Design of multicast controller for high-capacity Internet router

M. Blagojević and A. Smiljanić

Admissible multicast traffic can be efficiently handled in high-capacity Internet routers using packet circulation. Presented is a practical algorithm for the building and maintenance of the circulation tree, as well as a report on its implementation. Its scalability and speed were assessed, and shown to be satisfactory.

Introduction: Multicasting in high-capacity Internet has not been resolved yet. Multicast traffic that we consider has one source and multiple destinations. There are two main approaches for forwarding of the multicast traffic through a crossbar. In the case of unicast crossbars, an input has to replicate a multicast packet, as many times as there are output ports to which that packet is bound. So, if the input sources popular multicast sessions, it might become clogged easily. Alternatively, the crossbar might have a multicast capability, meaning that one input might simultaneously send packets to multiple outputs. However, it was shown that even if the optimal scheduling algorithm is applied to the router, the speedup required to pass an admissible traffic increases with the number of router ports [1]. On the other hand, the high-capacity router could handle the multicast traffic well when the packets are circulated through a unicast fabric [2, 3]. The input is allowed to send incoming packets just to the limited number of the ports belonging to the multicast set. In the next iteration, ports which received forwarded multicast packets keep forwarding them, to the limited number of ports from the multicast set which did not receive that packet in previous iterations. As a consequence, transmission load of the input port is spread over all ports in the multicast set. Packet circulation is simple to implement. Packets are forwarded according to the circulation tree corresponding to that multicast session. The circulation tree is updated when users in the network join or leave the multicast session. We propose a novel algorithm for the circulation tree calculation, and then report on the implementation of this new solution.

Balanced tree: Information about the circulation tree is kept distributed among all ports belonging to the tree. Each port has its multicast tree memory with one entry for each multicast session passing that port. The memory address of the entry named as the session identification number (SID) uniquely describes the session on that port. Only information about its parent and child(ren) ports are kept in one entry. The information about the port comprises its ID number (from 0 to N−1, where N is number of ports) and its SID number of the given multicast session. When a multicast packet arrives to the router (root port), lookup has to be performed and its long multicast address is replaced with the shorter SID. Then, the packet SID number is updated as this packet is forwarded through the tree, as each parent knows its child SID. In this way, the lookup that determines the next port based on its long multicast address is avoided as the packet is being circulated through the router. The multicast controller is the module that creates and maintains the circulation trees based on the requests for adding new ports and removing existing ports from their multicast sets. In order to service a new request, the ports exchange internal control messages. In [3, 4], the multicast controller determines the balanced tree which is the shallowest, thus providing the lowest delay. The balanced tree is achieved using branch fanout numbers. The branch fanout denotes the number of nodes that can be reached through the corresponding branch. Tree update is performed so that the difference between fanout numbers, at any node, is not greater than one. The main issue with updating the balanced tree arises when the request for excluding some port from a multicast session arrives. In order to keep the tree balanced, one of the leaf ports ought to replace the leaving port. The parent and children ports need to update their tree memories accordingly. In order to perform update, the SID corresponding to the multicast session of the leaving port has to be obtained. Therefore, each port should have a lookup table that maps the multicast address to the corresponding SID number. This lookup table is accessed only when the port is leaving. However, it consumes a considerable memory at each port.

Binary tree: When the request for adding a new port arrives, the control message is forwarded from the root node, through the tree, until the leaf node is reached. Each port forwards the control message according to its tree level and the new port ID. Namely, if we denote the *i*th bit of the new port ID as ID(i) ($i = 0$ denotes MSB), then the control message will be forwarded to the right child at level i if ID(i) = 1, and to the left child if ID(i) = 0. The described procedure is shown in Fig. 1. Bold font is used to show bits that define the path from the root to the corresponding port.

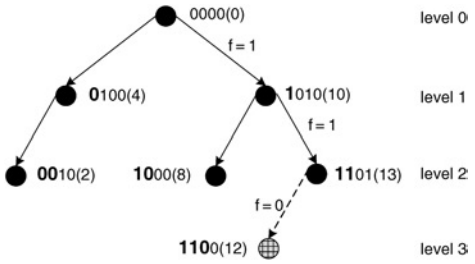

Fig. 1 *Handling request for adding port 1100(12) to forwarding tree*

The procedure for removing a port from the forwarding tree consists of two steps. First, the path from the root to the leaf node through the port on leave is selected (bold path in Fig. 2*a*). Then, the leaving port is removed from the tree and all ports down the selected path are moved one level up. Each of those ports is moved to the position of its parent node. The first part of the path, up to the leaving port, is selected using the leaving port ID. The remaining part of the path, from the leaving port to the leaf node, is selected arbitrarily. The complete procedure of replacing the port on leave is shown in Fig. 2.

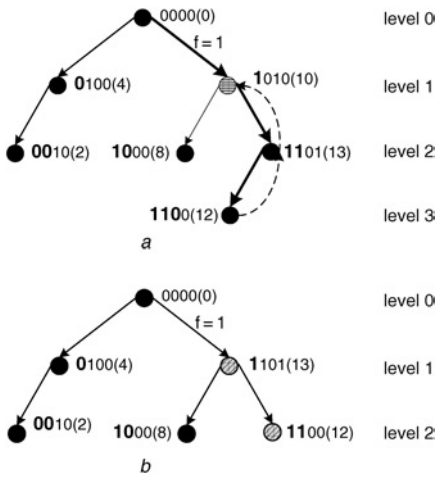

Fig. 2 *Handling request for removing port 1010(10) from forwarding tree*
a Patch selection
b Updated tree

Assuming the described procedures for adding and removing the ports, the position of each port in the tree is determined by the port ID. It is easy to see that the maximal number of tree levels is $\log_2 N - 1$, and it limits the total forwarding delay. In addition, the multicast controllers do not require additional lookup tables.

Performance and scalability assessment: A multicast controller based on the proposed binary tree algorithm is implemented on the FPGA device (Altera Cyclone II EP2C70F896C6). The implemented structure is shown in Fig. 3. Each port has its own multicast controller module. The crossbar scheduler, implemented in [5, 6] is included in that module. Each module can receive packets from two sources: new packets, which just enter the router and old packets, processed by other modules and passed through the crossbar. A multiplexer is included in each module to provide handling of packets arriving simultaneously from both sources. Multiple modules can fit the single chip. The number of modules on the single chip depends on the size of available memory, and the number of available pins. Simulations were performed for two different cases. In the first, one I/O pin is used for one signal, while in the second case two signals are multiplexed on the single pin. Simulation results are presented in Table 1. Numbers of modules per chip and processing time are denoted with N_{pm} and

T_S, respectively. From the column 'Lim' it can be seen which factor limits the number of input modules that fit one FPGA device: the size of available memory (M4k), or the number of available pins (Pin).

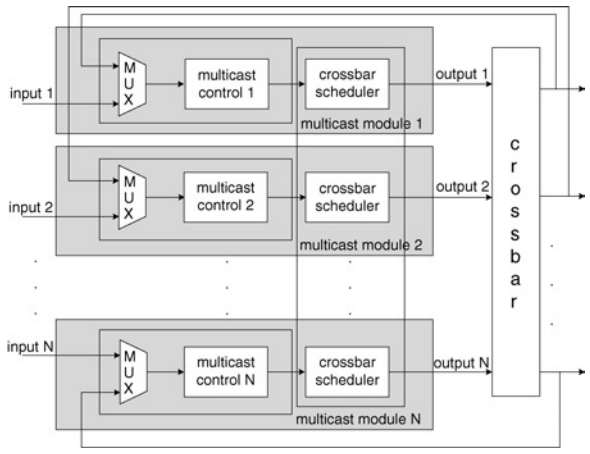

Fig. 3 *Multicast controller structure*

Table 1: Simulation results

	Simulation case 1			Simulation case 2		
N	N_m	T_s (ns)	Lim	N_{pm}	T_s (ns)	Lim
8	8	40.95	–	8	40.25	–
16	10	42	Pin	16	44.34	–
32	8	46.06	Pin	16	47.82	M4k
64	7	48.18	Pin	12	48.51	M4k
128	5	49.26	Pin	7	49.71	M4k

Conclusion: The proposed algorithm provides fast and flexible tree update. Control messages are always propagated in the same direction, from the root to the leaf node. Thus, there is no problem with obtaining SID numbers as in the case of the balanced tree. Scalability is assessed in terms of the number of modules that can be fit on the single chip and satisfactory results are obtained. Achieved processing time is less than 50 ns, which is tolerable by the delay sensitive applications.

28 September 2007
Electronics Letters online no: 20082789
doi: 10.1049/el:20082789

M. Blagojević and A. Smiljanić (*Belgrade University, Belgrade, Serbia*)

E-mail: milosdb@etf.bg.ac.yu

A. Smiljanić: Also with Stony Brook University, New York, USA

References

1 Marsan, M.A., Bianco, A., Giaccone, P., Leonardi, E., and Neri, F.: 'Optimal multicast scheduling in input-queued switches'. Proc. of IEEE Int. Conf. on Communication 2001, Helsinki, Finland, June 2001
2 Smiljanić, A.: 'Flexible bandwidth allocation in high-capacity packet switches', *IEEE/ACM Trans. Netw.*, April 2002, pp. 287–293
3 Smiljanić, A.: 'Scheduling of multicast traffic in high-capacity packet switches', *IEEE Comm. Mag.*, (IEICE/IEEE HPSR), November 2002, pp. 72–77
4 Blagojević, M., Smiljanić, A., and Petrović, M.: 'Design of the multicast controller for the high-capacity Internet router'. Proc. of IEEE High Performance Switching and Routing 2007, New York, USA, June 2007
5 Petrović, M., and Smiljanić, A.: 'Optimization of the scheduler for the non-blocking high-capacity router', *IEEE Comm. Lett.*, 2007, **11**, (6), pp. 534–536
6 Petrović, M., Blagojević, M., Joković, V., and Smiljanic, A.: 'Design, implementation, and testing of the controller for the terabit packet switch'. Proc. of IEEE Int. Conf. on Communications Circuits and Systems 2006, Guilin, China, June 2006