2012 Sixth IEEE/ACM International Symposium on Networks-on-Chip

Dynamic Flow Regulation for IP Integration on Network-on-Chip

Zhonghai Lu and Yi Wang
Department of Electronic Systems, School for ICT
KTH Royal Institute of Technology, Stockholm, Sweden
{zhonghai, yiwang}@kth.se

***Abstract*—Flow regulation is a traffic shaping technique, which can be used to achieve communication performance guarantees with low buffering cost when integrating IPs to network-on-chip architectures. This paper presents dynamic flow regulation, which overcomes the rigidity of static flow regulation that pre-configures regulation parameters statically and only once. The dynamic regulation is made possible by employing a sliding window based online flow (σ, ρ) characterization technique, where σ bounds traffic burstiness and ρ reflects the average rate. The characterization method is effective and can be implemented in hardware with small area and high speed. The resulting dynamic regulation can adaptively adjust the traffic regulation strength in response to real traffic workload scenarios. As such, it makes more efficient use of the system interconnect resources, leading to significant improvement in network performance.**

Keywords: Flow Regulation, IP Integration, Network-on-Chip

I. Introduction

To manage ever increasing complexity, advanced System-on-Chip (SoC) designs typically encompass a system integration process based on existing computation, storage and interconnect IPs . These IPs are designed separately, following a common interface, for example, AMBA, AXI, OCP-IP etc. Later, the computation and storage IPs are integrated together with the interconnect IP. While a common interface smooths the hardware integration, guaranteeing application performance during the integration remains an open challenge because of traffic diversity and unpredictability. This is becoming a critical issue as advanced SoC interconnects are moving from bus to network-on-chip (NoC) architectures in order to accommodate hundreds of IPs, which raise complexity and performance uncertainty in orders of magnitude [1].

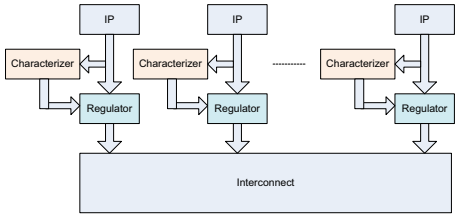

Fig. 1. Dynamic regulation with online characterization

To address the problem of IP integration on NoCs, traffic regulation was proposed by which traffic flows or streams can be shaped into desired characteristics before being admitted into the interconnect [2][3]. Figure 1 illustrates the flow regulation where a regulator is inserted between an IP and the interconnect to perform traffic shaping. Traffic regulation brings two main advantages: (1) It turns unknown or undesired traffic characteristics into desired ones, thus facilitating performance analysis and performance guarantees. (2) It can be used to decrease packet delay and buffer requirement since it can control network congestion status and reduce backlogs due to changing flows' timing and burstiness. To provide a solid foundation for the performance analysis and control, traffic regulation is preferably conducted under a mathematical formalism. (σ,ρ) based regulation, where σ bounds a flow's burstiness and ρ sustainable rate, is such a technique studied under *network calculus* [4][5][6], a flow-based queuing theory for performance guarantees in communication networks.

The (σ,ρ) based regulation requires characterizing flows' σ and ρ values before applying them to losslessly control the traffic admission. Depending on whether the parameters are determined statically or dynamically, we can apply static or dynamic regulation, respectively. While a static approach [2][3] is rigid and often cheaper to implement, it cannot adapt to dynamic scenarios. As the system gets more complex with more and more IPs communicating with each other, the system behavior is becoming more dynamic. In these cases, a dynamic approach is desired to enhance the network utilization and improve the application performance.

To realize dynamic traffic regulation requires dynamic traffic characterization. Based on a sliding window concept, we propose an online flow characterization technique, which samples incoming traffic with overlapped time windows and makes predictions to obtain a flow's (σ,ρ) values. With design optimizations, this technique can be implemented as a lightweight hardware module. Figure 1 shows the module as a *characterizer*. In our experiments with MMP (Markov Modulated Process) traffic flows on an NoC, we show the fidelity and adaptivity of our dynamic traffic characterization via comparing it with offline traffic profiling. We also demonstrate the delay efficiency of dynamic regulation in contrast to static regulation and no regulation.

Note that the static regulation is inflexible and may be overly restrictive, but it can be used to ensure per-flow worst-case delay bound through formal analysis. With the dynamic regulation, the traffic flows are not absolutely bounded, since the predication can only give an estimated upper bound. Violating the estimated bound is possible and allowed. As a consequence, the hard performance guarantee is lost and only

978-0-7695-4677-3/12 $26.00 © 2012 IEEE
DOI 10.1109/NOCS.2012.21

soft performance promise is possible.

The rest of the paper is organized as follows. Section II discusses related work. In Section III, we introduce the basics of network calculus and (σ,ρ) based regulation. Section IV details the sliding window based flow characterization from concept, design to implementation. In Section V, we describe the dynamic flow regulation. In Section VI, we report experiments and results. Finally, we conclude in Section VII.

II. Related Work

Network calculus was pioneered by Cruz [6] and Chang [5]. Since its creation, it has become a very active research area and been successfully applied to design QoS networks for ATM [4], Internet with differentiated and integrated services [4] [7], and recently for WSNs [8] etc. The stochastic version of network calculus can be found in [5][9]. Realtime calculus [10] combines network calculus with real-time scheduling theory to reason about the worst-case delay and backlog bounds under various scheduling policies (fixed priority, TDMA, EDF, FIFO, etc.) under different workload and service conditions.

Based on network calculus, traffic shaping was previously proposed for ATM and Internet to enable QoS guarantees for network users [4] [9], as it can control the traffic burstiness and rate from users. If a user injects traffic with a rate more than the agreed one, the excessive traffic may be dropped or sent out as best effort traffic. Based on such control mechanisms, network service providers can associate billing information with different service-level agreements for users.

Inspired by this technique, traffic shaping was introduced for SoC designs. In SymTA/S (Symbolic Timing Analysis for Systems) [11], traffic shapers were used to increase the minimum distance between events and thus reduce peak workload bursts so as to improve worst-case performance. In [12], greedy shapers were analyzed in distributed embedded systems and incorporated into a system-level modular performance analysis framework to allow deriving a timing guarantee for real-time traffic streams.

In [2][3], traffic shaping was proposed as a traffic regulation technique to deal with the QoS IP integration problem. As SoC applications typically do not allow dropping packets, lossless regulation is essential. In [2], regulation spectrum was formally defined to give a valid range for regulation parameters so as to shape (σ,ρ) flows without data loss. It is also shown that this spectrum can be exploited to improve delay and reduce backlog bounds. Because different flows may have conflicting regulation requirements due to sharing network resources, the problem of optimized regulation arises. In [3], the regulation problem was formulated with buffer optimization as objectives, and significant reduction in packet delay and required buffers can be achieved by solving the optimization problem. Both works [2][3] dealt with static regulation, requiring the characterization of traffic flows' (σ,ρ) values offline at design time.

Though offline characterization methods [13][14] such as static traffic analysis and trace-based profiling are possible, they are inflexible and can not capture traffic dynamism or only partially. Recently, dynamic traffic characterization and prediction has received increasing attention, thanks to its potential in guiding quality system design in achieving high performance and low power, as demonstrated in [15][16][17].

With a table driven approach, Kaxiras and Young [15] used coherence communication prediction in distributed shared-memory systems to speculate data needed by several processors and to deliver the data as soon as possible. In [16], Huang et al. proposed a table-driven predictor to predict end-to-end communication behavior in NoCs. Their method only predicted one future time interval and the predicted communication volume is either zero or the current quantity. In [17], a fuzzy logic based algorithm is used to search for similar traffic patterns in the traffic history to predict prospective data points. Since it aims to pinpoint the actual traffic volume and does not balance the consideration of current and past traffic information, it is very difficult to be accurate. The prediction accuracy for chaotic patterns with continuous behavior reaches up to 10 data points while for a discontinuous (high burstiness) real communication trace only up to two data points.

This paper proposes a hardware-oriented online flow characterization technique from concept, design to implementation. Our approach is based on a sliding window which can balance the consideration of current and history traffic information. Very different from the others, we aim to predict an *arrival curve* (refer to Section III-A) which provides an upper bound for the future traffic per time window rather than the actual data volume. Furthermore, our solution is designed in a way that is suitable for hardware implementation. Based on this online characterization, we further integrate it into a dynamic regulation architecture for NoCs.

III. (σ, ρ) based Flow Regulation

A. Network Calculus Basics

Network calculus deals with traffic flows and relies on two fundamental abstractions, *arrival curve* and *service curve*, for worst-case performance analysis.

In network calculus [4][6], a unicast traffic stream sent from a source to a destination is called a *flow*, which is denoted as $f(t)$ representing the *accumulated* number of bits transferred in the time interval $[0, t]$. Network calculus developed the abstraction of cumulative *arrival curve* $\alpha(t)$, also called envelop process, to upper-bound a traffic flow. A well-studied arrival curve is the linear function $\alpha(t) = \sigma + \rho t$, where σ bounds the traffic burstiness, and ρ the average (sustainable) rate. This affine function is very useful since it is simple and often enables to derive closed-form analytic results. Another abstraction is cumulative *service curve* $\beta(t)$ that models the minimum service behavior of a network element. A commonly used service model is the latency-rate function $\beta_{R,T}(t) = R(t-T)^+$, where R is the minimum service rate and T the maximum processing latency of the node [7]. Notation $x^+ = x$ if $x > 0$; $x^+ = 0$, otherwise.

Based on the two abstractions, network calculus delivers basic results in per-flow delay bound, backlog bound, and output arrival curve. Consider a single-node case. A flow $f(t)$

constrained by a (σ,ρ) arrival curve is served by a node guaranteeing a latency-rate service $\beta_{R,T}(t)$. According to [4], the maximum delay for the flow is bounded by Eq. (1) and the buffer required for the flow is bounded by Eq. (2):

$$\bar{D} = T + \frac{\sigma}{R} \tag{1}$$

$$\bar{B} = \sigma + \rho \cdot T \tag{2}$$

The output arrival curve can be described as $\alpha^*(t) = (\sigma + \rho T) + \rho t$. As can be observed, the output characterization follows the same affine model as the input flow. This nice property enables that the single node analysis can be extended to analyze cascaded multiple-node cases.

B. Flow Regulation

The (σ, ρ) traffic model is a powerful characterization which helps to ensure QoS guarantees in networks [4][5]. However, in reality, traffic flows are diverse and often they do not follow the (σ, ρ) envelop process. Because of this reason, traffic admission control has been extensively studied in order to shape an incoming traffic into the desirable flow characteristics. Specifically, the *leaky bucket* model [18] can be employed to shape an incoming traffic flow and make it conform to a (σ, ρ) envelop process.

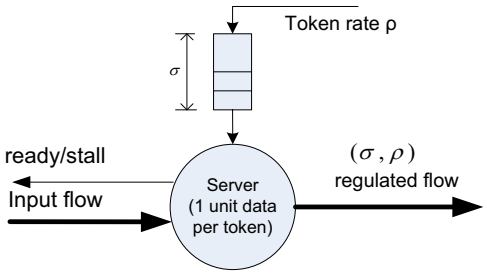

Fig. 2. A (σ, ρ) regulator model.

Figure 2 shows a leaky bucket (σ, ρ) regulator model. It works as follows: The server conditionally processes the incoming data stream. The condition is that, to process one unit of data, there needs exactly one token from the bucket, i.e., one token per unit of data. There is a "ready/stall" signal to control the service availability. The bucket has a maximum capacity of σ tokens. Initially, the bucket is full containing σ tokens and the token is filled at the rate of ρ. With this mechanism, during any time interval t, the generated number of tokens is bounded by $\sigma + \rho t$. Therefore, the amount of output traffic is enveloped by $\sigma + \rho t$.

C. Traffic Characterization

The leaky bucket model can be employed to shape traffic to conform to the (σ, ρ) characteristics. But, before shaping a traffic flow, we have to determine the two parameters, σ and ρ. This is the traffic characterization problem investigated in the paper. One way is to use *static traffic analysis*. Given an SoC application, the traffic characteristics over communication channels may be statically analyzed and annotated. As a result, a communication task graph, for example, a radio processing application [19], may have traffic bandwidth annotated on. This method is purely static, and thus only applicable to static SoC applications with good traffic knowledge.

Another way is to use *static traffic profiling*. Based on system simulation models, traffic flow characteristics may be obtained through analyzing simulated communication traces. However, each communication trace is specific to a particular system configuration, and to obtain good characteristics results, sufficiently long traces are required, leading the profiling process to be tedious and time-consuming.

In this paper, we propose a *dynamic traffic profiling* technique, which characterizes a flow's (σ, ρ) values online using hardware. While traffic profiling derives the characteristics of given traffic history, online characterizing involves also a prediction process, which projects the future traffic characteristics based on traffic history.

IV. Online Flow Characterization

In this section, we present our online flow characterization from concept, design to implementation.

A. Concept: Sliding Window based Characterization

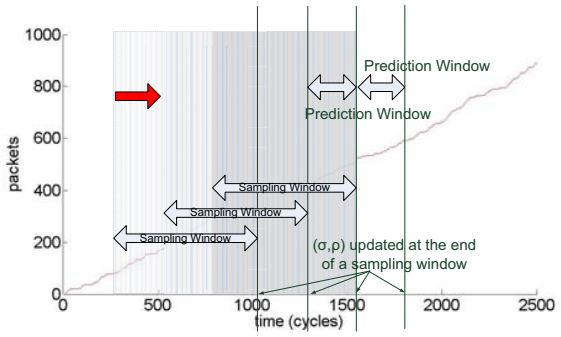

Fig. 3. Sliding window mechanism for traffic characterization

The characterization is based on a sliding window mechanism, as illustrated in Figure 3. The basic procedure involves three sequential steps: *sampling*, *characterizing*, and *shifting*. For each time window with length L_{sw}, an input flow is sampled cycle by cycle with values $(t, f(t))$ recorded; At the characterizing step (Section IV-B), the (σ, ρ) values are computed and updated at the end of each sampling window, and they are used for online regulation (Section V). After this, the sampling window is shifted forward at a step of L_{sw}/N, where N is a natural number. The step window is called *prediction window*. Its length equals to $L_{pw} = L_{sw}/N$, which gives the valid period for each projected (σ, ρ) pair. N is called *overlapping ratio*. Then the three steps repeat through the system execution.

Note that consecutive sampling windows may be overlapped. If $N = 1$, two consecutive sampling windows have no overlapping; If $N = 2$, two consecutive sampling windows overlap with a length of $L_{sw}/2$. Figure 3 shows the case with $N = 3$, where three consecutive sampling windows overlap

with a length of $L_{sw}/3$. Window overlapping is important to ensure that the window-by-window characterized results enjoy high continuity, taking into account both current and past states when predicting the next state.

B. Design: (σ, ρ) Characterization

We detail the characterizing process, first on ρ and then σ. We also describe design optimizations for characterizing σ.

*1) **Characterization of** ρ:* The online rate characterization consists of two steps: *rate profiling* and *rate prediction*. Both steps are conducted on a per-window timing basis.

Based on the window mechanism, the online profiling of ρ is straightforward. Within each sampling window, a flow is sampled at each time instant t_i for its traffic volume $f(t_i)$, where $t_i \in [1, L_{sw}]$, and these two values are maintained in two counters, one for each. At the end of each sampling window, ρ is computed by $f(L_{sw})/L_{sw}$ to obtain the per-window average rate of the flow.

The task of rate prediction is to speculate $\hat{\rho}_{n+1}$ based on previous profiled rate results ρ_n, ρ_{n-1}, $\cdots$, where n is the sequence number of a sampling window. Specifically, ρ_{n-1}, ρ_n, ρ_{n+1} represent the previous window, current window, and next window rate. The projected rate $\hat{\rho}_{n+1}$ is used to regulate the flow during time window $n+1$. In our current approach, we use ρ_{n-1} and ρ_n to project $\hat{\rho}_{n+1}$ in a simple way:

$$\hat{\rho}_{n+1} = \rho_n + (\rho_n - \rho_{n-1}). \tag{3}$$

The projected $\hat{\rho}_{n+1}$ is composed of a base value of ρ_n and an offset value of $\rho_n - \rho_{n-1}$, which captures possible rate variation. By using consecutive profiling results, this prediction exploits the continuity brought by the sliding-window mechanism to avoid abrupt change.

*2) **Characterization of** σ:* Similarly to rate characterization, the burstiness characterization also involves two steps: *burstiness profiling* and *burstiness prediction*.

The burstiness profiling involves finding the maximum burstiness value online such that $\sigma(t) = max\{\sigma(t_i) : \sigma(t_i) = f(t_i) - \rho \cdot t_i, 1 \le t_i \le L_{sw}\}$. This is done by first recursively performing a check on the critical instant, t_c, called *critical check*. A time instant t is considered *critical* if, at this instant, the traffic volume $f(t)$ surpasses the estimated traffic bound calculated with the previous t_c. The condition can be formulated as an inequality:

$$f(t_c) + \frac{f(t_i)}{t_i} \cdot (t - t_c) \ge f(t), \forall\, t \in [t_c,\, t_i]. \tag{4}$$

At the start of a sampling window, the first point t_1 is initiated as the first critical instant. As the time advances, the check is performed at each and every clock cycle until the end of the sampling window. If the critical check gets passed (the condition satisfied), nothing happens. Otherwise, the first t that fails the condition becomes the new t_c and the corresponding $f(t)$ will replace $f(t_c)$. At the end of the sampling window, the final value of $(t_c,\, f(t_c))$ will be used to calculate σ according to $\sigma = f(t_c) - \rho \cdot t_c$. Similarly to projecting $\hat{\rho}_{n+1}$, we use $\hat{\sigma}_{n+1} = \sigma_n + (\sigma_n - \sigma_{n-1})$ to project $\hat{\sigma}_{n+1}$.

The critical check is a key step for burstiness characterization. Condition 4 is recursive since it is performed for $\forall\, t \in [t_c,\, t_i]$ whenever t_i moves forward cycle by cycle. This is too computation intensive and impractical to implement in hardware. To simplify the condition and thus reduce hardware complexity, we use t_i to replace t, which turns the recursive check into a point-wise check. In this way, the condition is reduced to $f(t_c) + \frac{f(t_i)}{t_i} \cdot (t_i - t_c) \ge f(t_i)$. With simple transformation, it can be rewritten as

$$f(t_c) \cdot t_i \ge f(t_i) \cdot t_c. \tag{5}$$

This simplification enables an efficient hardware solution to conduct the critical check. We consider it acceptable, since the prediction itself is an approximation process.

C. Implementation: The Characterizer in Hardware

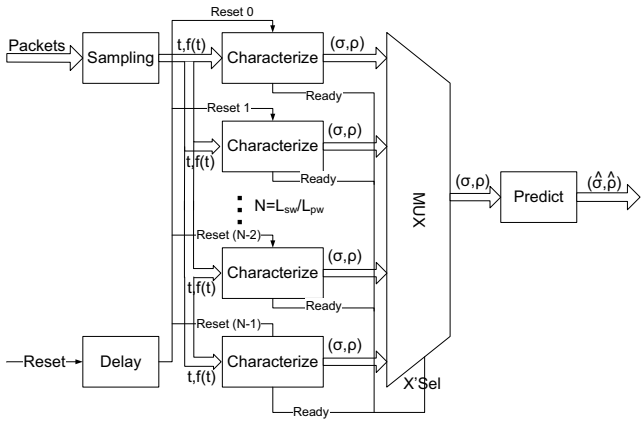

Fig. 4. The characterizer micro-architecture

Figure 4 shows the structure of the characterizer implementation. The *sampling* unit samples the values $(t, f(t))$ whenever an input packet arrives. Then this value is fed into N parallel *Characterize* units, where N is the overlapping ratio between the sampling window length L_{sw} and the prediction window length L_{pw}. The parallelization structure is to realize the overlapped window based concurrent characterization. The N Characterize units do not start simultaneously rather with an initial interval equal to L_{pw} one after another. This is realized by sequentially controlling the reset signal through the *Delay* unit. They sequentially compute the characterized (σ, ρ) values, based on the sliding window mechanism. Note that their executions are overlapping. A multiplexer with one-hot encoding as the control signal selects the results and feeds them to the *Predict* unit sequentially. This unit computes the estimated values of $(\hat{\sigma}, \hat{\rho})$ for each prediction window.

To optimize the design, we choose the sampling window length, L_{sw}, to be 2^m (m is a natural number). This brings two advantages. One is that we do not need to implement the boundary check and counter reset because overflow automatically achieves such effect. The other is that this value

can be directly used as the denominator for computing ρ and σ, and all divisions involved in the calculation of these two parameters can be replaced by a right shifter. The characterizer was synthesized in a 45 *nm* technology. It can run up to 2 GHz with an area of 12181 μm^2.

V. Dynamic Regulation

We integrate the online flow characterization into (σ, ρ) based regulation, resulting in dynamic flow regulation.

A. Overview of Dynamic Regulation

As already shown in Section I, Figure 1 presents an overall architecture of the dynamic regulation. There is one characterizer and one regulator per IP. The characterizer does the window-based online (σ,ρ) characterization as described in the previous section and uses the predicted (σ,ρ) values to configure the regulation parameters in the regulator. The regulator becomes a reconfigurable module which performs the leaky bucket regulation. As flows are continuously sent from IPs, (σ,ρ) values are dynamically updated window-by-window and used to shape traffic at run time. In view of this dynamic structure, the static regulation is a special case where there are no online characterizers, and it contains only regulators which are pre-configured with regulation parameters obtained at design time.

In fact, Figure 1 shows a *distributed* dynamic regulation architecture, where the characterizers make regulation decisions *locally* without global optimization. In contrast to such local-decision based distributed decision making, one can alternatively design and implement *centralized* decision making to globally optimize the regulation decisions [3]. In theory, this kind of global optimization is necessary as locally optimal decisions may lead to be globally suboptimal in performance and resource utilization due to lack of coordinations. However, such global optimization is not scalable for hardware implementation and thus preferably suits for static or semi-static regulation, since static decisions are made once via offline optimization.

B. The Reconfigurable Regulator Implementation

Following the leaky-bucket control mechanism (Section III-B), we implemented a dynamically reconfigurable regulator in HDL to quantify its hardware speed and area. In [20], a regulator implementation was discussed but only in theory. No hardware design and synthesis results were reported.

The reconfigurable regulator realizes dynamic regulation while admitting traffic into the interconnect. It contains three configuration hardware registers which store the regulation parameters, one for σ and two for ρ. A pair of integers is used to represent rate ρ, one as the numerator and the other denominator. All the three parameters are represented in 14-bit integers in the implementation. At the end of each time window, the (σ, ρ) values are updated by the characterizer.

For the regulator design, it is important to ensure that, given the same rate, tokens are evenly generated so as to remove unwanted bursts. For example, for $\rho = 0.5$, the regulator may receive values 2048/4096 or 1024/2048. A regulator configured with these two pairs of values should result in exactly the same token generation behavior. We use the following Algorithm 1 to achieve this.

Algorithm 1 Pseudo code of token generation

```
Let num be the numerator of ρ
Let den be the denominator of ρ
num ≤ den
Start:
compare = 0;
token = σ; //Admitting 1 packet consumes 1 token.
while TRUE do
  wait for clock rising edge;
  compare = compare + num;
  if compare ≥ den then
    compare = compare − den;
    if token < σ then
      token = token + 1;
    end if
  end if
end while
End
```

The numerator is self-added every cycle, and whether to generate a token or not is determined by the comparison between the numerator accumulation and the denominator. The algorithm is very simple, so the resulting ASIC can enjoy a high speed. After synthesized in a 45 *nm* library, the reconfigurable regulator can reach a maximum speed of 1.4 GHz, consuming 2238 μm^2.

C. Per-flow vs. Per-aggregate Regulation

In our system, there is one and only one pair of characterizer and regulator per IP. From an IP, there can be many flows sent to many different destinations. Different flows may have their own distinct characteristics. We can have two distinct regulation approaches:

1) Per-flow regulation: This means that each flow is treated differently. This is achievable by equipping a flow with a flow ID. The characterizer can be designed as a multi-flow characterizer that tracks and characterizes each flow individually online. A multi-flow regulator can be realized to implement the flow-sensitive regulation, regulating different flows according to their own regulation parameters.
2) Per-aggregate regulation: The characterizer and the regulator do not distinguish flows. All flows sent from the same IP are treated as a single flow *aggregate*. The characterizer profiles this aggregated flow, sends regulation decisions for the aggregate, and the regulator realizes the aggregate regulation.

Per-flow regulation can be more precise than per-aggregate regulation, but apparently, is less scalable in implementation. Suppose there are *n* nodes in the system and each node may

communicate with each other, then each node has up to n flows, resulting in a total of up to n^2 flows. With the per-flow regulation, the system implementation complexity is $O(n^2)$. While with the per-aggregate regulation, it is $O(n)$. In the following, we have adopted the per-aggregate regulation.

VI. Experiments and Results

We present experiments and results concerning the fidelity of our online characterization technique followed by the benefits of applying dynamic regulation in a NoC environment.

A. Effectiveness of Online vs. Offline Characterization

1) Experimental Purpose and Setup: The first experiment is to (1) validate the sliding window based online characterization against offline characterization; (2) investigate the effect of sampling window overlapping, as it is a key to ensure continuity in the predicted results.

We modeled the characterizer in Matlab. We used a two-state (on/off) MMP (Markov Modulated Process) [21] as the traffic model in the experiment. The reasons are two fold: (1) To validate the effectiveness of an online characterization technique, we need to have the traffic model known so that we can compare it with an offline characterization method. In other words, it is impossible or of no interest to validate a characterization method with completely random traffic; (2) The MMP can well represent traffic burstiness [21], which is a typical traffic characteristics in SoCs.

2) Effectiveness Result: We ran extensive MATLAB simulations to compare the effectiveness of online and offline characterization methods. We report detailed results for one case where the MMP traffic trace has a unit pattern length of 100 cycles, burst rate of 0.9 and burst proportion of 30%. The sampling window is set to 8192 cycles and the prediction window 2048 cycles. This means that four consecutive sampling windows overlap with each other.

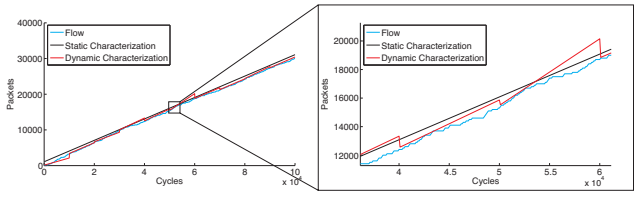

Fig. 5. Online characterization and offline characterization

Figure 5 illustrates the difference between the two characterization methods. Given complete MMP traces (the blue line), the offline characterization statically computes their (σ, ρ) values. This is to mimic the typical offline profiling method. Therefore, for a single trace, the offline method will result in only one arrival curve (the back line). The online prediction differs in two aspects. First, it does not require complete MMP traces beforehand. The profiling is based on window-by-window sampling and prediction. As can be observed, this online method results in one arrival curve per window (the red line). The resulting connected curve follows closely and adaptively to the dynamic traffic accumulation.

3) Effect of Sliding Window Overlapping: To show the impact of overlapped sampling windows, we conduct another set of tests in which we vary the ratio between the sampling window length and the prediction window length. In order to quantify the effect, we define a *deviation* metric, which measures the percentage of deviation occurrences among the total simulated cycles. We say that a deviation occurs whenever the real accumulated traffic volume at instant t_i is greater than that projected using the current arrival curve, i.e., $f(t_i) > \alpha(t_i)$.

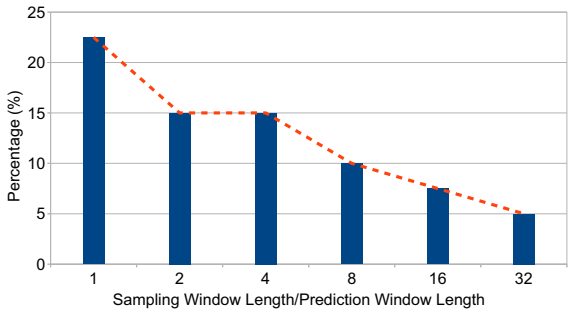

Fig. 6. Deviation in relation to window overlapping

Figure 6 shows the results with multiple overlapped windows. Clearly, as the overlapped windows increase, the deviation reduces. This means that with more historical information accounted, the projection can achieve higher continuity and thus higher fidelity. However, higher overlapping means more computation, which implies higher design complexity and thus a design tradeoff.

B. Effect of Dynamic vs. Static vs. No Regulation

1) Experimental Purpose and Setup: The second experiment is to compare the network performance of dynamic regulation against static regulation and no regulation. We shall investigate the performance gain with dynamic regulation in not only the observed maximum packet delay but also the average packet delay.

Our experimental platform is based on a cycle-accurate pin-accurate NoC using deflective routing [22] written in VHDL at the RT level. The synthesizable characterizer described in Section IV-C and the regulator in Section V-B are integrated onto the hardware modeling platform. With deflective routing, the router may transfer packets to unfavored links upon losing link arbitrations. Because of directing packets to other links rather than buffering them, the router requires only one buffer per port, minimizing area cost. The reason for using the deflective network is that the routing algorithm is fully adaptive to network congestion status. In comparison with a deterministically-routed network, the network performance is less sensitive to traffic regulation due to its capability of exploiting path diversity. Therefore it is a more challenging case for illustrating the effect of traffic regulation. The network ejects packets immediately once they reach destinations.

Figure 7 shows the experimental setup. It has 64 tiles (nodes) organized in an 8×8 mesh topology. Each tile can be

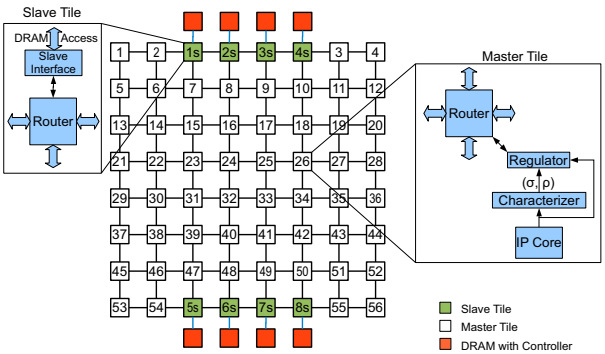

Fig. 7. A tile based 8×8 mesh with 56 masters and 8 slaves

either a master or slave tile. A master tile hosts a master IP, a router, together with a characterizer and a regulator to conduct regulation. A slave tile has a router connected to an off-chip memory module (DRAM controller plus DRAM). Master and slave tiles are numbered sequentially. In total, there are 56 masters numbered from 1, 2, ⋯ to 56, and 8 slaves numbered from 1s, 2s, ⋯ to 8s.

The performance metric we use is the packet delay, which is counted from when a packet is sent by its master until it is received by its slave. This delay includes two parts: *regulation delay* (delay due to regulation) and *network delay* (delay in the network).

In the following experiment, we use two kinds of traffic patterns: Synthetic traffic and Application benchmarks. For all traffic patterns, we compare three configurations:

- No regulation: The characterizer is disabled and the regulator provides a bypass to admit packets into the network as soon as possible.
- Static regulation: The characterizer is disabled and the regulator is configured once with constant regulation parameters (σ, ρ).
- Dynamic (online) regulation: The characterizer is enabled. The (σ, ρ) values are dynamically updated window by window and used to configure the regulator for run time regulation.

2) Selection of Operation Point: To show the effect of regulation, we need to select a proper network operating point with respect to the traffic injection rate. Flow regulation would make no sense if the network workload is too low, since the network has less contention and any holding back of traffic due to regulation is unnecessary and may result in negative impact on performance. On the other hand, the network workload should not be too high. Otherwise, the network gets saturated in throughput. The packet delay becomes exponentially up, and the network enters into an unstable state [21].

In view of the typical network behavior, we should select a proper traffic injection rate at which the network operates stably in the linear region in which, when the traffic injection rate increases, the network throughput linearly increases. From the regulation perspective, we should select a higher traffic injection rate. Combining these two requirements, we select a high traffic injection rate before but near to the network saturation point.

In the experiment, the injection rate per master is 0.166 packets/cycle. The temporal distribution of each aggregate flow follows the two-state MMP process described in the first experiment. We use the regulation parameters ($\sigma = 256$ packets, $\rho = 0.24$ packets/cycle) obtained from the offline characterization for the static regulation. For the dynamic regulation, the sampling and prediction window sizes still use 8192 and 2048 cycles, respectively.

3) Performance with Synthetic Traffic: We use hot spot traffic as the synthetic traffic pattern in which all 56 masters send packets to all 8 slaves with equal probability. Therefore the two slave regions are hot spots. Each master generates 8 flows, each targeting a slave. The 8 flows from the same master are treated as 1 aggregate. In total there are 448 (56×8) flows, 56 aggregates.

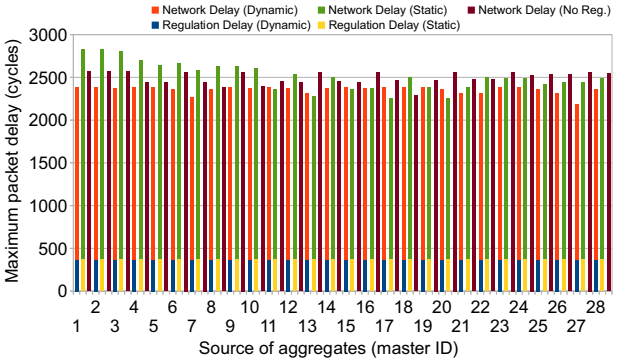

Fig. 8. Per-aggregate maximum packet delay comparison for hot spot traffic

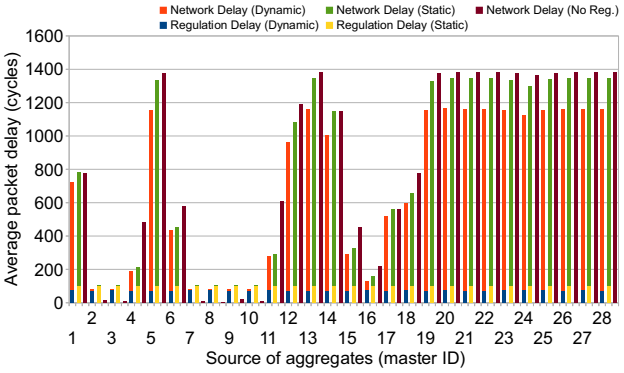

Fig. 9. Per-aggregate average packet delay comparison for hot spot traffic

Figures 8 and 9 compare the per-aggregate maximum and average packet delay for the 28 aggregates from the corresponding 28 masters, 1, 2, ⋯ to 28. We report the results only for half of the aggregates, because the other half (the other 28 aggregates from the other 28 masters, 29, 30, ⋯ to 56) have similar results due to the symmetric traffic pattern.

From Figure 8, we can observe: concerning the *per-aggregate maximum packet delay*: (1) The dynamic regulation outperforms the static regulation for 34 (61%) of the 56 aggregates, with the maximum and average reduction of 452

cycles (16%) and 146.8 cycles (5.8%), respectively; (2) The dynamic regulation outperforms no-regulation for 46 (82%) of the 56 aggregates. The maximum and average improvement is 435 cycles (17.4%) and 167.5 cycles (6.3%), respectively.

From Figure 9, we can observe: concerning the *per-aggregate average packet delay*: (1) The dynamic regulation outperforms the static regulation for all 56 aggregates, with the maximum and average reduction of 186 cycles (13.8%) and 108.6 cycles (14.5%), respectively. This average improvement is called *overall average packet delay improvement*, which is calculated by the sum of per-aggregate delay reduction ÷ number of aggregates; (2) The dynamic regulation outperforms no-regulation for 45 (80%) of the 56 aggregates. The maximum and average improvement is 332.8 cycles (54.6%) and 147.8 cycles (17.7%); (3) Neither static nor dynamic regulation improves the performance for nodes 2, 3, 7, 8, 9 and 10. This is because they are close to slaves, and the regulation delay occupies a larger portion of the total delay.

In the next, we show results with application benchmarks. All results (per-aggregate maximum and average packet delays) are consistent with the synthetic traffic study. Due to space limitation, we only report results for average packet delay.

4) Performance with Application Benchmarks: We experimented with the Stanford ParalleL Applications for SHared Memory benchmark suite 2 (SPLASH-2) to confirm the performance benefits brought by dynamic traffic regulation.

The first step is to obtain the communication traces for the SPLASH-2 benchmarks. We used the full-system simulator Simics together with GEMS [23] (for the memory system). According to Figure 7, we configured a CMP system with 56 cores (masters) and 8 slaves. Each core has L1 I/D Caches: 64KB, 4 way set-associative; L2 Cache: 256KB, 4 way set-associative, 64 Byte lines. The total memory size is 4 GB with each memory being 500 MB (4G/8). The cache coherency protocol follows the directory-based MOESI protocol. The configured CMP system runs Solaris 9 OS. After being compiled, the benchmark programs ran on top of the OS. The traces were recorded during the parallel execution phase, with each core running about 10 million instructions.

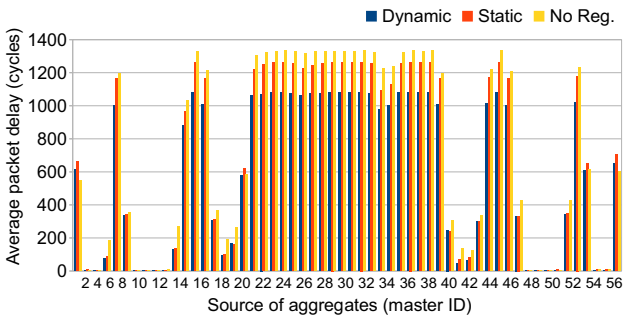

Fig. 10. Per-aggregate average packet delay comparison for FFT

Figure 10 compares the *per-aggregate average packet delay* for FFT. We can consistently observe: (1) Compared with the static regulation, the online regulation improves 53 (94.6%) of the 56 aggregates in per-aggregate average delay with a maximum of 183 cycles (14.5%) and an average of 90 cycles (26%). (2) Compared with no-regulation, the improvement is achieved for 54 (96.4%) of the 56 aggregates at maximum 259 cycles (19.5%) and on average 136 cycles (32%).

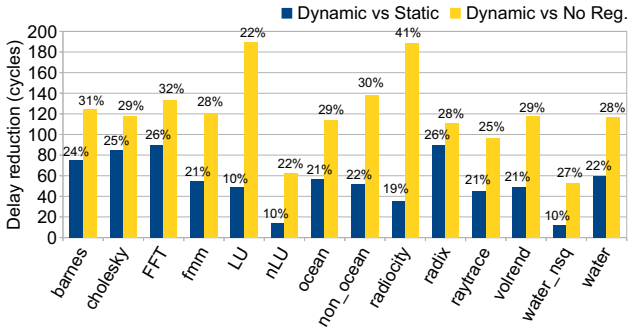

Fig. 11. Overall average packet delay reduction with dynamic regulation

To give a complete picture, we summarize the *overall average packet delay improvement* for the 14 benchmarks of SPLASH-2 in Figure 11. The dynamic regulation consistently improves the results of static regulation and no-regulation for all the benchmarks. Due to different traffic characteristics, the improvement is different case by case. In comparison with static regulation, the improvement in overall average packet delay ranges from 12 to 90 cycles, from 10% to 26% in percentage. In comparison with no regulation, it is from 53 to 190 cycles, from 22% to 41% in percentage. The case with less improvement is due to traffic locality which achieves much less network delay and the regulation delay takes a larger portion of the total delay.

C. Discussion

Static regulation can improve performance since it can control network admission and thus congestion [2][3]. The reason why static regulation performs worse than dynamic regulation in both maximum and average packet latency lies in its rigidity, which may result in two negative effects at some time intervals: (1) *Over regulation*: it does not allow supplying sufficient traffic even if the network can accept more workload; (2) *Under regulation*: it still injects more traffic even if the network cannot accept more workload. In contrast, the dynamic method can capture and adapt to traffic dynamism and thus avoid both over-regulation and under-regulation of the static method. With the dynamic regulation, the network workload can be adaptively adjusted reflecting closely the real workload scenarios. Consequently network contention is better mitigated, making more effective use of the interconnect resources.

To support the analysis above, Figure 12 shows the packet delay reduction of dynamic regulation against static regulation for FFT. For each regulation, we first recorded delays for 440,000 packets sent over time to the network. Then we sampled one packet every 100 packets, resulting in 4400

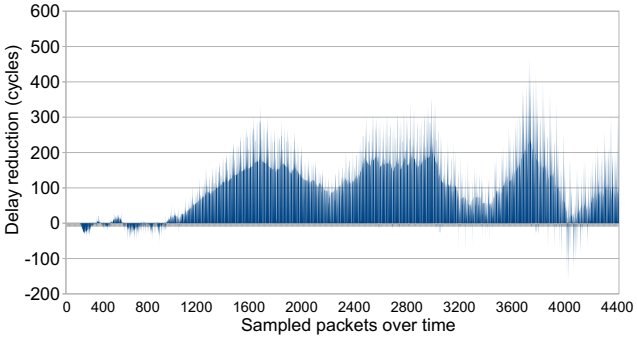

Fig. 12. Delay reduction of dynamic against static regulation for FFT

sampled packets and their respective delays. Figure 12 shows an un-even delay reduction, implying the effect of the dynamic adjustment. The static regulation is not always worse. For example, at the beginning of the system operation when the network is less loaded, the static regulation performs better since it allows more traffic injected into the network. This also suggests the need of combining the network status information for dynamic regulation as a future work.

VII. Conclusion

To overcome the weaknesses of static regulation, we have presented a coarse-grained dynamic regulation technique based on a sliding window based online flow characterization mechanism. With this technique, regulation decisions are dynamically made according to online traffic behavior. Using MMP flows, we have shown and discussed the fidelity of online characterization method together with the offline traffic profiling method. On a deflective routing NoC, our experiments with both synthetic traffic and SPLASH-2 benchmark traces show that the dynamic regulation can flexibly adjust regulation strength on demand. As a result, it makes more effective use of the system interconnect, achieving significant reduction in maximum and average packet delay. With the synthetic traffic, the maximum improvement in per-aggregate maximum packet delay reaches 16% against static regulation and 17.4% against no regulation. The maximum improvement in per-aggregate average packet delay is 13.8% against static regulation and 54.6% against no regulation. With the 14 SPLASH-2 benchmarks, the improvement in overall average packet delay reaches 26% against static regulation and 41% against no regulation.

The central idea for this work is to predict traffic characterization based on profiling the real traffic behavior, and use this information to conduct dynamic flow regulation. Another aspect to enhance dynamic regulation is to sense the network congestion status, for example on the buffer utilization or link activities, and then use such information to adaptively control the regulation strength online. We plan to investigate this in the future.

Acknowledgment

The research is sponsored in part by Intel Corporation through a research gift.

References

[1] Semiconductor Industry Association, "International technology roadmap for semiconductors (ITRS)," 2008 Update.

[2] Z. Lu, M. Millberg, A. Jantsch, A. Bruce, P. van der Wolf, and T. Henriksson, "Flow regulation for on-chip communication," in *Proceedings of the 2009 Design, Automation and Test in Europe Conference (DATE'09)*, Nice, France, April 2009.

[3] F. Jafari, Z. Lu, A. Jantsch, and M. H. Yaghmaee, "Buffer optimization in network-on-chip through flow regulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 29, no. 12, pp. 1973 – 1986, December 2010.

[4] J.-Y. L. Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. No. 2050 in LNCS, 2004.

[5] C. Chang, *Performance Guarantees in Communication Networks*. Springer-Verlag, 2000.

[6] R. L. Cruz, "A calculus for network delay, part I: Network elements in isolation; part II: Network analysis," *IEEE Transactions on Information Theory*, vol. 37, no. 1, January 1991.

[7] D. Stiliadis and A. Varma, "Latency-rate servers: A general model for analysis of traffic scheduling algorithms," *IEEE/ACM Transactions on Networking*, vol. 6, no. 5, pp. 611–624, October 1998.

[8] J. Schmitt, F. Zdarsky, and L. Thiele, "A comprehensive worst-case calculus for wireless sensor networks with in-network processing," in *IEEE Real-Time Systems Symposium (RTSS'07)*, 2007.

[9] Y. Jiang, "A basic stochastic network calculus," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, pp. 123–134, August 2006.

[10] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse, "System architecture evaluation using modular performance analysis - a case study," *Software Tools for Technology Transfer (STTT)*, vol. 8, no. 6, pp. 649 – 667, 2006.

[11] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis - the SymTA/S approach," *IEE Proceedings Computers and Digital Techniques*, vol. 152, no. 2, pp. 148 – 166, 2005.

[12] E. Wandeler, A. Maxiaguine, and L. Thiele, "Performance analysis of greedy shapers in real-time systems," in *Proceedings of Design, Automation and Test in Europe*, March 2006.

[13] L. Steffens, M. Agarwal, and P. Wolf, "Real-time analysis for memory access in media processing SoCs: A practical approach," in *Proc. of Euromicro Conference on Real-Time Systems*, July 2008, pp. 255 –265.

[14] A. Abdallah, W. Wolf, and G. Hellestrand, "Statistical characterization of execution time through simulation," in *International Workshop on Intelligent Solutions in Embedded Systems*, july 2008, pp. 1 –13.

[15] S. Kaxiras and C. Young, "Coherence communication prediction in shared-memory multiprocessors," in *Proceedings of Sixth International Symposium on High-Performance Computer Architecture (HPCA'06)*, 2000, pp. 156 –167.

[16] Y. Huang, K.-K. Chou, C.-T. King, and S.-Y. Tseng, "NTPT: On the end-to-end traffic prediction in the on-chip networks," in *Proceedings of 47th Design Automation Conference (DAC'10)*, June 2010, pp. 449–452.

[17] G. Thomas, B. Juurlink, and D. Tutsch, "Traffic prediction for NoCs using fuzzy logic," in *Proceedings of 2nd International Workshop on New Frontiers in High-performance and Hardware-aware Computing (HipHaC'11)*, Feb. 2011.

[18] P. P. Tang and T. yuan Charles Tai, "Network traffic characterization using token bucket model," in *Proceedings of the IEEE INFOCOM '99*, March 1999, pp. 51–62.

[19] Z. Lu and A. Jantsch, "TDM virtual-circuit configuration for network-on-chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 8, pp. 1021–1034, 2008.

[20] B. Akesson, L. Steffens, and K. Goossens, "Efficient service allocation in hardware using credit-controlled static-priority arbitration," in *Proc. of 15th IEEE Inter. Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'09)*, Aug. 2009, pp. 59 –68.

[21] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufman Publishers, 2004.

[22] E. Nilsson, M. Millberg, J. Öberg, and A. Jantsch, "Load distribution with the proximity congestion awareness in a network on chip," in *Proc. of the Design Automation and Test in Europe Conference*, 2003.

[23] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) toolset," *Computer Architecture News (CAN)*, September 2005.