equals to 6.) If we go up, mark that edge '0', if down, mark that edge '1'. We know that in this way, the sequence of the edge bits along the path from *root* to input I is exactly the binary representation of number I. (see the bits in the little rectangles of figure 6) We can easily find that setting these bits for the edge's right node's (AR) state will let these ARs select the corresponding edges with higher priority, according to how the AR works. Thus, we can see that setting the states of the k ARs from *root* to input I as a sequence of the binary representation of number I will make input I get the highest priority along the path from *root* to input I.

Second, we can see that, no matter how the states of other ARs are set, all other inputs except I will fail to input I in somewhere the ARs along the path from *root* to input I. Thus, input I gets the highest priority. ■

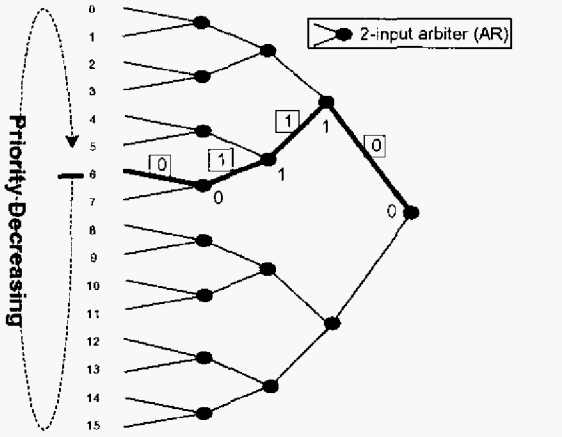

Figure 6: An example of rotating priority list

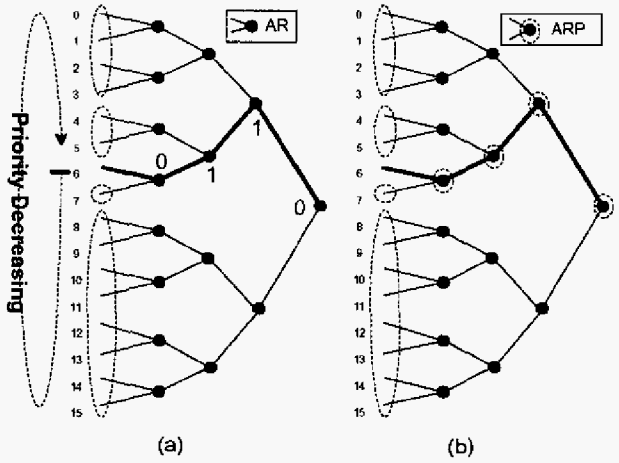

Figure 7: Grouped priority lists

Now, we have seen that input 6 gets the highest priority by setting the arbiter states in the path from root to input 6 as '0110'. But it still cannot produce the perfect rotating priority as {6, 7, ... N-1, 0, 1, ... 4, 5}. In fact, if we set all other arbiter states to '0', the priority list produced by figure 6 is {{6}, {7}, {4, 5}, {0, 1, 2, 3}, {8, 9, ... 14, 15}}. As shown in figure 7 (a), the inputs besides 6 form four groups and each group is prioritized as round robin. If we can prioritize the groups correctly, we can generate the exact priority list as {6, 7, ... N-1, 0, 1, ... 4, 5}.

From figure 7 (b) we find that the four groups are linked by exactly the path from the root to input 6, which indicates that we can prioritize the four groups at exactly the four arbiters of the root-to-6 path. We call the ARs along the path ARP.

ARP will do little more work than the normal ARs. ARP's structure is shown in figure 8, and the detailed algorithm of how the ARP works is shown in figure 9. We call this tree structured arbiter made of ARs and ARPs along a path as our Fast Fair Arbiter (FFA).

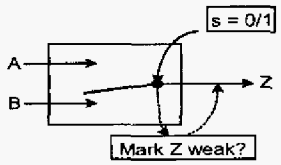

Figure 8: Structure of ARP

Algorithm ARP:
begin
if (s = 0)
Z = A > B > A.weak > B.weak
if (s = 1)
Z = B > A > B.weak > A.weak
if (s = 1 && Z = A)
Z = Z.weak // Z is marked weak
end

A.weak means A is marked weak.
Z = A > B is defined as in figure 4.
Z = A > B > C is defined recursively.

Figure 9: Behavior of ARP

The basic idea of how the FFA can guarantee rotating round robin priority list is as follows. A path from one input to the root can divide the tree into some sub-trees, forming some input groups. Setting all the AR states in a sub-tree can make that input group internally prioritized from top to bottom. Using ARPs along the path can further prioritize the input groups correctly.

Theorem: If we set the arbiters along the path from *root* to input x as ARPs with the states being the binary representation of x, and the other arbiters as ARs with states being '0', then the tree arbiter can produce an exact rotating round robin priority list as $\{x, x+1, \ldots N-1, 0, 1, \ldots x-1\}$.

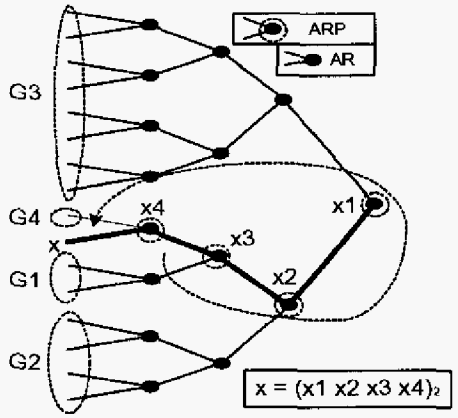

Figure 10: Fast Fair Arbiter, x is the starting point of the round robin priority list.

Proof: We use figure 10 to prove this theorem, where $N = 16$.

First, from lemma 2, we can prove that input x has the highest priority. We call the path from *root* to input x the critical *path* (the thick line in figure 10).

Second, the critical *path* will divide the tree into k ($=log_2N=4$ in our example) sub-trees. For example, we call these sub-trees as G1, G2, G3, and G4. Each sub-tree is made from ARs whose states are set as '0'. From lemma 1, we can prove that all the inputs of one sub-tree, e.g., G3, are prioritized internally from top to bottom.

Third, we prove that the ARPs along the critical *path* will prioritize the four groups (G1, G2, G3, and G4) correctly.

We can see from figure 10 that the critical *path* separates the groups into two parts, one above the path, and the other below it. For the groups above the *path* (G3 and G4), they are linked with the *path* at ARPs with states being '1'; for the groups below the *path* (G1 and G2), they are linked with the *path* at ARPs with states being '0'. According to our algorithm shown in figure 9, the above groups (G3 and G4) will be marked weak if they succeed in entering the ARPs. Because they are marked weak, they will always lose contention to the groups below the critical *path*. So, we can conclude that the below groups (e.g., G1 and G2) have higher priorities than the above ones (e.g., G3 and G4).

We now prove that the groups below the critical *path* have their proper priority in the round robin fashion. Consider any two groups G1 and G2, both below the *path*. We can see that, if G1 is closer from input x, then G1 links to the *path* at the ARP near from input x along the *path*. For example, in figure 10, $x3$ is closer than $x2$ from input x. So, any request from G1 will enter the *path* from ARP $x3$ and then defeat G2 at the ARP $x2$, thus making G1 prior to G2. This is exactly what we need from the round robin.

Similarly, we can prove that groups above the critical *path* will prioritize themselves exactly in a round robin fashion.

Now that we have proven that the below part of groups has higher priority than the above part, groups in either part prioritize themselves correctly, and inputs in every group prioritize themselves correctly, we can conclude that the tree arbiter can produce an exact rotating round robin priority list as $\{x, x+1, \ldots N-1, 0, 1, \ldots x-1\}$.

■

IV. Features of Fast Fair Arbiter

The two distinguished features of our Fast Fair Arbiter are: fairness and low time complexity. We achieve these goals by using rotating round robin arbiters and distribute the arbitration process into *O(logN)* levels in a tree architecture.

Note that the difference between AR and ARP is just for the clarity of discussion above. Actually, we need all the small 2-input arbiters to be ARP since every small 2-input arbiter can be in some critical *path*. Normally, if the small 2-input arbiter is not in the critical *path*, it just behaves as an AR with state being '0'.

We have proven the fairness of our FFA. Now we will analyze the time complexity. We do it in two phases: reset-

ting the states of all the ARPs and doing arbitration. According to the Theorem we have proven, we can see that there are two types of small 2-input arbiters to be set: ARPs in the critical path and those not in the critical path. Setting the states of ARPs not in the critical path is trivial since they are always '0' and can be reset in constant time before every cycle starts. For the states of those ARPs sitting on the critical *path*, the states sequence is just the binary representation of the input number x whose length is log_2N, so setting the states will cost $O(logN)$ time. We can improve the time to be *constant* by setting the ARPs in parallel using pre-determined memory (this is beyond the scope of this paper, and will be addressed in a sequel paper).

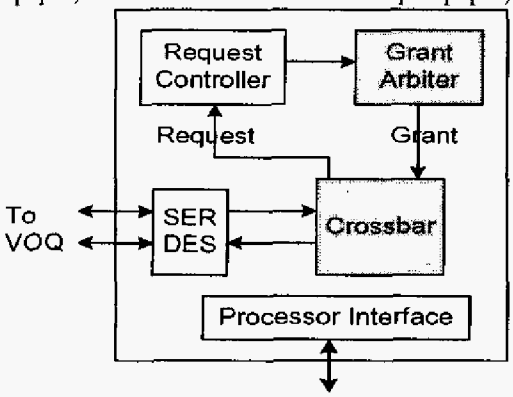

Figure 11: Traditional switch fabric with arbiter and crossbar

For the arbitration time, we can see that the winning request will go through log_2N small 2-input arbiters in total. So the arbitration time is $O(logN)$. We can use 4-input arbiters to replace all the small 2-input arbiters to improve the arbitration time a little. But the time complexity is still $O(logN)$.

Another feature of our Fast Fair Arbiter is that it can be embedded into the crossbar. The two main components of a switch fabric are scheduler and crossbar. As we can see from Figure 11, traditionally, they are separated chips. Communications between scheduler and crossbar always cause headaches in hardware implementation, especially when the number of inputs exceeds one hundred. In industry, many companies have claimed that they can manufacture integrated scheduler/crossbar switch fabrics. However, in most of their products, the scheduler and crossbar are just put together mechanically. Even they can be fabricated in one single chip, they are separately implemented. Communications between them are not avoidable.

One of our observations is that we can implement the crossbar using tree structures, just as we show in figure 12. For every output, it links to all the inputs in a tree fashion. It is natural to see that our FFA can be implanted in the ARPs in figure 12, thus making an arbiter to every output from all the inputs. By doing this, we remove the request controller module and grant arbiter module in figure 11. The complicated communications are not needed at all.

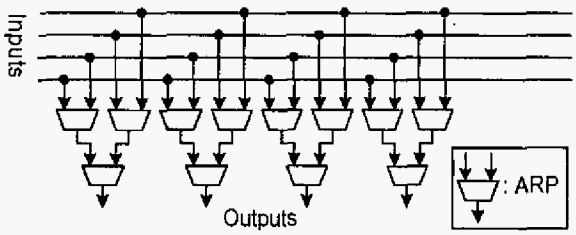

Figure 12: A 4x4 crossbar with implanted Arbiter

V. CONCLUSION

In this paper, we propose a Fast Fair Arbiter (FFA) design for output contention resolution. We first compare the two well-known arbiters: PPE and PPA. PPE is fair but slow. On the other hand, PPA can reduce the arbitration time significantly, but it cannot guarantee fairness.

We develop FFA from both PPE and PPA, taking the advantages of fairness from PPE and the low time complexity from PPA. To guarantee fairness, our FFA can provide rotating round robin priority list which is the basic requirement of most of scheduling algorithms, such as iSLIP, DRRM, FIRM, and so on. To be fast, our FFA can do the arbitration in $O(logN)$ time complexity by employing a binary tree structure. The basic idea is that we distribute the arbitration process into a layered architecture, thus decomposing the centralized arbitration process used by most arbiter designs.

We also propose that our FFA can be implanted into the switch crossbar if we design the crossbar using a tree-based architecture.

REFERENCES

[1] H. J. Chao, C. H. Lam and X. Guo, "Fast ping-pong arbitration for input-output queued packet switches", *International Journal of Communication systems*, 2001, pp. 663-678.

[2] J. Liu and M. Hamdi, "Stable and Practical Scheduling Algorithms for High Speed Virtual Output Queuing Switches", *The ACS/IEEE Conference International Conference on Computer Systems and Applications*, 2003.

[3] S. T. Chuang, A. Goel, N. McKeown and B. Prabhakar, "Matching output queuing with a combined input output queued switch", *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp 1030-1039, June 1999.

[4] N. Mckeown, P. Varaiya, and J. Warland, "The iSLIP Scheduling Algorithm for Input-Queued Switch", *IEEE Transaction on Networks*, 1999, pp. 133-167.

[5] Kangmin Lee Se-Joong Lee Hoi-Jun Yoo, "A distributed crossbar switch scheduler for on-chip networks", *Custom Integrated Circuits Conference*, 2003.

[6] D. N. Serpanos and P. I. Antoniadis, "FIRM: a class of distributed scheduling algorithms for high speed ATM switches with multiple input queues", in *Proc. IEEE INFOCOM*, 2000, pp. 548-555.

[7] H. J. Chao, J. S. Park, "Centralized contention resolution schemes for a large-capacity optical ATM switch", in *Proc. of the IEEE ATM workshop*, 1998.