A Model for the Development of AS Fabric Management Protocols*

Antonio Robles-Gómez, Eva M. García, Aurelio Bermúdez, Rafael Casado, and Francisco J. Quiles

Instituto de Investigación en Informática de Albacete (I^3A)
Universidad de Castilla-La Mancha
02071 Albacete, Spain
arobles@dsi.uclm.es

Abstract. Advanced Switching (AS) is a switching fabric architecture based on the PCI Express technology. In order to support high availability, AS includes important features, such as device hot addition and removal, redundant pathways, and fabric management failover. This work presents an AS model developed in OPNET. The contribution of this tool is that it can help researchers to design and evaluate management mechanisms for this new technology. It can also be used to analyze other key aspects of the architecture, such as routing, congestion, and quality of service.

Keywords: Advanced switching, modeling, network management, network availability.

1 Introduction

The Advanced Switching specification [1] has been developed by the Advanced Switching Interconnect Special Interest Group (ASI-SIG). It is a chip-to-chip and backplane interconnect switched fabric architecture. Unlike similar technologies, such as InfiniBand [5] and Quadrics [9], AS can be seen as the last step in the evolution of the traditional PCI bus [6]. In particular, AS inherits most of the physical and link characteristics of PCI Express [8]. However, it offers a bigger application space, including multiprocessing and peer-to-peer communications. The first commercial AS-compliant products have just started to appear in the marketplace [11].

To guarantee network availability, AS provides a fabric management mechanism, which basically configures and monitors the status of the network. Consider, for example, the occurrence of a failure in a network device. The management mechanism must detect that failure, discover the resulting topology, and finally obtain and distribute to the endpoints a new set of routes for packet delivery. All these tasks are performed by the fabric manager (FM), a software entity running on one or more AS endpoints.

* This work is supported by the following projects: TIC2003-08154-C06-02 (Ministerio de Ciencia y Tecnología), PBC05-007-1 (Junta de Comunidades de Castilla-La Mancha), and PCTC0622 (Universidad de Castilla-La Mancha).

W.E. Nagel et al. (Eds.): Euro-Par 2006, LNCS 4128, pp. 853 – 863, 2006.

The internal behavior of the management mechanism is currently an open issue for vendors and researchers. The AS specification only considers a set of configuration data structures –called *capabilities*– into each device, and the management packets –called *PI-4 packets*– used to exchange them among devices.

This paper presents a simulation model that provides the necessary support –capabilities and PI-4 packets– to develop management mechanisms. In order to be able to evaluate future proposals, our simulator allows measuring accurately control overhead and the time expended by the management process.

Our AS model is an evolution of a previous model [2] developed for the InfiniBand technology [5]. There are many differences between both technologies, such as source routing instead of distributing routing, and passive instead of active switches. These differences completely justify the development of a new tool to design specific management mechanisms for AS.

The AS model has been developed using the OPNET Modeler software [7]. This tool provides support to model and analyze communication networks and distributed systems. In OPNET, network devices are modeled through node models, which are built using basic modules. Fig. 1 shows an example. Each module can generate, send, receive, and consume packets from other modules. The behavior of a module is programmed via its process model. It consists of a finite state machine (see Fig. 4) containing blocks of C/C++ code and calls to the OPNET API.

The remainder of this paper is organized as follows. The next section describes the way we have modeled the AS network devices. Then, Section 3 introduces the modeling of the fabric management support. After that, we revise some tasks in the management mechanism that we plan to develop in the future. Finally, Section 5 gives some conclusions and future work.

2 Modeling the AS Architecture

Our model[1] is made up of AS x1 links, 16-port switches, and fabric endpoints. This section presents the way in which these network devices are modeled and it details some architectural issues closely related to the fabric management process, as flow control and the port state machine.

2.1 Network Components

We have defined AS links starting from the basic OPNET point-to-point bidirectional link model. The specified bandwidth for these links is 2.5 Gbps. However, bandwidth is reduced by 8b/10b encoding to 2.0 Gbps. So far, we have not considered transmission errors. To implement cut-through switching, we have programmed the link model in such a way that the receiver port can process a packet once the header has been received.

We have also modeled a multiplexed virtual cut-through switch [4]. Fig. 1 shows the modules implementing two switch ports –numbered as 7 and 8–, the switch arbitration unit, and the crossbar.

[1] The source code of our AS model will be freely available for the OPNET community, at the "Contributed Models" depot of the OPNET support center [7].

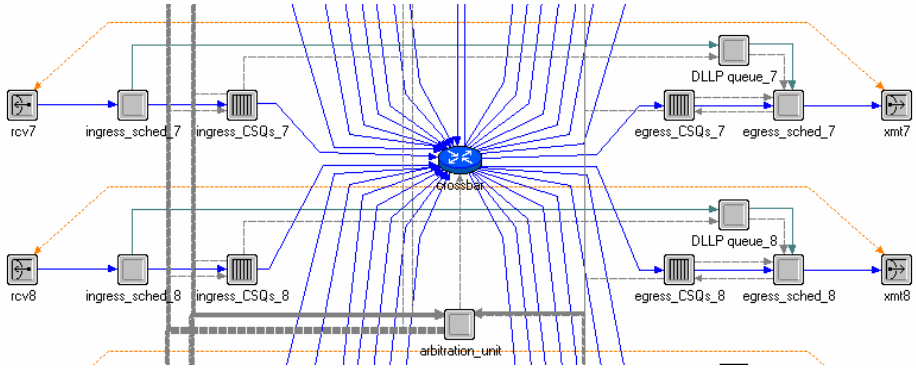


Fig. 1. A detail of the switch model

Each input channel contains a point-to-point receiver (*rcv* module in Fig. 1) connected to the link. A selector (*ingress_sched* module) delivers flow control packets (DLLP, data link layer packet) to the flow control unit. The rest of packets (TLP, transaction layer packet) are sent to the *ingress_CSQs* module.

AS defines three types of virtual channels: unicast bypassable VCs (BVC), unicast ordered VCs (OVC), and multicast VCs (MVC). Each BVC implements an ordered queue and a bypass one. Packets marked as "bypassable" (the *OO* field in Fig. 2 is unset) are delivered to the bypass queue if they cannot progress due to lack of credit. Packets at this queue can be "bypassed" by other packets at the ordered queue. On the other hand, OVCs and MVCs only support ordered queues. In our model, the number of virtual channels of each type and the size of the associated input and output buffers are defined as switch attributes.

A traffic class (TC) mechanism allows the grouping of traffic flows for similar treatment. The traffic class of a packet is defined at the source endpoint. When a packet reaches a port, the *Traffic Class* field at the header is used to obtain the corresponding VC, by using a set of fixed TC/VC mapping tables. The *ingress_CSQs* module in Fig. 1 performs this mapping, and stores the packet at the tail of the input buffer associated with the corresponding virtual channel.

In order to simplify the hardware, AS states that unicast packets use source routing. Endpoints include path information into the packets, by filling up the *Turn Pool*, *Turn Pointer*, and *D* (direction) fields in the packet routing header (shown in Fig. 2). These values are used at each intermediate switch to obtain the output port. In our model, unicast packets are routed when they reach the header of the input buffers. On the other hand, multicast packets require to look up into a specific forwarding table. These tables are stored at the switches and are defined by the management process.

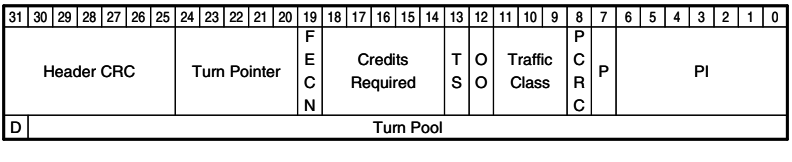


Fig. 2. AS routing header

The arbitration unit (*arbitration_unit* module in Fig. 1) receives requests from the input buffers and configures the crossbar, taking into account the space available at the output buffers (*egress_CSQs*) and the status of the internal channels.

AS defines several mechanisms for congestion management. First, it uses the credit-based flow control defined by the PCI Express architecture. The flow control unit (*DLLP queue* module in Fig. 1) processes incoming DLLPs and activates/deactivates the transmission of TLPs through the output channels. It must also inject periodically new DLLPs, in order to update the credit information at the neighbor port. The behavior of this module will be detailed in the next section.

Additional optional congestion mechanisms defined in AS are status-based flow control, minimum bandwidth scheduler, and endpoint source injection rate limiting. These mechanisms are not currently implemented in our simulator.

To conclude the description of the switch model, the output channel arbitration unit (*egress_sched* module in Fig. 1) receives requests from the port flow control unit and output buffers, and decides the packet that will be finally delivered to the physical link, through the transmitter module (*xmt*). Before sending a TLP, this module must consider the credit available at the corresponding neighbor input buffer, which is periodically notified by the flow control unit.

The endpoint model (not shown here) incorporates a communication port, including exactly the same modules as a switch port. There is also an *application* module which generates and consumes upper-level packets. Parameters for traffic generation, such as packet size and injection rate, are defined as simulation attributes.

2.2 Port Behavior and Flow Control Unit Model

Fig. 3 shows the set of possible states for a port, as defined in the AS specification. This behavior has been considered in our model. Once the device is powered-on, a port initialization phase starts. Each port tries to synchronize with a potential neighbor device. To do that, the port transits from *DL_Inactive* to *DL_Init*, and sends DLLPs through the link. If the port does not receive a response, it returns to the *DL_Inactive* state. After some time, it will try to synchronize again.

If the port receives a response from the neighbor, they must negotiate the number of virtual channels they are going to use in the communication. When the negotiation process finishes, the port transits to the *DL_Protected* state. In this state, the transmission of certain management packets (PI-0:0, for FM election, and PI-4, for device discovery and configuration) is allowed.

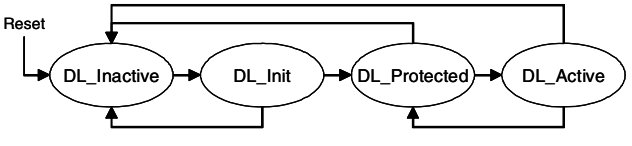


Fig. 3. Port state machine

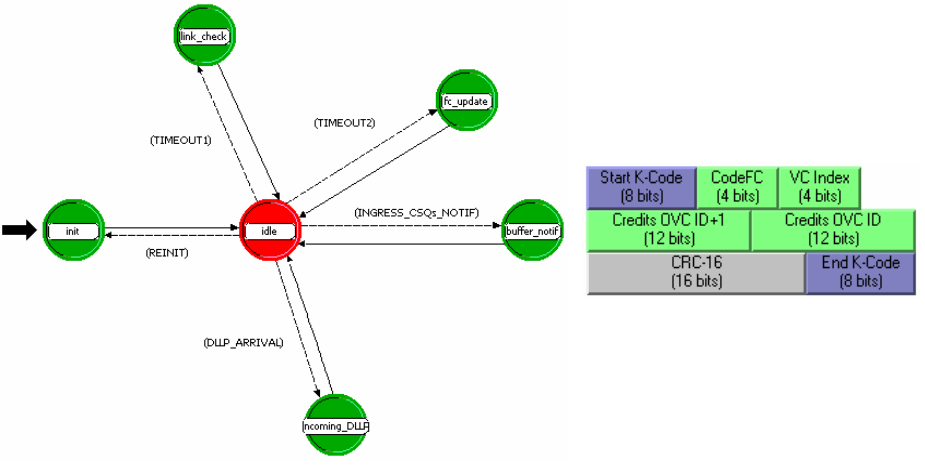


Fig. 4. (*left*) Flow control unit behavior and (*right*) a DLLP with credit information for two successive OVCs

The FM can order the port to transit to the *DL_Active* state by means of a PI-4 packet. In this state, the port is completely operational, allowing the transmission of all packet types. In the same way, the FM can order a transition from *DL_Active* to *DL_Protected*. Finally, the port will return to *DL_Inactive* if the link or the neighbor device is taken down, and DLLPs are not received during a period of time.

The flow control unit (*DLLP queue* module in Fig. 1) models the port behavior we have just described. Moreover, it implements the flow control tasks enumerated in the previous section. Fig. 4(*left*) shows the finite state machine in the corresponding OPNET process model.

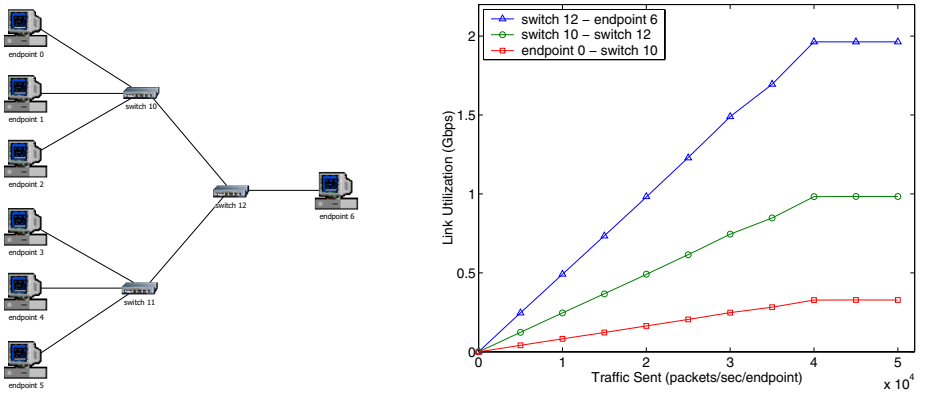
The *init* state performs some initialization tasks. In the *idle* state, the process model is waiting for the occurrence of some simulation event. Periodically, the machine enters the *link_check* state and begins the port initialization phase.

In order to inform the neighbor port about the credit available at the local input buffers, the process enters the *fc_update* state periodically. In this state, the corresponding DLLPs are generated and injected. Fig. 4(*right*) shows the format of an *FC_Update* DLLP defined in OPNET.

The *buffer_notif* state is reached when the flow control unit receives a notification from the *ingress_CSQs* module, reporting about a variation in the occupation of an input buffer.

When a DLLP arrives to the flow control unit, it is processed at the *incoming_DLLP* state. According to its type, the flow control unit either continues the initialization process, or communicates the available neighbor credit to the *egress_sched* module.

Finally, the process model returns to the *init* state if for a certain time interval the flow control unit has not received DLLPs with information about the neighbor credit.



(a) Scenario used (b) Simulation results

Fig. 5. Example of validation

2.3 Flow Control Validation

Several tests have been conducted in order to validate the AS model. As an example of this process, the topology in Fig. 5*(a)* has been used to check the correct implementation of the credit-based flow control mechanism. In this scenario, the six endpoints on the left side inject packets to the endpoint located on the right, assuming the existence of only one OVC in each device.

We have run several simulations varying the packet injection rate at the source endpoints. Fig. 5*(b)* shows the link utilization at each level in the topology. We can see that the maximum link bandwidth (2 Gbps) is never exceeded on the link connecting switch *12* and endpoint *6*.

Additionally, results for the other two series show that the flow control is correctly working. Note that the utilization of the link connecting switch *10* and switch *12* is exactly half of the utilization at the next hop, and the utilization of the link connecting endpoint *0* and switch *10* is the third part of the previous one.

3 Fabric Management Model

We are interested in the development of fabric management mechanisms for the AS technology. This section describes the aspects of the specification that provide support for this purpose, and how they have been modeled into the simulator. These features are the configuration space in each device, and the protocol that allows the FM to access it.

3.1 Device Configuration Space

The device configuration space is a storage area that contains a set of fields to specify device characteristics as well as fields used to control the device. This information is presented in the form of structures called capabilities. Each capability structure defines a specific characteristic of the device.

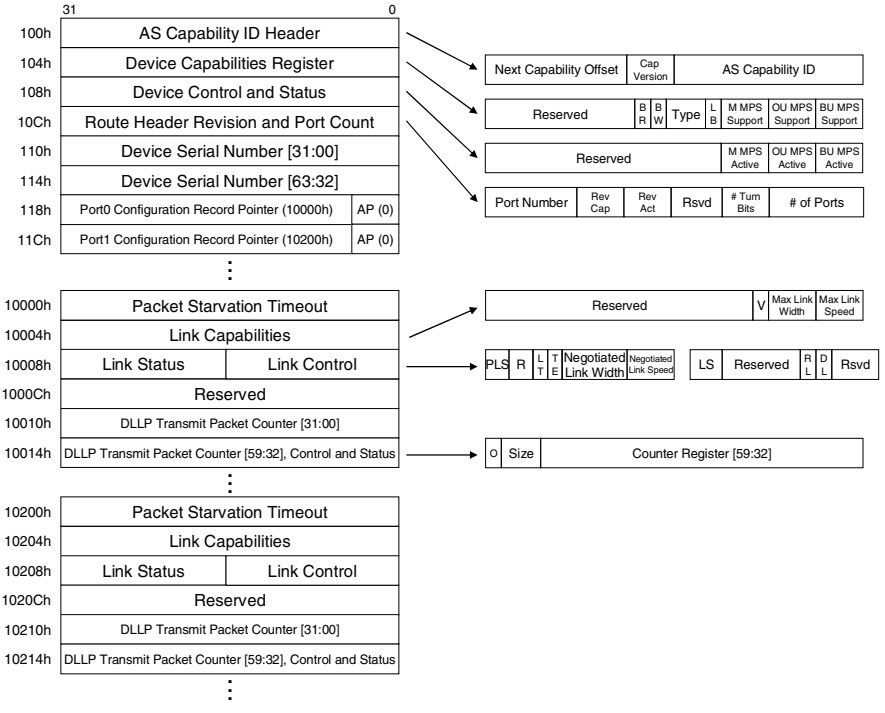


Fig. 6. Structure of the *baseline device* capability

The configuration space is made up of up to 16 blocks of 4 Gbytes of storage, called apertures. All the capability structures reside at the aperture 0. Additional data associated with the capabilities may be stored in any aperture.

Our model allows defining any AS capability. Currently, it includes the *baseline* and the *spanning tree* capabilities. The reason is that these capabilities are needed by several management processes, such as the topology discovery and the FM election.

In particular, the *baseline* capability includes device control and status information. Fig. 6 shows part of the contents of this capability. Each register is marked with the corresponding offset inside the aperture 0. The offsets could be different to the ones shown in this figure. The first six 32-bit blocks in the *baseline* capability contain general information for the device, such as its type –endpoint or switch– and serial number, the number of ports supported, and the maximum packet size. Next (from offset 118h in Fig. 6), we can find up to 256 32-bit blocks that point to the information about each particular port in the device. This information includes link speed and width, and current port state. In the Fig. 6 we only show the information corresponding to ports 0 and 1.

3.2 PI-4 – Node Configuration and Control Protocol

A *device_manager* module in the endpoint and switch models is defined. Its function consists of receiving requests from the FM, accessing to the capabilities in the device configuration space –by means of read and write operations–, and, if necessary,

generating and injecting the corresponding responses. This interaction is implemented by means of the protocol for node configuration and control.

The protocol defines PI-4 *read request* packets to obtain information from a capability into a device. A PI-4 *read completion with data* packet is returned by the device manager to the FM, containing the requested information. The path (in the opposite direction) and the traffic class used by the response is the same as those used by the request. If the read operation was not successful, a PI-4 *read completion with error* packet is returned.

Apart from read packets, the PI-4 protocol defines *write* packets that allow to the FM to modify any data in the device configuration space. However, in this case the specification does not define a response packet.

Our model incorporates all these management packets, and the support for their transmission through the fabric. As an example, Fig. 7 shows a PI-4 *read request* packet defined in OPNET. PI-4 *read* packets start with the common AS routing header (shown in Fig. 2). In the request packet, the *Apperture* and *Offset* fields determine the position of the information in the configuration space of the destination device that the FM is requesting for. In this packet, the *Req Code* field specifies the amount of data to read (up to eight 32-bit blocks). The *Transaction Number* field allows the FM to match completions with requests. Finally, in the completion packet, the *Data Payload* field contains the requested information.

Fig. 8 shows an example. The FM –located at the endpoint *7*– repeatedly accesses the baseline capability in switch *12*, in order to obtain information about the activity of its ports. Fig. 9 shows some of the packets exchanged between the FM and the device manager during this process.

The FM sends a first PI-4 *read request* packet to the device manager in switch *12* to get general information about this device (located at *Offset*=100h in Fig. 6). The corresponding response indicates that the destination device is a switch implementing a total of 16 physical ports. Then, the FM injects two request packets (*Offset*=118h and *Offset*=138h respectively) to obtain the pointers to every port information. Each response packet will contain a block of 8 pointers.

After receiving the pointers, the FM generates sixteen new requests to access to the information about the corresponding ports. The *Link State* field in each response packet reports about the activity of the corresponding port. In this case, the state of

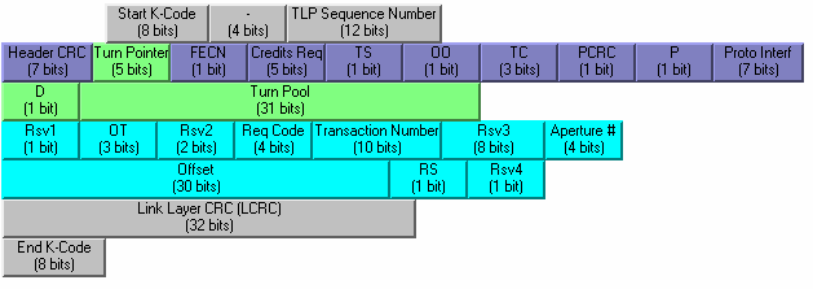


Fig. 7. PI-4 *read request* packet

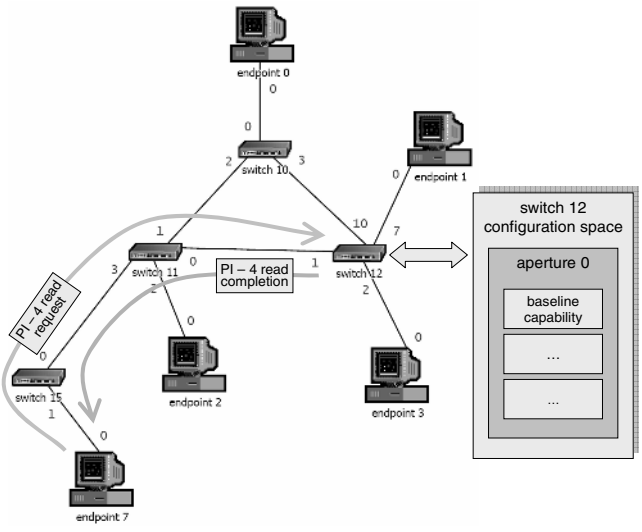


Fig. 8. Example of irregular fabric topology composed of 4 switches and 5 endpoints. Small numbers at link ends represent port numbers.

port *0* is *DL_Inactive* (we can see in Fig. 8 that it is unconnected) and the state of port *1* is *DL_Protected* (it is connected to switch *11*).

To sum it up, the FM has generated 19 PI-4 *read request* packets, and it has received 19 PI-4 *read completion with data* packets.

4 Fabric Management Tasks

The model described provides support to develop and evaluate management mechanisms for the AS technology. Network management involves a wide set of different tasks. Our work will be focused on those tasks related to network topology monitoring, computation of paths among devices, and their distribution to the source endpoints. In this section, we briefly describe the entire management process, and how it is modeled into our simulator.

As we have seen in Section 2.2, when a fabric device is powered-on, it enters to an initialization phase, exchanging credit information with potential neighbors and negotiating the available amount of virtual channels. When this negotiation concludes, it can transmit and receive management packets through its active links (*DL_Protected* state in Fig. 3).

If the device runs a FM driver, it triggers a FM election process. This process elects the primary and secondary fabric managers, the only endpoints that can configure the fabric. If the primary FM fails, the secondary one takes over. The FM election process is completely defined in the AS specification.

The first task of the (primary) FM is to discover the fabric topology. The discovery process is performed by using the PI-4 *read* packets described in the previous section. The particular implementation of this task is not detailed in the specification, and can be performed in either a centralized or distributed way [10].

```
FM sends a packet to discover a device
    Turn Pointer: 8  Turn Pool: ECh  Direction: 0  Transaction Number: 0
    Aperture: 0  Offset: 100h  Request Scale: 1  Request Code: 6
FM receives a packet from switch 12
    Turn Pointer: 8  Turn Pool: ECh  Direction: 1  Transaction Number: 0
    Next Capability Offset: 900h  CapVersion: 0  ID: F000h
    Type: SWITCH   Block Write: 1   Block Read: 1  Loopback: 1
    MVC MPS Support: 0  OVC MPS Support: 8  BVC MPS Support: 8
    BVC MPS Active: 8  OVC MPS Active: 8  MVC MPS Active: 0
    # of Ports: 16  # of Turn bits: 4  Rev Act: 0  Rev Cap: 0  Port Number: 1
FM sends a packet to obtain the pointer block 1
    Turn Pointer: 8  Turn Pool: ECh  Direction: 0  Transaction Number: 1
    Aperture: 0  Offset: 118h  Request Scale: 1  Request Code: 8
FM sends a packet to obtain the pointer block 2
    Turn Pointer: 8  Turn Pool: ECh  Direction: 0  Transaction Number: 2
    Aperture: 0  Offset: 138h  Request Scale: 1  Request Code: 8
FM receives a packet including the pointer block 1
    Turn Pointer: 8  Turn Pool: ECh  Direction: 1  Transaction Number: 1
    Port 0 Configuration Record Pointer: 10000h  AP: 0
    Port 1 Configuration Record Pointer: 10200h  AP: 0
    ...
FM receives a packet including the pointer block 2
    Turn Pointer: 8  Turn Pool: ECh  Direction: 1  Transaction Number: 2
    Port 8 Configuration Record Pointer: 11000h  AP: 0
    Port 9 Configuration Record Pointer: 11200h  AP: 0
    ...
FM sends a packet to obtain port 0 information
    Turn Pointer: 8  TurnPool: ECh  Direction: 0  Transaction Number: 3
    Aperture: 0  Offset: 10000h  Request Scale: 1  Request Code: 6
FM sends a packet to obtain port 1 information
    Turn Pointer: 8  Turn Pool: ECh  Direction: 0  Transaction Number: 4
    Aperture: 0  Offset: 10200h  Request Scale: 1  Request Code: 6
...
FM receives a packet including port 0 information
    Turn Pointer: 8  TurnPool: ECh  Direction: 1  Transaction Number: 3
    Timeout: -1  VLink: 0  MaxLink Width: 1  MaxLink Speed: 1
    Peer Link State: DL_Inactive   Training Prog: 0  TError: 0   Link Width: 1
    Link Speed: 1  Link State: DL_Inactive  Retrain Link: 0   Disable Link: 0
FM receives a packet including port 1 information
    Turn Pointer: 8  TurnPool: ECh  Direction: 1  Transaction Number: 4
    Timeout: -1  VLink: 0  MaxLink Width: 1  MaxLink Speed: 1
    Peer Link State: DL_Protected   Training Prog: 0  TError: 0   Link Width: 1
    Link Speed: 1  Link State: DL_Protected  Retrain Link: 0   Disable Link: 0
...
```

Fig. 9. Sequence of PI-4 packets to obtain topological information about switch *12* in Fig. 8

After discovery, the FM configures fabric devices. This task includes, for example, the distribution of paths to endpoints. Moreover, fabric ports must be activated in order to allow the reception and transmission of all packet types (*DL_Active* state in Fig. 3). In this case, PI-4 *write* packets are used.

Once the network has been configured and activated, the FM remains monitoring its state. The specification provides an event-reporting mechanism to notify topology changes. In particular, the device manager in a detecting device can report the FM about a change in the state of a local port, through a PI-5 packet. After detecting a change, the FM must update again the set of fabric routes.

In our model, we have defined a *FM* module at the endpoint model, which models the behavior of a centralized fabric manager. At this moment, the election process has not been modeled. We indicate the endpoint that hosts the primary FM by activating a particular node attribute. In the remaining endpoints, the *FM* module is inactive.

The *FM* module handles several data structures to store the fabric topology, the set of paths between endpoints, and other configuration information. At this moment, it

can discover the configuration information about a particular device, and detect a topological change (i.e. the addition or removal of any fabric component) by means of the event-reporting mechanism. Currently, we are developing a discovery algorithm which can obtain the entire fabric topology. Later, we will focus on the path computation and dynamic distribution tasks, without stopping upper-level traffic.

5 Conclusions and Future Work

The model presented in this paper embodies key physical and link layer features of Advanced Switching. Unlike classical simulation tools, our model incorporates the fabric management entities defined in the specification and the packets that allow the fabric manager to access to the configuration information in fabric devices. It also includes the behavior of a port upon a change in its neighbor. At this moment, a basic fabric management mechanism is being developed. As future work, we plan to improve each management task, in order to optimize the performance of the entire process. In particular, we plan to reuse previous proposals [3], and to design specific protocols for this architecture.

References

1. Advanced Switching Interconnect Special Interest Group, Advanced Switching Core Architecture Specification Revision 1.0. December 2003, http://www.asi-sig.org
2. Bermúdez, A., Casado, R., Quiles F. J., Pinkston T. M., Duato, J.: Modeling InfiniBand with OPNET. In Proceedings of the 2nd Annual Workshop on Novel Uses of System Area Networks, February 2003
3. Bermúdez, A., Casado, R., Quiles F. J, Duato, J.: Handling topology changes in InfiniBand. IEEE Transactions on Parallel and Distributed Systems (accepted for publication)
4. Duato, J., Yalamanchili, S., Ni, L.: Interconnection Networks: An Engineering Approach. Morgan Kaufmann Publishers, 2003
5. InfiniBand Architecture Specification (1.2). November 2002, http://www.infinibandta.com/
6. Mayhew, D., Krishnan, V.: PCI Express and Advanced Switching: evolutionary path to building next generation interconnects. In Proceedings of the 11th Symposium on High Performance Interconnects (HOTI'03), 2003
7. OPNET Technologies, Inc., http://www.opnet.com/
8. PCI-SIG, PCI Express Base Specification Revision 1.0a. April 2003, http://www.pci-sig.org
9. Petrini, F., Frachtenberg, E., Hoisie, A., Coll, S.: Performance evaluation of the Quadrics interconnection network. Journal of Cluster Computing, 6(2): 125-142, April 2003
10. Rooholamini, M.: Advanced Switching: a new take on PCI Express. October 2004, http://www.asi-sig.org/press/Articles/
11. Stargen, http://www.stargen.com/