A Network Calculus Based Analysis for the PCI Bus

Rodolfo Pellizzoni, Min-Young Nam, Richard M. Bradford*, and Lui Sha
Department of Computer Science,
University of Illinois at Urbana-Champaign, Champaign, IL 61801
{mnam, rpelliz2}@uiuc.edu, rmbradfo@rockwellcollins.com, lrs@cs.uiuc.edu

Abstract

I/O configurations play a crucial role in many critical embedded systems. In particular, avionics systems are often I/O bounded and I/O scheduling problems have been a major source of integration problems. A deterministic analysis of communication between CPU and peripherals is needed to derive correct schedulability tests. In this paper, we detail an analysis based on network calculus theory for the PCI bus, which is becoming more and more popular in the embedded market. Our efforts illustrate the difficulties in applying a rigorous analysis framework to COTS components which often lack precise specification.

1. Introduction

In many hard real-time systems, such as avionics, more and more features are added to faster and cheaper hardware, leading to rapid increases in system size and complexity. One important challenge is how to analyze the schedulability impacts of different software and hardware design decisions early in the architecture design process. The schedulability equations are functions of software task designs and hardware designs. Building such analysis is complex as modern embedded systems are increasingly built by using Commercial Off-The-Shelf (COTS) components in an attempt to reduce costs and time-to-market: it is becoming difficult to rely on completely specialized hardware and software solutions since development time and costs raise dramatically while performance is often lower when compared to equivalent COTS components commonly used for general purpose computers. For example, the specialized SAFEbus backplane [5] used in the Boing777 is capable of transferring data up to 60 Mbps, while a modern COTS interconnection such as PCI Express 2.0 [1] can reach transfer speeds over three orders of magnitude greater at 16 Gbyte/s.

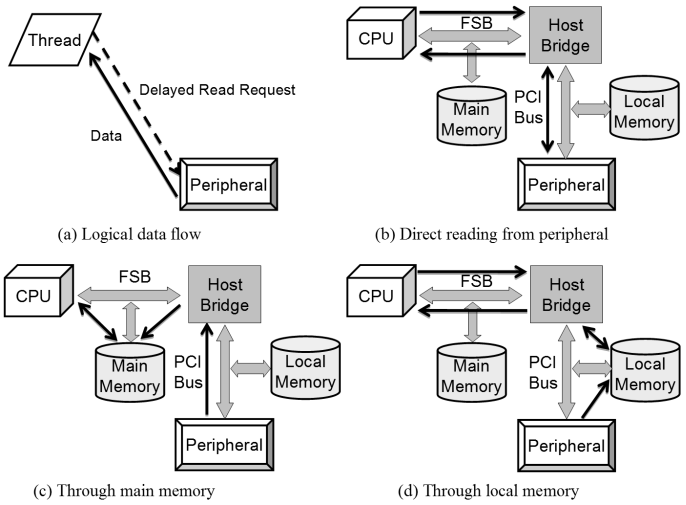

Figure 1: Input/Output Configuration Example

Unfortunately, COTS components employ a variety of techniques, like deep pipelining and caching, that are designed to improve average case performance but makes it difficult to obtain the safe worst-case bounds that are required for critical real-time embedded systems.

In this context, I/O peripheral interconnects and I/O configurations play a crucial role. Avionics systems are often I/O bounded and I/O scheduling problems have been a major source of integration problems found in practice. When data is exchanged through a COTS interconnection such as the PCI bus [1], the overall delay for data transfer must be inserted in the schedulability equation as a blocking time whenever a thread must wait for data to be updated. However, this value greatly depends on the architecture of the hardware including the bus topology and the behavior of peripheral devices that share the bus. The effect of I/O devices is becoming one of the major causes of wrongful analysis for tightly scheduled systems: for example, in [3] it is shown that contention for access to main memory between the CPU and peripherals can increase the execution time of a task up to 44%.

At the same time, a hardware engineer is often forced to choose among many alternative I/O configurations early in

* R. M. Bradford is with Rockwell Collins Inc., Cedar Rapids, IA 52498, U.S.A.

the design phase. Figure 1 shows an example of a PCI read transaction that can be implemented in multiple topologies. The thread executing on the processor can read data directly from the peripheral device or from a memory connected to the front side bus or the PCI bus, while the peripheral device is writing to the memory. During this operation, there will be other interferences from different devices connected to the bus. Clearly, an efficient and automated way to analyze multiple configurations of bus topology and communication flows is required.

In this paper, we describe an analysis framework that is able to compute deterministic worst-case performance bounds on communication through the widely used PCI bus. We estimate the worst-case delay time of all communication flows based on the PCI topology and all other interactions from PCI peripheral devices, and calculate for each PCI bridge the buffer size required to maintain system stability. Our analysis works by mapping PCI physical elements (bridges and bus segments) to network elements (routers and point-to-point connections) and then applying the theory of network calculus [2], which is able to compute deterministic delay bounds for network traffic.

We would like to stress that our contribution is not in the development of network calculus theory itself; on the contrary, network calculus has already been applied and extended to compute deterministic bounds in real-time systems and embedded architectures (for example, see [6] and [7]). Our objective is to show how a rigorous analysis methodology can be applied to a COTS system in order to extract safe bounds in spite of the extreme complexity of the architecture and the lack of a precise specification: most industrial standards are only concerned with preserving logical equivalence, but most timing and performance related aspects are left open to manufacturer implementation. As such, it is essential to make sound assumptions when the precise working of each COTS component can not be determined, often because manufacturers are wary to release implementation details for fear of losing their competitive edges. In this sense, we believe that our methodology can be helpful to produce analysis frameworks for other COTS interconnections. Our final goal is to build a tool that is able to apply the described analysis to a high level system specification, thus enabling efficient space exploration of bus configurations.

The rest of the paper is organized as follows. In Section 2 we briefly describe the PCI standard. In Section 3 we detail our analysis and provide relevant examples. Finally, we provide concluding remarks in Section 4.

2. PCI Bus Standard

The Peripheral Component Interconnect (PCI) is the current standard family of communication architectures for motherboard-peripheral interconnection in the personal computer market; it is also widely popular in the embedded domain [1]. The standard can be divided into two parts: a *logical* specification, which details how the CPU configures and accesses peripherals through the system controller, and a *physical* specification, which details how peripherals are connected to and communicate with the motherboard. In this section we focus on the PCI/PCI-X physical specification, which uses a shared bus architecture with support for multiple bus segments connected by bridges. A typical PCI-based platform is shown in Figure 2. The CPU and main memory are connected through the Front Side Bus (FSB), which has a CPU-manufacturer dependent implementation. A host bridge is also connected to the FSB and offers access to the rest of the system using the PCI standard. Each further PCI-to-PCI bridge connects two PCI bus segments together. The architecture resembles a tree, where the host bridge represents the root, bridges are intermediate nodes, and peripherals represent leaves. Data transfers are carried out on each bus segment as non-preemptive bus *transactions*; the entity that starts a transaction (either a peripheral or the CPU) is known as the *initiator*, while the entity that receives the transaction (either another peripheral or a memory) is known as the *target*. Each bus segment has a separate arbiter which determines the order in which initiators are allowed to transmit. If the initiator and target of a transaction reside on different bus segments, then the transaction data is stored and forwarded by intermediate bridges; note that due to the tree-shaped structure of PCI, there is a single path between any two bus segments. The PCI standard offers support for several types of transactions between different bus segments: *posted writes*, *delayed* writes and reads, and *locking* transactions. The latter are considered obsolete as in the last PCI specification and can further cause very high worse-case delay, hence we do not con-

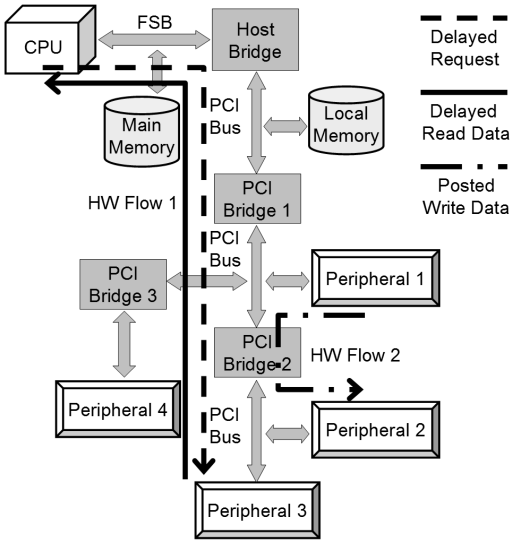

Figure 2: PCI-based platform Example

sider them.

The simplest type of PCI transaction is *posted write*. A posted write completes at the initiator before it completes at the target: in other words, after data has been moved from the initiator to the first intermediate bridge, the initiator can disconnect from the bus and considers the transaction successfully completed. In this way, data is posted from bridge to bridge until the target is reached, similarly to what happens in a packet-switched network; this is the main reason why we base our analysis on network calculus theory [2]. The drawback is that the initiator has no way to know when the transaction completes at the target. When an acknowledgement is required, such as for all read operations, a delayed transaction can be employed. In a delayed transaction, the initiator first sends a write/read request to the first intermediate bridge, which buffers it and then terminates the transaction with a *retry* command. The retry command notifies the initiator that the transaction has not completed successfully, and forces it to retry sending the same request over and over until the bridge acknowledges it (since write requests contains actual data, to avoid saturating the bus the bridge terminates further retries before the data is sent). The first intermediate bridge then sends the request to the second bridge on the transaction path using the same retry mechanism. In this way the request is propagated through the intermediate bridges until it reaches and completes at the target; at this point the last intermediate bridge buffers the completion of the transaction (which includes the actual data for a read transaction), and acknowledges the next request received by the next-to-last intermediate bridge. In this way, the acknowledgement is propagated back until it reaches the initiator and the transaction ends. To preserve coherence, the standard further imposes a detailed set of precedence rules on posted and delayed transactions; in particular, a posted write must always be allowed to pass a buffered delayed transaction, while a delayed transaction can never pass a buffered posted write. The rule is typically enforced by assigning strictly higher priority to posted writes than delayed transactions. We refer the interested reader to [1] for addition details on the PCI standard.

3. PCI Communication Analysis

For ease of exposition, we first describe our analysis for a system where all transactions are posted writes. We treat all system communication as a set of N *flows* $\{f_1, \ldots, f_N\}$, each between a specified source (an initiator) and destination (a target). For each flow f_i, let $R_i(t)$ be its *cumulative function*, i.e. the total number of bytes transmitted by transactions that belong to the flow during interval $[0, t]$ (with the assumption that $R_i(0) = 0$). We then model each flow f_i using an arrival curve $\alpha_i(t) : \forall s, t, s \leq t : R_i(t) - R_i(s) \leq \alpha_i(t - s)$. Intuitively, $\alpha_i(t)$ is a worst-case bound on the amount of traffic generated by the initiator in any interval of length t. The arrival curve captures the burstiness of the initiator; as the flow goes through the network, it is buffered multiple times and as a result the worst case burstiness increases. Network calculus provides a way to compute the burstiness at each stage and relates it to the maximum delay encountered by any byte in the flow. More specifically, if f_i goes through M_i network elements, we can define $M_i + 1$ *subflows*, where f_i^0 represents the flow produced at the initiator, $f_i^j, 0 < j < M_i$ is the flow after passing through j network elements, and $f_i^{M_i}$ is the output flow. The goal is then to compute an arrival curve α_i^j for each subflow f_i^j, with $\alpha_i^0 = \alpha_i$. There remains a problem: network calculus considers a system comprised of multiple network elements, or routers, connected by point-to-point lines, but the PCI architecture employs shared buses. We can still rely on the network calculus model using the following transformation: we treat each bus segment, together with all bridges that transmit data on it, as a unique network element. Network elements are then connected if the corresponding bus segments are interconnected by a bridge. A clarifying picture is shown in Figure 3, for a flows f_i that goes through $M_i = 4$ bus segments numbered B_1, B_2, B_3, B_4. To simplify the analysis, we introduce the following notation: for each subflow $f_i^j, 0 < j \leq M_i$, we let out_i^j be its output network element / bus segment, and for each subflow $f_i^j, 0 \leq j < M_i$, we let in_i^j be its input network element / bus segment. For example, in Figure 3 $out_i^2 = B_2, in_i^2 = B_3$. It follows that $in_i^j = out_i^{j+1}$, which expresses the condition that subflows are concatenated.

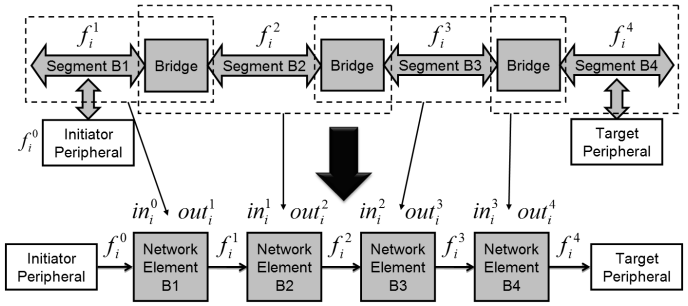

Figure 3: Equivalence of Bus segments and Subflows

Using the network calculus approach, for each subflow f_i^j we then define a *service curve* $\beta_i^j : \forall s, t, s \leq t : R_i^j(t) - R_i^{j-1}(s) \geq \beta_i^j(t - s)$. The intuitive idea is that in any interval of length t, the "service" provided to f_i^j (i.e., how much more traffic is seen on f_i^j at the end of the interval compared to the traffic seen on f_i^{j-1} at the beginning of the interval) is at least $\beta_i^j(t)$. Service curves are important because of the following three essential network calculus theorems:

Theorem 1 (backlog bound, Theorem 1.4.1 in [2])
The maximum backlog (number of buffered bytes) suf-

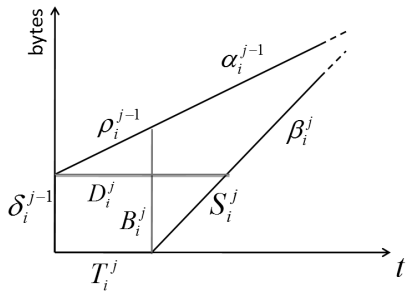

Figure 4: Linearized arrival and service, with theorem bounds

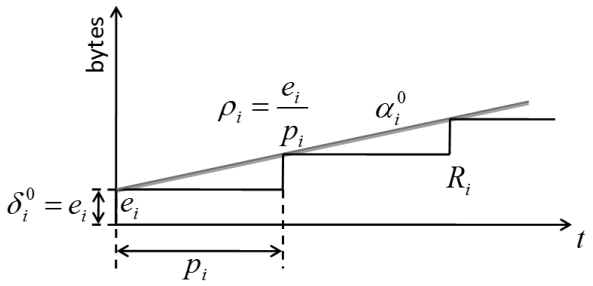

Figure 5: Periodic task Arrival Curve derivation for the Initiator

fered by bytes going from subflow f_i^{j-1} to subflow f_i^j is $B_i^j = \sup_{s \geq 0}\{\alpha_i^{j-1}(s) - \beta_i^j(s)\}$.

Theorem 2 (delay bound, Theorem 1.4.2 in [2]) *The maximum delay suffered by a byte going from subflow f_i^{j-1} to subflow f_i^j is* $D_i^j = \sup_{s \geq 0}\{t : \alpha_i^{j-1}(s) = \beta_i^j(s+t), t \geq 0\}$.

Theorem 3 (flow concatenation, Theorem 1.4.3 in [2]) $\alpha_i^j(t) = (\alpha_i^{j-1} \oslash \beta_j^j)(t)$ *where* $(\alpha \oslash \beta)(t)$ *is called the min-plus deconvolution of* α *by* β *and is defined as* $\sup_{u \geq 0}\{\alpha(t+s) - \beta(s)\}$.

The flow concatenation theorem lets us compute the arrival curve α_i^j at each step. Delay bounds are used to obtain the total flow delay $D_i = \sum_{1 \leq j \leq M_i} D_i^j$. The backlog expresses an important condition on buffer sizes: in particular, our analysis is valid only under the assumption that buffers never overflow, which is true if buffer sizes are at least equal to the computed backlog. Unfortunately, Theorems 1-3 are hard to compute for general arrival and service curves. To simplify the analysis, we therefore propose to adopt a linearized representation. We consider an upper bound to the arrival curve $\alpha_i^j(t)$ of the form $\delta_i^j + t\rho_i^j$, where δ_i^j represents the *burstiness* and ρ_i^j the *arrival rate* for $\alpha_i^j(t)$. As for service curve $\beta_i^j(t)$, we consider a lower bound (which itself is a valid service curve) of the form $\max(0, (t - T_i^j)S_i^j)$, where T_i^j is the *service delay* and S_i^j is the *service rate*. For simplicity, given this representation we write $\alpha_i^j(t) \equiv (\delta_i^j, \rho_i^j)$ and $\beta_i^j(t) \equiv (T_i^j, S_i^j)$. Figure 4 shows example arrival and service curves α_i^{j-1} and β_i^j, respectively. Theorems 1-3 can then be rewritten as follows:

Theorem 4 (linearized backlog bound) *The maximum backlog B_i^j is bounded iff $\rho_i^{j-1} \leq S_i^j$, in which case* $B_i^j = \delta_i^{j-1} + \rho_i^{j-1}T_i^j$.

Proof.
Refer to curves $\alpha_i^{j-1}(t), \beta_i^j(t)$ in Figure 4. If $\rho_i^{j-1} > S_i^j$ (intuitively, $\alpha_i^{j-1}(t)$ increases faster than $\beta_i^j(t)$), then $\lim_{s \to \infty} \alpha_i^{j-1}(s) - \beta_i^j(s) = \infty$, hence the backlog is unbounded. If instead $\rho_i^{j-1} \leq S_i^j$, then the maximum of $\alpha_i^{j-1}(s) - \beta_i^j(s)$ is clearly for $s = T_i^j$, from which $B_i^j = \delta_i^{j-1} + \rho_i^{j-1}T_i^j$. The proofs for linearized delay bound and flow concatenation can be similarly derived (see [2] for details). □

Theorem 5 (linearized delay bound) *The delay D_i^j is bounded iff $\rho_i^{j-1} \leq S_i^j$, in which case* $D_i^j = T_i^j + \frac{\delta_i^{j-1}}{S_i^j}$.

Theorem 6 (linearized flow concatenation) $\alpha_i^j(t)$ *is bounded iff $\rho_i^{j-1} \leq S_i^j$, in which case* $\rho_i^j = \rho_i^{j-1}, \delta_i^j = \delta_i^{j-1} + \rho_i^{j-1}T_i^j$.

Intuitively, the delay bound is the maximum horizontal deviation of $\alpha_i^{j-1}, \beta_i^j$ and the backlog and burstiness are both equal to the maximum vertical deviation. Also note that $\rho_i^j = \rho_i^{j-1}$, i.e. only the burstiness changes among subflows of the same flow, not the arrival rate. We can therefore use a single rate ρ_i for the entire flow and compute just δ_i^j for each subflow. Finally, note that delay and backlog are bounded if and only if the rate provided by the service curve is at least equal to the rate required by the subflow, which is to be expected.

The rest of the discussion is organized as follows. In Section 3.1 we detail how the compute the arrival curve α_i^0 for the input subflow f_i^0. In Section 3.2 we then show how to compute the service curves β_i^j for all successive subflows of f_i. Section 3.3 shows a detailed example of our methodology, and Section 3.4 shows how to improve the computed end-to-end delay D_i. Finally, Section 3.5 extends the analysis to cover delayed transactions.

3.1. Arrival curve derivation

The computation of α_i^0 depends on the characteristics of the initiator and its produced data. Assume that initiator traffic is strictly periodic, i.e. the initiator generates e_i bytes of traffic every p_i time units. A clarifying example is provided in Figure 5, where we plot the cumulative function $R_i(t)$ for the flow and its arrival upper bound (δ_i, ρ_i).

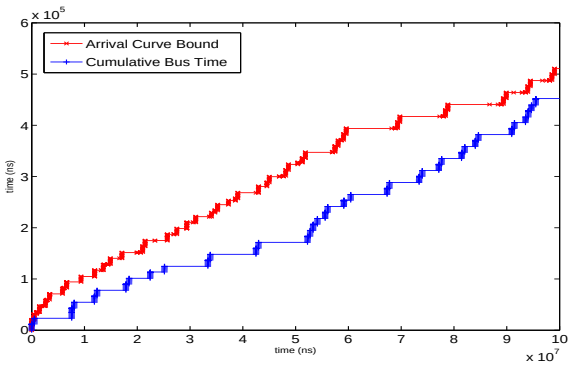

Figure 6: Measured Arrival Curve.

Lemma 7 *Consider an initiator with strictly periodic traffic e_i, p_i. Then $\alpha_i^0 \equiv (\delta_i^0 = e_i, \rho_i = \frac{e_i}{p_i})$ is an arrival curve for the peripheral traffic.*

Proof.
Let $\bar{R}(t)$ be the cumulative function where e_i bytes of data are first produced at time $t = 0$, and at each subsequent period thereafter. It is easy to verify that $\bar{R}(t)$ captures the critical instant, i.e. for any other cumulative function $R(t)$, $\bar{R}(t - s - 0) \geq R(t) - R(s)$. To show that α_i^0 is a valid arrival curve, we then only need to check that $\alpha_i^0(t) \geq \bar{R}(t)$ for all t, which is trivial. □

However, often such detailed information is not available, for example because the average transmission rate ρ_i of the peripheral is known, but the transmitted traffic is not exactly periodic. In this case, we propose a testing methodology that is well suited to the analysis of COTS peripherals. A trace of activity for a PCI/PCI-X peripheral can be gathered monitoring the bus with a logic analyzer. For example, Figure 6 shows the first 100ms of a measured trace for a 100MB/s ethernet network card in term of cumulative time taken by peripheral transactions on a 32bit, 33Mhz PCI bus segment; the whole recorded trace consisted of 1000 transactions. We developed a simple algorithm that computes the arrival curve α_i^0 from a trace in quadratic time in the number of bus transactions. The algorithm performs a double iteration over all transactions, computing at each step the amount of peripheral traffic in an interval between the beginning of any transaction and the end of any other successive transaction. The computed values are inserted into a list ordered by interval length, and all non maximal values are culled. Figure 6 shows the result in the interval $[0, 100ms]$; note that the computed arrival curve is expressed in term of required bus time rather than bytes, so to obtain α_i^0 we must multiply it by the speed of the bus. If multiple traces are recorded, then an upper bound can be computed by merging the computed arrival curves for each trace and again removing all non maximal values. Finally, if the average transmission rate ρ_i is known, then a burstiness value δ_i^0 can be obtained as the minimum value such that $\delta_i^0 + t\rho_i$ is greater than or equal to the computed bound for all t.

3.2. Service curve derivation

We now focus on the derivation of a service curve β_i^j for subflow f_i^j. Assume that f_i^j is produced by element B_k, i.e. $out_i^j = B_k$. Intuitively, the service curve β_i^j depends on two elements: the subflows that are input to B_k, and the transmission policy followed by all elements that comprise B_k (bus segment arbiter and bridges that transmit on B_k). The PCI standard does not impose any strict constraint on bus segment arbitration, instead mentioning only a weak fairness condition, in the sense that a backlogged subflow must eventually be allowed to transmit. We therefore consider only the following work-conserving constraint: if subflow f_i^{j-1} is backlogged and B_k is free, then f_i^j must be transmitted. This is equivalent to assuming that f_i^{j-1} has the lowest priority among all other input flows to B_k (f_i^j is transmitted iff no other input flow is backlogged). To simplify the analysis, we define the set of *interfering* input flows as follows:

$$inter_i^j = \{f_l^q : l \neq i, in_l^q = out_i^j\}. \quad (1)$$

In other words, $inter_i^j$ is the set of all input flows to B_k except for f_i^{j-1}. Furthermore, let C be the speed of bus segment B_k in bytes/s. The following result can then be proved:

Theorem 8 *A valid service curve β_i^j is $(T_i^j = \frac{\sum_{f_l^q \in inter_i^j} \delta_l^q}{C - \sum_{f_l^q \in inter_i^j} \rho_l}, S_i^j = C - \sum_{f_l^q \in inter_i^j} \rho_l)$.*

Proof.
A function $x(t)$ is said to be a strict service curve if for any interval of length t during which f_i^{j-1} is backlogged, f_i^j transmits at least $x(t)$ bytes. We now show that the defined β_i^j is a strict service curve; it is then trivial to see that any strict service curve is also a valid service curve for the flow (see Proposition 1.3.5 in [2]).

Due to the work-conserving assumption, we can obtain a strict service curve by computing the total amount of traffic that B_k can transmit in any interval of time t minus traffic from all interfering flows. Therefore:

$$\beta_i^j = \max(0, Ct - \sum_{f_l^q \in inter_i^j} \alpha_l^q) \quad (2)$$

$$= \max(0, (C - \sum_{f_l^q \in inter_i^j} \rho_l)t - \sum_{f_l^q \in inter_i^j} \delta_l^q) \quad (3)$$

$$= \max(0, (C - \sum_{f_l^q \in inter_i^j} \rho_l)(t - \frac{\sum_{f_l^q \in inter_i^j} \delta_l^q}{C - \sum_{f_l^q \in inter_i^j} \rho_l})) \quad (4)$$

which concludes the proof. □

We can then apply Theorem 6 to obtain arrival curve α_i^j.

Corollary 9 *$\alpha_i^j(t)$ is bounded iff $\sum_{f_l^q : in_l^q = out_i^j} \rho_l \leq C$, in which case:*

$$\delta_i^j = \delta_i^{j-1} + \rho_i \frac{\sum_{f_l^q \in inter_i^j} \delta_l^q}{C - \sum_{f_l^q \in inter_i^j} \rho_l}. \quad (5)$$

The inequality $\sum_{f_l^q : in_l^q = out_i^j} \rho_l \leq C$ expresses the necessary condition that the sum of the rates of all incoming flows to a bus segment must be smaller than C, i.e. the bus segment utilization must not exceed one.

If we know more about the implementation of arbiters and bridges, we can improve the bound of Corollary 9. First of all, assume that all posted writes that are input to the same segment B_k are buffered inside a unique FIFO buffer in each bridge, which is typically true for commercial bridges since it simplifies design. We can obtain a tighter bound by removing all subflows in the same buffer from the interfering set, since a newly arrived byte in the FIFO can not interfere with bytes from different subflows that are already stored in the FIFO. In network calculus terminology, subflows stored in the same FIFO queues are referred to as *aggregate* traffic. We can formally define aggregate subflows as follows:

$$aggr_i^j = \{f_l^q : l \neq i, out_l^q = out_i^{j-1} \wedge in_l^q = in_i^{j-1} = out_i^j\}. \quad (6)$$

In other words, aggregate subflows are all subflow between the same input and output network elements as f_i^{j-1}. We can then obtain a valid service curve for the aggregate using Theorem 8 with the following interfering set:

$$inter_i^j = \{f_l^q : l \neq i, in_l^q = out_i^j, f_i^q \notin aggr_i^j\}, \quad (7)$$

which is the same as Equation (1) except that we removed the aggregate flows. Unfortunately, Theorem 6 does not hold for aggregate flows, but we can instead prove the following:

Theorem 10 (linearized concatenation with aggregate flow) *$\alpha_i^j(t)$ is bounded iff $\rho_i \leq S_i^j$, in which case $\delta_i^j = \delta_i^{j-1} + \rho_i T_i^j + \rho_i \frac{\sum_{f_l^q \in aggr_i^j} \delta_l^q}{S_i^j}$.*

Proof.
The theorem follows directly from Theorem 6.4.1 in [2], applying it to the case of multiple aggregate flows and to a linearized service curve. □

Applying Theorems 8, 10 to compute δ_i^j yields the following corollary:

Corollary 11 *$\alpha_i^j(t)$ is bounded iff $\sum_{f_l^q : in_l^q = out_i^j} \rho_l \leq C$, in which case:*

$$\delta_i^j = \delta_i^{j-1} + \rho_i \frac{\sum_{f_l^q \in inter_i^j} \delta_l^q + \sum_{f_l^q \in aggr_i^j} \delta_l^q}{C - \sum_{f_l^q \in inter_i^j} \rho_l}. \quad (8)$$

Note that the difference between Equations 8 and 5 is that the rate of aggregate flows does not need to be subtracted from C.

A second burstiness bound can be obtained by adding additional constraints on bus segment arbitration. While the standard does not require it, it suggests that bus arbitration be round-robin. Therefore, in what follows we assume that arbitration follows a simple round-robin model where each bridge and each peripheral that transmit on the system is given one round[1]. To model round-robin arbitration, we need additional information: for each flow f_i, assume that we know the minimum length in bytes of any transaction $L_i^{\min}$ and the maximum length $L_i^{\max}$. Furthermore, since bridges can merge transactions together, let L^{bridge} be the maximum length in bytes of transactions generated by a bridge. Then we can obtain a service curve considering that in the worst case all bridges and all peripherals connected to the bus transmit for the maximum amount of time (which we call RR_i^j), except for the subflow f_i^j which transmits for its minimum bit length $L_i^{\min}$.

Theorem 12 *Consider a network element with K connected bridges. A valid service curve β_i^j is $(T_i^j = \frac{RR_i^j}{C}, S_i^j = \frac{CL_{elem}^{\min}}{L_{elem}^{\min} + RR_i^j})$, where:*

$$L_{elem}^{\min} = \min(L_i^{\min}, \min_{f_l^q \in aggr_i^{j-1}} \{L_l^{\min}\}) \quad (9)$$

and

$$RR_i^j = KL^{bridge} + \sum_{f_l^0 \in inter_i^j} L_l^{\max} \text{ for } j = 1; \quad (10)$$

$$RR_i^j = (K-1)L^{bridge} + \sum_{f_l^0 \in inter_i^j} L_l^{\max} \text{ for } j > 1; \quad (11)$$

Proof.
We distinguish two cases: 1) $j = 1$, meaning that the input subflow $f_i^{j-1} = f_j^0$ is buffered in a peripheral; 2) $j > 1$, meaning that f_i^{j-1} is buffered in a bridge.

Consider $j = 1$ first. We show that the effect of round-robin arbitration is to force a periodic schedule where f_i^j can not transmit for at most $\frac{RR_i^j}{C}$ time units, and can then transmit for at least $\frac{L_{elem}^{\min}}{C}$ time units. This implies that the

1 In practice, most commercial bridges employ as default a double-level round-robin scheme where the first level arbitration is between the upstream bridge and all other peripherals/bridges; however, the proposed model can be easily generalized.

defined β_i^j is a lower bound to a strict service curve for f_i^j, and therefore it is a lower bound to a service curve which is itself a valid service curve for f_i^j. Assume that all other peripherals and bridges that transmits on the bus segment are backlogged, and are enqueued before f_i^j in the round-robin queue. Then a maximum amount of traffic RR_i^j will be sent before f_i^j is allowed to transmit assuming that all K bridges transmit L^{bridge} amount of traffic and all interfering peripheral with input subflows f_l^0 transmits their maximum length $L_l^{\max}$. Since furthermore for $j = 1, L_{elem}^{\min}$ reduces to $L_i^{\min}$, this concludes the proof for the first case.

Now consider the $j > 1$ case. We follow the same approach, except that in the periodic schedule we consider the minimum time $\frac{L_{elem}^{\min}}{C}$ during which all aggregate flows can transmit. Therefore, $\frac{RR_i^j}{C}$ now considers transmissions from all peripherals and all other $K-1$ bridges, and a safe lower bound on $L_{elem}^{\min}$ can be obtained by taking the minimum transaction length among all aggregate flows. □

The obtained β_i^j can be used with Theorems 6 and 10 to compute α_i^j. We can therefore ask ourselves whether using the service curve of Theorem 8 or Theorem 12 yields a lower bound on the burstiness δ_i^j. In general, this is not a trivial question, since the service rates S_i^j computed by Theorems 8, 12 are not comparable. Obviously if ρ_i is smaller than the S_i^j for a service curve, we can not use it. In the case where we can apply both service curves, we have to consider two cases. If all burstiness values δ_l^q required to compute δ_i^j are known, then we can simply use both service curves and take the minimum of the computed δ_i^j. However, as we show in Section 3.3 there are cases when the δ_l^q are unknown, and a system of equations must be solved to obtain them. In this case, it is preferable to consider a single service curve to avoid inserting a minimum function in the system. It is possible to prove that when no aggregate flow is considered, Theorem 12 always provides a smaller service delay T_i^j and therefore a lower δ_i^j. For aggregate flows, this is not true since the burstiness depends also on the service rate. However, using Theorem 12 reduces the number of variables δ_l^q that δ_i^j depends on. As we show in Section 3.3, mutual dependencies between flows can reduce the available bus utilization, hence this is desirable.

3.3. Example

We now provide an example to show how the analysis can be applied to a concrete case. It is important to note that the burstiness bounds of Equations 5, 8 directly hold only for *feed-forward* configurations, where there are no circular dependencies among flows. In this case, Equations 5, 8 can be applied to iteratively compute all burstiness terms d_i^j. In practice, many configurations do have circular dependencies among flows. Consider the system with

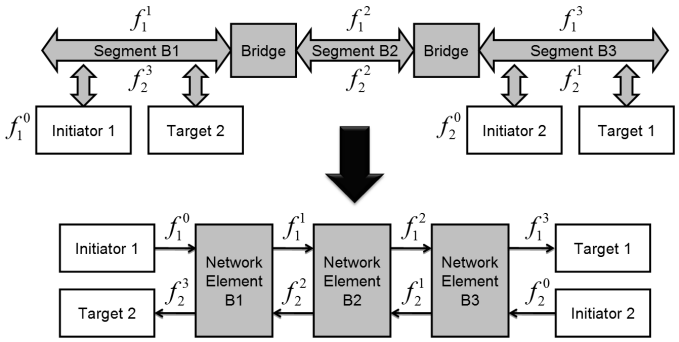

Figure 7: Two Flows Circular Example

two posted write flows f_1, f_2 and three bus segments depicted in Figure 7, where we assume that δ_1^0, δ_2^0 are known and $\rho_1 = \rho_2 = \rho \leq 0.5C$. Using Corollary 9 would yield six equations for burstiness $\delta_1^1, \delta_1^2, \delta_1^3, \delta_1^1, \delta_1^2, \delta_1^3$:

$$\begin{array}{ll} \delta_1^1 = \delta_1 + \rho_1 \frac{\delta_2^2}{C-\rho_2} & \delta_2^1 = \delta_2 + \rho_2 \frac{\delta_1^2}{C-\rho_1} \\ \delta_1^2 = \delta_1^1 + \rho_1 \frac{\delta_2^1}{C-\rho_2} & \delta_2^2 = \delta_2^1 + \rho_2 \frac{\delta_1^1}{C-\rho_1} \\ \delta_1^3 = \delta_1^2 + \rho_1 \frac{\delta_2}{C-\rho_2} & \delta_2^3 = \delta_2^2 + \rho_2 \frac{\delta_1}{C-\rho_1} \end{array} \quad (12)$$

Note that there is a circular dependencies between δ_1^2, δ_2^1 and between δ_2^2, δ_1^1, hence we cannot compute them directly. To solve this problem, we follow the methodology delineated in Proposition 2.4.1 in [2]. We first note that Equations 5, 8 are linear in the burstiness terms. We can therefore use them to obtain a linear system of equations of the form $\vec{x} = A\vec{x} + \vec{b}$, where $\vec{x}$ is a vector of unknown burstiness values, A is a square matrix and $\vec{b}$ is a vector of known values. Furthermore, it can be shown that following Equations 5, 8 both A and $\vec{b}$ are non negative. Hence, if all eigenvalues of A are within the unit circle, $(I-A)^{-1}$ is also non negative and $\vec{x}$ can be computed as $\vec{x} = (I-A)^{-1}\vec{b}$. By following the *time stopping* principle [2], it can then be proven that $\vec{x}$ indeed provides a valid upper bound for burstiness.

Following the above technique, we can compute $\delta_1^1, \delta_1^2, \delta_1^1, \delta_1^2$ using the following system:

$$\vec{x} = \begin{pmatrix} \delta_1^1 \\ \delta_1^2 \\ \delta_2^1 \\ \delta_2^2 \end{pmatrix}, A = \begin{pmatrix} 0 & 0 & 0 & k \\ 1 & 0 & k & 0 \\ 0 & k & 0 & 0 \\ k & 0 & 1 & 0 \end{pmatrix}, \vec{b} = \begin{pmatrix} \delta_1 \\ 0 \\ \delta_2 \\ 0 \end{pmatrix}, \quad (13)$$

where $k = \frac{\rho}{C-\rho}$. The eigenvalues are $\pm\sqrt{k^2+k}$ and $\pm\sqrt{k^2-k}$; by solving for ρ we obtain that a solution exists if $\rho \leq \frac{3-\sqrt{5}}{2}C \approx 0.382C$. It is interesting to see that the condition implies that we are not able to find delay bounds even when bus utilization is as low as 77%. Note, however, that the derived condition on A is only sufficient: if all eigenvalues are within the unit circle, then we can compute bounded delay and backlog, but even if the eigenvalues are outside the unit circle, delay and backlog can still be

bounded. In fact, the derivation of necessary stability conditions for general networks is still an open problem.

A final consideration is relative to parametric analysis. So far in this section we have assumed that all inputs (data sizes e_i and periods p_i) were constants, but in practice, we are interested in the case where some of the inputs are variables. Assume for example that a data size e_i is allowed to vary in an interval $[e_i^{\min}, e_i^{\max}]$. Assuming that delay bounds can be computed for $e_i = e_i^{\max}$, we would like to solve the system in Equation 13 symbolically as a function of the variable e_i. However, for this to be possible, we have to prove that if delay bounds can be computed for $e_i = e_i^{\max}$, then they can also be computed for any $e_i \in [e_i^{\min}, e_i^{\max}]$.

Note that according to Lemma 7 and 22, reducing the data size e_i will decrease the input arrival curve α_i^0. We now prove that decreasing the arrival curve leads to both a reduction in the delay (Corollary 14) and in the output arrival curve (Theorem 15) for each network element. The time stopping principle can then again be applied to show that the delay bound computed for $e_i = e_i^{\max}$ is a valid upper bound for any $e_i \in [e_i^{\min}, e_i^{\max}]$. Hence, as long as delay bounds are finite for $e_i = e_i^{\max}$ the symbolic solution of the system $\vec{x} = A\vec{x} + \vec{b}$ will yield a correct value.

Theorem 13 *Consider a node that offers a service curve β and suppose the node is stable for some arrival curve α,* i.e. *for all sufficiently large t, $\beta(t) \geq \alpha(t)$. Now consider a second arrival curve α_ℓ such that $\alpha_\ell(t) \leq \alpha(t)\ \forall t$. Assume for simplicity that α, α_ℓ, and β are all continuous. Then the worst-case delay bound, given by the maximum horizontal distance between the arrival curve and service curve, is no larger when α_ℓ is the arrival curve than when α is the arrival curve.*

Proof.
For any t, let $\hat{d}(t)$ denote the horizontal distance between $\alpha(t)$ and β, and let $\hat{d}_\ell(t)$ denote the horizontal distance between α_ℓ and β. Then $\alpha(t) = \beta(t + \hat{d}(t))$. By assumption, $\alpha_\ell(t) \leq \alpha(t)$. Then $\beta(t + \hat{d}_\ell(t)) = \alpha_\ell(t) \leq \alpha(t) = \beta(t + \hat{d}(t))$. Since β must be nondecreasing by definition, this implies that $t + \hat{d}_\ell(t)$ must be no greater than $t + \hat{d}(t)$ and therefore that $\hat{d}_\ell(t) \leq \hat{d}(t)$. Since this holds for an arbitrary t, the maximum of $\hat{d}_\ell(t)$ must be no larger than the maximum of $\hat{d}(t)$. □

Corollary 14 *Under the assumptions of the previous theorem, the worst-case backlog bound, given by the maximum vertical distance between the arrival curve and service curve, is no larger when α_ℓ is the arrival curve than when α is the arrival curve.*

Proof.
By assumption, $\alpha_\ell(t) \leq \alpha(t)\ \forall t$. Then $\alpha_\ell(t) - \beta(t) \leq \alpha(t) - \beta(t)\ \forall t$. □

According to (Theorem 1.4.3) in [2], when a flow having arrival curve α passes through a node that offers a service curve β, then the output flow is constrained by the min-plus deconvolution of α by β. We use this to prove a similar result to the previous theorems, this time for the output flow.

Theorem 15 *Consider a node that offers a service curve β and suppose the node is stable for some arrival curve α. Now consider a second arrival curve α_ℓ such that $\alpha_\ell(t) \leq \alpha(t)\ \forall t$. Assume for simplicity that α, α_ℓ, and β are all continuous. Then the worst-case output envelope given by the min-plus deconvolution when α_ℓ is the arrival curve for the node is no larger than when α is the arrival curve.*

Proof.
Recall that the min-plus deconvolution of α and β is $(\alpha \oslash \beta)(t) = \sup_{u \geq 0} \{\alpha(t+u) - \beta(u)\}$. By assumption, the node is stable, so the supremum happens within a bounded interval of u. Moreover, α and β are continuous by assumption. Thus, by Weierstrass's theorem, which states that a continuous function on a nonempty compact set (such as a closed and bounded set of real numbers) attains its supremum, we conclude that for every t there is at least one value of u such that $(\alpha \oslash \beta)(t) = \alpha(t+u) - \beta(u)$.

Consider an arbitrary t, and let u^* be the value of u that maximizes $\alpha(t+u) - \beta(u)$. Now consider $(\alpha_\ell \oslash \beta)(t) = \sup_{u \geq 0} \{\alpha_\ell(t+u) - \beta(u)\}$. By the same arguments as above, there exists at least one value of u, call one of them u_ℓ^*, that maximizes $\alpha_\ell(t+u) - \beta(u)$. From the assumption that α upper-bounds α_ℓ, it follows that $(\alpha_\ell \oslash \beta)(t) = \alpha_\ell(t+u_\ell^*) - \beta(u_\ell^*) \leq \alpha(t+u_\ell^*) - \beta(u_\ell^*) \leq \alpha(t+u^*) - \beta(u^*) = (\alpha \oslash \beta)(t)$. □

3.4. Improving End-to-End Delay

Following the methodology described in Section 3.3, all subflow burstiness values d_i^j can be computed, and therefore the service curve parameters S_i^j, T_i^j can also be determined. Following Theorem 5, the end-to-end delay D_i for flow f_i can then by determined by computing the delay bound D_i^j on each element out_i^j and then summing over all $1 \leq j \leq M_i$, which yields:

$$D_i = \sum_{1 \leq j \leq M_i} \left(T_i^j + \frac{\delta_i^{j-1}}{S_i^j} \right). \tag{14}$$

However, the obtained end-to-end delay is pessimistic. Network calculus provides a way to reduce two adjacent network elements to a unique element by concatenating the respective service curves for flow f_i. We can therefore obtain a new end-to-end delay bound by first concatenating

all service curves $\beta_i^1, \ldots, \beta_i^{M_i}$ together, and then applying Theorem 5. This newly obtained bound is less pessimistic that the one of Equation 14, an effect known as "pay burstiness only once".

Theorem 16 (service concatenation, Theorem 1.4.6 in [2]) *Assume flow f_i traverses elements out_i^j, out_i^{j+1} with service curves $\beta_i^j(t), \beta_i^{j+1}(t)$. Then the concatenation of the two elements offers a service curve $\beta_i^{j,j+1}(t) = (\beta_i^j(t) \otimes \beta_i^{j+1})(t)$, where $(\beta_1 \otimes \beta_2)(t)$ is called the min-plus convolution of β_1 and β_2 and is defined as* $\inf_{0 \le s \le t} \{\beta_1(t-s) + \beta_2(s)\}$.

Theorem 17 (linearized service concatenation) $\beta_i^{j,j+1}(t) = \max(0, (t - T_i^j - T_i^{j+1}) \min(S_i^j, S_i^{j+1}))$.

By iteratively applying Theorem 17, a global service curve $\beta_i^{1,M_i} \equiv (T_i^{1,M_i}, S_i^{1,M_i})$ can be obtained, with $T_i^{1,M_i} = \sum_{1 \le j \le M_i} T_i^j$ and $S_i^{1,M_i} = \min_{1 \le j \le M_i} S_i^j$. Finally, using β_i^{1,M_i} the following end-to-end delay bound can be obtained:

$$D_i = \sum_{1 \le j \le M_i} T_i^j + \frac{\delta_i^0}{\min_{1 \le j \le M_i} S_i^j}. \quad (15)$$

3.5. Delayed Transactions

We now extend the analysis to deal with delayed transactions. If the initiator and target reside on different bus segments, a delayed transaction requires two flows: a forward flow from initiator to target, and a backward flow from target to initiator. In the case of delayed writes, the forward flow carries the real data, while the backward flow provides acknowledgement to the initiator. We can therefore first model the forward flow in the same way as we did for posted write and compute burstiness bounds for all other flows. Then, we can compute delay for the backward flow and sum it to the delay of the forward flow to obtain the overall flow delay. Delayed reads are more difficult to analyze: the forward flow only carries a read request, while the real data is carried by the backward flow. A clarifying example of delayed read is shown in Figure 8, where $\{f_i^0, \ldots f_i^4\}$ represent the forward flow, and $\{\bar{f}_i^0, \ldots, \bar{f}_i^4\}$ represent the backward flow. Note that $f_i^{M_i-1} \equiv \bar{f}_i^0$ and $f_i^{M_i} \equiv \bar{f}_i^1$, i.e. the forward subflow on the last bus segment, which obtains the data from the peripheral, is now the part of the backward flow which receives its input flow from $\bar{f}_i^0$. As shown in Figure 8, we can interpret the backward flow to be a posted write in the opposite direction with $\bar{f}_i^0$ as the flow generated by an initiator. In what follows, we first show how the analysis can be extended to account for delayed writes, and then we explain how delayed reads can be modeled.

When delayed transactions are present in the system, we must consider the effect of the associated retries on all transactions. The PCI specification imposes that posted writes be allowed to have priority over delayed request, while there

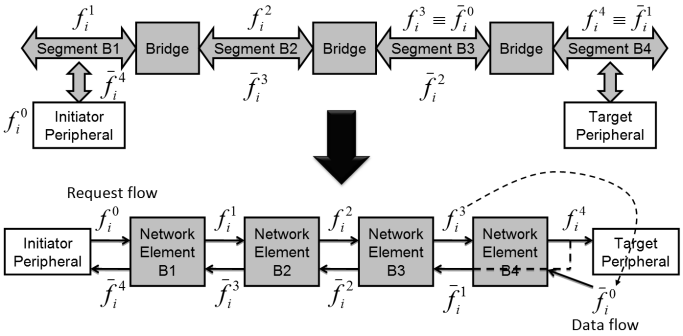

Figure 8: PCI Delayed Read Transaction

is no requirement on the ordering of delayed transaction within themselves. We shall again make very general assumptions, as dealing with each special case would be otherwise extremely complex: we assume that posted writes are buffered in a FIFO queue as before, and that this queue has strictly higher priority than all delayed transactions in the same bridge. Therefore, the $inter_i^j$ and $aggr_i^j$ definitions for a posted write f_i^j, when $j > 1$, must be changed to remove delayed transactions in the same bridge. As for each delayed subflow, we simply use the pessimistic assumption that all other transactions have higher priority. It is useful to introduce the set of *higher priority* subflows for a delayed subflow f_i^j as follows:

$$hp_i^j = \{f_l^q : l \ne i, out_l^q = out_i^{j-1}, in_l^q = in_i^{j-1}\}. \quad (16)$$

Note that since we do not make any assumption on the relative ordering of delayed transactions among each other, the aggregate set for a delayed subflow is always void. Finally, all retries are comprised by the same number of bytes r. Since retries can be continuously retransmitted on the bus, we need to impose some assumptions on the bus to bound the bandwidth consumed by retries. We shall therefore assume that the bus arbitration is round-robin, and furthermore that $L_i^{\min} \ge r$ for all flows f_i. Under this assumptions we can compute service curves for all flows, using the strategy of either Theorem 8 or Theorem 12. We start with the latter.

Theorem 18 *Consider a subflow f_i^1 (either posted or delayed), and let $(T_i'^1, S_i'^1)$ be the service curve computed by Theorem 12. Then $(T_j^1 = T_i'^1, S_j^1 = S_i'^1)$ is a valid service curve for f_i^1.*

Proof.
Since $L_q^{\min} \ge r$ for all flows f_q (and thus also $L^{bridge} \ge r$), the computed RR_j^1 in Theorem 12 is still a worst case interference time under round-robin. The proof then directly follows. □

Theorem 19 *Consider a posted write subflow $f_i^j, j > 1$, and let $(T_i'^j, S_i'^j)$ be the service curve computed by Theorem 12. Then $(T_i^j = T_i'^j + \frac{L^{bridge}}{S_i'^j}, S_j^i = S_i'^i)$ is a valid service curve for f_i^j and all its aggregate subflows.*

Proof.
Posted writes have strictly higher priority than delayed transactions coming from the same bridge, hence they can get interference from them. However, since transactions are non preemptive, there might be a delayed transaction transmitting at time $t = 0$, which can block f_i^j while transmitting at most L^{bridge} bytes. A valid strict service curve for f_i^j can therefore be computed as:

$$\beta_i^j = (t - T_i'^j)S_i'^j - L^{bridge} = \big(t - \big(T_i'^j + \frac{L^{bridge}}{S_i'^j}\big)\big)S_i'^j, \tag{17}$$

which concludes the proof. □

Theorem 20 *Consider a delayed write forward subflow f_i^j, and let $(T_i'^j, S_i'^j)$ be the service curve computed by Theorem 12 with:*

$$L_{elem}^{\min} = \min(L_i^{\min}, \min_{f_l^q \in hp_i^j}\{L_l^{\min}\}). \tag{18}$$

Then $(T_i^j = \frac{T_i'^j S_i'^j + \sum_{f_l^q \in hp_i^j} \delta_l^q}{S_i'^i - \sum_{f_l^q \in hp_i^j} \rho_l}, S_j^i = S_i'^i - \sum_{f_l^q \in hp_i^j} \rho_l)$ is a valid service curve for f_i^j.

Proof.
The service curve $(T_i'^j, S_i'^j)$ computed by Theorem 12 is a valid strict service curve for the aggregate of f_i^j and the all subflows in hp_i^j. Since we assumed that f_i^j has strictly lower priority than subflows in hp_i^j, we can obtain a strict service curve for f_i^j by subtracting the maximum traffic requested by all subflows in hp_i^j. Therefore:

$$\beta_i^j = (t - T_i'^j)S_i'^j - \sum_{f_l^q \in hp_i^{j-1}} (t\rho_l + \delta_l^q) \tag{19}$$

$$= t(S_i'^j - \sum_{f_l^q \in hp_i^j} \rho_l) - T_i'^j S_i'^j - \sum_{f_l^q \in hp_i^j} \delta_l^q, \tag{20}$$

$$= (t - \frac{T_i'^j S_i'^j + \sum_{f_l^q \in hp_i^j} \delta_l^q}{S_i'^i - \sum_{f_l^q \in hp_i^j} \rho_l})(S_i'^j - \sum_{f_l^q \in hp_i^j} \rho_l). \tag{21}$$

□

In the case where $\rho_i > S_i'^j$ and therefore the round-robin based bound can not be applied, another service curve can be derived using the same idea as Theorem 8.

Theorem 21 *Consider a network element with K connected bridges. Define:*

$$L_{elem}^{\min} = \min(L_i^{\min}, \min_{f_l^q \in inter_i^j}\{L_l^{\min}\}) \tag{22}$$

$$RR_i^j = Kr \text{ for } j = 1; \tag{23}$$

$$RR_i^j = (K-1)r \text{ for } j > 1; \tag{24}$$

$$T' = \frac{RR_i^j}{C}; S' = \frac{CL_{elem}^{\min}}{L_{elem}^{\min} + RR_i^j} \tag{25}$$

Then a valid service curve for f_i^j, where f_i^j is a delayed subflow or $j = 1$, is:

$$(T_i^j = \frac{T'S' + \sum_{f_l^q \in inter_i^j} \delta_l^q}{S' - \sum_{f_l^q \in inter_i^j} \rho_l}, S_j^i = S' - \sum_{f_l^q \in inter_i^j} \rho_l). \tag{26}$$

Furthermore, a valid service curve for a posted write subflow f_i^j with $j > 1$ and all its aggregate subflows is:

$$(T_i^j = \frac{T'S' + \sum_{f_l^q \in inter_i^j} \delta_l^q}{S' - \sum_{f_l^q \in inter_i^j} \rho_l} + \frac{L^{bridge}}{S_i^j},$$

$$S_j^i = S' - \sum_{f_l^q \in inter_i^j} \rho_l). \tag{27}$$

Proof.
The proof for this theorem uses a similar methodology as the proofs of Theorems 12,18-20; in what follows, we provide a proof sketch. We first compute a service curve (T', S') for f_i^j and all interfering traffic assuming round-robin arbitration with the maximum number of retrying bridges/peripherals. Equations (22)-(25) can thus be computed using the same idea as Theorem 12. A strict service curve for just f_i^j can then be derived using the same strategy as Theorem 20, assuming that all interfering traffic has higher priority. Finally, note that posted writes in a bridge can still be blocked at time $t = 0$ by at most one delayed transactions, hence we need to add a term $\frac{L^{bridge}}{S_i^j}$ as in Theorem 19. □

Note that when $L_{elem}^{\min} >> r$ (that is, retries are much smaller than normal transactions), we obtain $T' = 0, S' = C$, and the service curve of Theorem 21 is equivalent to Theorem 8. Given the new service curves, equations for burstiness can be derived using either Theorem 10 (for posted writes buffered in a bridge) or Theorem 6 (for all other subflows) and the system can be solved using the approach described in Section 3.3; it is easy to see that all equations are still linear in the burstiness. In turn, this lets us compute the delay for the forward flow.

To compute the delay for the backward flow, we can use the following solution: we remove all forward subflows of

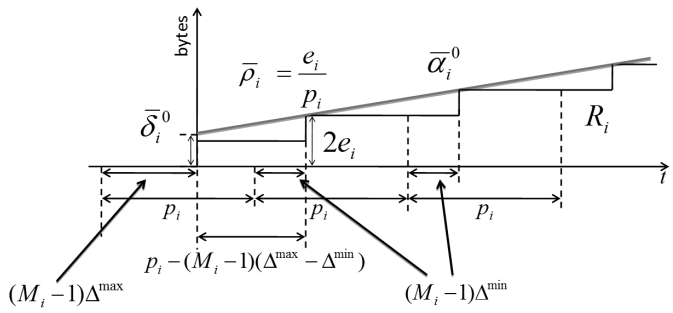

Figure 9: Periodic task Arrival Curve derivation for the Target

f_i from the system, and we introduce backward subflows $\bar{f}_i^2, \ldots, \bar{f}_i^{M_i}$, each with an arrival curve $(\bar{\delta}_i^j = r, \bar{\rho}_i = 0)$ which represents a retry operation. Since after waiting for the forward delay the transaction has completed at the target, the backward delay at each step is the delay of transmitting a single retry. We then compute service curves $\bar{\beta}_i^2, \ldots, \bar{\beta}_i^{M_i}$ for each backward subflow according to Theorems 18-21, and subflow delay using Theorem 5. The overall backward delay can then be obtained by summing the delay of subflows $\bar{f}_i^2, \ldots, \bar{f}_i^{M_i}$; note that we do not consider subflows $\bar{f}_i^0, \bar{f}_i^2$ since their delay is part of the forward flows $f_i^{M_i-1}, f_i^{M_i}$ as shown in Figure 8.

Let us know consider delayed reads. Since delayed reads use the retries in the same way as delayed writes, we can reuse Theorems 18-21 to compute service curves. However, since the real data is carried by the backward flow, we must first use backward subflows to compute system-wide bounds on burstiness. Forward delay can then be computed using the same substitution technique used for backward delay in delayed writes. To simplify the analysis, assume that both the maximum and minimum delay $\Delta^{\max}, \Delta^{\min}$ required to propagate a read request at each stage is known, such that the maximum forward delay for the first $M_i - 1$ segments is $(M_i - 1)\Delta^{\max}$. Since the output forward flow $f_i^{M_i}$ carries data, we need to analyze the last bus segment ($B4$ in the example of Figure 8); this implies deriving an arrival curve for $f_i^{M_i-1} \equiv \bar{f}_i^0$ (f_i^3 in the example). Again suppose that the flow is periodic with e_i bytes of traffic in every period p_i. The worst case arrival pattern for the cumulative function $R_i(t)$ for subflow $\bar{f}_i^0$ is shown in Figure 9, where we assume that in one period the read request reaches the target after the maximum delay $(M_i - 1)\Delta^{\max}$, and in all subsequent periods it reaches the target after the minimum delay $(M_i - 1)\Delta^{\min}$. The following result can then be proved using the same strategy as Lemma 7.

Lemma 22 *Consider an initiator issuing a delayed read for e_i bytes every period p_i. Then $\bar{\alpha}_i^0 \equiv (\bar{\delta}_i^0 = e_i(1 + \frac{(M_i-1)(\Delta^{\max}-\Delta^{\min})}{p_i}), \bar{\rho}_i = \frac{e_i}{p_i})$ is an arrival curve for subflow $\bar{f}_i^0 \equiv f_i^{M_i-1}$.*

Using the result of Lemma 22 we can first compute burstiness for $f_i^{M_i} \equiv \bar{f}_i^1$, and then for all remaining backward subflows. The remaining problem is how to determine $\Delta^{\max}, \Delta^{\min}$. In many cases, the value of $\Delta^{\max}$ can be bounded based on the bus topology and bridge specification using a framework similar to the one described in this section (for example, if all bridges that outputs subflows of f_i do not buffer any other transaction, $\Delta^{\max}$ can simply be set as the maximum of all RR_i^j as computed in Theorem 12). Otherwise, assuming that flow f_i has a deadline equal to its period p_i, a safe assumption is to set $\Delta^{\max} = \frac{p_i}{M_i-1}, \Delta^{\min} = \frac{r}{C}$. A third and possibly more efficient alternative would be to perform a fixed point iteration, starting from the described safe value and then setting a new $\Delta^{\max}$ at each step based on the computed forward delay. While we do not detail such technique in this paper, we plan to explore this direction as part of our future work.

4. Conclusions

In this paper, we have introduced an analysis to compute deterministic performance bounds for the PCI bus. While our presented analysis is based on network calculus framework, we believe that the presented methodology could also be applied to extract a model for the PCI and other COTS interconnections based on different analysis frameworks, such as real-time calculus [6] and delay algebra [4]. Finally, we are developing a tool to automatically extract schedulability equations and compute flow delays based on a high level description of the hardware architecture and logical communication.

References

[1] Conventional PCI 3.0, PCI-X 2.0 and PCI-E 2.0 Specifications. http://www.pcisig.com/specifications/.

[2] J.-Y. L. Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet Series*. Springer, 2001.

[3] R. Pellizzoni, B. D. Bui, M. Caccamo and L. Sha Coscheduling of CPU and I/O Transactions in COTS-based Embedded Systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, Dec 2008.

[4] P. Jayachandran and T. Abdelzaher. Delay Composition Algebra: A Reduction-based Schedulability Algebra for Distributed Real-Time Systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, Dec 2008.

[5] K. Hoyme and K. Driscoll. Safebus(tm). *IEEE Aerospace Electronics and Systems Magazine*, pages 3439, Mar 1993.

[6] L. Thiele, S. Chakraborty and M. Naedele. Real-Time Calculus for scheduling hard real-time systems. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 4, pp. 101104, 2000.

[7] E. Wandeler, L. Thiele, M. Varhoef and P. Lieverse. System architecture evaluation using modular performance analysis: a case study. In *International Journal on Software Tools for Technology Transfer*, vol. 8-6, pp. 649-667, 2006.