

RapidIO Traffic Management and Flow Arbitration Protocol

Syed Ijlal Ali Shah, Freescale Semiconductors

Shashank Khanvilkar and Ashfaq Khokhar, University of Illinois at Chicago

ABSTRACT

The RapidIO data-streaming logical layer provides a segmentation and reassembly service to higher layers. Since a RapidIO receiver can potentially receive 64-kbyte PDUs from 64-kbyte senders, the memory requirements for successfully reassembling fragments is huge (~4 Gbytes). As RapidIO-based hardware chips have limited memory, efficient arbitration algorithms/protocols are needed that can fairly share memory among competing traffic streams, under constraints imposed by the standard. In this article we show that absence of proper arbitration can lead to deadlocks and system underutilization. We then develop a Flow Arbitration Protocol (now a part of RapidIO 2.0) as an extension to existing specification. A defining characteristic of this protocol is that memory resources can be reserved for single- and multi-PDU transfers.

INTRODUCTION

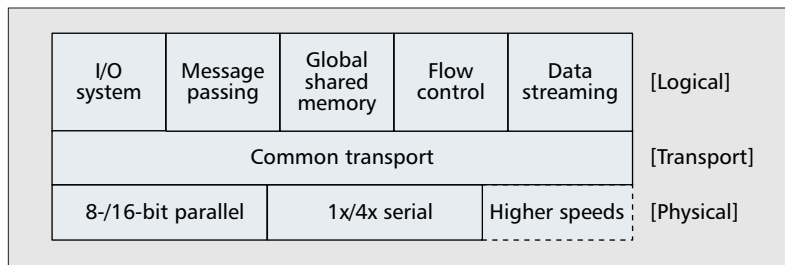
Selecting an ideal communication fabric for connecting various components within a system is perhaps one of the most important decisions made by system designers. In the past, much effort was invested in research and development of in-house communication fabrics that used proprietary communication protocols. These fabrics were costly and often required bridging devices that would convert standard protocols in to those supported by the fabric, thus adding to the overall cost of building the system. At the other end of the spectrum, Standard Interconnect technologies have been continually adding novel features. This has been the direct result of market pressures to deliver scalable, reliable and high-performance interconnects that can support applications such as IPTV and VoIP. As such, companies are now slowly moving toward adopting standard technologies as their system interconnects of choice. RapidIO is one such emerging standard that provides high-performance interconnect for chip-to-chip, board-to-board, and chassis-to-chassis communications. In this article we focus on the traffic management capabilities of RapidIO. In particular, we present a Flow Arbitration Protocol used by

RapidIO endpoints for arbitrating memory resources necessary for transferring packets. The protocol has been designed to provide complete compatibility and interoperability with existing RapidIO specifications.

Several open standard models for system interconnects do exist. For example, PCI Express (PCIe) is a high-bandwidth backward-compatible version of the popular PCI bus found in commercial PC systems [1]. Since PCI is very popular, PCIe already has a good market base. However, a major drawback of PCIe is that it has to be backward compatible with legacy PCI. As a result it is not possible to add several features that are essential in modern fabrics such as peer-to-peer communication, classes of service, and source-based routing, all of which are well defined in the RapidIO architecture. Another popular interconnect fabric mainly found in low-end cost-sensitive network devices is Ethernet. The main disadvantage here is that Ethernet is primarily developed as a LAN technology, and does not provide the reliability and traffic management functionality increasingly becoming standard requirements for communication fabrics. Other interconnect technologies that are more exclusive and targeted toward specific markets are HyperTransport, Fibre Channel, and InfiniBand.

RapidIO, initially conceived as a processor interconnect technology, has evolved in to a cost-effective, reliable, and efficient technology for communication fabrics. With the addition of the data streaming and flow control logical layers as well as enhancements to the physical and transport layers, RapidIO provides a packet-switched interconnect that can support thousands of flows across the system. Some of these enhancements include source-based routing, bandwidth management on links, multicasting, and end-to-end congestion management/avoidance.

One of the basic services supported by the RapidIO architecture (and defined in the data streaming logical layer [2]) is the transparent bridging of multiple protocols (e.g., Utopia, CSIX, SPI, and Ethernet) over a system fabric. Here, packets belonging to different protocols are segmented and encapsulated at a RapidIO



■ **Figure 1.** Three-layered protocol stack for RapidIO.

ingress endpoint (called sender), and reassembled and decapsulated at the egress (receiver). Since the complete segmentation and reassembly has to be done in hardware, this presents a unique set of problems.

Reassembly of packets at the receiver requires memory to be reserved in structures called *SAR-Contexts*, which need to be separately allocated to every flow. The size of an *SAR-Context* needs to be at least equal to the size of the original un-segmented packet. The current RapidIO specification has no provision to determine this size (*getting the packet size means parsing the packet headers, which cannot be done without penalizing the speed*), as a result of which the receiver ends up reserving just enough space to accommodate the maximum possible packet size (which is ~64 kbytes). Anything less than this can result into a deadlock situation as illustrated by the following example:

Consider a RapidIO data communication system consisting of two senders and one receiver. Let us assume that each sender sends packets that get fragmented into three segments (s_{ij} , i th sender, j th segment). Let us also assume that the receiver has memory to accommodate just 4 segments at a time. In this case, if the receiver allocates an *SAR-Context* that can accommodate 2 segments from every sender and the segments are received in the order $\{s_{01}, s_{02}, s_{10}, s_{12}, s_{03}, s_{13}\}$, a deadlock condition occurs where all memory is used but no complete packets have been reassembled.

Moreover since a receiver can potentially receive packets from 64,000 flows, the memory requirements for successfully reassembling all received packets is huge (~4 Gbytes). Most RapidIO-based hardware chips will have memory far less than this requirement, prompting the need to have efficient arbitration protocols/algorithms that can allow sharing of memory among competing traffic streams. The current data streaming specification does not have any provision for such arbitration and solely depends on simple X_{ON}/X_{OFF} congestion avoidance mechanisms defined in the flow control logical layer [3] for this purpose, causing system underutilization. Lack of an arbitration protocol also inhibits the receiver from intelligently arbitrating between different senders. This intelligent arbitration may be required to meet the service level agreements of services and applications, and prevent greedy and aggressive sources from consuming the receiver bandwidth.

To circumvent this problem, we define a Flow Arbitration Protocol as an extension to the flow control logical layer. Here, a sender has to get

explicit permission from the intended receiver before sending a packet. Such permission allows a sender to send either a single or multiple packets. We also simulated this protocol using Opnet and found high improvement in system throughput. The main challenge in designing this protocol lay in balancing the complexity of the protocol with system efficiency and throughput. The protocol is intended to be implemented in hardware with link speeds exceeding 2.5 Gb/s. At such high speeds, payload header parsing and processing of control messages becomes a tedious and costly task. We begin with a review of the RapidIO architecture with a focus on current traffic management capabilities. We provide further details on the Flow Arbitration Protocol along with analysis. We present the simulation results followed by conclusions and future work.

REVIEW OF RAPIDIO TECHNOLOGY

RapidIO is a point-to-point packet-oriented switched interconnect. As shown in Fig. 1, it has a modular three-layer architecture consisting of logical, transport, and physical layers. It is not required for compliant devices to implement all three layers. In fact, RapidIO makes a distinction between endpoint processing devices and interconnecting switching elements. While an endpoint has to implement all three layers of the protocol, the switching element only needs to implement the transport and physical layers. This delineation between endpoints and switches reduces the processing burden required to switch packets at high speeds and shifts the burden of terminating the protocol to endpoints.

The topmost logical layer defines the protocols and packet formats used for different types of communication. Several functional specifications have been defined. For example, the *I/O system* specification is used to describe the input/output operations (e.g., *read* and *write*) as a series of request/response transaction pairs using type 2 packets. Similarly, the *message passing* specification defines the protocol for passing messages between end systems (Table 1 provides a brief overview of all). Traffic management in the logical layer is provided by the data streaming and flow control specifications, described later.

The middle transport layer is responsible for routing packets from one end-point to another. Every RapidIO endpoint is assigned a unique 8- (or 16-) bit transport address used for routing. The standard itself is not biased toward any specific routing protocol, and implementers can even use direct routing for small systems. The two different sizes of the transport address allow two different scalability points to optimize packet header overhead. Thus, smaller systems with less than 256 devices can use an 8-bit transport address, saving 2 bytes in the transport header.

The bottom physical layer describes the device-level interface specifications including connector types and electrical characteristics. Separate specifications have been developed for parallel and serial interfaces having speeds ranging from 1 to 2.5 Gb/s and a roadmap to higher speeds currently being debated within the RapidIO Trade Association (RTA). The physical

Specification	Functionality
Input/output system (I/O)	Defines a rich variety of transaction types, such as DMA-style read and writes, that allow efficient I/O systems to be built.
Message passing	Defines message passing or software-based coherency functions that enable support for distributed I/O and processing requirements.
Global shared memory	Describes the standard approach for providing directory based coherent memory in a RapidIO-based multiprocessor system.
Data streaming	Defines a method for protocol independent encapsulation/SAR of payloads up to 64 kbytes and end-to-end traffic management.
Flow control	Describes optional extensions that provide X_{ON}/X_{OFF} support to provide traffic management capabilities.

■ **Table 1.** Logical layer specifications.

layer also offers two modes for packet delivery. In *reliable* mode, an error recovery protocol is used to deliver packets error-free between peers, while *continuous* mode tolerates packet loss due to congestion and bit errors during transmission. The available physical bandwidth is also divided into 16 virtual channels (VCs) with predefined priorities. VCs 0–7 (0 being the highest priority) always operate in reliable mode, while 8–15 can be configured to operate in continuous mode by the system designer.

Traffic management functionality has also been included in the physical layer in the form of two flow control mechanisms for congestion management/avoidance. In *receiver-based flow control*, no information is provided to the sender on the amount of buffering available at the receiver. The receiver accepts/rejects packets based on buffer availability, and this decision is conveyed to the sender using acknowledgments. On the other hand, the sender has complete information on the availability of receiver buffers in *transmitter-based flow control*. Such information is routinely transmitted by the receiver, piggybacked on acknowledgments.

The traffic management capabilities of data streaming and flow control specifications are discussed in more detail below.

DATA STREAMING SPECIFICATION

The data streaming logical specification defines a mechanism for transporting an arbitrary protocol over a standard RapidIO interface. Packets belonging to the protocol are encapsulated in a data streaming header (type 9) and transported transparently across the fabric. Since the maximum transmission unit (MTU) for RapidIO is 256 bytes, an encapsulation specification is defined for segmentation and reassembly of packets larger than this size. According to this specification, all incoming packets of length greater than the MTU size are segmented before being transported. The type 9 packet format used for encapsulating these segments contains 1-bit start (S) and end (E) fields that identify them as SOM ((S, E) = 10, start of message), COM (00, continuation of message), EOM (01, end of message), or FP (11, full packet).

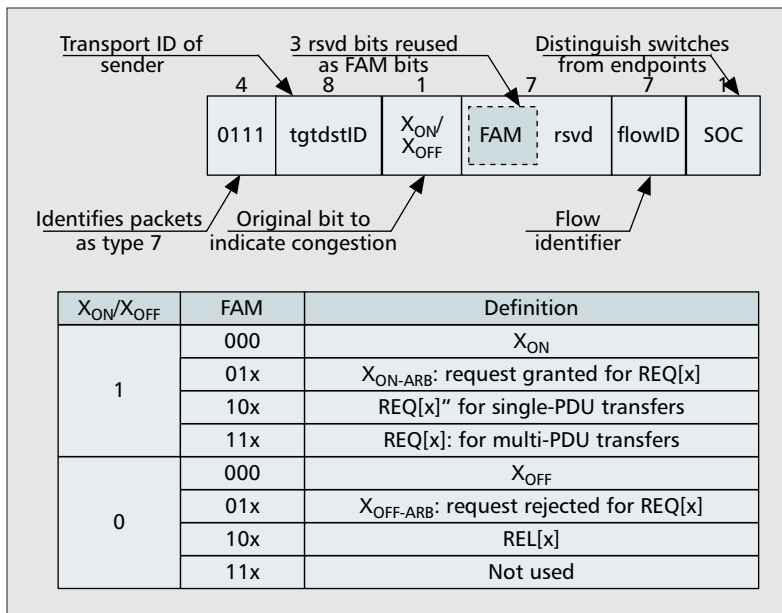
As soon as the sender starts getting a packet from an external protocol, it starts filling up the SOM segment. If the size of the packet happens to be less than the MTU size, the SOM segment is updated to represent a full packet before being transmitted. In all other cases the packets are sent as a sequence of SOM, COM, and EOM messages. The receiver, on its part, allocates a new SAR-Context as soon as it receives an SOM, and deallocates it as soon as it receives the EOM segment. The number of SAR-Contexts available at the receiver is thus one of the major factors that determine the chances of successfully reassembling a segmented packet. For example, if the receiver gets segments from N different flows and has M SAR-Contexts available, the probability of getting a free SAR-Context is M/N for $M < N$ and 100 percent for $M \geq N$. The unpredictability and randomness of access to SAR-Contexts causes the system to oscillate between periods of congestion and underload, resulting in lower system utilization.

FLOW CONTROL SPECIFICATION (TYPE 7)

The current flow control specification, also referred to as the type 7 specification, has been designed to handle transient congestion in the switches and endpoints by buffering packets and throttling the transmitting endpoints. The throttling mechanism uses a pair of X_{ON}/X_{OFF} control messages that turn sources ON/OFF, respectively. X_{OFF} messages are usually generated when the receiver buffer exceeds a certain congestion threshold and is sent to all the devices whose packets arrived at the queue after the congestion threshold was exceeded. Once the queue occupancy falls below this threshold, the receiver sends X_{ON} messages to all devices that were turned OFF.

While this type of traffic management can reduce congestion, it is inadequate for flow arbitration. The throttling mechanism is reactive and is invoked after the data segments have reached the receiver. The data segments could be either dropped, flow controlled, or accepted, depending on the availability of SAR-Contexts. This uncertainty reduces overall system throughput and does not allow the receiver to intelligently assign SAR-Contexts to senders. To solve these

FAP is a dynamic buffer allocation protocol designed to arbitrate and allocate SAR-Contexts to flows for certain periods of time. This period can be as short as the time required to transmit a single packet or could be larger to accommodate multiple packets.



■ Figure 2. Type-7 modified packet format for FAP.

issues, we define a Flow Arbitration Protocol, described next.

FLOW ARBITRATION PROTOCOL

In this section we explain the basic working of the Flow Arbitration Protocol (FAP). This protocol is being ratified by the RTA and is expected to be part of the Flow Control Specification version 2.0 (the current version is 1.3).

FAP is a dynamic buffer allocation protocol designed to arbitrate and allocate SAR-Contexts to flows for certain periods of time. This period can be as short as the time required to transmit a single packet or could be larger to accommodate multiple packets. Note that here *packet* refers to the unsegmented protocol data unit (PDU). FAP allows fewer SAR-Contexts to be efficiently shared among a large number of flows. The FAP essentially extends the current flow control logical specifications by adding the following four commands in addition to the existing X_{ON}/X_{OFF} commands:

- **REQ**: Request SAR-Context for PDU transfers.
- **X_{ON-ARB}**: Sent in response to REQ if context is available.
- **X_{OFF-ARB}**: Sent if no contexts are available. The sender is expected to resend REQ after some random time.
- **REL**: Used to indicate completion of data transfer only in the multi-PDU case.

These extra commands have been defined by extending the type 7 packet format currently used for defining X_{ON}/X_{OFF} congestion management in the flow control specification. As shown in Fig. 2, this packet format has a 7-bit reserved field, of which 3 bits have been redefined as the flow arbitration message (FAM) field. This field, along with the original X_{ON}/X_{OFF} bit, is used to define the above commands. For example, X_{ON}/X_{OFF} = 1 and FAM = 100 indicates an REQ message for single-PDU transfers. Note that the type 7 packet format has no fields to

indicate either PDU size or the number of PDUs it is willing to transmit. This is deliberate and based on the difficulty in parsing the PDU header to get to the size field, especially in high-speed devices and cut-through situations.

The FAP protocol is fairly simple and operates in two modes, as described below.

SINGLE-PDU TRANSFERS

In this mode the sender has to take permission from the receiver before sending every packet. The receiver, in turn, will grant this permission only if it has SAR-Contexts available for this purpose. The single-PDU transfer scenario is illustrated in Fig. 3a.

Here, if the transmitting endpoint has a packet to transmit, it first sends an REQ message (specifying single-PDU transfer) and waits for a response. If the receiver has sufficient resources, it responds with an X_{ON-ARB}, following which the sender can begin transmission. If no resources are currently available, the receiver may immediately send an X_{OFF-ARB}, asking the sender to try again after some random time. Some implementation may decide to use a hold queue to hold the request messages from the sender and send X_{OFF-ARB} only if the hold queue is full. For single-PDU transfers, the receiver automatically de-allocates the SAR buffers once it receives the last segment of the PDU. Thus, if the transmitter needs to send another PDU, the same procedure has to be repeated.

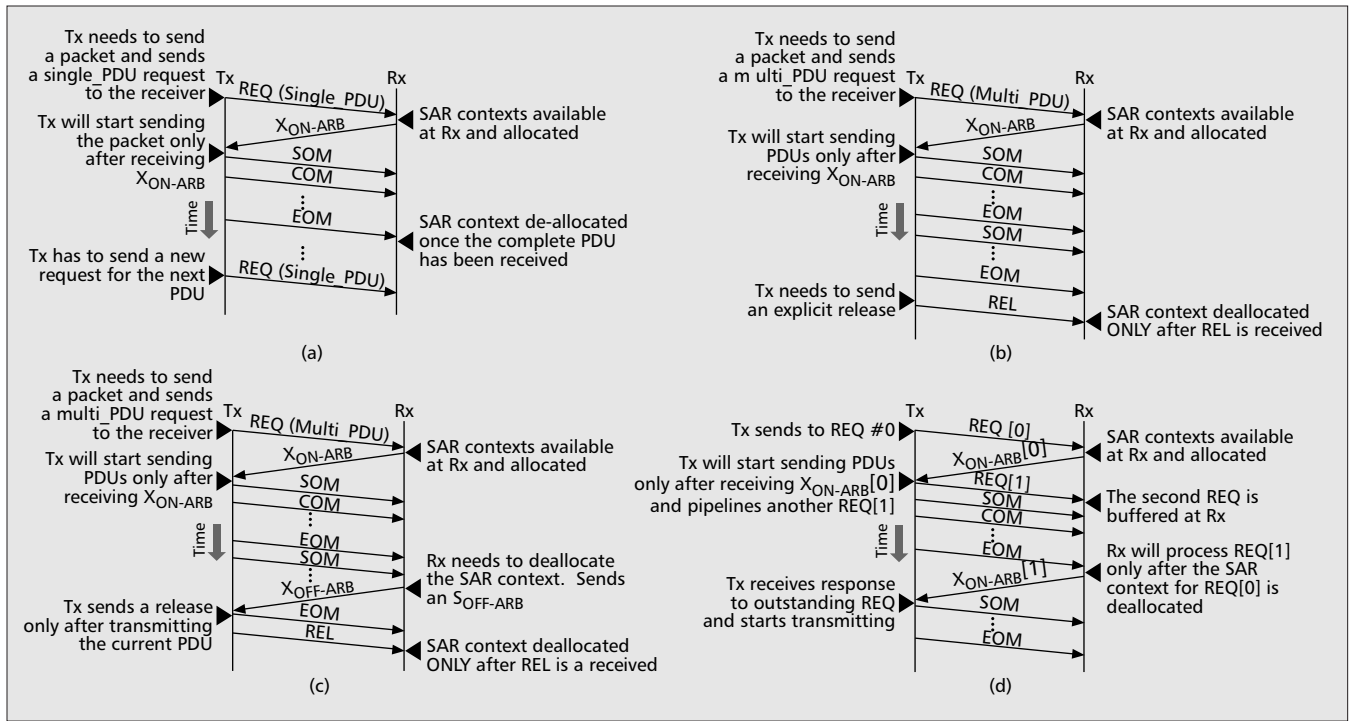
Since in the single-PDU transfer case the sender has to wait at least one round-trip time (*rtt*) before transmission can begin, it may seem that this would result in very low bandwidth utilization. However, since RapidIO architecture is designed for onboard communication fabrics, endpoints cannot be very far from each other, and the *rtt* overhead is minimal. From classical texts [4] the bandwidth utilization (*u*) for single-PDU transfers can be derived as

$$u = \frac{T_{PDU}}{2 * T_{prop_delay} + n * T_{seg}},$$

where T_{PDU} is the actual time to transmit the whole PDU (without segmentation), T_{prop_delay} is the one-way propagation delay from the sender to the receiver, and $n * T_{seg}$ is the total time to transmit n segments of this PDU. Since $T_{PDU} = n * \alpha * T_{seg}$, where α is a factor less than 1 to account for the overhead involved in segmentation, we have

$$u = \frac{\alpha}{1 + (2 * T_{prop_delay}) / (n * T_{seg})}$$

A conservative estimate for T_{prop_delay} is ~ 10 ps (assuming speed of electrons in metal as 3×10^8 m/s and maximum distance of 3 m), T_{seg} is ~ 1 μ s (assuming 2.5 Gb/s bandwidth link and segment size of 256 bytes), and α is ~ 1 (assuming segmentation overhead as contributed by the 44 bits of headers added by transport and physical layers). Thus, T_{seg} is at least 100 times higher than T_{prop_delay} , as a result of which the bandwidth utilization even for the case where there are single segments (i.e., $n = 1$) is close to α (~ 1).



■ Figure 3. a) Single-PDU transfer; b) multi-PDU transfer; c) multi-PDU transfers with release; d) pipelined request.

MULTI-PDU TRANSFERS

Although the theoretical bandwidth utilization for single-PDU transfers is quite high, it still requires a sender to send a REQ message for every packet it wishes to send. This might prove to be very cumbersome if the sender has a constant source of packets. Moreover, as high-speed physical layers are developed, the bandwidth utilization of single-PDU transfer will eventually decrease (T_{seg} will be reduced). To circumvent this, the FAP protocol has provisions for accommodating multi-PDU transfers. The timeline diagram for this is illustrated in Fig. 3b and is very similar to the single-PDU case. The main difference is when a receiving endpoint responds with an X_{ON-ARB} , the transmitter can send multiple PDUs without having to renegotiate the SAR-Context. When the sender is done transmitting, it sends an explicit REL message to indicate completion. The receiver will free the allocated SAR-Context only on receiving an REL. As before, the bandwidth utilization for multi-PDU transfers can be derived as:

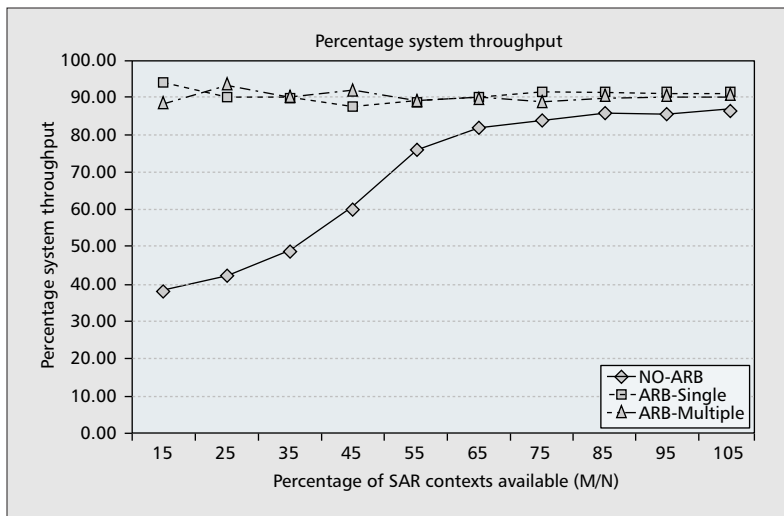
$$u = \frac{\alpha}{1 + (3 * T_{prop_delay}) / (m * n * T_{seg})}$$

where m is the number of PDUs that get transferred during a single transaction, and 3 accounts for the additional delay involved in sending the extra REL message. A disadvantage of multi-PDU transfer is that it is sender-controlled, and a receiver cannot release a resource unless the sender explicitly releases it. Under these circumstances, it is possible for an aggressive or greedy sender to starve others. To prevent this, the FAP protocol uses a feedback mechanism to prematurely stop multi-PDU transfers. This is useful if there are other flows waiting for a resource that

is held by another flow. Here, the receiver informs the transmitting endpoint of its desire to deallocate the SAR-Context by sending a premature $X_{OFF-ARB}$ message, as shown in Fig. 3c. In this case the transmitter has to send an REL message as soon as it finishes transmission of the current PDU.

In both the single- and multi-PDU transfers discussed above, the transmitting endpoint has to wait at least a round trip time after it has sent the REQ message, before it can start transmitting. This delay may be undesirable in high performance systems. To avoid this delay, FAP allows overlapping the request phase with the data transmission phase; i.e. the transmitter is allowed to have a maximum of one outstanding request in the system. The requests and the corresponding responses are identified by a 1 bit sequence number. Consider the exchange shown in Fig. 3d. The transmitting endpoint issues a REQ[0] (either for Single or Multi-PDU transfer). The request is processed by the receiving endpoint and an $X_{ON-ARB}[0]$ corresponding to REQ[0] is issued. Once the transmitter receives this message, it can start transmitting the data and it may also pipeline another request; i.e. REQ[1]. The pipelined request, however, is not honored until the SAR-Context already allocated to the transmitter has been de-allocated. In Fig. 3d, REQ[1] is sent after the transmitting endpoint has received the $X_{ON-ARB}[0]$ message for the previous request (REQ[0]). The request message is not restricted by the same bandwidth constraints as the data segments. If the receiving endpoint for some reason cannot queue/process the requests, it can send an $X_{OFF-ARB}[1]$ message immediately to indicate lack of resources.

The simple flow-arbitration mechanism has several advantages that make it better than the



■ Figure 4. Percentage improvement in system throughput.

current situation, where FAP is not used (this scenario is referred to as NO-ARB case). Below we list a few of these.

Prioritized Access — The FAP, unlike the NO-ARB case, cleanly separates the arbitration phase from the data transmission phase. This separation allows the receiver to selectively choose sources, based on some criteria set by the system designer, before data is transmitted. The FlowID field (Fig. 2) in the REQ messages identifies the transaction requesting the SAR-Context and allows the receiver to prioritize certain flows over others, on the basis of policies set by the designer. Since most systems would cater to a multitude of applications and services, including real-time applications such as VoIP and IPTV, this ability to prioritize flows (real-time or non-real-time, say) is necessary to achieve application-level service agreements across the system while maintaining high network utilization. The ability to choose sources and flows that get access to the SAR-Contexts also allows the system designer to grant access to some sources more often than others, depending on their bandwidth requirements.

Guaranteed Resource Provisioning — Consider the NO-ARB case, where sources have no prior knowledge on the state of the SAR-Context availability at the receiver before it starts transmitting the PDU. This lack of information could result in the PDU being either dropped at the receiver or flow controlled (using X_{ON}/X_{OFF}). In the first case the only recourse for the application is to recover the PDU through retransmission. Retransmitting a whole or partial PDU can consume a significant portion of the bandwidth and also cause unnecessary congestion in the system as sources retransmit dropped packets. At very high speeds, it is not difficult to imagine a scenario where sources transmit, back off, and retransmit PDUs in lockstep, causing the network to go into periods of system overload. Flow controlling the sources has another set of issues that, if not impossible, are difficult to solve. First, the receiver

Simulation parameter	Value
Number of senders (N)	20
Number of receivers	1
Available SAR-Contexts at receiver	Varied from 3 to 20
Average packet size	5000 bits
Average packet interarrival time	50 ms
MTU size	2048 bits
Max resource allocation time	50 ms
Min resource de-allocation time	25 ms

■ Table 2. Simulation parameters.

er has to make sure that enough buffering is available to buffer packets that are already in flight. Since the receiver has no knowledge of the number of in-flight packets, the only options are to either reserve enough space to accommodate all possible scenarios or apply flow control at the physical layer. None of the above possibilities are desirable, as they would either lead to overprovisioning of already limited memory resources or face a possible deadlock situation.

Fairness — The No-ARB case does not provide the receiver with the information to choose the sources to which it wants to allocate the SAR-Contexts. Thus, it is possible for a few aggressive sources to hog the receiver, denying the rest of the sources their fair share of bandwidth.

SIMULATION RESULTS

We simulated FAP using Opnet [5]. Here the RapidIO communication fabric consisted of 20 senders ($= N$) sending segmented packets to just a single receiver. The receiver was configured to contain M SAR-Contexts (with M varying from 3 to 21). All the senders were configured to send at the same data rate so that the receiver was mildly congested. Other relevant simulation parameters have been provided in Table 2.

Figure 4 illustrates the system throughput for three cases: NO-ARB: no flow arbitration is used; ARB-Single: senders use only single-PDU transfer mode; and ARB-Multiple: senders use multi-PDU transfer mode. The x-axis represents the ratio of the number of SAR-Contexts to the number of senders. As seen, the system throughput for NO-ARB is much lower for low values of M and increases as more SAR-Contexts become available. This is quite expected, since with a low value of M , the probability of getting a free SAR-Context decreases and packets get dropped. FAP, on the other hand, shows consistent system throughput (in both cases) even for small M . The single-PDU and multi-PDU transfers have the same throughput because of the low value of propagation delay between the two endpoints, as explained earlier. Since in FAP the sender does not send packet unless it gets per-

mission from the receiver, no packets are dropped. Figure 5 shows the corresponding packet drops for all three cases.

Another way of interpreting the results shown in Fig. 6 is to consider the average number of retransmissions required in the no-loss case; that is, the number of times higher-layer protocols retransmit a PDU when the PDU is lost or dropped during the previous transmission. Let the number of such retransmissions be represented by X and the probability of dropping a packet be p . In this case the expected number of retransmissions is simply given as

$$E[X] = \sum_{i=1}^N i * (p^{i-1} * (1-p)) = \frac{1}{(1-p)}$$

The loss probability is the same as the percentage of packets lost. Thus, the expected number of retransmissions decreases (from 1.67 down to 1) as the number of available SAR-Contexts increase from 15 to 100 percent. A side-effect of using FAP in the case where lost packets are not retransmitted, however, is increased average packet delay. This is mainly due to the increase in queuing delay faced by the packets at the senders. This delay, however, is not present in the NO-ARB case.

Figure 6 shows the average wait time experienced by senders while sending packets in the single- and multi-PDU cases. Multi-PDU shows slightly better results than single-PDU, because fewer requests are transmitted.

CONCLUSIONS/FUTURE WORK

In this article we present the Flow Arbitration protocol for reserving SAR-Contexts at the receiver. Although the protocol is developed for RapidIO and uses a RapidIO-specific messaging format, it is generic enough to be used by any protocol that allows encapsulation of packets. FAP is simple and consists of only four additional messages over type 7 flow control messages. Our simulation results indicate that FAP improves system performance significantly over the NO-ARB case. It also allows the system designer to maintain service- and application-level service level agreements across the fabric and endpoint processing units, and prevents aggressive sources from hogging system resources at the receiver.

More work needs to be done in researching resource allocation/buffer management schemes (conventional schemes may not be suitable for such high-speeds), which can improve system performance, while having a small memory footprint. Also, interaction between FAP with traffic management is another interesting problem that needs a second look. We plan to explore these problems in future research.

REFERENCES

- [1] White Paper, "RapidIO Technology and PCI Express — A Comparison," http://www.rapidio.org/education/documents/RapidIO-PCI-Ex_WP_v02.pdf
- [2] RapidIO Specs. 1.3, Data Streaming Logical Layer, Phase I.
- [3] RapidIO Specs 1.3, Flow Control Extensions.
- [4] W. Stallings, *Data and Computer Communications*, 5th ed., Prentice-Hall, 1998.
- [5] Opnet Technologies, Network Simulator, <http://www.opnet.com>

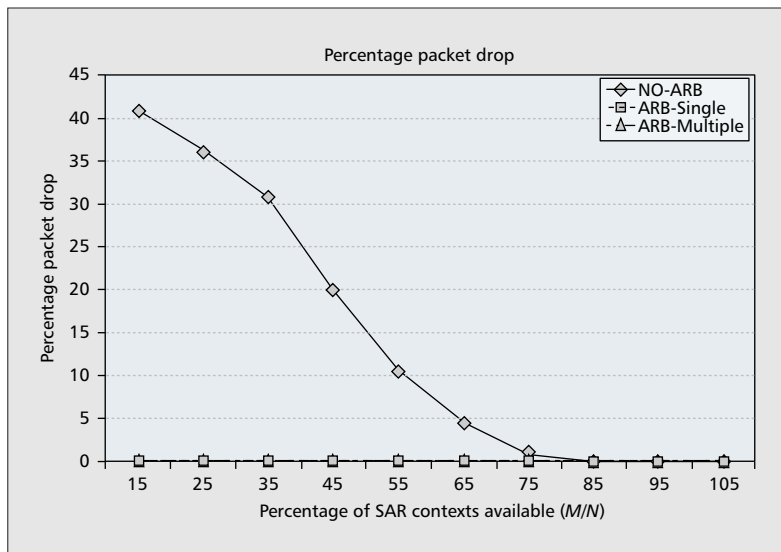


Figure 5. Percentage packet drop.

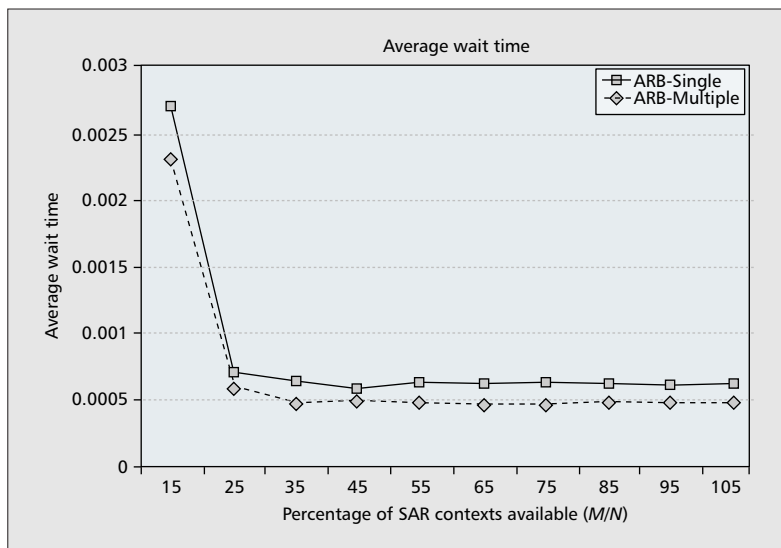


Figure 6. Average wait time.

BIOGRAPHIES

SYED SHAH (syed.shah@freescale.com) has more than 12 years of experience in the telecommunications and data-com industry. As a data path architect for Freescale Semiconductor's Digital Systems Division, he has defined and architected Freescale's traffic management co-processor, the RapidIO Fabric, and has been a key contributor in defining and architecting several other network-related projects. Prior to Freescale/Motorola, he was an associate professor at Lahore University of Management Sciences (LUMS). Prior to LUMS he was at Nortel, where he helped define and architect packet-switching products, and QoS and network dimensioning methodologies. He holds a Ph.D. and an M.S. in electrical engineering from Columbia University and a Master's in engineering management from the University of Ottawa. He holds three patents on call admission control algorithms for switches/routers and dynamic IP/ATM congestion management with several more pending. He has presented on IP/ATM congestion control and traffic management at various conferences and invited sessions, and has published in numerous conference and journal publications on various aspects of QoS, network design and architecture, and all-optical networks. He is also Traffic Management Editor for the RapidIO Trade Association. His research interests include sensor network design, QoS in IP networks, optical networks, and wireless networking.

SHASHANK KHANVILKAR (skhanv1@uic.edu) received his M.S. and Ph.D. degrees in computer engineering from the University of Illinois at Chicago in 2002 and 2006, respectively. He received his Bachelor's in electronics from Ramrao Adik Institute of Technology, University of Mumbai, in 1997. During 1997–1999 he worked as a software engineer for Patni Computer Systems, India. His research interests include networking, multimedia and security.

ASHFAQ A. KHOKHAR (ashfaq@uic.edu) received his Ph.D. in computer engineering from the University of Southern California in 1993. He spent two years as a visiting assistant professor in the Departments of Computer Science, and Electrical and Communications Engineering at Purdue University. In 1995 he joined the Electrical and Communications Engineering Department of the University of Delaware. In fall 2000 he joined the University of Illinois

at Chicago, where he currently serves as a professor in the Departments of Computer Science, and Electrical and Communications Engineering. He has published over 100 technical papers and book chapters in refereed conferences and journals in the area of wireless networks, multimedia systems, data mining, and high-performance computing. He is a recipient of the NSF CAREER award in 1998. His paper entitled "Scalable S-to-P Broadcasting in Message Passing MPPs" won the Outstanding Paper award at the International Conference on Parallel Processing in 1996. He has served as Program Chair of the 17th Parallel and Distributed Computing Conference, 2004, Vice Program Chair for the 33rd International Conference on Parallel Processing, 2004, and General Chair of the Workshop on Frontiers of Information Technology, 2004. His research interests include: wireless and sensor networks, multimedia systems, data mining, and high-performance computing.

IEEE COMMUNICATIONS MAGAZINE CALL FOR PAPERS FEATURE TOPIC: WIRELESS MESH NETWORKS

Next generation communication networks are facing the challenge of providing adaptive, flexible, and reconfigurable architectures capable of catering to the dynamics of the network, while providing cost effective solutions for service providers. Wireless mesh networking (WMN) has emerged as a promising concept to meet such challenges of next generation networking. In WMN, no cabling is needed to connect the routers together. All routers self-organise and auto-configure themselves wirelessly, and form a rich radio mesh connectivity among themselves that is difficult to provision in wired networks. While wireless router connectivity significantly reduces the up-front deployment and subsequent maintenance costs, the rich mesh connectivity helps delivering high level of reliability and robustness. Due to these attractive features, WMN is being considered for a wide variety of application scenarios such as backhaul connectivity for cellular radio access networks, high-speed metropolitan area mobile networks, community networking, building automation, intelligent transport system networks, defence systems, and city-wide surveillance systems. Despite significant advances in the last few years, many challenging research issues, such as sharp drop in available throughput as the number of wireless hop increases, remains to be resolved. Industry, academia, and standard bodies are all actively working together on bringing this technology to its maturity.

The aim of this Feature Topic is to feature the recent advances in theory and application of wireless mesh network. Articles from both academia and industry are solicited. Topics of interests include, but are not limited to:

- Throughput enhancing techniques for paths with many wireless hops
- Latency reduction techniques
- Multipath routing in wireless mesh networks
- Advanced antenna technologies
- Manageability and interoperability issues
- Self organization and network configuration
- Test-bed design, experimental results, implementation and prototypes
- Novel application and deployment scenarios
- Standardisation activities (e.g. IEEE 802.11s)
- Security architectures and protocols for wireless mesh networks
- Mobile mesh networks

SUBMISSION GUIDELINES

Articles should be tutorial in nature and written in a style comprehensible to readers outside the specialty of the article. Articles may be edited for clarity and grammatical accuracy, and will be copyedited according to the Magazine's style. Complete guidelines for prospective authors can be found at www.comsoc.org/pubs/commag/sub_guidelines.html. All articles to be considered for publication must be submitted through IEEE Manuscript Central (<http://commag-ieee.manuscriptcentral.com>).

SUBMISSION SCHEDULE

Manuscripts Due: August 01, 2006
Acceptance Notification: November 01, 2006
Final Revised Manuscripts Due: January 01, 2007
Manuscripts to Publisher: March, 2007
Publication Date: May 2007

GUEST EDITORS

Mahbub Hassan
School of Computer Science and Engineering
University of New South Wales and National ICT Australia
Email: mahbub@cse.unsw.edu.au

Prasant Mohapatra
Department of Computer Science
University of California, Davis
Email: prasant@cs.ucdavis.edu

Sajal K. Das
Department of Computer Science and Engineering
University of Texas at Arlington
Email: das@cse.uta.edu

Charles E. Perkins
Communication Systems Laboratory
Nokia Research Center
Email: charles.perkins@nokia.com