Dynamic Traffic Regulation in NoC-Based Systems

Zhonghai Lu, *Member, IEEE*, and Yuan Yao

***Abstract*— In network-on-chip (NoC)-based systems, performance enhancement has primarily focused on the network itself, with little attention paid on controlling traffic injection at the network boundary. This is unsatisfactory because traffic may be over injected, aggravating congestion, and lowering performance. Recently, traffic regulation is proposed as an orthogonal means for performance improvement. Rather than as soon as possible admission, traffic regulation may hold back packet injection by admitting packets into the network only when the accumulated traffic volume at any time interval does not exceed a threshold. These regulation techniques are, however, often static, likely causing overregulation and underregulation. We propose dynamic traffic regulation to improve the system performance for NoC-based multi/many-processor systems-on-chip (MPSoC) and chip multi/many-core processor (CMP) designs. It can be applied to MPSoCs for intellectual property integration in an open-loop fashion by injecting traffic according to its run-time profiled characteristics. It can also be applied to CMPs in a closed-loop fashion by admitting traffic fully adaptive to the traffic and network states. Through extensive experiments and results, we show that both the open-loop and closed-loop dynamic regulation techniques can significantly improve the network and system performance.**

***Index Terms*— Chip multi/many-core processor (CMP), fuzzy control, multi/many-processor systems-on-chip (MPSoC), network-on-chip (NoC), traffic engineering.**

I. Introduction

REPLACING buses and crossbars, network-on-chip (NoC) is becoming the *de facto* pervasive interconnect for application-specific multi/many-processor systems-on-chip (MPSoCs) and general-purpose chip multi/many-core processors (CMPs). In customized computing, advanced SoC designs typically encompass a system integration process based on existing computation, storage, and interconnect intellectual property (IP) designs to manage ever increasing complexity. These IPs are designed separately by possibly different design teams or vendors, following a common interface, for example, AMBA, AXI, OCP-IP, and so on. Later, the computation and storage IPs are integrated together with the interconnect IP, such as NoC, to achieve common system-level design goals. In general-purpose computing, NoC also becomes an essential platform for CMPs to exploit core-level parallelism, since fat uniprocessors were no longer scalable in performance and power consumption [1]. Advanced high-performance CMPs integrate various components, such as processing cores, L1 and L2 caches (some also contain L3 caches, e.g., the IBM POWER7 & 8 multiprocessor) together onto an NoC architecture.

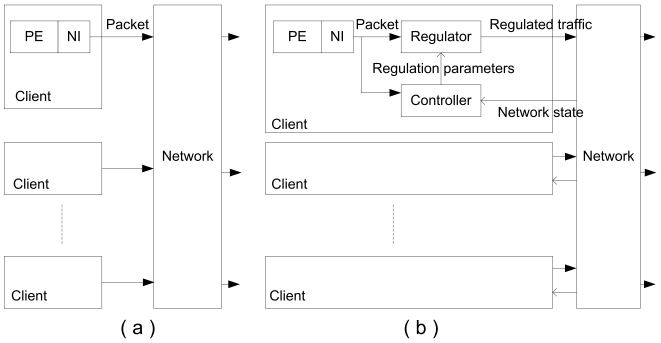


Fig. 1. Traffic injection from client to network (a) without regulation and (b) with regulation.

Since IP blocks, CPU cores, DSP cores, and memory nodes communicate with each other via the shared network, designing an efficient network in the NoC-based systems is of primary importance. Traditional approaches in performance enhancement have been network centric, where various techniques in topology, routing, flow control, switching, router microarchitecture, and circuit optimization have been developed over years to achieve the goal. However, as an orthogonal approach, traffic engineering has been paid less attention. This is unsatisfactory because the network performance becomes only the passive result of packet delivery and is not subject to control. The lack of injection control may in turn hurt the application performance. Recent research works [2], [3] have shown that traffic regulation, i.e., controlling traffic injection rather than free-to-go admission, can reduce packet delay and buffer requirement. However, both works employ static regulation requiring traffic knowledge a prior. While a static approach is rigid, suitable for guaranteed performance and often cheaper to implement, it cannot adapt to dynamic behavior that a typical complex system has and is thus less performing due to overregulation and underregulation. As the central suggestion of this paper, we advocate dynamic traffic regulation as a means of traffic engineering to adaptively control traffic admission into the network so as to enhance the network and system performance in MPSoCs and CMPs.

In contrast to conventional unregulated traffic injection in Fig. 1(a), Fig. 1(b) shows a generic scheme of regulated traffic injection in an NoC-based system. In the figure, PEs refer to processing elements in general, such as IP blocks, CPU cores, and DSP cores. NI stands for network interface which bridges a PE to a network router. PEs are the clients of the network, sending and receiving packets via NIs. Without regulation in Fig. 1(a), PE injects packets through its NI

Manuscript received January 5, 2016; revised April 26, 2016; accepted June 3, 2016. Date of publication July 26, 2016; date of current version January 19, 2017. This work was supported in part by the Swedish Research Council under Grant E0509501.

The authors are with the School of Information and Communication Technology, KTH Royal Institute of Technology, Stockholm 10044, Sweden (e-mail: zhonghai@kth.se; yuanyao@kth.se).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2016.2584781

to the network in a best-effort manner, i.e., as soon as the network can admit packets. There is no constraint on traffic injection from the client side. The limitation is enforced only from the network side through back pressure. With regulation in Fig. 1(b), PE injects packets through its NI to the network in a controlled manner, i.e., the packet injection has to conform to a traffic constraint in, for example, rate or burstiness or both. As such, the packet admission may be delayed in order to respect the traffic constraint. The constraint conformance is fulfilled by a pair of additional components per PE, namely, a regulator and a controller. The controller takes the traffic and possibly network congestion state information as inputs and makes decisions on regulation parameters, such as rate and burstiness as outputs. The regulator implements the regulation decision according to the regulation parameters from the controller. In short, the controller is the decision maker, such as a judge, and the regulator the decision implementer, such as a police.

To provide a solid foundation for performance control and analysis, traffic regulation is preferably conducted under a mathematical formalism. (σ, ρ)-based regulation, where σ bounds a traffic stream's burstiness and ρ sustainable rate, is such a technique [4]. The (σ, ρ) traffic regulation was originally developed in network calculus, a queuing theory for performance guarantees in communication networks [4]–[6]. It stipulates that during any time interval $(t_1,\ t_2]$, the amount of traffic denoted by $A_i(t_1,\ t_2)$ of a traffic arrival process $A_i(t)$ from source i entering the network is upper bounded by

$$A_i(t_1,\ t_2) \leq \sigma_i + \rho_i \cdot (t_2 - t_1). \tag{1}$$

In this paper, based on the (σ, ρ) regulation model, we elaborate two different traffic regulation schemes for MPSoCs and CMPs, namely, open-loop regulation for MPSoCs and closed-loop regulation for CMPs. The adaptations are needed because of different requirements and contexts of the two different kinds of systems. With open-loop regulation, we treat the IP blocks and network as black box in which only the output signals of the IP blocks are visible to design the regulation mechanism. With closed-loop regulation, we treat the cores and network as clear box where we can monitor the internal states of traffic and network, and then, the regulation can be fully adapted according to the traffic and network status.

The rest of this paper is organized as follows. We discuss the related work in Section II. In Section III, we give a system overview of the open-loop and closed-loop schemes and their common regulation models. We present the specifics for the open-loop scheme in Section IV and for the closed-loop scheme in Section V. In Section VI, we report full-system simulation experiments and results. Finally, we summarize and conclude in Section VII.

II. Related Work

1) $(\sigma,\ \rho)$ Model in Network Calculus: The (σ, ρ) model was first proposed in [4] as a special case (linear function) of an envelop process defining that the traffic volume within any time interval is constrained to a general function. The traffic envelop process was developed as arrival curve later in network calculus. Together with the concept of service curve modeling a network element, such as channels and arbiters, it is used to calculate the upper delay bound and maximum backlog of a flow using min-plus algebra. Because of its powerful expressiveness and simplicity, the (σ, ρ) model has been widely studied and adopted as a QoS contract specification between a network client and the network [5], [7], i.e., as long as the client injects traffic following the (σ, ρ) model, the network guarantees delay and backlog bounds. In this paper, we utilize the (σ, ρ) model to control traffic injection. Network calculus has recently been applied to NoCs [8]. Its stochastic version can be found in [6] and [9].

2) Traffic Regulation in Communication Networks and SoCs: Traffic regulation is a shaping technique, which turns unknown or undesired traffic characteristics into desired ones. In communication networks, traffic shaping was previously proposed for ATM and Internet to enable QoS guarantees for network users [5], [9], as it can control the traffic burstiness and rate from users. If a user injects traffic with a rate more than the agreed level of service, 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 and NoC designs. In 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 the 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 a data flow analysis [13], traffic regulators were used to shape prioritized real-time flows to control the minimum interval between prioritized events and hence avoid starvation.

In [2], [3], and [14], traffic shaping was proposed to deal with the QoS problem in NoCs. As computing applications typically do not allow dropping packets, lossless regulation is essential. In [2], regulation spectrum was formally defined to give a valid range of (σ, ρ) parameters for data lossless regulation. 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 a significant reduction in packet delay and required buffers can be achieved by solving the optimization problem. In [14], $(\sigma,\ \rho)$-based flow regulation is used in the data NoC of Kalray's MPPA-256 processor (a 28-nm chip integrating 256 processing cores and 32 resource management cores) to achieve guaranteed communication services in per-flow delay and bandwidth. The above works dealt with static regulation, meaning offline characterization and design-time configuration of traffic flows' (σ, ρ) values. Though inflexible and likely overrestrictive, static regulation facilitates ensuring determinis-

tic per-flow worst case delay bound through a formal analysis. Our dynamic regulation does not aim for performance guarantees. Rather, we show that it is also a powerful technique to improve the network and system performance by smartly controlling traffic admission.

3) Traffic Characterization: To enable application-dependent regulation, traffic characteristics have to be profiled. One way is to use the 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 [15] and an MPEG-2 video decoder [16], may have bandwidth annotated on its communication channels. This method is purely static, and thus only applicable to static SoC applications with good traffic knowledge *a prior*. Another way is to use trace-based traffic profiling [17]. Based on the system simulation models, traffic characteristics may be obtained through analyzing collected 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 summary, though offline characterization methods, such as static traffic analysis and trace-based profiling, are effective, they are inflexible and cannot or only partially capture traffic dynamism.

Dynamic traffic characterization has received increasing attention, thanks to its potential in guiding the quality system design for high performance and low power, as demonstrated in [18]–[21]. Bogdan and Marculescu [18] advocated a statistical physics inspired approach to express critical NoC traffic dynamics in nonstationary and fractality, which are not captured in widely used memoryless Poisson and monofractal self-similar models [21], [22]. By well capturing the time-dependent and space-dependent traffic characteristics, this approach significantly improved the performance evaluation results in, e.g., buffer overflow probability, packet delay deadline violation probability, and average packet delay (four orders of magnitude improvement), and opens up larger space for intelligently optimizing NoC designs. The multifractal traffic behavior has also been observed in the data request-reply latencies on NoC-based CMPs running various SPEC CPU2006 [23] programs [19]. Based on the model in [18], [20] proposed an efficient method to estimate traffic fitness parameters at runtime, achieving a high degree of accuracy without significant hardware overhead. A recent book [21] on modeling, analysis, and optimization of NoC architectures provides a short survey and more examples on traffic modeling and application.

In our open-loop regulation scheme, we propose an online traffic characterization technique. Our approach is based on a sliding-window mechanism, which can balance the consideration of current and history traffic information. Very different from others, we aim to predict an envelope process, which provides an upper bound for the future traffic per time window rather than the actual data volume. Based on this online characterization, we further embed it into a dynamic regulation architecture for NoC-based IP integration. Furthermore, our solution is approximate to be beneficial for hardware implementation.

4) Feedback-Based Fuzzy Control in NoCs: Besides using the traffic knowledge, the closed-loop regulation uses the network congestion state to adaptively control the regulation strength. Unfortunately, there is no formal model to precisely describe when a network is congested. Often, network congestion is recognized through the measurement metrics, such as packet latency, link utilization, or throughput. With a terminal instrumentation measurement approach, a network congestion state can be inspected by examining average packet latency and average throughput against the offered traffic [17]. Olukotun *et al.* [1] proposed a speculative method to detect congestion status and to accelerate single-thread applications in CMP. Chen *et al.* [24] developed a PID control theory-based DVFS technique integrated with an average memory access time-based monitoring technique to improve the system energy efficiency with a negligible impact on system performance. Since these works aim to assert the congestion status through the measurement values, they are difficult to be accurate in some situations. For example, suppose n cycles of average packet delay indicates network congestion, can we claim that the network is not saturated if the average packet delay becomes $n-1$ or $n-3$ cycles? Essentially, we consider that the network congestion recognition has a fuzzy nature.

In our closed-loop regulation scheme, we propose a fuzzy traffic regulation technique. Based on the fuzzy logic theory [25], it can recognize and exploit network congestion state to make new regulation policy. Very different from the others, we aim to mimic the behavior of an expert that validly recognizes the network state and controls the traffic admission. From the congestion control perspective, our traffic regulation can be viewed as a proactive rather than reactive approach to control network congestion.

5) Our Previous Works: This paper is a unification and extension of our two previous conference publications [26], [27]. The work in [26] presents a traffic regulation scheme for NoC-based IP integration in complex SoC designs. The work in [27] discusses a traffic regulation mechanism in NoC-based CMPs. Going beyond the two papers, we give a systematic account of the traffic regulation techniques for MPSoCs and CMPs. In particular, we unify the two separate techniques in one general traffic regulation framework, one as open-loop regulation for MPSoCs and the other as closed-loop regulation for CMPs. We also motivate the design choices in the different design contexts. Through the unification, we expose and contrast the strength, weakness, and complexity of the traffic regulation techniques in a structured way, thus deepening our understanding of their values and broadening their applicability. Another major extension is the new experiments, results, and new insights. Section VI is completely new. For the first time, we apply the two schemes to the same 64-core system running full-system simulations in GEM5 [28] with well-accepted PARSEC benchmark programs [29] to understand and contrast the benefits of both schemes. We show the significant potential of both schemes in network and system performance enhancements and further discuss how these benefits correlate with application characteristics. It is

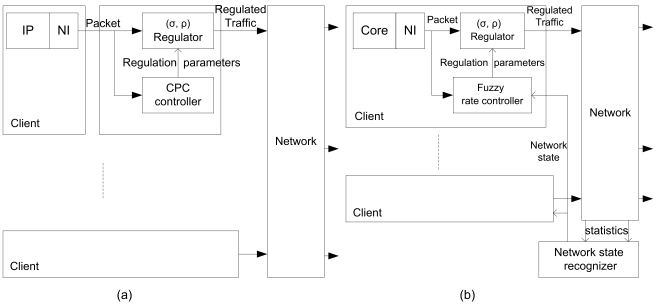


Fig. 2. Traffic regulation schemes. (a) Open-loop scheme in MPSoC. (b) Closed-loop scheme in CMP.

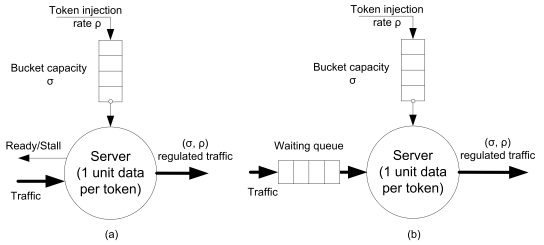


Fig. 3. Leaky-bucket (σ, ρ) regulator models. (a) Nonbuffering regulator. (b) Buffering regulator.

worth mentioning that the exposed new insights are impossible to obtain from the two individual conference papers.

III. Traffic Regulation in System Architectures

A. System Overview

Fig. 2 shows two variants of traffic regulation in NoC-based MPSoC and CMP system architectures. One is called open-loop regulation [Fig. 2(a)] because the characterize-predict-compare (CPC) controller makes the regulation decision using traffic characteristics and a preset network workload threshold without runtime feedback from the network. The other is called closed-loop regulation [Fig. 2(b)], because the fuzzy controller (FC) utilizes traffic characteristics and network feedback to make regulation decision. For both the schemes, the regulator that controls packet admission according to regulation parameters from the controller is common with a minor difference. While the open-loop scheme generally regulates both traffic burstiness (σ) and rate (ρ), the closed-loop scheme only controls rate (ρ), because traffic burstiness can be predetermined.

The NI realizes the general functions required to bridge an IP or core to the packet router. In particular, it conducts: 1) protocol conversion, i.e., transforming an IP interface protocol, e.g., AXI, to packet protocol and vice versa, so-called packetization (from message to packet) for transmission and depacketization (from packet to message) for reception; NI may further perform flitization (from packet to flit) and deflitization (from flit to packet) according to packet/flit size constraints; 2) message/packet queuing, i.e., buffering is needed for messages/packets to be sent and received due to temporary unavailability of communication or computation resources; and 3) multiplexing for forwarding and demultiplexing for receiving in the case of parallel organization of multiple queues.

In Fig. 2, the regulator is inserted between the NI and the network. This is a preferable organization, even though it is possible to have the regulator inserted before the NI. This, however, means that the regulator has to deal with protocol-related interfacing functions and is thus not favorable. We also note that the regulator could be designed as part of the functions in the NI. In this way, the regulator becomes transparent. In this paper, since we focus on regulation mechanisms, we make this component explicitly visible as a separate component.

In principle, both open-loop and closed-loop schemes can be applicable to application-specific MPSoCs and general-purpose CMPs. However, due to different design practices and contexts, one scheme can be preferable than the other. In particular, the open-loop regulation scheme is more appropriate to use for MPSoCs, because MPSoC designs typically involve an integration process based on IPs (ASIC blocks, DSPs, ASIPs, and interconnect) from different design teams or vendors. Once the IPs are provided as they are, it can be difficult to insert monitoring points and add new functions in the already-proven IPs and network. Therefore, a regulation scheme treats the IPs and network as black boxes (without touching their internal structures) is more suitable. The closed-loop regulation scheme can be more attractive to use in CMPs, because CMPs are typically in-house designs. The cores and the network are internally designed hardware modules. It is feasible and often desirable to add new functions to make designs as competitive as possible. Therefore, the network and the core can be treated as clear boxes, and hence, the internal network state and the traffic state information can be collected and analyzed as inputs to the controller.

Next, we describe the common regulation module. Then, we present specifics for the open-loop scheme in Section IV and for the closed-loop one in Section V.

B. (σ, ρ) Model-Based Traffic Regulation

Both open-loop and closed-loop schemes use the (σ, ρ) leaky-bucket model [6], [17] based regulation. Fig. 3(a) shows a nonbuffering leaky-bucket (σ, ρ) regulator model. It works as follows. The server processes the incoming data stream under the condition: to process one unit of data requires exactly one token from the bucket, i.e., one token for one unit of data. The ready/stall signal controls the service availability. If the token queue is nonzero, the ready/stall signal is asserted. Otherwise, the ready/stall signal is deasserted to stop traffic arrival. The bucket has a maximum capacity of σ tokens. Initially, the bucket is full containing σ tokens. Later, as long as the bucket is not full, tokens are filled in at a rate of ρ. With this mechanism, during any time interval Δt, the number of generated tokens is bounded by $\sigma + \rho \cdot \Delta t$. Therefore, the amount of output traffic is enveloped by $\sigma + \rho \cdot \Delta t$. When $\rho = 1$, the regulator can produce one token each cycle. This means a special case of no regulation that the service is always available, virtually providing a bypass path for the traffic.

Instead of using the ready/stall signal to enable/disable the traffic arrival, one can use a waiting queue to store the incoming packets if the token queue has no token available, as shown

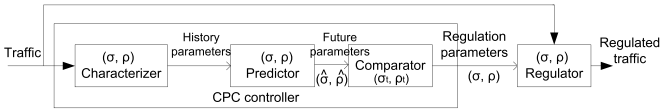


Fig. 4. Open-loop regulation scheme.

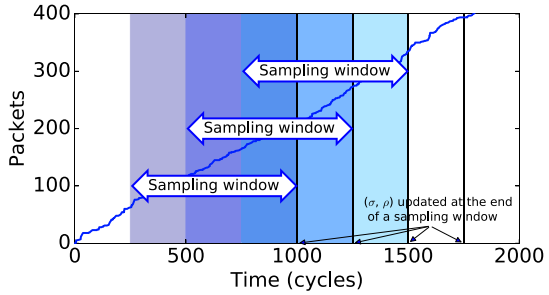


Fig. 5. Sliding window mechanism for traffic characterization.

in Fig. 3(b). The third option of achieving the (σ, ρ)-based regulation is that the PE or NI injects traffic autonomously (without external regulation) following the (σ, ρ) specification. This would be mostly suitable for newly designed IPs or cores with static or semistatic (σ, ρ) [multiple pairs of (σ, ρ) values for multiple operating modes] configuration.

We note that both regulator models can be used in the open-loop and closed-loop regulation schemes. However, the buffering regulator can be more suitable for the open-loop scheme because of the nature of open system design and integration in MPSoCs, while the lower cost nonbuffering regulator is preferable for the closed-loop scheme because of the nature of closed-system design in CMPs.

IV. Open-Loop Traffic Regulation for MPSoCs

A. Overview

In the open-loop regulation scheme, a CPC controller and a regulator work jointly in each node to achieve adaptive traffic regulation. The CPC controller performs online traffic (σ, ρ) characterization, prediction, and comparison to determine the regulation parameters of the regulator. As shown in Fig. 4, the controller contains three modules in series: a characterizer, a predictor, and a comparator. The characterizer does the time-window-based online (σ, ρ) characterization by sampling the incoming traffic. The predictor estimates the (σ, ρ) values of traffic in the next time window. The comparator compares the estimated $(\hat{\sigma}, \hat{\rho})$ values against preset (σ_t, ρ_t) threshold values to determine their output values, which are used to update the regulation parameters in the regulator. Next, we detail the operation principle of the CPC controller.

B. CPC Controller

1) Characterizer: The characterizer realizes a sliding-window-based characterization, as illustrated with a packet arrival process over time in Fig. 5. The basic procedure involves three sequential steps: sampling, characterizing, and shifting. For each time window with length L_{sw}, input traffic is sampled cycle by cycle with values $(t, f(t))$ recorded, where t is time and $f(t)$ is traffic volume. Afterward, the sampling window is shifted forward with a step of L_{sw}/N, where N is a natural number defining the overlapping ratio of consecutive sampling windows. The step window length gives the valid period for each projected $(\hat{\sigma}, \hat{\rho})$ pair, which is used as input to the comparator. Hence, the step window is effectively the regulation window. The three steps repeat through the system execution. Note that, when $N > 1$, consecutive sampling windows overlap. Fig. 5 shows the case with $N = 3$, where three consecutive sampling windows overlap with a length of $L_{sw}/N = 250$ cycles ($L_{sw} = 750$ cycles and $N = 3$). 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. With more historical information counted, the projection can achieve higher continuity and thus higher fidelity [26]. However, higher overlapping ratio means higher computation complexity and buffering cost, thus implying a design tradeoff.

a) Characterization of ρ: Based on the window mechanism, the online profiling of ρ is straightforward. Within each sampling window, a traffic stream is sampled at each time instant t_i for its traffic volume $f(t_i)$, where $t_i \in [0,\ L_{sw}]$, and these two values are maintained in two counters, one for each value. At the end of each sampling window, ρ is computed by $(f(L_{sw}) - f(0))/L_{sw}$ to obtain the per-window average traffic rate. $f(0)$ is reset to 0 for each sampling window, and thus, only the incremental $f(L_{sw})$ above $f(0)$ is recorded.

b) Characterization of σ: The burstiness profiling involves finding the maximum burstiness value online, such that $\sigma(t) = \max\{\sigma\ (t_i) : \sigma\ (t_i) = f(t_i) - \rho \cdot t_i, 0 \le t_i \le L_{sw}\}$. This is done by first recursively performing a critical check on a critical instant, t_c. 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

$$f(t_c) + \frac{f(t_i)}{t_i} \cdot (t - t_c) \ge f(t) \quad \forall\ t \in [t_c,\ t_i]. \tag{2}$$

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 $\sigma = f(t_c) - \rho \cdot t_c$.

The critical check is a key step for burstiness characterization. Inequality 2 is recursive since it is performed for $\forall\ t \in [t_c,\ t_i]$ whenever t_i moves forward cycle by cycle. This is computation intensive and impractical to implement in efficient hardware. To simplify the condition and thus reduce hardware complexity, we use t_i to replace t, which turns the recursive check into a pointwise check. In this way, the condition is reduced to $f(t_c) + (f(t_i)/t_i) \cdot (t_i - t_c) \ge f(t_i)$. With simple transformation, it can be rewritten as

$$f(t_c) \cdot t_i \ge f(t_i) \cdot t_c. \tag{3}$$

This simplification enables an efficient hardware solution to conduct the critical check (see the details in [26]).

2) Predictor: The predictor projects the values of rate and burstiness for the next time window.

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

$$\hat{\rho}_{n+1} = \rho_n + (\rho_n - \rho_{n-1}). \tag{4}$$

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

Similarly, to projecting $\hat{\rho}_{n+1}$, we use $\hat{\sigma}_{n+1} = \sigma_n + (\sigma_n - \sigma_{n-1})$ to project the next-window burstiness $\hat{\sigma}_{n+1}$.

3) Comparator: The projected $(\hat{\sigma}, \hat{\rho})$ values enter into a comparator, which determines the final (σ, ρ) values to the regulator. The comparator has a reference rate value ρ_t and a reference burstiness value σ_t, which are predetermined thresholds indicating an optimal network operating point. If $\hat{\rho} > \rho_t$, then $\rho = \rho_t$, meaning that the traffic admission rate is to be temporally reduced; otherwise, $\rho = \hat{\rho}$, meaning an adaptive rate control. If $\hat{\sigma} > \sigma_t$, then $\sigma = \sigma_t$, meaning that the traffic burstiness is temporally reduced; otherwise, $\sigma = \hat{\sigma}$, meaning an adaptive burstiness control.

The thresholds (σ_t, ρ_t) are predetermined offline. The burstiness threshold σ_t in either packet or flit can be set according to the system architecture characteristics. In MPSoCs, if the NI uses, e.g., the AXI protocol which allows at maximum 16 outstanding transactions, we set σ_t to 16 messages. If one message is encapsulated into one packet which has up to four flits, then σ_t is set to 16 packets or 64 (16×4) flits. In the cache-based CMPs, σ_t depends on the cache line size and packet/flit size. One cache line as one message is often encapsulated as one packet. If one packet has a maximum of eight flits, then σ_t is set to eight flits. The injection rate threshold ρ_t can be analyzed or determined empirically from network simulation. Since uniform random traffic can stress network uniformly, it facilitates load balancing and achieves ideal performance in average delay and throughput [17]. We inject uniform traffic to a given network architecture, with each node injecting traffic at the same rate and with the same probability to all other nodes. As a typical network behavior, network throughput grows linearly with the injection rate until network saturation [17]. Then, we record an injection rate, which nearly saturates the network as the injection rate threshold ρ_t measured in either packets or flits per cycle.

V. Closed-Loop Traffic Regulation for CMPs

A. Introduction to Fuzzy Processing

The closed-loop traffic regulation scheme is based on a fuzzy logic theory [25]. We first introduce the fuzzy processing procedure before detailing our fuzzy regulation scheme.

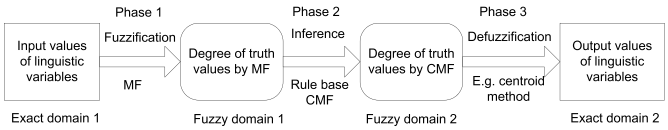


Fig. 6. Three-phase procedure of fuzzy processing.

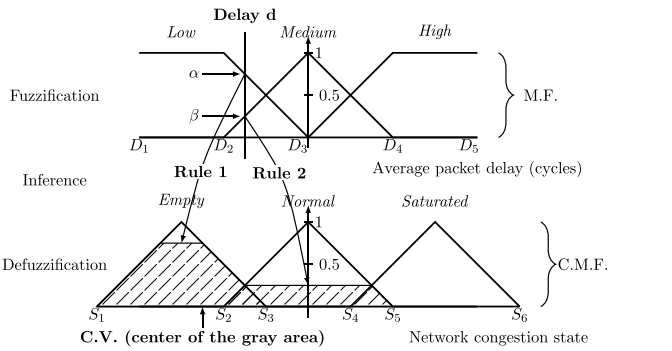


Fig. 7. Fuzzy network status recognition.

1) Procedure: Fig. 6 shows the three-phase (stage) procedure of fuzzy processing. With fuzzification, the input values of linguistic values in the exact logic domain are translated to the degree of truth values in the fuzzy logic domain according to a set of member functions (MFs). In the inference phase, the degree of truth values in the fuzzy logic domain is mapped to those in another fuzzy logic domain represented by consequence MFs (CMFs) using a set of rules. In the defuzzification phase, the degree of truth values are translated to the output values of linguistic values in another exact logic domain using a calculation method, e.g., the centroid method. In the following, we exemplify the fuzzy logic concepts, processing procedure and also explain the centroid method.

2) Example: Our example is to show the relationship between average packet delay and network congestion state. This can be better handled by using fuzzy logic, because there is no exact correspondence between an average packet delay and the network congestion state. Rather, a delay value gives a degree of truth about the network congestion state.

Fig. 7 shows the fuzzy network status recognition based on the packet delay. The fuzzy set is a pair $(\mathcal{U}, M)$, where $\mathcal{U}$ is a set of exact values (average packet delay ranging from D_1 to D_5) and M a set of MFs with the mapping $m_i : \mathcal{U} \rightarrow [0, 1]$, $m_i \in M$. In the figure, there are totally three MFs for assessment of low, medium, and high delay, respectively. Each MF maps the measured delay value to a degree of truth value in the range [0, 1]. This is the fuzzification phase. Note that one exact value can invoke multiple MFs. For example, average delay d is mapped to two degree of truth values, α and β, by MFs for low delay and medium delay, respectively. These degree of truth values further invoke rules 1 and 2 (illustrated by solid lines), which show the consequences of the delay. For example, if average packet delay is low, then network is

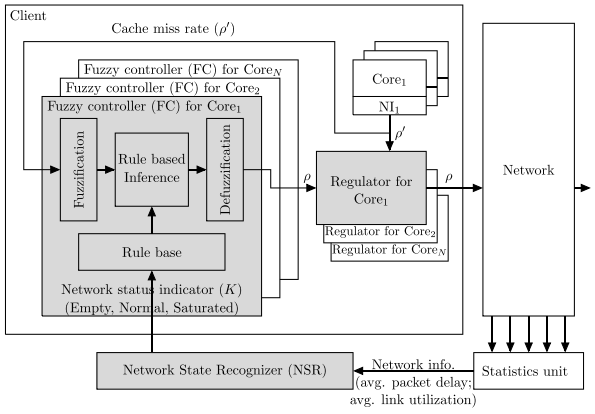


Fig. 8. Fuzzy control-based closed-loop traffic regulation.

empty. This is the second phase called inference using the rule base.

To quantify the consequence, each rule is assigned a CMF, which defuzzifies all consequences of the invoked rules back to one consequence value (CV). This phase is called defuzzification. CMF can be the same or different shapes as the corresponding MF, but often with both ends closed [30]. There are dozens of approaches to implement defuzzification, each with various advantages or drawbacks [30]. A most popular one is the centroid method, in which the center of mass of the results provides the CV. The mass (gray area in Fig. 7) of each CMF is determined by the degree of truth value from the corresponding MF. It is shown in Fig. 7 that given the average packet delay d, we can say that the network is in $Empty$ state, since the final CV falls into the $Empty$ CMF. If the CV falls into the intercrossed section $[S_2, S_3]$, the network status can be interpreted in either way.

B. Overview of Closed-Loop Traffic Regulation

In the closed-loop regulation scheme, the controller is a fuzzy rate controller, which uses the cache miss rate as input traffic rate information. Together with the network congestion state information, it determines the controlled rate for the outgoing traffic. The traffic burstiness is predetermined according to the system architecture parameters (cache block size and packet/flit size). The fuzzy controller (FC) uses the output rate to configure the rate parameter of the $(\sigma,\ \rho)$ regulator.

As shown in Fig. 8, three kinds of components: FC, network state recognizer (NSR), and regulator collaborate to achieve fuzzy regulation. Note that the NSR is one per CMP, but the FC/regulator is one per core. The NSR detects network congestion state from the network performance statistics (link utilization and packet delay), and the FC updates new regulation policy (ρ) of the regulator based on both the network congestion state and the arrival rate (ρ') of unregulated traffic of the local processing core.

Here, we use a fuzzy control theory as an instance to realize the closed-loop regulation mechanism. Other control theories, e.g., PID, can also be applied. We choose fuzzy control, because the network congestion state and traffic regulation strength do not necessarily need precise quantification and thus can be properly expressed in fuzzy logic. Another consideration is that fuzzy control can be more tolerable to the control decision-making latency and the latency for the control decision to take effect in the time-window-based regulation scheme, since the control decision is not meant to be precise, and each decision is made for a relatively large window size.

C. Traffic Rate Characterization

Before detailing the FC, we shortly describe the input rate (ρ') characterization, which is still based on the sliding-window mechanism. In the cache-based CMPs, network data message generation is always indicated by cache read/write misses. Since data traffic is originated from cache misses, the average cache miss rate [31] indicates how often network messages will be launched. In the state-of-the-art nonblocking/lock-up free caches, when an L1/L2 cache miss occurs, an access request to the L2/DRAM is created by first allocating a miss status holding register (MSHR) entry [17]. As each cache miss results in an entry in the MSHR, we monitor the number of allocated MSHR entries per sampling window, obtaining message arrival rate (one cache line is one message), which is in turn transformed into packet arrival rate. In the NoC-based CMPs, one message is usually encapsulated into one single-flit or multiflit packet. In this case, a message arrival rate equals to packet arrival rate.

D. Fuzzy Controller

1) Function: The FC takes the unregulated traffic arrival rate (ρ') and network congestion state from the NSR as inputs, and generates the target injection rate (ρ, regulation rate) as output. Here, the network congestion state information is used as an additional input to adjust the regulation decision.

It operates as follows. At the end of each sampling window, three stages are invoked in each FC: an input stage (fuzzification), a processing stage (inference using rule base), and an output stage (defuzzification). In the input stage, inputs (injection rate ρ' of the unregulated traffic from each core) are mapped by the appropriate membership functions to degree of truth values. In the processing stage, appropriate rules are invoked and a result for each rule is generated. In this stage, the network status indicator K, which has three values: $Empty$, $Normal$, $Saturated$, generated by the NSR, influences the mapping of each rule. The principle is that if the network is $Empty$, the traffic regulator casts a relaxing policy; if the network is $Saturated$, the traffic regulator tightens up the incoming flow; if the network is $Normal$, the traffic regulation policy casts no effect. Finally, the output stage converts the combined results back into a new regulation policy, which is used to update the old one in the leaky-bucket regulator. Next, we detail the three stages of the FC.

2) Fuzzification: Let m_i be the ith MF of the linguistic variable ρ' defined over the universe of discourse $\mathcal{P}'$. In our design, ρ' denotes unregulated arrival rate and $\mathcal{P}'$ all the possible values of ρ'. The fuzzy set S of the FC is a pair: $S = (\mathcal{P}', M)$, where $M = \{m_i : i = 1, 2, \ldots, N\}$ (N is the number of MFs). We assume that $\mathcal{P}' = [0, \infty]$ and $N = 7$. There are in total seven MFs representing an arrival rate being

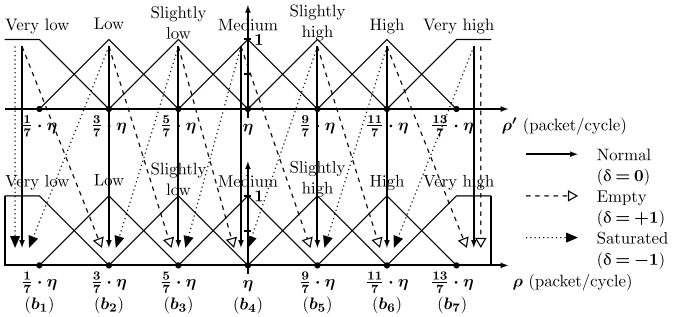


Fig. 9. Fuzzification through seven membership functions with different mappings ($\delta = 0, \pm 1$).

possibly very low, low, slightly low, medium, slightly high, high, and very high. Similarly, let c_j be the jth CMF of the linguistic variable ρ defined over the universe of discourse $\mathcal{P}$. Here, ρ denotes regulated injection rate and $\mathcal{P}$ all the possible values of ρ. Similarly, $j \in [1, 7]$ and $\mathcal{P} = [0, \infty]$.

During fuzzification, S samples of ρ' are recorded and denoted by ρ'_k, $k = 1, 2, \ldots, S$. Furthermore, the degree of truth of each sample is denoted by $m_i(\rho'_k)$, where m_i is the ith MF. The mass of each $m_i(\rho'_k)$ in the corresponding CMF is denoted by $c_j(m_i(\rho'_k))$, where c_j is the jth CMF.

3) Inference Using Rule Base: As exemplified in Section V-A, the mapping of the inputs to the outputs is characterized by a set of rules, or formally, in the If–Then form. In FC, the rules in the rule base have the form

$$\textbf{IF } \rho' \textbf{ is } m_i(\rho'), \ \textbf{THEN } \rho \textbf{ is } c_j(m_i(\rho')), \ i, \ j \in [1, 2, \ldots, 7].$$

By iterating i and j, we get a set of linguistic rules that specify how to regulate the traffic.

The inferencing of each rule in the rule base is influenced by the network status. Network status indicator $K \in [Empty, Normal, Saturated]$ determines the extent to which rule is relevant to the current situation. In particular, the mapping principle is as follows.

1) When K is $Empty$, mapping of $m_i(\rho'_k)$ to $c_j(m_i(\rho'_k))$ is done by $f : m_i \to c_j, j = i + \delta, \delta > 0$. It means that the network can accept more packets and increasing of ρ is accepted. One example of $\delta = +1$ is shown by the dashed lines in Fig. 9.
2) When K is $Normal$, mapping of $m_i(\rho'_k)$ to $c_j(m_i(\rho'_k))$ is done by $f : m_i \to c_j, i = j$, and is shown by the solid lines in Fig. 9. It means that the network works fine with the current arrival rate ρ', and the traffic just gets through without regulation.
3) When K is $Saturated$, mapping of $m_i(\rho'_k)$ to $c_j(m_i(\rho'_k))$ is done by $f : m_i \to c_j, j = i + \delta, \delta < 0$. It means that the network tightens up the injection rate to recover from saturation. One example of $\delta = -1$ is shown by the dotted lines in Fig. 9.

From the above mapping rules, we can observe that $|\delta|$ captures the regulation strength. A small $|\delta|$ results in mild regulation while a large $|\delta|$ sharp regulation. Note that η in Fig. 9 denotes the long-term average injection rate of traffic source obtained offline. In the MFs and CMFs, η is used as a reference to determine whether an arrival rate is considered to be one of the seven possible values, namely, very low, low, slightly low, medium, slightly high, high, and very high.

4) Defuzzification: During defuzzification, the centroid method is used to combine $c_j(m_i(\rho'_k)), i, j, \in [1, 2, \ldots, 7]$ into a single result by the following formula:

$$\rho = \frac{\sum_{j,i,k}^{N,N,S} b_j \ c_j(m_i(\rho'_k))}{\sum_{j,i,k}^{N,N,S} c_j(m_i(\rho'_k))} \tag{5}$$

where b_j is the center of c_j. The new ρ is passed to configure the traffic regulator, and the new regulation policy becomes effective during the next regulation window.

E. Network State Recognizer

The NSR takes both average link utilization and average packet delay as inputs to judge the network status K as output. The NSR design is also based on fuzzy logic, due to the fact that it is hard to evaluate NoC congestion status by the accurate mathematical/empirical model. We monitor network congestion state through online collected network statistics (average packet delay and average link utilization) by the statistics unit shown in Fig. 8. Similar to FC, NSR is also realized in three stages: fuzzification, inference using rule base, and defuzzification. The difference lies in that the NSR makes fuzzy control decision based on two inputs while the FC on one input. Details about the NSR can be found in [27].

VI. Experiments and Results

A. Experimental Setup

1) Target System Platform: We first introduce the target 64-core architecture we exploit through all experiments in this paper. Fig. 10 shows a typical architecture for a 64-core system, where the NoC interconnects PEs, NIs, traffic regulation modules, and on-chip routers. Each PE contains a core (a CPU core with its private L1 cache) and a secondary on-chip shared L2 cache bank. Routers are organized in a popular mesh topology on which the XY routing algorithm is implemented to achieve simplicity and deadlock free. There are eight memory controllers, which are connected to the middle four nodes on the top and bottom rows for architectural symmetry. To support cache coherency, a directory-based MOESI protocol is implemented. On the target system, once a core issues a memory request, the private L1 is first checked for the presence of the requested data. If this fails, depending on the location of the data block, the request is then forwarded to the local shared L2 or via the NoC to remote shared L2/DRAM.

2) Simulation Platform Configuration: We implemented and integrated our open-loop and closed-loop traffic regulation mechanisms with the timing-detailed full-system simulator **GEM5** [28], which simulates the target many-core system in Fig. 10 with configurations summarized in Table I.

Fig. 11 shows the open-loop and closed-loop experimental setups with corresponding timing measurement points. In Fig. 11(a), the open-loop setup uses an input buffer to decouple the traffic source (PE and NI) from the network. Ideally, to achieve complete traffic source-network decoupling,

TABLE I
SIMULATION PLATFORM CONFIGURATION

Item	Amount	Description
Processor	64 cores	Alpha based 2.0 GHz out-of-order cores. 32-entry instruction queue, 64-entry load queue, 64-entry store queue, 128-entry reorder buffer.
L1-Cache	64 banks	Private, 32 KB per-core, 4-way set associative, 128 B block size, 2-cycle latency, split I/D caches, 32 MSHRs.
L2-Cache	64 banks	Chip-wide shared, 1 MB per-bank, 16-way set associative, 128 B block size, 6-cycle latency, 32 MSHRs.
Memory	8 ranks	4 GB DRAM, 512 MB per-rank, up to 16 outstanding requests for each processor, 8 memory controllers.
NoC	64 nodes	8×8 mesh network. Each node consists of 1 router, 1 network interface (NI), 1 core, 1 private L1 cache, and 1 shared L2 cache. X-Y dimensional routing. Router is 2-stage pipelined, 6 VCs per port, 4 flits per VC. 128-bit data path. Directory based MOESI cache coherence protocol. One cache block consists of 1 packet, which contains 8 flits. One coherence control message consists of 1 single-flit packet.
Regulation	–	16,384 (2^{14}) cycles of sampling window, 4096 (2^{12}) cycles of regulation window. The regulation module is one for each PE. The network state recognizer (NSR) and the statistics unit are one for the whole chip.

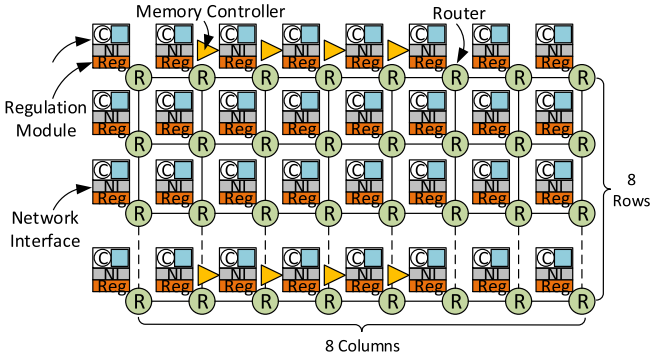


Fig. 10. 64-core architecture.

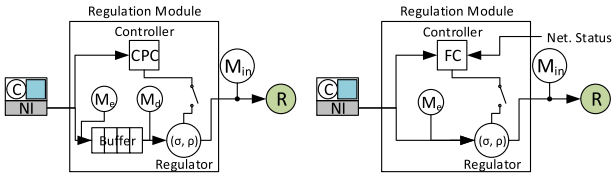


Fig. 11. Open-loop and closed-loop setups. (a) Open-loop setup. (b) Closed-loop setup.

TABLE II
BENCHMARK CHARACTERISTICS AND PROBLEM SIZE

Application	Problem size (all `simlarge`)	Injection rate	Group
blackscholes	65,536 options	low	1
bodytrack	4 frames, 4,000 particles	low	
dedup	184 MB data	low	
facesim	80,598 particles, 372,126 tetrahedra	mid	2
ferret	256 queries, 34,973 images	mid	
fluidanimate	5 frames, 300,000 particles	mid	
freqmine	990,000 transactions	mid	
canneal	400,000 netlist elements	high	3
streamcluster	16,384 input points, 128 point dimensions	high	
swaptions	64 swaps, 20,000 simulations	high	
vips	2,662×5,500 pixels	high	
x264	128 frames, 640×360 pixels	high	

an infinite input buffer should be used, albeit it is unrealistic to implement. However, through real application executions, we find that input buffers with a size of 128 packets are big enough for the target many-core platform and never fully filled by traffic source. Thus, as a reasonable approximation, we use 128 packets as the input buffer size in the experiments. To measure the packet waiting time in the input buffer, we set up two measurement points, M_e and M_d, to record the time that a packet enters and departs from the input buffer, respectively. As shown in Fig. 11(a), the difference between M_e and M_d is the input buffer queuing delay. In the closed-loop measurement, the network congestion state directly affects the input progress of traffic source. To this end, as shown in Fig. 11(b), no input buffers are used to isolate traffic sources from the network. Hence, only a single measurement point M_e is inserted to record the time when a packet enters into the network. To measure the actual packet injection rate, we insert a measurement point M_{in} after the regulator and before the router. To make the two regulation mechanisms fairly comparable, both controllers (CPC and FC) characterize the arrival rate by monitoring MSHR entries as traffic input. Both schemes use the same regulator controlling only injection rate (ρ) without regulation control on burstiness σ. σ is predetermined by the cache block and packet sizes. The controllers and the regulator are realized as behavior models.

3) Benchmark Configuration: We use well-accepted **PARSEC** [29] as our benchmark suite. To scale the PARSEC benchmarks well to 64 cores, we choose large input sets (`simlarge`) for all programs. Table II summarizes the problem size of the benchmarks. For data validity, we only report the results obtained from the parallel execution phase of application [called region of interest (ROI)] in the experiments. Table II also shows the characteristics of the PARSEC programs in terms of network traffic injection rate (average number of packets injected into network per cycle) based on the data in [29]. As can be observed, three programs (*blackscholes*, *bodytrack*, and *dedup*) show relatively low traffic injection rate, while five (*canneal*, *streamcluster*, *swaptions*, *vips*, and *x264*) relatively high injection rate, leaving the other four (*facesim*, *ferret*, *fluidanimate*, and *freqmine*) with the middle-level traffic injection rate.

4) Comparison Studies: In the experiments, we consider four comparison cases: two baseline schemes plus our two dynamic traffic regulation mechanisms, as listed in the following.

1) The first baseline scheme (denoted *NoReg*) considers no traffic regulation policy by setting $\rho = 1$ to virtually bypass all traffic regulation modules in Fig. 10.
2) The second baseline (denoted *StaReg*) implements a static traffic regulation mechanism, in which each regulator is configured with constant regulation parameters (σ, ρ), which represent per-application offline characterized

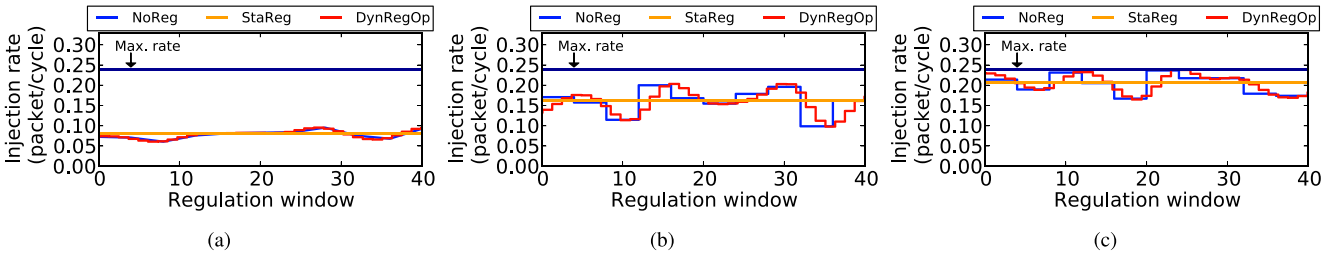


Fig. 12. Open-loop injection rates of three representative applications under the three mechanisms. (a) Injection rate of *bodytrack*. (b) Injection rate of *freqmine*. (c) Injection rate of *x264*.

long-term average traffic burstiness and injection rate of each PE. For simplicity and fairness, the same average (σ, ρ) parameters of all PEs are used for *StaReg* in both the open-loop and closed-loop experiments. Note that in the static regulation experiments, ρ is configured only once with the offline characterized value.

3) The third mechanism is our open-loop dynamic traffic regulation mechanism (denoted *DynRegOp*) and the fourth our closed-loop fuzzy traffic regulation mechanism (denoted *DynRegFz*). To avoid saturation, we adopt an injection rate upper bound (0.24 packet/cycle) for both the static and dynamic traffic regulation mechanisms, which operates the network just before saturation in offline experiments under uniform random traffic.

B. Open-Loop Experimental Results

1) Impact on Packet Injection Rate: Fig. 12 shows the regulation rates under *StaReg* and *DynRegOp* within the first 40 regulation windows for three representative applications: *bodytrack*, *freqmine*, and *x264*, with one in each group shown in Table I. To evaluate the difference between *StaReg* and *DynRegOp*, we also depict the traffic injection rate of *NoReg* in each figure. As shown in Fig. 12, without traffic regulation, the injection rate of *NoReg* varies with application execution.

Fig. 12(a) shows that in Group 1 application *bodytrack*, because the overall injection rate of the application is relatively low, there is marginal difference between the regulation rate adopted by *StaReg* and *DynRegOp*. However, in Group 2 application *freqmine* shown in Fig. 12(b), the difference between *StaReg* and *DynRegOp* becomes more obvious. In *StaReg*, as the regulation parameters are one-time configured, regulation rate stays constantly at the long-term average value (0.16 packet/cycle). As such, when the short-term injection rate is lower than the long-term average value, *StaReg* does not restrain the outgoing traffic. However, when the short-term injection rate is higher than the long-term average, *StaReg* forces the traffic injection to obey the long-term average rate, which may overregulate the outgoing traffic. By contrast, in *DynRegOp*, the regulation rate changes adaptively according to real traffic characteristics. As shown in Fig. 12(b), because *DynRegOp* adopts a sliding-window based mechanism to predict injection rate for the next regulation window, it smooths the underregulated traffic in *NoReg*, reducing burst traffic injection that can lead to short-term network congestion. Consistent trends in *StaReg* and *DynReg* are shown in Fig. 12(c), which is obtained by executing a Group 3 application *x264*. Fig. 12(c) also shows that in the three mechanisms, the maximum short-term injection rate is upper bounded by the packet injection rate threshold of 0.24 packet/cycle.

2) Impact on Queuing Delay and Network Delay: Fig. 13 shows the average queuing delay and network delay of the three comparison mechanisms, namely, *NoReg*, *StaReg*, and *DynRegOp*. As shown in the figure, the effects of different traffic regulation mechanisms depend consistently on application characteristics. Fig. 13(a) shows the average queuing delay across different applications, which represents the average packet waiting time in the input buffer shown in Fig. 11(a). For Group 1 benchmarks, average queuing delay of the three mechanisms achieves similar values, which are 2.8 cycles in *NoReg*, 3.5 cycles in *StaReg*, and 2.4 cycles in *DynRegOp*. This is because in Group 1 benchmarks, which have relatively low packet injection rate, the network is lightly loaded and packets can be quickly transmitted without incurring much queuing delay.

In Group 2 benchmarks, which have the middle-level traffic injection rate, the differences among the three mechanisms become more obvious, with average queuing delay reaching 13.1 cycles in *NoReg*, 17.9 cycles in *StaReg*, and 9.7 cycles in *DynRegOp*, showing that *StaReg* achieves the maximum queuing delay and *DynRegOp* the minimum across Group 2 applications. This is because in *StaReg*, outgoing traffic is overregulated and packets are unnecessarily held waiting in the input buffer. Being contrary to *DynRegOp*, in *NoReg*, the outgoing traffic is unconstrained and passes through the input buffer as soon as possible. However, by underregulating the outgoing traffic, although the packet queuing delay is reduced in the short-term, the network can easily become congested under traffic bursts and increases the queuing delay in the long-term. In *DynRegOp*, which adaptively regulates the traffic according to dynamic traffic injection characterization, both overregulation and underregulation are reduced, and the packets enjoy minimum queuing delay. The effectiveness of the *DynRegOp* is further confirmed by Group 3 applications, in which the average queuing delay reaches 20.5 cycles in *NoReg*, 28.1 cycles in *StaReg*, and *14.9* cycles in *DynRegOp*.

Fig. 13(b) shows the influence of different traffic regulation mechanisms on average network delay, which represents the average packet transmission time through the network. We can observe that being similar to average queuing delay, average network delay results also depend consistently on the characteristics of benchmarks. In Group 1 applications,

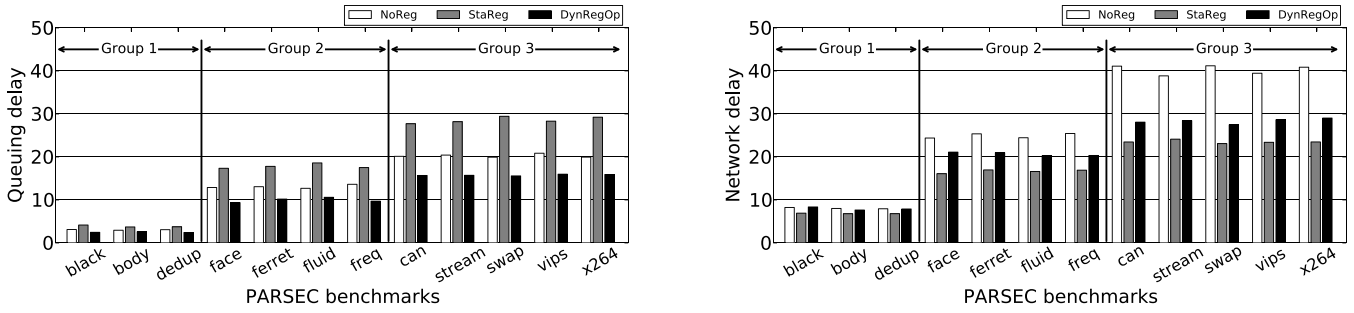


Fig. 13. Comparison of average queuing and network delay across all benchmark programs in the open-loop experiments. (a) Comparison of average queuing delay. (b) Comparison of average network delay.

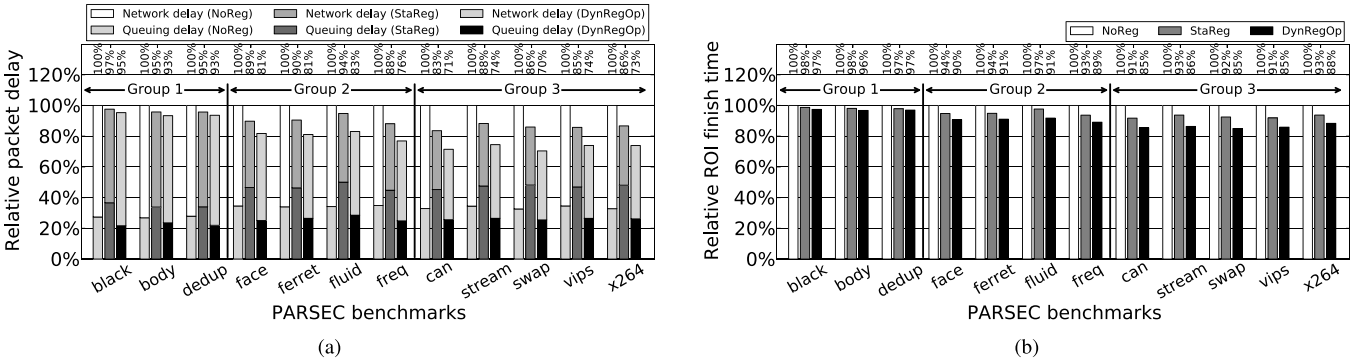


Fig. 14. Relative average packet delay and ROI finish time comparisons across all benchmarks in the open-loop experiments. (a) Comparison of relative average packet delay. (b) Comparison of relative ROI finish time.

as expected, average network delays of the three mechanisms achieve similar lower values, which are 8 cycles in *NoReg*, 6.3 cycles in *StaReg*, and 7.8 cycles in *DynRegOp*. In Group 2 applications, the average network delay of the three mechanisms increases to 24.8 cycles, 16.3 cycles, and 20.7 cycles, respectively. As shown in Fig. 13(b), *NoReg* achieves the maximum network delay and *StaReg* the minimum across Group 2 programs. This is because in *NoReg*, the network continuously admits packets in a best-effort manner, which may adversely incur severe traffic congestion inside the network, consequently increasing the average network delay. Moreover, in *StaReg*, network traffic is overly regulated, which pessimistically stops injecting additional packets to the network once the short-term injection rate reaches the long-term average rate. The consequence is that *StaReg* creates relatively light workload for the network, and the average packet delay achieves the minimum value. In *DynRegOp*, by reducing both overregulation and underregulation, the average network delay achieves the intermediate results between *NoReg* and *StaReg*. The same trends are consistently observed in Group 3 applications, in which the average network delay for *NoReg*, *StaReg*, and *DynRegOp* are 40.1 cycles, 23.5 cycles, and 27.6 cycles, respectively.

3) Impact on Packet Delay and ROI Finish Time: Fig. 14 shows the relative average packet delay (sum of packet queuing delay and network delay) and application ROI finish time results. Fig. 14(a) compares the relative average packet delay of the three comparison mechanisms. As shown in Fig. 14(a), albeit achieving higher network delay than *StaReg*, *DynRegOp* achieves the lowest packet delay values in the three mechanisms. In Group 1 applications, because the network is lightly loaded, the average packet delay achieved by *DynRegOp* is only 6.3% less than *NoReg*, and 2% less than *StaReg*. In Group 2 applications, *DynRegOp* improves the packet delay averagely by 19.8% (maximally by 24%) over *NoReg* and averagely by 9% (maximally by 12%) over *StaReg*, indicating that with the network injection rate increasing, *DynRegOp* achieves more significant packet transmission acceleration. Furthermore, in Group 3 applications, the average improvement in packet delay attained by *DynRegOp* reaches 27.6% over *NoReg* and 13.2% over *StaReg*, with maximum improvement reaching 30% and 16%, respectively.

Fig. 14(b) compares the relative ROI finish time across different applications. Because each application only spends a limited fraction of instructions communicating network packets, improvements in terms of ROI finish time achieved by *DynRegOp* are less than the improvements in terms of average packet delay. Across Group 1 programs, the ROI finish time is negligibly affected by the three comparison mechanisms. However, applying *DynRegOp*, the ROI finish time is averagely reduced by 9.8% (maximally by 11%) over *NoReg* and averagely by 4.3% (maximally by 6%) over *StaReg* across Group 2 applications. Furthermore, in Group 3 programs, the average ROI finish time reduction increases to 14.2% over *NoReg* and 6.8% over *StaReg*, with the corresponding maximum reduction reaching 15% and 7%,

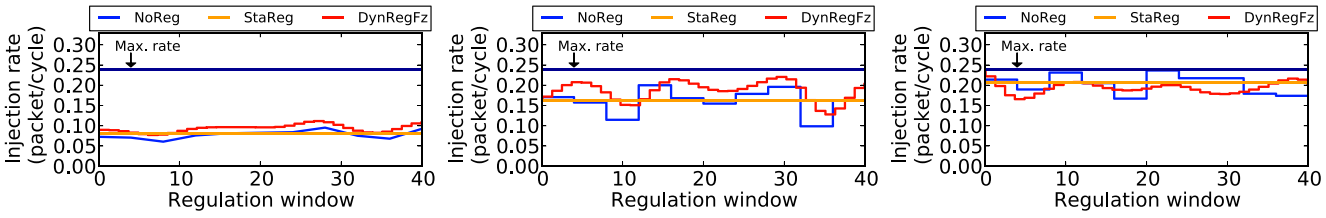


Fig. 15. Closed-loop injection rates of three representative applications under the three mechanisms. (a) Injection rate of *bodytrack*. (b) Injection rate of *freqmine*. (c) Injection rate of *x264*.

showing consistent application execution acceleration brought by *DynRegOp*.

C. Closed-Loop Experimental Results

1) Impact on Packet Injection Rate: Fig. 15 shows the injection rate comparison of the fuzzy traffic regulation *DynRegFz* and the static traffic regulation *StaReg* in the closed-loop measurement shown in Fig. 11(b). For comparison purpose, we also depict the traffic injection rate of *NoReg* in each figure, which shows that without traffic regulation, the injection rate of *NoReg* varies with application execution.

Fig. 15(a) compares the injection rates under *StaReg* and *DynRegFz* for Group 1 application *bodytrack*, which has relatively low long-term average packet injection rate. As shown in Fig. 15(a), the regulation rate of *StaReg* is statically set to the long-term average value, achieving similar regulation results as in the open-loop experiments. Being different from the open-loop traffic regulation mechanism *DynRegOp*, the closed-loop fuzzy traffic regulation mechanism *DynRegFz* considers both injection traffic's characteristics and network status to predict the regulation rate. As the network is generally lightly loaded when executing *bodytrack*, the injection rates of *DynRegFz* are thus higher than the results achieved by *DynRegOp* in Fig. 12(a), accelerating packet injection to achieve higher network utilization. The same phenomenon is shown in Fig. 15(b), which compares the regulation rates achieved by *StaReg* and *DynRegFz* in Group 2 application *freqmine* with the middle-level long-term average traffic injection rate. Although the network is more heavily loaded when running *freqmine* than running *bodytrack*, Fig. 15(b) shows that *DynRegFz* achieves consistently higher injection rates than *DynRegOp* in Fig. 12(b), more frequently approaching the maximum injection rate of 0.24 packet/cycle and thus maximum network utilization.

Fig. 15(c) shows the injection rate comparison in Group 3 application *x264*. Because *x264* has high-level average packet injection rate, it operates the network near the maximum injection rate of 0.24 packet/cycle close to network saturation point. Taking the network status into account to regulate outgoing traffic, *DynRegFz* decreases the regulation rate to proactively avoid network saturation occurrence. Thus, as shown in Fig. 15(c), the injection rates achieved by *DynRegFz* are lower than the results achieved by *DynRegOp* in Fig. 12(c).

2) Impact on Packet Delay and ROI Finish Time: Fig. 16 shows the relative average packet delay and ROI finish time comparisons across all PARSEC programs in the closed-loop experiments. Fig. 16(a) shows the relative average packet delay comparison of *NoReg*, *StaReg*, and *DynRegFz*. Across all benchmarks, *DynRegFz* consistently reduces the average packet delay against *NoReg* and *StaReg*. Furthermore, because Group 1 and 2 applications have low- or middle-level long-term traffic injection rate, they generally stress the network with light or moderate workloads. By adapting to the actual workload characteristics, *DynRegFz* reasonably increases the regulation rate, accelerating traffic injection and averagely reducing packet delay by 11% (maximally by 12%) over *NoReg* and averagely by 8% (maximally by 9%) over *StaReg* in Group 1 applications. Across Group 2 applications, packet delay is averagely decreased by 29.3% (maximally by 30%) over *NoReg* and averagely by 20.3% (maximally by 22%) over *StaReg*. However, in Group 3 applications, which have high-level long-term traffic injection rate, *DynRegFz* decreases the regulation rate to avoid network saturation. Thus, across Group 3 programs, *DynRegFz* achieves averagely 23.8% (maximally 27%) packet delay reduction against *NoReg* and averagely 11.4% (maximally 13%) against *StaReg*, resulting in slightly less but still satisfactory packet delay improvements compared with *DynRegOp*.

Fig. 16(b) shows the relative ROI finish time comparisons across *NoReg*, *StaReg*, and *DynRegFz*. Being similar to the results in Fig. 14(b), ROI finish time reductions achieved by *DynRegFz* are less than the packet delay reductions. This is because each application only spends a fraction of instructions communicating network packets. Compared with *NoReg*, the ROI finish time is marginally improved by *StaReg* and *DynRegFz* in Group 1 programs. Furthermore, in Group 2 applications, *DynRegFz* averagely reduces the ROI finish time by 15.8% (maximally by 17%) over *NoReg* and averagely by 11.2% (maximally by 12%) over *StaReg*. Moreover, in Group 3 programs, average ROI finish time reductions achieved by *DynRegFz* are 9.8% over *NoReg* and 3.2% over *StaReg*, with the corresponding maximum improvements of 11% and 5%, showing consistent application execution acceleration brought by *DynRegFz*.

D. Comparison of Open-Loop and Closed-Loop Regulation

Fig. 17 compares the open-loop dynamic traffic regulation mechanism *DynRegOp* and the closed-loop fuzzy traffic regulation mechanism *DynRegFz*. To complete the evaluations, we also implement an ideal case (denoted *Ideal*), which applies the input buffers in *DynRegOp* [see Fig. 11(a)] to the closed-loop fuzzy traffic regulation policy *DynRegFz* [Fig. 11(b)].

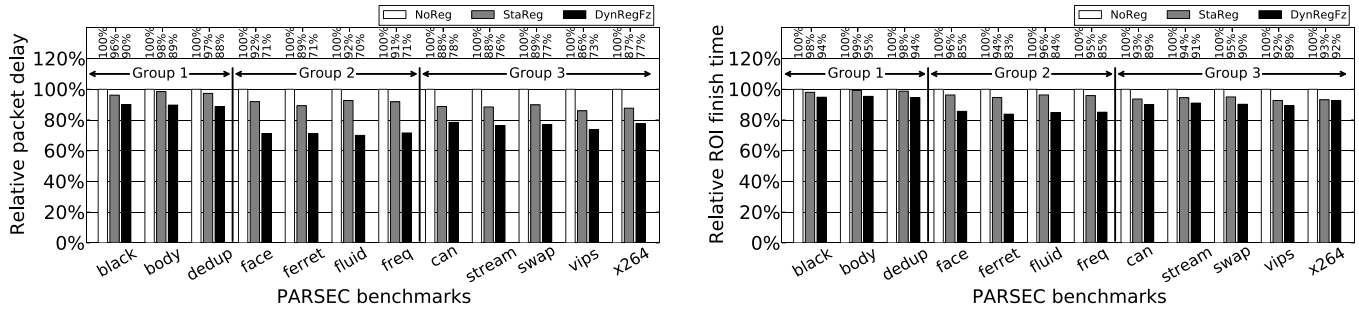


Fig. 16. Relative average packet delay and ROI finish time comparisons across all benchmarks in the closed-loop experiments. (a) Comparison of relative average packet delay. (b) Comparison of relative ROI finish time.

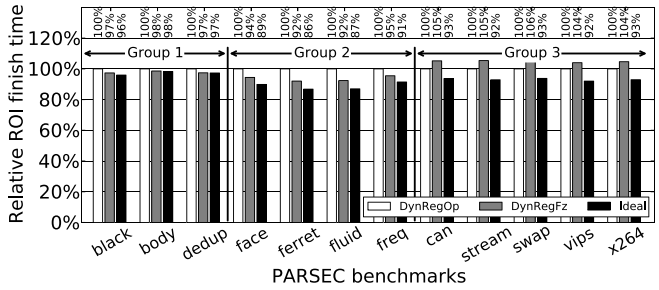


Fig. 17. Application performance comparison of open-loop and closed-loop regulation schemes.

TABLE III
COMPARING OPEN-LOOP AND CLOSED-LOOP REGULATION

Components/Signals	Open-loop	Closed-loop
Regulator	(σ, ρ)	(σ, ρ)
Controller	CPC controller	Fuzzy controller
Controller input	Traffic	Traffic, Network status
Controller output	(σ, ρ) or ρ	ρ
Network state recognizer	-	Fuzzy analysis

As such, *Ideal* empowers the privilege of buffering excessive traffic to achieve network congestion-state aware traffic regulation, and consistently improves ROI finish time than *DynRegOp* and *DynRegFz* across all benchmarks. However, to implement *Ideal* as a real scheme is hard, because in the open-loop scenario, it can be difficult to get the network congestion state if the underlying network IP does not support congestion awareness. Furthermore, in the closed-loop scenario, big input buffers can add considerable area and power consumption to the CMP chip.

As shown in Fig. 17, the effectiveness of *DynRegOp* and *DynRegFz* varies with the characteristics of different benchmarks. In Group 1 benchmarks, because each program has low average traffic injection rate, although *DynRegFz* achieves shorter ROI finish time than *DynRegOp*, the two mechanisms exhibit insignificant difference. By stressing the network with moderate workloads, Group 2 applications highlight the ROI improvements achieved by *DynRegFz*. Considering the network congestion state to regulate outgoing traffic, *DynRegFz* increases the network's potential in expediting packet transmission, thus achieving shorter ROI finish time than *DynRegOp*. In Group 3 applications, where the network works near the saturation point, input buffers in *DynRegOp* decouple the network congestion state from traffic sources, allowing cores to continuously progress even when the network is temporarily congested. However, in *DynRegFz*, the fuzzy regulation policy directly tightens up the traffic source once network congestion occurs. Thus, in Group 3 programs, *DynRegOp* outperforms *DynRegFz* in the ROI execution acceleration.

VII. SUMMARY AND CONCLUSION

We have presented two dynamic traffic regulation schemes in a unified framework, which can flexibly adjust regulation strength on demand. As a result, they make more efficient use of the system interconnect, achieving significant reduction in network packet delay and improving system throughput. While the central idea on traffic admission is the same, the two schemes are applied to different contexts for MPSoC and CMP designs. To show a clear qualitative comparison, we summarize the components used in both regulation schemes in Table III. Because IP integration is a post-IP design process, the open-loop regulation that only monitors the outgoing traffic from the IP and makes regulation decisions according to the online profiled traffic characteristics is more feasible for MPSoCs. For CMPs, the closed-system design nature allows the regulation scheme to be more aggressive using both traffic and network state information.

By realizing and evaluating our techniques within the same 64-core system architecture simulated in GEM5 running all 12 PARSEC benchmarks, we find that the improvements in average packet delay and application ROI finish time correlate consistently with the long-term average packet injection rate characteristics of the benchmarks. There are no or marginal improvements in three programs (Group 1: *blackscholes*, *bodytrack*, and *dedup*), because these programs have low long-term average packet injection rate, stressing the network with light workloads. However, for the other nine programs (Group 2: *facesim*, *ferret*, *fluidanimate*, and *freqmine*, and Group 3: *canneal*, *streamcluster*, *swaptions*, *vips*, and *x264*), our techniques gain significant improvements in average packet delay and ROI finish time, because these programs have medium (Group 2) or high (Group 3) long-term average packet injection rate. Indeed, in the open-loop experiments, our technique enhances the average packet delay by 23.7% on average (Group 2: 19.8%; Group 3: 27.6%) and 30% in maximum against no traffic regulation, and 11.1% on average (Group 2: 9%; Group 3: 13.2%) and 16% in maximum against static traffic regulation. In terms of ROI finish time,

our technique outperforms averagely by 12% (Group 2: 9.8%; Group 3: 14.2%) and maximally by 15% against no regulation, and averagely by 5.6% (Group 2: 4.3%; Group 3: 6.8%) and maximally by 7% against static traffic regulation. In the closed-loop experiments, compared with no regulation, our fuzzy traffic regulation averagely reduces the average packet delay by 26.6% (Group 2: 29.3%; Group 3: 23.8%) and maximally by 30%, accelerating ROI execution averagely by 12.8% (Group 2: 15.8%; Group 3: 9.8%) and maximally by 17%. Compared with static regulation, our fuzzy traffic regulation averagely reduces the packet delay by 15.9% (Group 2: 20.3%; Group 3: 11.4%) and maximally by 22%, accelerating ROI execution averagely by 7.2% (Group 2: 11.2%; Group 3: 3.2%) and maximally by 12%.

We have also contrasted the two regulation mechanisms both qualitatively and quantitatively, showing no single winner. We believe that traffic regulation (traffic engineering in general) as a complement to network-centric approaches should and can be equally exploited to enhance the application performance in NoC-based computing systems.

REFERENCES

[1] K. Olukotun, L. Hammond, and J. Laudon, *Chip Multiprocessor Architecture: Techniques to Improve Throughput and Latency*. San Rafael, CA, USA: Morgan & Claypool Publishers, 2007.

[2] Z. Lu, M. Millberg, A. Jantsch, A. Bruce, P. van der Wolf, and T. Henriksson, "Flow regulation for on-chip communication," in *Proc. Design, Autom. Test Europe Conf. Exhibit. (DATE)*, Nice, France, Apr. 2009, pp. 578–581.

[3] F. Jafari, Z. Lu, A. Jantsch, and M. H. Yaghmaee, "Buffer optimization in network-on-chip through flow regulation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 12, pp. 1973–1986, Dec. 2010.

[4] R. L. Cruz, "A calculus for network delay, part I: Network elements in isolation," *IEEE Trans. Inf. Theory*, vol. 37, no. 1, pp. 114–131, Jan. 1991.

[5] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet* (Lecture Notes in Computer Science), vol. 2050. Berlin, Germany: Springer, 2004.

[6] C.-S. Chang, *Performance Guarantees in Communication Networks*. London, U.K.: Springer-Verlag, 2000.

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

[8] X. Zhao and Z. Lu, "Heuristics-aided tightness evaluation of analytical bounds in networks-on-chip," *IEEE Trans. Comput.-Aided Design Integr.*, vol. 34, no. 6, pp. 986–999, Jun. 2015.

[9] Y. Jiang, "A basic stochastic network calculus," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, pp. 123–134, Oct. 2006.

[10] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse, "System architecture evaluation using modular performance analysis: A case study," *Int. J. Softw. Tools Technol. Transf.*, 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 Proc.-Comput. Digit. Techn.*, vol. 152, no. 2, pp. 148–166, Mar. 2005.

[12] E. Wandeler, A. Maxiaguine, and L. Thiele, "Performance analysis of greedy shapers in real-time systems," in *Proc. Design, Autom. Test Eur.*, Mar. 2006, pp. 444–449.

[13] M. H. Wiggers, M. J. G. Bekooij, and G. J. M. Smit, "Modelling run-time arbitration by latency-rate servers in dataflow graphs," in *Proc. Int. Workshop Softw. Compil. Embedded Syst. (SCOPES)*, Apr. 2007, pp. 11–22.

[14] B. D. de Dinechin, Y. Durand, D. van Amstel, and A. Ghiti, "Guaranteed services of the NoC of a manycore processor," in *Proc. Int. Workshop Netw. Chip Archit.*, 2014, pp. 11–16.

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

[16] G. V. Varatkar and R. Marculescu, "On-chip traffic modeling and synthesis for MPEG-2 video applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 12, no. 1, pp. 108–118, Jan. 2004.

[17] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks*. San Mateo, CA, USA: Morgan Kaufmann, 2004.

[18] P. Bogdan and R. Marculescu, "Non-stationary traffic analysis and its implications on multicore platform design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 4, pp. 508–519, Apr. 2011.

[19] P. Bogdan, M. Kas, R. Marculescu, and O. Mutlu, "QuaLe: A quantum-leap inspired model for non-stationary analysis of NoC traffic in chip multi-processors," in *Proc. 4th ACM/IEEE Int. Symp. Netw.-Chip (NOCS)*, May 2010, pp. 241–248.

[20] A. Bansal, S. Gupta, and T. Majumder, "Efficient estimation of non-stationary traffic parameters on networks-on-chip," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshop (IPDPSW)*, May 2015, pp. 425–433.

[21] U. Y. Ogras and R. Marculescu, *Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures* (Lecture Notes in Electrical Engineering), vol. 184. New York, NY, USA: Springer, 2013.

[22] U. Y. Ogras, P. Bogdan, and R. Marculescu, "An analytical approach for network-on-chip performance analysis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 12, pp. 2001–2013, Dec. 2010.

[23] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006.

[24] X. Chen *et al.*, "In-network monitoring and control policy for DVFS of CMP networks-on-chip and last level caches," *ACM Trans. Design Autom. Electron. Syst.*, vol. 18, no. 4, pp. 47:1–47:21, 2013.

[25] L. Zadeh, "Fuzzy sets," *Inf. Control*, vol. 8, no. 3, pp. 338–353, 1965.

[26] Z. Lu and Y. Wang, "Dynamic flow regulation for IP integration on network-on-chip," in *Proc. Int. Symp. Netw. Chip (NOCS)*, May 2012, pp. 115–123.

[27] Y. Yao and Z. Lu, "Fuzzy flow regulation for network-on-chip based chip multiprocessors systems," in *Proc. 19th Asia South Pacific Design Autom. Conf. (ASPDAC)*, Singapore, Jan. 2014, pp. 343–348.

[28] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

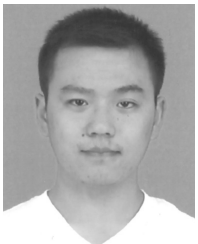
[29] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proc. Int. Conf. Parallel Archit. Compil. Techn. (PACT)*, Aug. 2008, pp. 72–81.

[30] K. M. Passino and S. Yurkovich, *Fuzzy Control*. Reading, MA, USA: Addison-Wesley, 1997.

[31] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed. San Mateo, CA, USA: Morgan Kaufmann, 2011.

Zhonghai Lu (M'05) received the B.S. degree in radio and electronics from Beijing Normal University, Beijing, China, in 1989, and the M.S. degree in system-on-chip design and the Ph.D. degree in electronic and computer system design from the KTH Royal Institute of Technology, Stockholm, Sweden, in 2002 and 2007, respectively.

He was an Engineer in the area of electronic and embedded systems from 1989 to 2000. He is currently an Associate Professor with the Department of Electronic and Embedded Systems, KTH Royal Institute of Technology. He has authored over 120 peer-reviewed papers. His current research interests include interconnection networks, performance analysis, and real-time systems.



Yuan Yao received the B.S. degree in microelectronics from Northwestern Polytechnical University, Xi'an, China, in 2009, and the M.S. degree in system-on-chip design from the KTH Royal Institute of Technology, Stockholm, Sweden, in 2014, where he is currently pursuing the Ph.D. degree with the Department of Electronic and Embedded Systems.

His current research interests include quality-of-service provisioning, formal performance analysis, and power management in network-on-chip based chip multi/many-core processors.