WHITE PAPER

Speculative Forwarding: An Implementation for Reduced Latency in HyperTransport™ Technology-Based Systems

Jose Duato
Universidad Politecnica de Valencia, Spain,
Simula Research Laboratory, Norway

Federico Silla
Universidad Politecnica de Valencia, Spain

Brian Holden
Vice President and Chair, Technical Working Group
HyperTransport Technology Consortium

Executive Summary

HyperTransport interconnect technology has become a highly pervasive technology due to its ability to provide a high-performance, point-to-point link with the lowest communication latency for chip-to-chip and board-to-board interconnects. This white paper discusses a new method for further reducing latency in systems based on the HyperTransport interconnect technology standard. It describes the advantages and disadvantages of speculative forwarding, an alternative way of designing HyperTransport devices that could significantly reduce latency in long chains without requiring changes to the HyperTransport Link Specification.

Introduction

In order to understand the advantages of speculative forwarding, it is important to first review how HyperTransport systems are architected and how links traditionally operate. HyperTransport technology-enabled devices include processors, which are referred to as hosts; tunnels, which implement two HyperTransport link interfaces and provide connectivity to HyperTransport peripherals, as well as forward traffic to and from other daisy-chained HyperTransport peripheral controllers in the system; and bridges, which are used to logically “bridge” the HyperTransport link with peripherals complying with other interconnect standards.

HyperTransport links are bidirectional, consisting of two separate unidirectional sets of signals that connect two HyperTransport devices in point-to-point fashion. Each unidirectional set of signals can be 2, 4, 8, 16, or 32 bits wide. Additionally, each set contains one or more CTL (Control) signals used to differentiate control and data transmission, and one or more CLK (Clock) lines for clocking packet transmission. Refer to Section 2 of the HyperTransport I/O Link Specification Revision 3.0 for a complete description of HyperTransport signalling.

Communication among HyperTransport devices is carried out by exchanging packets. Every time a HyperTransport device needs to communicate with another HyperTransport device, it creates a packet that contains the command or data to be transmitted. Command packets may include read and write requests or responses; atomic read-modify-write commands; and flush, fence and target-done commands intended for better controlling the transactions in the link. When one of these

commands requires some data to be transferred, a second packet containing the data is sent to the link, following the corresponding command packet.

In the HyperTransport protocol some command packets, like requests, are 8 bytes long, while other command packets, such as responses, fence, or flush packets, are 4 bytes long. Packets can be extended by using an address extension, a source identifier extension, or both. Since packet extensions are always 4 bytes long, the length of command packets exchanged by HyperTransport devices may be 4, 8, 12, or 16 bytes, depending on the type of request and the number of extensions added to the packet.

In order to provide the HyperTransport protocol with robustness against transmission errors, a 32-bit Cyclic Redundancy Code (CRC) is appended to packets for protection. If a packet's corresponding CRC reports a transmission error, the packet is immediately discarded. Note that this feature was first implemented based on the latest HyperTransport 3.0 specification, although it existed in earlier HyperTransport revisions also. All revisions have a link-based CRC as well that is not aligned with the packets. Refer to Section 10 of the HyperTransport I/O Link Specification Revision 3.0 for more details.

A HyperTransport system consists of one or more daisy chain interconnected HyperTransport devices. In the simplest connection scheme, only one daisy chain is implemented. In this case, the chain extends from the host interface at one end, to a cave device at the other end. A cave is a HyperTransport link termination device that implements only one link interface. Between the host and the cave there may be zero or more tunnels providing connectivity to peripherals. The maximum number of devices interconnected in a single HyperTransport chain is 32. Figure 1 shows an example of such a chain containing a host, a couple of tunnel devices, and a cave device.

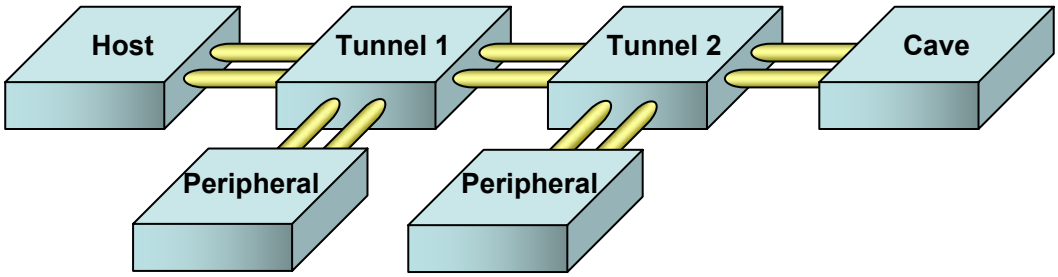


Figure 1: HyperTransport fabric composed of a host, two tunnels, and one cave

Basic HyperTransport Device Behavior

As mentioned above, HyperTransport devices communicate by exchanging packets. When a device needs to send a packet to another device, it puts it onto the link. After some wire propagation delay, the first packet slice arrives at the next device in its path towards the final destination. As the slices of the packet arrive (the link could be 2, 4, 8, 16, or 32 bits wide), they are temporarily stored in an input buffer until the whole packet has been received. At that moment, the receiving device compares the received per-packet CRC for that packet with the one that it has computed in order to check if the packet was received correctly or otherwise. If the reception was error-free, the device analyzes the packet header and decides whether to process the packet or forward it to the next device on the link. If the current device is not the packet's intended destination, it forwards the packet towards its destination by passing it onto the next link. This process repeats as many times as point-to-point links exist between the source and the intended target device.

The way HyperTransport devices store arriving packets and later forward them is traditionally known as packet switching. Packet latency exhibited by this switching technique is proportional to both packet length and path length. Thus, if packet length is *n* bytes (including per-packet CRC), link width is *w* bits, and path length is *d* hops, then the overall packet latency from source to destination will be given by the formula:

$$L = d * (n * \frac{8}{w} + t_{wire} + t_{proc})$$

where t_{wire} represents the propagation delay along the wire and t_{proc} is the processing time required by the device to check the per-packet CRC and analyze whether the packet must be forwarded or not. Table 1 shows packet latency for several path lengths and link widths.

	Link Width				
Path Length	2	4	8	16	32
5	165 245 405	85 125 205	45 65 105	25 35 55	15 20 30
10	330 490 810	170 250 410	90 130 210	50 70 110	30 40 60
20	660 980 1620	340 500 820	180 260 420	100 140 220	60 80 120
30	990 1470 2430	510 750 1230	270 390 630	150 210 330	90 120 180

Table 1: Packet latency for several link widths and path lengths. Numbers in red, green, and blue represent latency for 4, 8, and 16-byte packets respectively. A 32-bit per-packet CRC has been considered for computing latencies. Latency is measured in clock cycles. Path length is measured in hops. Propagation delay along the link and processing time at the tunnels has been assumed to be zero and one clock cycle, respectively.

The basic device behavior described above is the simplest possible implementation. However, in order to reduce packet latency, two improvements may be simultaneously applied to the design. The first improvement is the use of cut-through switching instead of packet switching. The second improvement is the use of speculation.

Cut-through Device Design

With cut-through switching, packets are forwarded before they are completely received. In this case, once the packet header is received and decoded, and the output port is computed, the packet is forwarded through that output port at the same time it is being received through the input port. This behavior is described in Section 10.3.3 of the HyperTransport I/O Link Specification Revision 3.0. Effectively, a tunnel may begin forwarding a packet before validating it in the case when the packet is a valid command, and the per-packet CRC for that packet can be checked before sending the per-packet CRC on the next link. In this case, if the packet being received presents no transmission

error, once the per-packet CRC has been checked, it is forwarded onto the next link. Typically, in the event of a transmission error, the packet would be discarded. However, since the packet has already been forwarded, it is impossible to discard it. Therefore, in order to signal a transmission error, the packet is stomped by sending the inverse of the correct per-packet CRC.

This cut-through implementation - commonly used in HyperTransport-based designs - reduces packet latency because the packet must not be completely received before being forwarded. However, the first bytes of the packet containing the destination identifier must be received before forwarding the packet in order to compute the output port. This is required because devices have two possible outputs for an incoming packet: if the packet is destined for the device, then it is accepted and no forwarding is performed. Instead, if the current device is not the destination of the packet, it is forwarded. Note that waiting for the destination identifier to arrive is only necessary for control packets but not for data packets, as the latter immediately follow the former and the destination for the latter is specified in the preceding control packet.

The destination identifier for a control packet is not always located at the same position in the packet. Thus, depending on the type of control packet, one, two, or more bytes must be received before being able to compute the output port for the packet. Table 2 shows the number of bytes that must be received before computing the output port for the different types of control packets. Accordingly, Table 3 shows packet latency for several path lengths and link widths. In this table, 16-byte packets are not shown because their latency is the same as in Table 1.

Packet length	Packet type	Bytes required before forwarding
4	Flush	1
	Fence	1
	Target Done	2
	Read/Write Response	2
8	Broadcast	1
	Read/Write Request	8
	Atomic RMW	8
12	Packet with address extension	12
	Packet with source ID extension	Depends on control packet
16	Packet with both source ID and address extension	16

Table 2 – Number of bytes that must be received before computing the output port for the different types of control packets

	Link Width				
Path Length	2	4	8	16	32
5	53 69 69	29 37 37	17 21 23	13 13 15	11 11 12
10	78 114 94	44 62 52	27 36 33	23 23 25	21 21 22
20	128 204 144	74 112 82	47 66 53	43 43 45	41 41 42
30	178 294 194	104 162 112	67 96 73	63 63 65	61 61 62

Table 3: Packet latency for several link widths and path lengths. Numbers in red and black represent latency for 4-byte packets that require 1 or 2 bytes, respectively, to be received before being forwarded. Numbers in green represent latency for 8-byte packets that require 1 byte to be received before beginning forwarding. 32-bit per-packet CRC has been considered for computing latencies. Latency is measured in clock cycles. Path length is measured in hops. Propagation delay along the link and processing time at the tunnels has been assumed to be zero and one clock cycle, respectively.

In general, when using cut-through switching in the context mentioned above, packet latency from source to destination can be computed using the following formula:

$$L = d * (\max(1, \frac{8 * h}{w}) + t_{wire} + t_{proc}) + \frac{8}{w} * (n - \max(h, \frac{w}{8}))$$

In this formula, packet length is *n* bytes (including per-packet CRC), link width is *w* bits, and path length is *d* hops. Additionally, t_{wire} represents the propagation delay along the wire and t_{proc} is the processing time required by the device to analyze whether the packet must be forwarded or not. On the other hand, *h* is the number of bytes to be received before forwarding the packet. This parameter is the same as the one presented in the right-most column in Table 2. Comparing this formula with the one for packet switching presented previously, one can see how distance and packet length are combined in an additive manner instead of in a multiplicative way, thus noticeably lowering latency.

According to the numbers in Tables 2 and 3, cut-through switching is expected to significantly reduce latency in cases where read responses and write requests carry data. In these cases, once the first 2 or 8 bytes of the packet have arrived (for read responses and write requests, respectively), the device can start forwarding both the control packet and the associated data packet. In the case of flush and fence control packets, as their length is only 4 bytes and, moreover, 2 bytes must be received before forwarding the packet, latency will not be so noticeably reduced since an important part of the packet must be received before starting to forward it. This is even worse in cases where the destination identifier is located in such a way that the whole packet must be received before

starting to forward it, as in the cases for read requests, atomic RMW, or packets with source and address extensions.

Speculative Device Design

Cut-through switching is based on forwarding the packet before it is completely received. In this way, latency can be drastically reduced for packets carrying data. However, cut-through switching does not help much when packets do not contain data. Fortunately, we can reduce latency even more by observing that in cut-through switching the packet is validated and the device is intrinsically speculating. This is because a received per-packet CRC could indicate a transmission error, thereby revealing that the packet should have never been forwarded. Moreover, Section 10.3.3 of the HyperTransport I/O Link Specification Revision 3.0 implicitly introduces another kind of speculation, much more interesting from the latency point of view. The only condition required by the specification is that packets must be valid commands to be eligible for speculative forwarding. Since the command field arrives before the destination identifier, it is possible to forward a packet as soon as the command field has been completely received. In this case, the destination of the packet is not known by the time the packet is being forwarded. Therefore, a device might speculate on the correctness of the received packet at the same time that it speculates on the output port for that packet. As with cut-through designs, if output port speculation is wrong and the packet is erroneously forwarded because the current device is actually the packet destination, the only solution is for the current device to stomp the packet in order to signal such routing mistake.

Speculatively forwarding a packet before the output port is computed by the routing logic is only useful if the probability of selecting the correct output port is very high. For most topologies, that probability is relatively low. However, for a linear array, as in the case of HyperTransport chains (assuming that speculation is not implemented for the secondary port of a bridge), such probability is very high. In fact, a non-cave device in a HyperTransport chain must always forward a packet arriving at one of its ports to the other port unless the device is the packet's destination. It is this high probability that makes speculation truly useful for reducing latency when there are several devices in the chain.

As mentioned above, packets must be valid commands to be eligible to be speculatively forwarded. As the command field appears in the first byte of the packet, receiving the first byte is enough to start forwarding the packet. Thus, this speculative implementation is especially useful for links equal to or wider than 8 bits because the command field is received in the first slice of the packet. However, for 2- or 4-bit wide links, the receiving device cannot begin forwarding the packet until 4 or 2 slices of it, respectively, have arrived. Table 4 presents packet latency for the same path lengths and link widths as Table 1. Numbers in Table 4 can be also compared with the ones in Table 3. In this case, please note that Table 3 only shows latencies for 4 and 8-byte packets, while Table 1 and Table 4 also show latencies for 16-byte packets.

The data in these tables illustrates how this technique effectively reduces packet latency. In fact, if packet length is *n* bytes (including per-packet CRC), link width is *w* bits, and path length is *d* hops, then the overall packet latency from source to destination will be:

$$L = d * (\max(1, \frac{8}{w}) + t_{wire} + t_{proc}) + \frac{8}{w} * (n - \max(l, \frac{w}{8}))$$

	Link Width				
Path Length	2	4	8	16	32
5	53 69 101	29 37 53	17 23 29	13 15 19	11 12 14
10	78 94 126	44 52 68	27 33 39	23 25 29	21 22 24
20	128 144 176	74 82 98	47 53 59	43 45 49	41 42 44
30	178 194 226	104 112 128	67 73 79	63 65 69	61 62 64

Table 4 – Packet latency for several link widths and path lengths. Numbers in red, green, and blue represent latency for 4, 8, and 16-byte packets, respectively. 32-bit per-packet CRC has been considered for computing latencies. Latency is measured in clock cycles. Path length is measured ir hops. Propagation delay along the link and processing time at the tunnels has been assumed to be zero and one clock cycle, respectively.

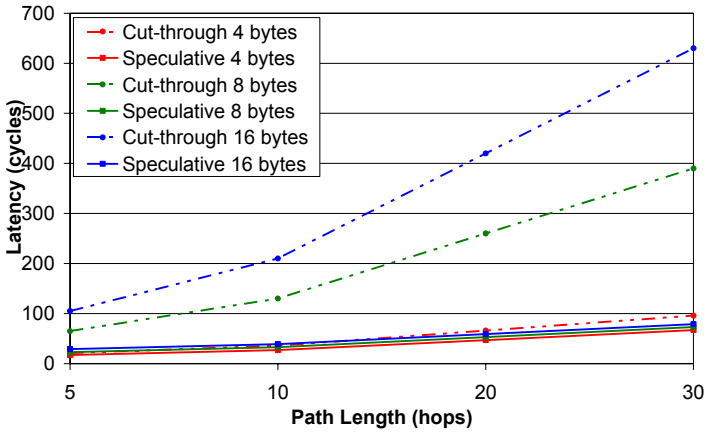
where t_{wire} represents the propagation delay along the wire and t_{proc} is the processing time required by the device to set the output port. Additionally, the formula above assumes that the command field is located in the first byte of the packet (which is the case in HyperTransport), thus being necessary to receive just the first byte of the packet to start forwarding it. If this formula is compared with the one for cut-through presented in the previous section, it can be seen that both of them are very similar, the only difference being the removal of the factor *h* in the last formula, due to the fact that, in this case, it is only necessary to receive the first byte of the packet, containing the command field. Indeed, this difference is the essence of speculation as it has been presented in this section.

In the case for each of the different link widths defined in HyperTransport, the formula can be simplified by making the max terms constant. For example, for a 4-bit wide link, the formula would be:

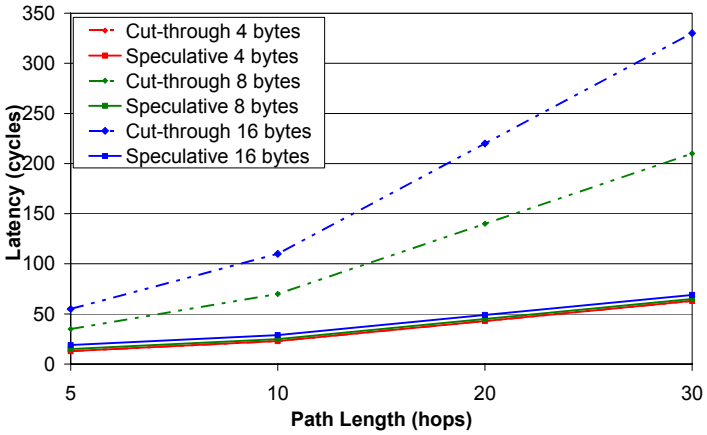
$$L = d * (2 + t_{wire} + t_{proc}) + 2 * (n - 1)$$

Figure 2 merges numbers in Tables 1, 2, 3, and 4 graphically, for 8-bit and 16-bit wide links. Note that in the case of 4-byte packets, numbers refer to the case when it is required to receive the first two bytes of the packet before forwarding it. In the case of 8-byte and 16-byte packets, the number of bytes required is 8 and 16 bytes, respectively. This choice may seem to be unfair, however, remember that when using cut-through, some types of packets require that the whole packet is received before starting to be forwarded. Figure 2 is intended to point this out.

Ultimately, HyperTransport specifications allow speculative devices to coexist with cut-through devices. When different devices work together, the simplest ones will manage stomped packets as if they had suffered a transmission error and will discard them, therefore limiting the effects of cut-through and/or speculation.



a)



b)

Figure 2: Packet latency comparison for cut-through and speculative implementations. a) 8-bit wide link. b) 16-bit wide link.

Speculating One Step Beyond

The idea behind speculatively forwarding a packet before having completely received and validated it can be extended further in order to use a more general and aggressive model of speculation. Instead of waiting for the reception and processing of the command field before starting to forward a packet, speculation can be generalized so that a packet is forwarded as soon as its first slice has been received. By first slice, we mean the first piece of information, which could be 2, 4, 8, 16, or 32 bits depending on link width.

Such generalization reduces latency with respect to the model presented in the previous section for cases in which receiving the first byte of the packet takes several clock cycles. More precisely, when link width is 2 or 4 bits, a greater reduction in latency is achieved than in the previous case. For 8, 16, and 32-bit links, latency remains the same as in the previous case.

Using this more aggressive speculation, packet latency will be given by the formula:

$$L = d * (1 + t_{proc} + t_{wire}) + \frac{8}{w} * n - 1$$

As indicated above, *d* is the path length in hops, *w* is link width, and *n* is the packet length. The formula shows that packet length and path distance are not directly combined, therefore lowering latency as in the previous section. Additionally, latency is only proportional to path length without taking into account the amount of time required to receive the first byte.

	Link Width				
Path Length	2	4	8	16	32
5	41 57 89	25 33 49	17 23 29	13 15 19	11 12 14
10	51 67 99	35 43 59	27 33 39	23 25 29	21 22 24
20	71 87 109	55 63 79	47 53 59	43 45 49	41 42 44
30	91 107 139	75 83 99	67 73 79	63 65 69	61 62 64

Table 5 – Packet latency for several link widths and path lengths. Numbers in red, green, and blue represent latency for 4, 8, and 16-byte packets, respectively. 32-bit per-packet CRC has been considered for computing latencies. Latency is measured in clock cycles. Path length is measured in hops. Propagation delay along the link and processing time at the tunnels has been assumed to be zero and one clock cycle, respectively.

Table 5 shows packet latency for several path lengths and link widths. Numbers for link widths equal to 8, 16, and 32 are the same as in Table 4, as explained above. They are presented in the table just for the sake of completeness. Also, Figure 3 (in the next page) shows a comparison between both speculation techniques for 2 and 4-bit links.

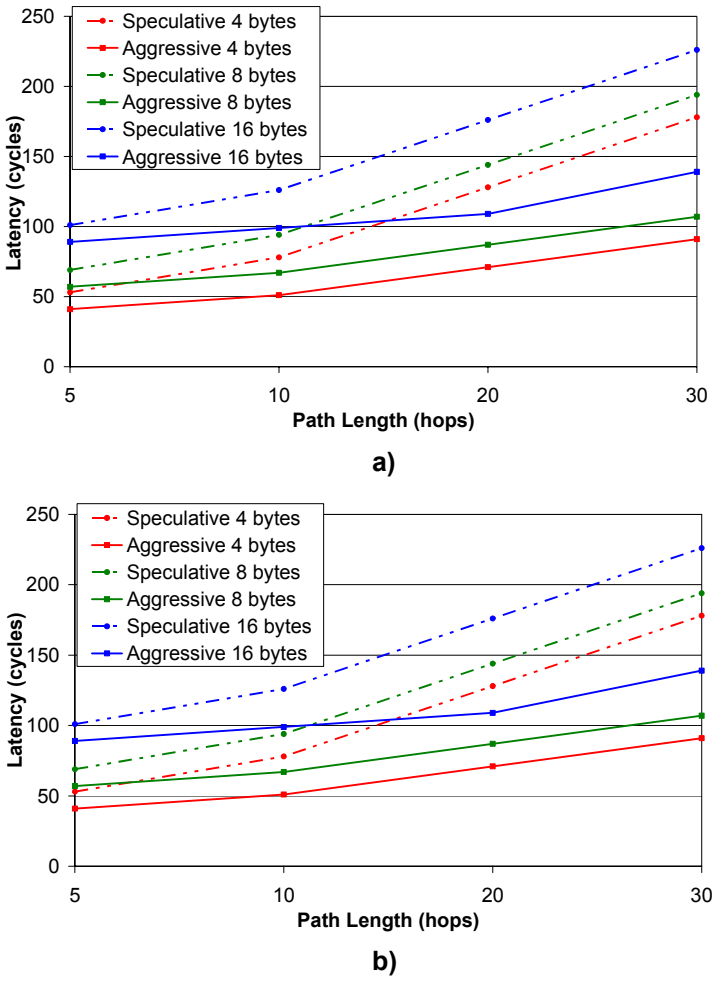


Figure 3 – Packet latency comparison for the legacy speculative forwarding and the more aggressive implementation. a) 2-bit wide link. b) 4-bit wide link.

Other Considerations

Speculation reduces packet latency at the expense of wasting some bandwidth in the chain. A device forwards a packet before knowing whether or not it is the packet destination. Since the packet is already travelling along the next link before this is known, it is impossible to discard it. At that point, the only thing that the device can do is to stomp the packet, so that when it reaches a non-speculative device, it is discarded. In the worst case, this non-speculative device could be the host or the cave, and therefore the packet would have been forwarded along the whole chain wasting resources. This is not really as bad as it seems. If DirectRoute is used (refer to Section 4.9.6 of the HyperTransport I/O Link Specification Revision 3.0 for a complete description of peer-to-peer traffic) then most of the traffic will be statistically concentrated in the central part of the chain, leaving the ends of the HyperTransport chain less congested. In this case, the extra bandwidth used at the ends of the chain by speculatively forwarding packets introduces no overhead (except for power consumption) because such extra bandwidth would have unlikely been used due to the nature of traffic, unless a peer-to-peer communication takes place at the very end of the chain. On the other hand, if DirectRoute is not used, then Host-Reflected routing must be used. In this case, at the host

side of the chain no bandwidth is wasted because all the packets must arrive at the host. At the opposite end, the use of extra bandwidth is not a problem because that bandwidth in the downstream direction would have never been used otherwise - extra power consumption being the only consideration. When mixing DirectRoute with HostReflected traffic, the central part of the chain will experience most of the traffic while the ends will be less congested. Thus, central part of the chain is the natural system bottleneck, limiting overall performance. In this context, the extra bandwidth used at the ends of the chain will not degrade performance.

Speculatively forwarding packets, in any of the two versions presented in this white paper, is just an alternative way of designing HyperTransport devices that could significantly reduce latency in long chains. Although speculative forwarding envisages a different HyperTransport device implementation, it does not require any change in the HyperTransport Link Specification, which even implicitly introduces this feature. Furthermore, devices implemented using these techniques can be directly connected to and transparently interoperate with non-speculative devices. In this case, packets would be quickly forwarded along the sub-chain of speculative devices until they reach a non-speculative one, where the packet (or at least, its header) would be stored, validated, and later forwarded. If efficiently planned, such intermixing of devices may help to discard stomped packets, thus avoiding wasted bandwidth. A trade-off between latency, bandwidth, and power consumption may be achieved by conveniently locating non-speculative devices near the ends of the chain, so that they may stop stomped packets and prevent them from advancing further in the chain.

On the other hand, speculative forwarding might be treated as a programmable feature, so that programmers could select the way HyperTransport devices behave. This could be achieved by using a device-specific control bit - a common practice in the HyperTransport ecosystem. This is possible by means of extra control bits available in the device's control registers that are not controlled by the HyperTransport specification and that are commonly used to handle these type of extra features. In this way, if the behavior of the application is known, then the desired behavior for each HyperTransport device could be programmed so that parts of the chain do forward packets speculatively while other devices do not, thereby acting as walls that split packet traffic in accordance with different latency/bandwidth requirements.

Finally, it should be noted that speculative forwarding is one example of reversible behaviors that are often useful for system optimization. A similar method can be employed in the endpoint as well, with a device taking only "reversible" actions before the packet is known to be good. Another comparable use of reversible behaviors is "speculative execution" in pipelined microprocessors that employ branch prediction. In this kind of optimization approach, a classic trade-off that the system architect must deal with is gain versus complexity of the optimization.

Conclusions

In this paper we have presented the benefits (and drawbacks) of speculatively forwarding packets. This technique is achieved by means of a new device implementation that does not require any change the HyperTransport I/O Link Specification. In fact, such kind of implementation is implicitly introduced in Section 10.3.3 of the HyperTransport specification. A more aggressive speculative implementation has also been proposed.

The benefits of this technique are mainly a noticeable reduction in packet latency. This reduction depends on link width, packet length, and path length. For example, for 8-bit links, packet latency is reduced by a factor of 2.6 (worst case) to 8 (best case). For 16-bit links, the reduction in packet latency can be expected to be between 2 and almost 5 times. In all case, this speculative forwarding technique achieves lower packet latency.