High-Speed Cell-Level Path Allocation in a Three-Stage ATM Switch.

Martin Collier

School of Electronic Engineering, Dublin City University, Glasnevin, Dublin 9, Ireland.
email address: collierm@eeng.dcu.ie

ABSTRACT Path allocation in a three-stage ATM switch is the process whereby bandwidth is reserved through the second stage of the switch for each cell. Cell-level path allocation, performed once in every time slot, ensures that cells are routed through the second stage of the switch without delay or contention. A new algorithm for cell-level path allocation was recently proposed by the author. The motive for supporting intermediate channel grouping in the algorithm is described here. The results of the path allocation process must be forwarded to the appropriate cells by a routing tag assignment network. A fast method of routing tag assignment is described, which employs a non-blocking copy network. This reduces the clock rate required of the circuitry, for a given switch size.

1. Introduction.

The use of three stages of switching to allow a large ATM switch to be constructed using smaller switch modules has received considerable attention [1-7]. A key consideration in the design of such switches is the routing algorithm used. The process of determining a routing pattern through the second or intermediate stage of the switch which results in the avoidance of blocking in that stage is referred to as path allocation by the author. Cell-level path allocation algorithms have featured in a number of switch designs [3-7]. Such algorithms require special hardware, so that reconfiguration can be performed at the necessary rate (once per time slot). One such technique, proposed by Collier and Curran, supports the use of intermediate channel grouping, i.e., channel grouping at both the input and output sides of the intermediate stage modules.

This algorithm may be applied to the three-stage switch shown in Fig. 1. This is an n_1L_1 x n_2L_2 switch, with L_1, m, and L_2 modules in the input, intermediate and output stages respectively. There are S_1 links in the channel group connecting input and intermediate stage modules, and S_2 links in the channel group connecting intermediate and output stage modules.

The algorithm requires the following hardware, as discussed in [5].

- Request counting circuitry. A total of L_1 copies of this circuitry is required (one for each input module). This circuitry has n_1 inputs and L_2 outputs, and counts the number of cells requesting each output module. This data is required to initialise the algorithm in each time slot.
- The *atomic*() processor array. This array contains L_1L_2 processors, which implement the algorithm in parallel. Processor X_{ij} deals with requests from the i-th input module for the j-th output module.
- Circuitry for routing tag assignment. A total of L_1 copies of this circuit is also required. This circuitry forwards the results of the path allocation algorithm from the processors to the relevant cells.

The *atomic*() processor array was described in [5] for the special case where the same number of modules is used in each stage. This description shall be generalised in Section 2 of this paper, so that the motivation for using intermediate channel grouping shall become apparent.

The hardware described in [5] for performing request counting and routing tag assignment operates sequentially, and so is relatively slow. This limits the maximum size of switch which can be implemented for a given system clock rate. Details of a faster method of request counting were presented in [6]. A fast method of routing tag assignment will be described in Section 3 of this paper. The performance of a switch employing this algorithm for path allocation has been evaluated in [6].

2. The path allocation algorithm.

The algorithm requires m' iterations, where $m' = max\ (L_1, L_2, m)$, preceded by an initialisation step. It operates on the following quantities:

A_{ir} : the number of channels available from input module i to intermediate switch module r;

B_{rj} : the number of channels available from intermediate switch module r to output module j;

K_{ij} : the number of requests from input module i for output module j.

It is implemented by an $L_1 \times L_2$ array of processors. The processor in row i and column j of the array is called X_{ij}. Processor X_{ij} executes the procedure *atomic*(i, $(i+j-k)$ mod m', j) during iteration k of the algorithm. The procedure *atomic*(i,r,j) is defined as

$$\left.\begin{array}{r} R_{irj} = \min(K_{ij}, B_{rj}, A_{ir}) \\ K_{ij} \leftarrow K_{ij} - R_{irj} \\ B_{rj} \leftarrow B_{rj} - R_{irj} \\ A_{ir} \leftarrow A_{ir} - R_{irj} \end{array}\right\} \quad atomic(i, r, j)$$

where R_{irj} is the number of cells which will be routed from input module i to output module j via intermediate module r. Each processor contains three registers (A, B and K). These are initialised as follows.

K ¬ K_{ij}

A ¬ S_1

B ¬ S_2

After iteration k of the algorithm, the updated value of B is forwarded to $X_{(i+1) \bmod m', j}$ and the new A value is sent to $X_{i, (j+1) \bmod m'}$. The K register value is retained locally. If $L_1 < m'$, X_{ij} is not an *atomic*() processor, for $i \geq L_1$, but simply a B register. Similarly, if $L_2 < m'$, X_{ij} is not an *atomic*() processor, for $j \geq L_2$, but simply an A register. These extra registers introduce delays to ensure that processor X_{ij} receives A_{ir} and B_{rj} simultaneously. An example of such an array is shown in Fig. 2.

A large value for m will be required if intermediate channel grouping is not used. Hence, the execution time of the algorithm, in clock cycles, will be long, and so a high system clock rate will be required. This penalty can be avoided if the bandwidth of the intermediate stage is increased by increasing S_1 and S_2, rather than by increasing m.

3. Routing tag assignment using a copy network.

3.1 Principles of operation.

The functions to be performed during routing tag assignment resemble those carried out by the 'allocation network' in Pattavina's switch [8]. Hence, fast routing tag assignment could be performed using a running sum adder network, similar to that employed by Pattavina. An alternative method for fast routing tag assignment will now be described, which uses a modified version of Lee's copy network [9]. This network is used to broadcast a routing tag simultaneously to all the address generators which should receive it.

A Batcher network and a copy network are used, as shown in Fig. 3. The copy network has $n_1 + L_2$ inputs and outputs. The routing packet generators are connected to L_2 of the copy network inputs, and the remaining inputs are idle. Routing packet generator RPG$_j$ receives the value of K_{ij} from the appropriate *atomic*() processor. The *atomic*() processor X_{ij} generates a sequence of K_{ij} values, one after every iteration of the algorithm, commencing with K_{ij}^0 (the initial value of K_{ij}, determined by the request counting hardware) and decrementing, after every iteration, in accordance with the *atomic*() procedure, as paths are allocated to cells.

The Batcher network merges the data cells (arriving at the input ports of input module i) with a set of control packets (one for each output module) in such a way that the data cells requesting output module j appear at higher-numbered output ports of the Batcher network than control packet j. Hence, the cells requesting output module j appear at outputs $D_{j-1} + 1$ through $D_{j-1} + K_{ij}^0$ of the Batcher network, where the value of D_j is the address of the sorter output port at which control packet j appears.

Evidently

$$D_j = \sum_{u=0}^{j} K_{iu}^0 + j\ .$$

The routing packet generator for output module j (RPG_j) must forward the relevant routing tags to the data cells at outputs $D_{j-1} + 1$ through $D_{j-1} + K_{ij}^0$ of the Batcher network. The request counting hardware described in [5,6] can be (trivially) modified to generate D_{j-1}.

After each iteration of the path allocation algorithm (including the initialisation step), RPG_j submits a routing packet to the copy network, to be broadcast to address generators $D_{j-1} + 1$ through $D_{j-1} + K_{ij}$, containing in the data field the token address, i.e., the address of the intermediate switch module through which routes were allocated during that iteration. If $K_{ij} = 0$, an inactive packet is submitted. An address generator may thus receive a number of routing tokens; only the most recently received token is used to generate a routing tag.

A set of null tokens is broadcast to those cells which cannot be routed when the path allocation algorithm has terminated. This requires only a few clock cycles. The hardware described in [5] introduced an additional delay, since the broadcasting of tokens occurred sequentially rather than simultaneously.

The routing packets submitted to the copy network do not collide, since (as shown in [10]) they satisfy the condition for avoiding internal contention in Lee's copy network, presented in [9].

3.2 The Modified Copy Network

An apparent difficulty with this method of routing tag assignment is the amount of data which must be processed during each broadcast. In general, Lee's copy network must process two bits (one each from the upper and lower address) in addition to the activity bits, at each stage. Hence, the interval between successive passes of the algorithm, in bit times, will be quite large. The speed of the algorithm can be increased by observing that, in this application of the copy network, the lower address bit processed at each node never changes after the first (initialisation) step of the algorithm. Hence, on subsequent passes of the algorithm, there is no need to distribute the lower address, so that the header on the routing packet may be shortened, reducing the delay through the copy network.

The proof of the assertion that the lower address bit to be processed at each node of the copy network never changes after the first iteration is given in Appendix A.

3.3 An Example of Routing tag Assignment

The success of this approach to routing tag assignment shall be demonstrated by an example described in Tables 1 through 3. The example considered features 4 modules in each stage of the switch.

Table 1 indicates the number of cells from input module 0 (IM_0) which have requested each of the four output modules and a possible outcome of the path allocation process. Table 2 indicates the contents of the K register of each *atomic*() processor associated with IM_0 after initialisation (0^-) and after each iteration (0^+, 1^+, 2^+ and 3^+). The resulting values of the routing packet headers are shown in Table 3.

The broadcasts for each iteration of the algorithm are illustrated in Figs. 4 (a) through 4 (e). The value printed beside each link on the broadcast trees represents the lower address bit to be processed by the next stage of the copy network. It can be seen that this never changes after initialisation. After five broadcast operations, the correct number of cells has been assigned a path via each intermediate switch module.

Output module no. (j)	0	1	2	3
No. of requests (K_{0j})	4	7	0	1

(a) Number of cells arriving to input module 0 requesting each output module.

	Destination			
	OM_0	OM_1	OM_2	OM_3
Routed via ISM_0	0	1	0	0
Routed via ISM_1	1	3	0	0
Routed via ISM_2	1	2	0	0
Routed via ISM_3	2	0	0	1
Total cells routed	4	6	0	1
cells losing contention	0	1	0	0

(b) Results of path allocation

Table 1: An example of path allocation

	iteration				
	0^-	0^+	1^+	2^+	3^+
K_{00}	4	4	2	1	0
K_{01}	7	4	3	3	1
K_{02}	0	0	0	0	0
K_{03}	1	0	0	0	0

Table 2: Contents of *K* registers during path allocation.

3.4 Clock cycles required

The length of each routing packet (after the first) is

$$L_r = 1 + \lceil \log_2(m+1) \rceil + \lceil \log_2(n_1 + L_2) \rceil,$$

i.e., one activity bit, enough bits to represent the token address, and sufficient bits to represent the requested upper copy network output.

Hence, routing packets may be submitted to the network at the rate of one every L_r clock cycles. If this exceeds the number of clock cycles required for one iteration of the path allocation algorithm, an undesirable delay is introduced, whereby the path allocation can only proceed at the rate of one iteration per L_r clock cycles.

A solution to this difficulty involves a reduction in the value of L_r. The token address is not broadcast, except during initialisation. The address generator stores the token address received then, and on subsequent iterations calculates the token address by decrementing the previous value. Therefore, the value of L_r can be reduced by $\lceil \log_2(m+1) \rceil$ bits.

Once path allocation is complete, the tag assignment process requires only a further

$$L_r + 2\lceil \log_2(n_1 + L_2) \rceil$$

clock cycles to terminate (this is the time required to generate the final null routing packet, and to propagate it through the copy network). This compares favourably with the corresponding number of clock cycles for the method described in [5], which, in the worst case, requires

$$[n_1 - min(S_1, S_2)].D$$

clock cycles, where Δ (typically equal to one) is the number of clock cycles required to propagate a routing packet through an address generator. Which of the two strategies is to be preferred depends on the required operating speed of the circuitry.

	Iteration				
	0^-	0^+	1^+	2^+	3^+
Routing packet for OM_0	1,0,3	1,0,3	1,0,1	1,0,0	0,0,-1
Routing packet for OM_1	1,5,11	1,5,8	1,5,7	1,5,7	1,5,5
Routing packet for OM_2	0,13,12	0,13,12	0,13,12	0,13,12	0,13,12
Routing packet for OM_3	1,14,14	0,14,13	0,14,13	0,14,13	0,14,13

Table 3: Routing packet header contents (Activity bit, Lower broadcast address, Upper broadcast address).

4. Conclusions.

The principal difficulty in implementing a cell-level path allocation algorithm for three-stage ATM switches is the high bit rate required of the circuitry. The fast method of routing tag assignment described here, together with the fast method of request counting presented in [6], allows the bit rate of the algorithm described in [5] to be reduced considerably, for a given switch size. It was estimated, in [5], that a switch with $L_1 = m = L_2 = 32$, $n_1 = 96$, $S_1 = 4$ and $S_2 = 8$ would require a system clock rate of 290 MHz. The use of the faster hardware techniques described here and in [6] (in conjunction with a faster implementation of the *atomic*() processor described in [10] requiring only 9 clock cycles per iteration) allows this rate to be reduced to about 130 MHz. Hence, it should be possible to implement the algorithm in CMOS VLSI.

Appendix A

The lower address bit need not be transmitted through the copy network on the second and subsequent passes of the routing tag assignment algorithm. This may readily be demonstrated if the Boolean interval splitting algorithm proposed by Lee [9] is described in the following terms.

Let the lower and upper outputs of a switch element of the copy network be referred to as OUT_0 and OUT_1 respectively. Consider the switching which occurs at stage k of the network. The incoming packet is described by three quantities, A (the activity; A=1 if a packet is present; A=0 for an idle input), L (the lower address) and U (the upper address). The corresponding quantities for OUT_0 are $A(OUT_0)$, $L(OUT_0)$ and $U(OUT_0)$, and for OUT_1 are $A(OUT_1)$, $L(OUT_1)$ and $U(OUT_1)$. Let l_k be the lower address bit inspected at stage k.

Using this notation, it follows that

$$A(\mathrm{OUT}_0) = 1 - sgn(L - 2^k);$$
$$A(\mathrm{OUT}_1) = sgn(U - 2^k);$$

$$L(\mathrm{OUT}_0) = L;$$
$$L(\mathrm{OUT}_1) = max(L, 2^k);$$

$$U(\mathrm{OUT}_0) = min(U, 2^k - 1);$$
$$U(\mathrm{OUT}_1) = U,$$

where

$$sgn(x) = \begin{matrix} 1, & x \geq 0 \\ 0, & x < 0 \end{matrix}.$$

Note that when A=0, the values of L and U are 'don't cares'. There are three possible outcomes of the Boolean interval splitting algorithm (routing to OUT_0, routing to OUT_1, or routing to both outputs). It may be shown that the above description of the data on OUT_0 and OUT_1 gives results consistent with those described by Lee, in all three cases. This proves the validity of this description of Lee's algorithm.

It is apparent that the values of $L(\mathrm{OUT}_0)$ and $L(\mathrm{OUT}_1)$ are dependent only on L and k. Hence it follows that the value of l_k for each node in the tree consisting of copy network links which are carrying a packet which originated at a single input is dependent only on L.

It can be concluded that two broadcasts, both from the same input port, and with (A, L, U) equal to $(1, L, U_1)$ and $(1, L, U_2)$ respectively, present the same bit as l_k to switching elements which lie on the tree common to $(1, L, U_1)$ and $(1, L, U_2)$. However, if $U_1 < U_2$, the tree associated with $(1, L, U_1)$ contains only links which are shared with the $(1, L, U_2)$ tree. Therefore, if the $(1, L, U_1)$ broadcast is preceded by that for $(1, L, U_2)$, the value of L need not be transmitted, provided the values of l_k are stored in the relevant switch elements.

In the routing tag assignment application, the broadcasts on successive iterations of the algorithm involve a non-decreasing fanout, with an unchanging lower address. Hence, after the first iteration, the lower address need not be transmitted. Since the network is contention-free, the lower address bit stored in a switch element will never be read by a packet from an input port other than that from which the bit was received.

References

[1] Y. Sakurai *et al.*, "Large scale ATM multi-stage switching network with shared buffer memory switches," *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. IV, pp. 121-126.

[2] M. Collier and T. Curran, "The strictly non-blocking condition for three-stage networks," to be presented at ITC-14, Antibes, May 1994.

[3] A. Cisneros, "Large packet switch and contention resolution device," *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. III, pp. 77-83.

[4] K. Eng and M. Karol, "The growable switch architecture: a self-routing implementation for large ATM applications," *Proc. ICC '91*, pp. 1014-1020.

[5] M. Collier and T. Curran, "Path allocation in a three-stage broadband switch with intermediate channel grouping," *Proc. Infocom '93*, pp. 927-934, San Francisco, Mar.-Apr. 1993.

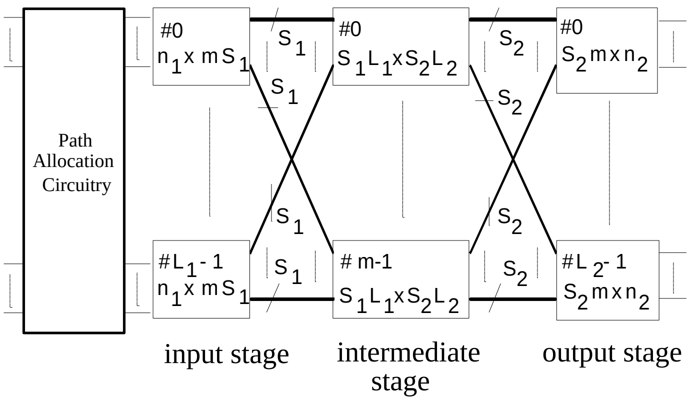
[6] M. Collier and T. Curran, "Cell-level path allocation in a three-stage ATM switch," to be presented at *ICC 94*, New Orleans, May '94.

[7] J. Hui and T.-H. Lee, "A large-scale ATM switching network with sort-banyan switch modules," *Proc. Globecom '92*, pp. 133-137.

[8] A. Pattavina, "A broadband packet switch with input and output queueing," *Proc. International Switching Symposium*, Stockholm, May-June 1990.

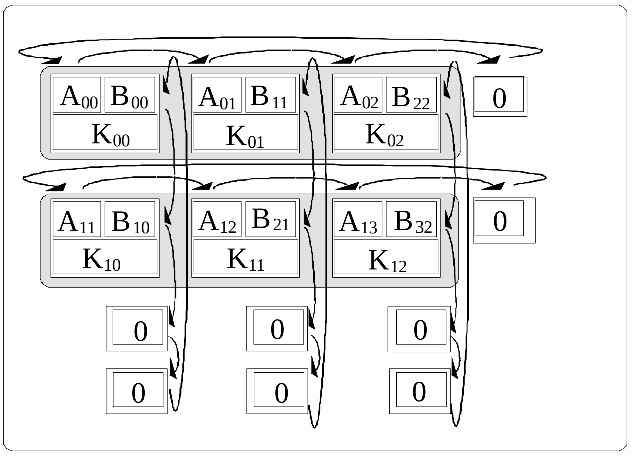
[9] T.T. Lee, "Nonblocking copy networks for multicast packet switching," *Journal of Select. Areas Commun.*, vol. 6, no. 9, pp. 1455-1467, Dec. 1988.

[10] M. Collier, "Switching Techniques for Broadband ISDN," PhD thesis, Dublin City University, 1993.

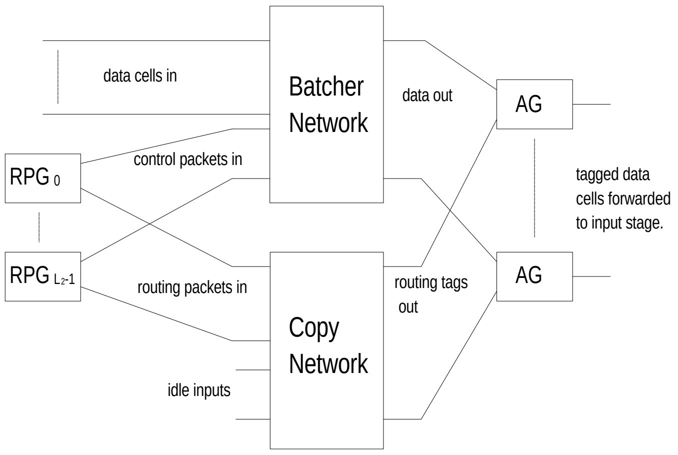


V:	channel rate (155 Mb/s)
n_1 (n_2) :	the number of input (output) ports per input (output) module;
L_1 (L_2) :	the number of input (output) modules;
m:	the number of intermediate switch modules;
S_1 (S_2) :	the number of channels in the channel group connecting each input (output) module to each intermediate switch module.

Fig. 1: A three-stage switch with intermediate channel grouping.


A_{00} B_{00} K_{00}
A_{01} B_{11} K_{01}
A_{02} B_{22} K_{02}
A_{11} B_{10} K_{10}
A_{12} B_{21} K_{11}
A_{13} B_{32} K_{12}
0

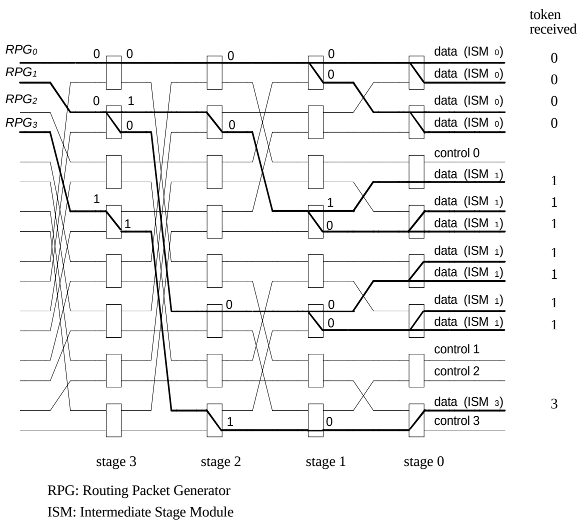
Initial conditions for L_1 =2, m = 4, and L_2 = 3.
Fig. 2: Example of a processor array.



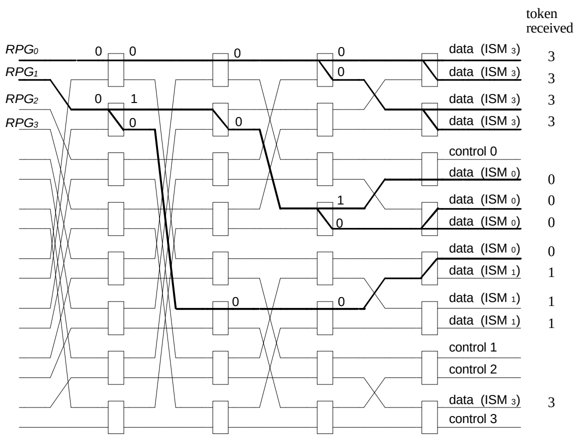
RPG : routing packet generator

AG : address generator

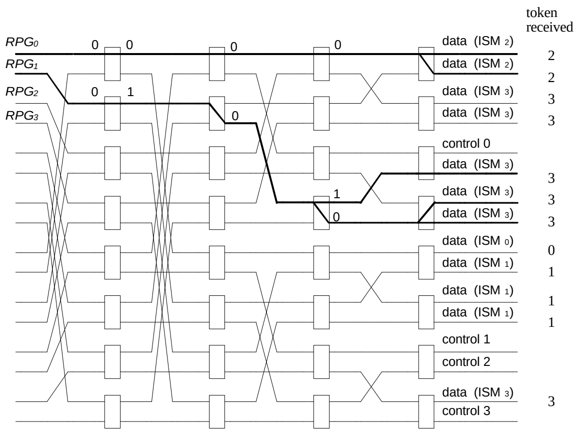
Fig. 3: A faster method of routing tag allocation.



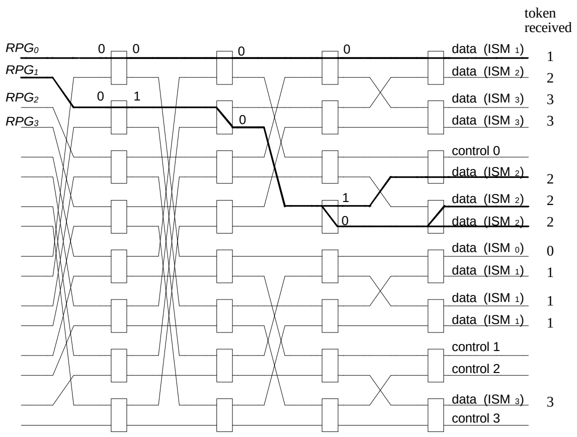
(a) Iteration 0^-.



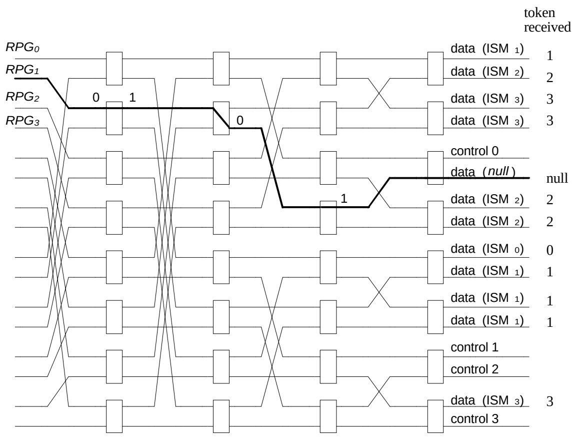
(b) Iteration 0^+.



(c) Iteration 1^+.



(d) Iteration 2^+.



(e) Iteration 3^+.

Fig. 4: An example of routing tag assignment.