Predictive-Flow-Queue Based Energy Optimization for Gigabit Ethernet Controllers

Hwisung Jung, *Student member*, *IEEE*, Andy Hwang, and Massoud Pedram, *Fellow*, *IEEE*

Abstract - This paper presents an energy-efficient packet interface architecture and a power management technique for gigabit Ethernet controllers, where low-latency and high-bandwidth are required to meet the pressing demands of very high frame-rate data. More specifically, a predictive-flow-queue (PFQ) based packet interface architecture is presented, which adjusts the operating frequency of different functional blocks at a fine granularity so as to minimize the total system energy dissipation while attaining performance goals. A key feature of the proposed architecture is the implementation of a runtime workload prediction method for the network traffic along with a continuous frequency adjustment mechanism, which enables one to eliminate the latency and energy penalties associated with discrete power mode transitions. Furthermore, a stochastic modeling framework based on Markovian decision processes and queuing models is employed, which make it possible to adopt a precise mathematical programming formulation for the energy optimization under performance constraints. Experimental results with a designed 65nm gigabit Ethernet controller show that the proposed interface architecture and continuous frequency scaling result in system-wide energy savings while meeting performance specifications.

***Index Terms* — Energy optimization, gigabit Ethernet controllers, predictive-flow-queue, semi-Markov process, workload prediction**

I. Introduction

Ongoing advances in computer networks and hardware designs have resulted in the introduction of multi-gigabit Ethernet links. Commensurate with this trend, the network interface cards (NICs) are becoming ever more complex in order to satisfy the high-functionality and high performance demands of today's applications. For example, hardware support for manageability features such as ASF (Alert Standards Format) and DMWG (Desktop and Mobile Work Group) are being integrated into the NIC to provide system management capabilities [1], which in turn necessitates more processors to be included in the NICs. A close look at today's high-speed NICs (a.k.a., gigabit Ethernet controllers) reveals that these controllers must also conserve power since excessive power dissipation creates many problems, from increased operational cost to reduced hardware reliability.

A multi-gigabit Ethernet controller must be able to support high frame-rate data processing and low-latency access for the frame data. However, these trends also are translated into higher power density, higher operating temperature, and consequently lower system reliability. The power consumption of the gigabit Ethernet controller increases rapidly with the increase in link speed. In addition, as the Ethernet port density of a server system increases for a given form-factor, designers of the gigabit Ethernet controller must resort to more energy-efficient architectures as they attempt to add more ports to the server system. For example, the Sun Blade server consumes around 55W power for a Network Express module which includes 20 ports [2]. Typically the power number for the gigabit Ethernet controllers ranges from 2 to 3W per port, depending on the link speed [3][4].

Power saving in gigabit Ethernet controllers has been commonly achieved by transitioning to more advanced semiconductor process technologies, (e.g., using 65nm and 45nm technology nodes) and/or by utilizing low power design techniques (e.g., using clock gating and static voltage scaling techniques). However many opportunities for reducing energy dissipation at the system level exist. For example, modern circuit design technologies allow a number of different clock and voltage domains to be specified on the same chip. As a result, significant power saving can be achieved if, at runtime and under the control of a power management unit, suitable operating voltage and frequency values are assigned to various functional blocks (FBs) inside an Ethernet controller to trade performance for lower power dissipation.

As more and more of the FBs inside an Ethernet controller (e.g., MAC, PHY, PCI-E) are being designed to support multiple power-performance modes (i.e., different supply voltage and clock frequency settings), it is becoming possible to realize full chip energy saving by employing advanced system-level power management strategies. This solution, however, requires development of dedicated means and methods for realizing runtime power management policies. In particular, the following issues must be considered when utilizing a dynamic power management policy which changes the *voltage-frequency* settings of different FBs in order to minimize energy dissipation while attaining a performance goal:

i) Typically a lot of performance is sacrificed in order to achieve lower power dissipation; this is especially a concern for demanding applications such as the Ethernet controller,
ii) There is a significant latency and energy dissipation overhead associated with the mode transition, e.g., the overhead of acquiring the lock in a phase locked loop (PLL) once a new frequency target is set or the overhead of

Manuscript submitted to the IEEE Transactions on Very Large Scale Integration Systems in Oct. 2007. A preliminary version of this work was published in the 2007 Proc. of Asia and South Pacific-Design Automation Conference [52]. Much of the formulation, discussions, and results presented here are different from that preliminary paper.

DC-DC conversion to change the supply voltage level [6], and

iii) The power management routine, which is likely residing in the operating system (OS), can itself become a heavy duty task, which can consume sizeable computational and energy resources since it has to continually monitor the system workload, make decisions about the next set of voltage-frequency settings for the various FBs, and communicate the decision to the appropriate hardware [7].

In this paper, we propose a predictive-flow-queue (PFQ) based packet interface architecture to minimize the energy dissipation of a gigabit Ethernet controller. Generally, 802.3 Medium Access Control (MAC) [8] sub-layer offers an Ethernet level flow control mechanism among a pair of full-duplex end points, while routing certain classes of network traffic, which are generated, processed, and terminated at the specific processor inside the gigabit Ethernet controller. In the proposed architecture, the packet interfaces inside MAC and between MAC and Direct Memory Access (DMA) engine are targeted for energy-saving opportunities. A dynamic frequency adapter, which provides a continuously varying frequency based on a workload prediction technique, is utilized to achieve energy saving without much overhead. The proposed architecture is modeled with semi-Markov chain (SMC) [9] and queuing models to enable formulation of a mathematical program for optimizing the total system energy dissipation under performance constraints.

In this paper, we shall only utilize a dynamic frequency scaling (DFS) technique to minimize the power consumption of the system. This is because dynamic voltage scaling (DVS) technique, although it can provide a near cubic reduction in power dissipation, tends to incur a large transition time overhead. This overhead can be on the order of tens of microseconds, for example from the specification for the AMD Athlon chip [10]. In the 80200 XScale processor chip, the latency for switching the CPU voltage is 6 microseconds [11]. For systems where execution is blocked during a transition (which is common in many existing commercial processors including the AMD Athlon chip), this translates into tens of thousands of lost execution cycles. Another related problem is transition energy overhead, which can actually cause the system's energy consumption to increase if DVS is not used judiciously.

The transition time overhead for frequency scaling is much shorter, i.e., frequency change can indeed take effect in one cycle. For example, IBM researchers recently introduced a dynamic power management technique called PowerTune [12] which uses a single PLL driving divider circuitry to produce multiple frequencies for the PowerPC 970 family. This allows the PLL to stay locked at a given frequency while the processor core frequency is dynamically scaled from initial frequency level of f to $f/2$, $f/4$, and $f/64$ within one cycle without phase shift. When the processor switches frequency, PowerTune switches back and forth between the old (lower) and new (higher) frequencies, resulting in more cycles of the new frequency. This reduces the bounce noise and allows the packages to effectively react to the change in current. PowerTune differs from other work (e.g., the power management solution for the PowerPC 750 [13]) in that it allows system-wide dynamic frequency scaling without stopping the core. System wide control of clock frequency achieves excellent power savings as reported. PowerTune is closest to what we propose here, except that our design allows for continuous frequency adjustment, and not simply switching among a small number of frequency levels.

As another case study, we can mention the Intel's Montecito design [14], which attempts to tap unused power by dynamically adjusting the processor voltage and frequency setting to ensure the highest frequency within temperature and power constraints [15]. The chip is capable of changing its supply voltage level in 12.5mV increments. However, it takes 100ms to respond in the voltage control loop to a request for voltage change by an on-chip microcontroller, which runs a real-time scheduler to support multiple tasks - calibrating an on-chip ammeter, sampling temperature, performing power calculations, and determining the new voltage level. In the Montecito design, the supply voltage throughout the chip (which is affected by current-induced droops and DC voltage drops) is constantly monitored with 24 voltage sensors, and a frequency level is selected to match the lowest voltage reported by any sensor. A digital frequency divider provides the selected frequency, in nearly 20-MHz increments and within a single cycle, without requiring the on-chip PLLs to resynchronize [16].

Another difficulty with dynamic voltage scaling is the fact that the amount of load current change after mode transition can be significant. Even specialized DC-DC converters cannot completely avoid the instantaneous output voltage drop (loss of output regulation) due to a sudden and dramatic change in the load current demand. For example, the DC-DC converter of [17], which uses a reactance switching technique for fast-response load regulation, encounters about 150mV of voltage deviation from a target output voltage of 3.3V when the load changes from 1 to 30A. This supply voltage droop may last 100's of microseconds.

Note that we are not arguing against DVFS in general. Instead what we are stating is that for the target application (i.e., the Gigabit Ethernet Controller), given the current state of the art in DC-DC conversion and voltage control loop response time, the overhead of voltage scaling is too high, and hence, we resort to frequency scaling only.

The remainder of this paper is organized as follows. Section II provides a brief background of Ethernet controller and related work while section III describes the details of proposed PFQ-based architecture. In section IV, we present a workload prediction technique. Section V provides an analysis of the system based on MDP and queuing models and a performance optimization formulation. Experimental results and conclusion are given in section VI and section VII.

II. Preliminaries

A. Background on Ethernet Controller

The main purpose of an Ethernet controller is to transport network traffic between the host system and the physical Ethernet links. Sending and receiving the network traffic over local interconnect, i.e., the PCI-E bus [19], is handled by the Ethernet controller and device driver in the host operating system. In general, the Ethernet controller typically has a DMA engine to transfer data between the host system memory and the network interface memory. In addition, the Ethernet controller includes a MAC unit to implement the link level protocol for the underlying network, and uses a signal processing engine to implement the physical (PHY) layer of the network stack (where the 802.3 frame format is supported).

To satisfy the high-functionality of today's network applications, enterprise-class Ethernet controllers must handle other classes of network streams, e.g., remote *management traffic*, which is required to terminate at the host computer, and not necessarily at the host operating system [1]. Management technology essentially allows IT administrators to remotely access a user's system, i.e., via the network that the system is connected to, and perform necessary provisioning, maintenance, and repairs. Hence, an additional CPU-memory sub-system is required to handle the management bound traffic. This sub-system is fundamentally independent of the Ethernet side functionalities. More specifically, remote management and/or fast message transfer may be accomplished through a remote direct memory access (RDMA) engine [20] which allows data to move directly from the memory of one host system to that of another without involving either one's operating system, thereby realizing high-throughput and low-latency in data transfer. In addition, the latest Ethernet controllers include an integrated IP security encryption engine (IPSec) [21] to secure internet protocol communications. Fig. 1 shows a simplified block diagram of the Ethernet controller, where the statistics functional module provides relevant information about the packet flows.

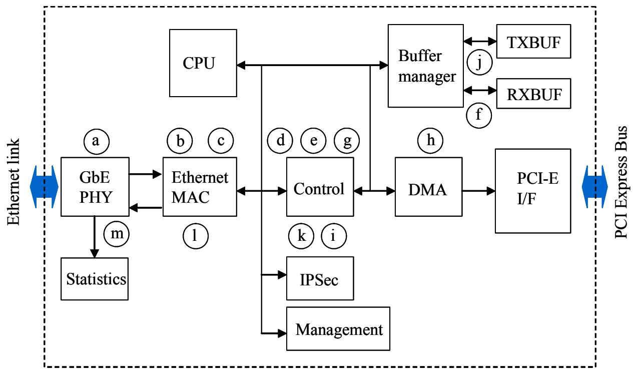

Fig. 1. Simplified block diagram of an Ethernet controller.

To understand the functionality of Ethernet controller inside, the process of receiving a packet over the network is explained next (see Fig. 1 where various steps are shown near the blocks that execute them). In step (a), the Ethernet controller receives a data stream from the selected physical layer interface. It performs address checking, Cyclic Redundancy Check (CRC), and Carrier Sense Multiple Access / Collision Detection (CSMA/CD) functions [8] in step (b). In step (c), the Ethernet controller calculates checksum and parses Transport Control Protocol / Internet Protocol (TCP/IP) headers. This is followed by the classification of the frame based on a set of matching rules in step (d). In step (e), the Ethernet controller strips the Virtual Local Area Network (VLAN) tag, and then temporarily places the packet data and header into a pre-allocated receive buffer (i.e., RXBUF) in step (f). After that, the Ethernet controller completes the buffer descriptors, which contain information about the starting memory address and the length of the data packets, in step (g). Finally, in step (h), the PCI-E interface initiates the DMA transfer of the packet data and descriptors to the host memory by interrupting the device driver.

Sending packets is analogous to receiving them except that the device driver first creates the buffer descriptors for the packets to be transmitted. Next, the Ethernet controller completes the buffer descriptor for the data packets in step (i) of Fig. 1. In step (j), the packets are stored into the temporary buffer (i.e., TXBUF) via the DMA. The Ethernet controller updates the frame descriptor with checksum, VLAN tag, and header pointers in step (k), and executes CSMA/CD functions to transmit the frame in step (l). Finally, data is formatted to the selected physical layer interface in step (m). Details about the Ethernet controller architecture and the processes of traffic management, RDMA, and IPSec are omitted for brevity. Interested readers may refer to [20]-[22] for additional information.

B. Related Work

Dynamic voltage and frequency scaling (DVFS) has been the subject of many investigations [23]-[29]. In the following, we provide a quick review of some the work which is most directly related to ours.

An application-level power management technique was presented in [23], where the authors described an operating system interface that can be used by applications to achieve energy savings. In this approach, an application is allowed to specify its desired voltage-frequency value, and the operating system ensures that the application will run under that setting. The research work in [24] considered a DVFS managed processor executing packet producing tasks and a power-managed network interface. The authors introduced an approach for minimizing the energy consumed by the network resource through careful selection of voltage-frequency settings for the processor. A key consideration was to dynamically balance the processor and network energy dissipations.

The authors in [25] described a method of profile-based power-performance optimization by DVFS scheduling in a high-performance PC cluster. The authors divide an application program's execution into several regions (CPU-intensive or communications-intensive) and choose the best voltage-frequency setting for the processors to execute each region according to the profile information about the execution time and power dissipation of a previous trial run. The authors consider the latency and energy dissipation overhead of changing voltage-frequency settings. The work in [26] investigated software techniques to direct run-time power optimization. The authors targeted network links, the dominant power consumer in parallel computer systems, allowing DVFS instructions extracted during static compilation to coordinate link voltage and frequency transitions for power savings during the application execution. Concurrently a hardware online mechanism measures network congestion levels and adapts these off-line voltage-frequency settings to optimize network performance.

The authors of [27] present a compiler-driven approach to optimize the power consumption in communication links by using DVFS. In this approach, an optimizing compiler analyzes the data-intensive application code and extracts the data communication pattern among parallel processors. This information along with network topology is used for identifying the link access patterns. The link access patterns and inherent data dependence information are used to determine optimal voltage-frequency settings for the communication links at any given time frame. In [28], the authors presented a DVFS technique for a soft real-time system where the voltage-frequency setting is updated at variable-length (instead of fixed-length) intervals. The proposed voltage-frequency setting method is based on the notion of an effective deadline for a task, which is predicted adaptively and is used to provide fast tracking for abrupt workload changes. In [29], the authors described a DVFS technique targeted at non-real-time applications running on an embedded system. This approach makes use of runtime information about the external memory access statistics in order to perform CPU voltage and frequency scaling. The proposed DVFS technique relies on dynamically constructed regression models that allow the CPU to calculate the expected workload and slack time for the next time frame and, thus, adjust its voltage and frequency in order to save energy, while meeting soft timing constraints.

All of the above techniques perform DVFS, where the performance overhead of such DVFS mechanisms is rather high. For example, the authors in [26] assume that no network traffic (i.e., packets) can cross the link during the power-mode transitions, resulting in 20 to 100 bus cycle penalty each time a voltage-frequency scaling is executed. Furthermore since the voltage-frequency commands are issued by the operating system, the time interval between two successive commands is high. For example, reference [29] invokes a power management kernel, which is a part of the OS code, to change the voltage-frequency setting every 50ms (corresponding to a Linux time quantum).

Little attention has been paid to doing DVFS by using purely hardware-based mechanisms. This is clearly a promising direction since hardware-based DVFS mechanisms produce low latency and energy dissipation overheads. They can thus be invoked much faster (i.e., two successive adjustments to voltage and frequency can be made with a much shorter interval in between).

III. Proposed Architecture

In this section, we present details of *Predictive Flow Queue* (PFQ) based power management architecture, which is comprised of a performance monitor, a power manager, and a dynamic frequency adapter. We also describe our energy-efficient packet interface architecture.

A. *PFQ-based Power Management Architecture*

Defragmenting/filtering packets of various communication protocols inside the Ethernet controller is a particularly complex and demanding task. Thus, the Ethernet controller needs many FBs and specialized hardware units that efficiently process and transfer data between the local interconnect and the network [30].

The PFQ architecture for the most part provides a first-in first-out (FIFO) mechanism between the state machines realizing various FBs. Each state machine essentially reacts to the content of its corresponding PFQ to initiate and direct the processing activities of the state machine as depicted in Fig. 2, where we assume that each FB has three number of active state (e.g., S_1, S_2, and S_3) which is controlled by dynamic frequency scaling (DFS) values. A FB is shut down (power gated) when it is in sleep mode. In contrast, the FB is assigned the lowest allowed frequency when it is in idle mode. The content of PFQ includes pointers that are used to indicate where the frame data is located within the temporary buffers. When the PFQ is empty, the state machine has no work to perform and is in its idle state.

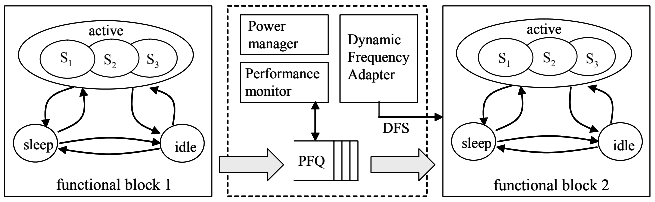

Fig. 2. Concept of predictive-flow-queue (PFQ).

Attempting to greedily respond to the workload changes so as to provide an optimal DFS value can result in significant energy and delay overheads associated with the power-mode transitions. To solve this problem, a software component, which accurately predicts the required performance level of the system has been incorporated into the power management systems [31][28]. Although these prediction methods help reduce the energy/delay overheads, they suffer from a few disadvantages, that is, i) a software-oriented prediction algorithm increases the computational complexity of the power manager that resides in the driver or the OS, and ii) when using a Phase-Lock Loop (PLL) to effect a frequency change, the FB may be stalled during the lock time of the PLL. Consequently, use of the PLL to realize the DFS setting commanded by the power manager may result in a sizeable performance penalty.

The main advantage of the proposed PFQ-based power management architecture is that we predict the workload level for the next time step while processing the incoming traffic and ramp up (or down) the operating frequency in a *continuous* manner until the target operating frequency value is achieved. As a result, there is never a need for stalling the FB. The details of the power management architecture which include a performance monitor, a power manager and a dynamic frequency adapter, are explained next.

1) Performance Monitor

The performance monitor profiles and analyzes characteristics of the workload by examining the corresponding PFQ. The service time behavior of each FB is captured in the form of the service time distribution for the FB when it is in the active mode. Similarly, the input request behavior (i.e., workload) of each FB is modeled by the request interarrival time distribution at the corresponding input queue. In our problem setup, the PFQ of each FB is represented by the G/M/1 queuing model, whereby the interarrival times are arbitrarily distributed and the service times are exponentially distributed [32]. The justification for adopting this model is that the PFQ receives different and arbitrary sizes of frame data or frame descriptors with different link speeds whereas the corresponding FB executes its function with a fixed speed.

2) Power Manager

The main goal of the power manager is to determine and execute a power management policy (i.e., one that maps workloads to power state transition commands so as to minimize the total system energy dissipation under a performance constraint) based on the information provided by the performance monitor. The power manager performs workload prediction and policy optimization. Details of the proposed workload prediction technique are explained in section IV while the performance optimization formulation, required to implement a workload-frequency mapping table, is discussed in section V.

3) Dynamic Frequency Adapter

When the workload of an FB changes greatly and frequently, the task of deciding what frequency value to assign to the FB becomes increasingly difficult. Furthermore, the conventional PLL-based frequency scaling techniques waste energy when they change the frequency values. To overcome these shortcomings, we present a workload-aware dynamic frequency adapter (DFA) to generate a *continuously varying* frequency for each FB.

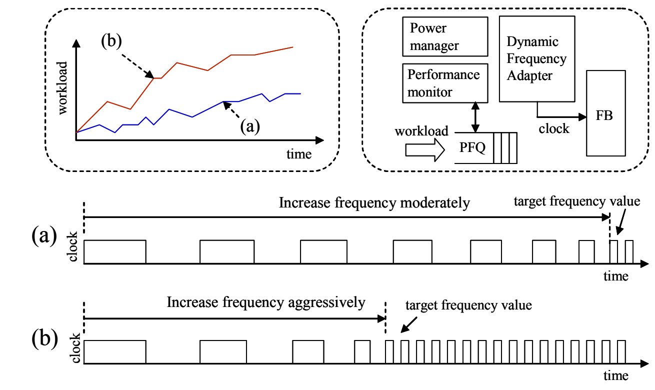

Fig. 3. Continuous frequency adjustment at a slow pace (a) or fast pace (b).

One benefit of using a variable frequency is that the DFA enables each FB to remain operational even when its frequency is being adjusted. The DFA is able to increase (or decrease) the operating frequency value at a slow or fast rate with the help of the performance monitor, depending on how slow or fast the workload is changing and what the user preferences are (cf. Fig. 3). The procedure for continuously adjusting the frequency is explained next.

The power manager examines the workload of each FB at *decision epoch*[1] $n+1$ for the time interval ranging from decision epoch n to $n+1$, and subsequently, sets the frequency value of each FB for the next time ranging from $n+1$ to next decision epoch at time $n+2$ (see below for an explanation of the frequency prediction algorithm). Assume that a mapping table for selecting an optimal operating frequency as a function of the present workload (i.e., required performance) has been provided. If the workload change is fast (slow), the interval during the frequency adjustment is performed will be shortened (lengthened) to improve the DFA responsiveness. In the proposed framework, determining which frequency level to use in what time interval is implemented *in hardware*.

The proposed DFA method is implemented in hardware inside the Ethernet controller chip. In this way, we also control noise and manage signal integrity. This is because if the variable clock signal is produced outside the chip, jitter (which is caused by several factors, e.g., crosstalk, power supply noise) will pose a significant challenge to board designers who must prevent sudden functional failures of the chip. In other words, by implementing the DFA inside the chip, we can reduce the impact of jitter on the chip's performance [33].

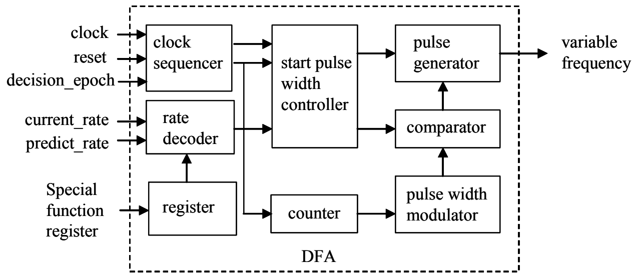

Fig. 4. Block diagram of the proposed dynamic frequency adapter module.

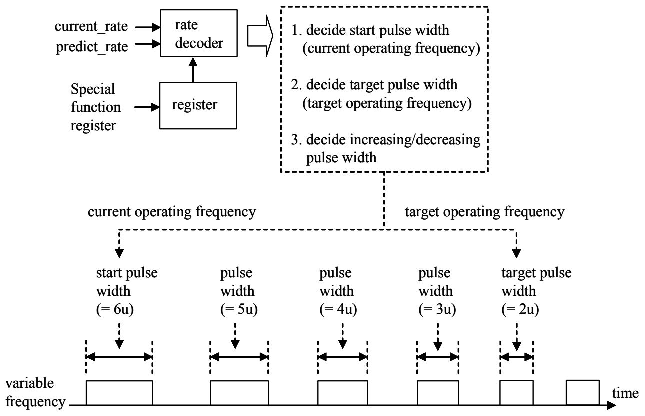
current_rate
predict_rate
rate decoder
Special function register
register
1. decide start pulse width (current operating frequency)
2. decide target pulse width (target operating frequency)
3. decide increasing/decreasing pulse width
current operating frequency
target operating frequency
start pulse width (= 6u)
pulse width (= 5u)
pulse width (= 4u)
pulse width (= 3u)
target pulse width (= 2u)
variable frequency
time

Fig. 5. Procedure for generating a variable frequency.

The block diagram of the DFA is depicted in Fig. 4 and is explained next. At each decision epoch, the power manager inputs values of the current and predicted frequencies to the DFA block, which translates these two values into a *start pulse width* and a *target pulse width*. The frequency adjustment is achieved by steadily changing the frequency from its start value toward the target value (see the register setting technique illustrated in Fig. 5). In our design, the frequency is increased when the pulse width is lowered. For example, the DFA generates a variable frequency between a minimum frequency set by a start pulse width of 6*usec* and a maximum frequency set by a pulse width of 2*usec*. The DFA uses a digital PWM (Pulse Width Modulator) [34] by means of a fast-clocked counter, which is loaded by input digital code (i.e., *current_rate* and *predict_rate* signals) at the beginning of the process. The variable frequency signal is input to a clock buffer

[1] Any regular or interrupt-based power management decision time instance is called a decision epoch.

(not shown in the figure) before it is supplied to any functional block. The DFA changes the frequency faster than a conventional PLL since it eliminates the lock-time of feedback loop of standard PLLs. Detailed functional simulation results (cf. Fig. 19) will be reported in section VI.

B. PFQ-based Packet Interface Architecture

We apply the proposed PFQ-based power management architecture to the packet interface modules inside the Ethernet controller, which includes interfaces between MAC and DMA. In this paper, we consider the packet interface between MAC and DMA (without involving IPSec and management function) to capture energy-saving opportunities by using the proposed architecture since this interface amply exhibits the competing requirements of low-latency and high-bandwidth processes. In general, the frame data is provisionally stored in memory buffers before being sent to local interconnect or network, while the control data is processed by a series of FBs, each requiring low-latency as shown in Fig. 1 (see steps (d), (e), and (g) in the packet receive path). Thus, this architecture targets the control dominated tasks rather than the storage and forwarding of the frame data. The event-queue mechanism of the PFQ enables multiple operating frequencies for the FBs, satisfying the low-latency control data access and the high-bandwidth frame data access. The interested reader should refer to the research work in [35] if interested in the DMA/PCI-related packet interfaces.

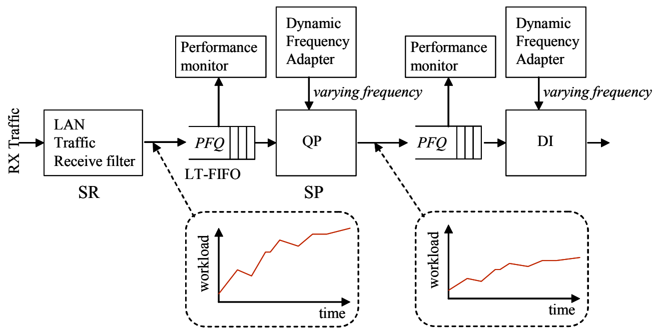

Fig. 6. Adaptation of PFQ-based power management structure to packet receive path.

Fig. 6 illustrates the adaptation of PFQ-based power management architecture to the packet receive path. For example, considering LT-FIFO (which receives control data of the LAN Traffic), the performance monitor observes the contents of PFQ (i.e., LT-FIFO); subsequently, the dynamic frequency adapter adjusts the operating frequency of the corresponding FB (i.e., QP) under the power manager's direction. Control blocks such as the Queue Placement (QP) and the Data Initiator (DI) interact with the RISC processor or the buffer manager for the packet receive path, while transferring memory buffer pointers to the ensuing PFQ so as to advance the sequence of tasks. The LAN traffic receive filter and the QP block are considered as the service requestor (SR) and the service provider (SP), respectively. Note that when the QP block is considered as a SR, then the DI block will play the role of a SP. In the following, details of the packet interface are described.

1) Packet Receive Interface

Fig. 7 shows the complete configuration of energy-efficient packet interface architecture based on the proposed PFQ. Detailed power management modules, which include the performance monitors, power manager, and dynamic frequency adapters, are omitted to simplify the figure. The RX-MAC determines exactly what type of in-bound traffic is routed to a host system through a series of packet receive control blocks, where a programmable filter placed in the receive MAC layer is responsible for filtering and tagging the in-bound traffic. The receive filters have special features to analyze and classify the incoming packets. The received packets, appearing in the form of a 64-bit word stream, are en-queued into the RX-FIFO, where MAC applies a programmed set of filters to such streams.

While the frame data is en-queued in the RX-FIFO, the control data (i.e., receive buffer descriptors) which are obtained by matching and filtering, are en-queued into a LT-FIFO, where these FIFOs have PFQ-based power management structure. Note that the receive buffer descriptors are used to keep track of packets being received from the Ethernet interface, where the packet receive interface is responsible for placing the received packets in the temporary memory buffer (i.e., RXBUF) along with an associated buffer descriptor. The receive buffer descriptor, which includes information about the starting address, end status, and packet length, is updated by hardware in order to indicate to the driver or the OS where the received packets are located. Hence, for every complete frame that resides in the RX-FIFO, there is a corresponding receive buffer descriptor bearing the filtering results in the LT-FIFO.

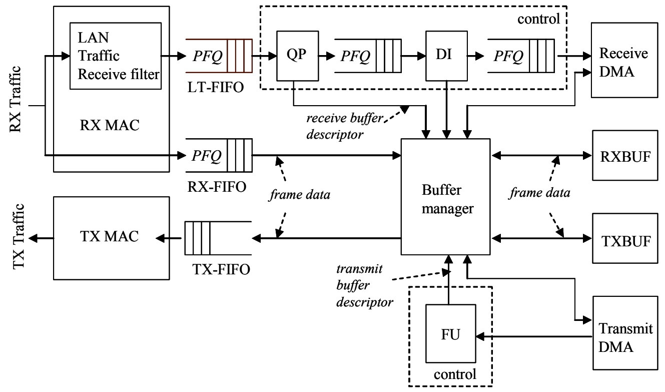

Fig. 7. The configuration with PFQ for packet interface architecture.

When a new frame starts filling the RX-FIFO, the MAC requests buffers (i.e., temporary space) from the RXBUF for the frame data (before it is sent to the host system via a receive DMA) and starts placing the in-bound frame into these buffers. At the end of the frame, the MAC pops the LT-FIFO, and takes appropriate actions based on the contents of the FIFO entries. As shown in Fig. 7, the frame data, which is stored in the RX-FIFO, is transferred to the temporary RXBUF via a buffer manager, and the control data used as a pointer to indicate the buffer location of its corresponding frame data is processed through the QP and DI control blocks. The QP and DI control blocks are used to monitor several indicators (e.g., diagnostics) during the reception of a packet and to update the information of buffer descriptor.

2) Packet Transmit Interface

The configuration of packet transmit interface is simpler than that of the receive interface. In order to transmit a packet through the MAC, the host system needs to construct the packet in the TXBUF. At the same time, the control data (i.e., transmit buffer descriptors) are configured by the driver or the OS in order to indicate to the packet transmit interface where the packets that are to transmitted are located in the TXBUF. Information such as the starting address, packet length, and VLAN tag is included in the transmit buffer descriptors. Next the MAC commits the frame by en-queuing the frame data into TX-FIFO, while the Frame Updater (FU) modifies the frame header with the VLAN tag and checksum fields. After the frame is transmitted, the MAC requests the buffer manager to de-allocate the list of buffers for the freshly transmitted packet. Simply speaking, the packet transmit interface is responsible for transmitting packets to the Ethernet link by reading the associated transmit buffer descriptors and the packet from the local temporary memory buffer (i.e., TXBUF). Note that the FPQ-based power management architecture is not applied to the packet transmit interface since the power manager already has knowledge about the rate of transmitting traffic, which enables a DFS technique to be easily performed based on this information alone.

IV. Workload Prediction-based Frequency Adjustment Technique

We present a frequency adjustment technique based on workload prediction for FPQ-based architecture, which is formulated as an initial value problem (IVP) [36]. We also describe our workload-driven dynamic frequency adjustment design.

A. IVP-based Workload Prediction

Assume that power manager of the PFQ-based architecture is able to monitor the current workload of the traffic at the decision epochs $t_1, \ldots, t_n$ where $t_{i+1} = t_i + T$. Let $w(t)$ denote the workload (i.e., the arrival rate of traffic) of a target FB at time t and let f be a function providing the operating frequency for the FB in every interval $[t_i, t_{i+1}]$. Then, an initial value problem (IVP) may be defined to predict $w(t)$ as follows:

$$\partial w / \partial t = f(t, w), \qquad w(t_i) = w_i \tag{1}$$

where $t \in [t_i, t_i + T]$, and w_i denotes the workload at the beginning of the current interval. The IVP limits the solution by an initial condition, which determines the value of solution at all future time t in the current interval [36]. Although f can be any general function, in practice, we assume a linear function form: $f=aw(t)+b$ where a and b are appropriately calculated slope and offset coefficients. Since the initial workload value is specified by the power manager, it is possible to integrate (1) to obtain $w(t)$ in the interval $[t_i, t_{i+1}]$. The standard solution method for the IVP is to approximate the solution of the ordinary differential equation by calculating the next value of w, i.e., $w(t+h)$ as the summation of the present value $w(t)$ plus the product of the size of a time step h and an estimated slope $w'(t)$ i.e.,

$$w(t+h) = w(t) + h \cdot w'(t) \tag{2}$$

where the smaller this time step h is, the more accurate the results will be. The difference between different ODE solvers is in how they approximate $w'(t)$ and whether and how they adaptively adjust h.

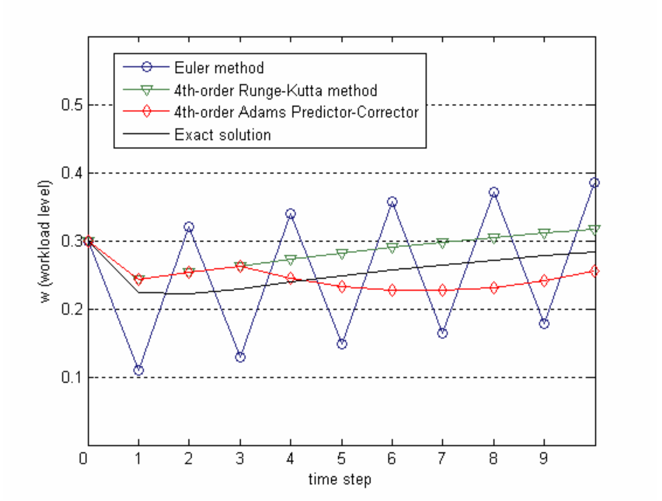

Fig. 8. Evaluation of various IVP solutions.

Considering the accuracy and overhead, we have evaluated a number of methods for solving the IVP, which include the Euler's method, the 4th-order Runge-Kutta method, and the 4th-order Adams predictor-corrector method (cf. Fig. 8). In this figure, we assume that $w(0) = 0.3$ as an initial value. The time step size is defined as $h = T/K$, where the time interval $[t_i, t_i + T]$ is divided into K equal-length segments. It is clearly seen that the Euler method, the simplest approach for solving the IVP, shows low accuracy (i.e., high error) in predicting the workload value, where the error is defined as the difference between the exact values and the computed approximates. However, the 4th-order Runge-Kutta method exhibits low error and consistent stability in predicting the workload value. The 4th-order Adams predictor-corrector method is also accurate, but has higher computational complexity.

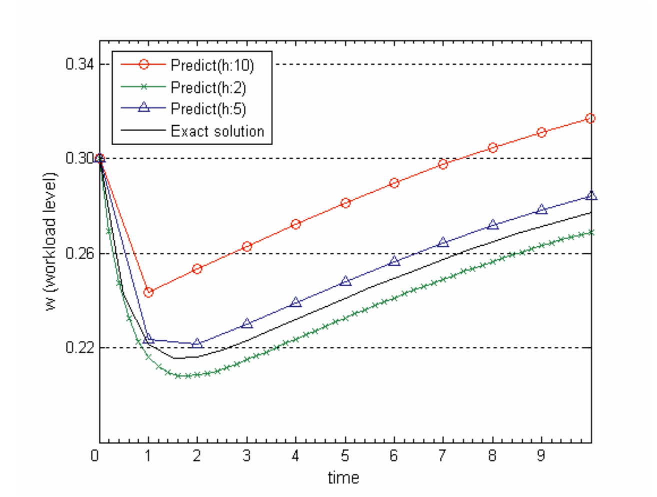

Fig. 9. Trade-off between the performance and time step h.

Fig. 9 shows the trade-off between the accuracy and time step h in terms of performance of the workload prediction technique, where time (the *x*-axis) is defined in terms of successive time steps. In this evaluation, the 4th-order Runge-Kutta method is used with an initial value of $w(0) = 0.3$. Determination of the time step size is crucial since although a small time step increases the computational overhead, it also improves accuracy, i.e., the 4th-order Runge-Kutta method requires four evaluations per time step h but its accuracy is improved. We use various values for time step size h (= 2, 5, and 10), where T is fixed, while monitoring the error in predicting the workload values. The time step of size 2 indicates great accuracy, but increases computational efforts by the software (due to more computations in the same interval), whereas step size of 10 exhibits lower computational efforts with lower accuracy. In our problem setup, we have empirically observed that a time step size of 5, which exhibits around 13% error, provides a reasonable trade-off point, where the time step size of 2 consumes around 4% and 9% increased CPU time compared to the size of 5 and 10, respectively, based on our simulations. Notice that the time steps of size 2 and 10 present around 4% and 60% error, respectively.

To make the workload prediction technique more suitable for online implementation, an efficient one-step method known as the *midpoint method* [36] is utilized to solve the IVP. Specifically, at time instance t, we predict the workload value for time $t + h$, based on the value at time $t + h/2$, which is obtained by using the midpoint method, as depicted in Fig. 10. First, the current workload at time t is monitored by the performance monitor and a frequency value is read from a pre-characterized *workload-frequency mapping table* (cf. Section V) by the power manager. Note that we do not use the predicted value for time t, which was previously computed at time $t - h$, because we can achieve the exact frequency value at time t. Next, the workload value at time $t + h/2$ is estimated by using a moving average method, for example, if the window size of the moving average calculator is 2, then, $w^{\text{pred}}(t + h/2) = 1/2\cdot(w^{\text{exact}}(t) + w^{\text{exact}}(t-h))$. This workload value is subsequently used as the midpoint estimate of the workload in the upcoming period. In particular, it is used along with $w^{\text{exact}}(t)$ to compute $w^{\text{pred}}(t + h)$ by applying the IVP.

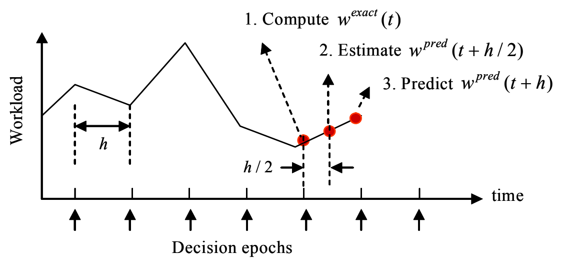

Fig. 10. Workload prediction technique based on the midpoint method and IVP.

The advantage of this prediction method is that we do not attempt to predict $w^{\text{pred}}(t + h)$ directly by using a moving average method only. Instead, we estimate the workload value for a nearer time in the future (which should provide higher accuracy) and use that value to initially estimate the rate of workload change in the upcoming period, followed by finally computing $w^{\text{pred}}(t + h)$ by solving the IVP.

B. Workload-Driven Frequency Adjustment

In our problem setup, mapping from a workload (i.e., the arrival rate of traffic) level to a corresponding optimal frequency value is performed based on prior (offline) simulations, as explained in section V. Since an Ethernet controller includes a number of FBs which run at different frequency values, the power manager must first predict the workload for each FB. Next, by utilizing a pre-characterized workload-frequency mapping table the DFA assigns the optimal frequencies to the corresponding FB.

The decision about the frequency adjustment interval is made based on the difference between $w^{\text{exact}}(t)$ and $w^{\text{pred}}(t + h)$. For example, if $w^{\text{pred}}(t + h) >> w^{\text{exact}}(t)$ ($w^{\text{pred}}(t + h) << w^{\text{exact}}(t)$), then the dynamic frequency adapter (DFA) will increase (decrease) the frequency quickly. On the other hand, the DFA increases (decreases) the frequency slowly if $w^{\text{pred}}(t + h)$ is only a little larger (smaller) than $w^{\text{exact}}(t)$. Fig. 11 shows the flow of dynamic frequency adjustment technique, where $f^{\text{pred}}(t + h)$ is the frequency value obtained from the predicted workload and the workload-frequency mapping table for the two cases where $w^{\text{pred}}(t + h)$ > or >> $w^{\text{exact}}(t)$. In Fig. 11, we have omitted the case of $w^{\text{pred}}(t + h)$ < or << $w^{\text{exact}}(t)$, which can be handled in a similar way. Note that when $w^{\text{pred}}(t + h) = w^{\text{exact}}(t)$, the current frequency value is maintained. It is worthwhile to mention that the DFA is capable of handling the throughput and power budget. If there is a target throughput, for example, the DFA will slowly increase the frequency up to a target frequency value that results in just-enough throughputs and the minimum power dissipation.

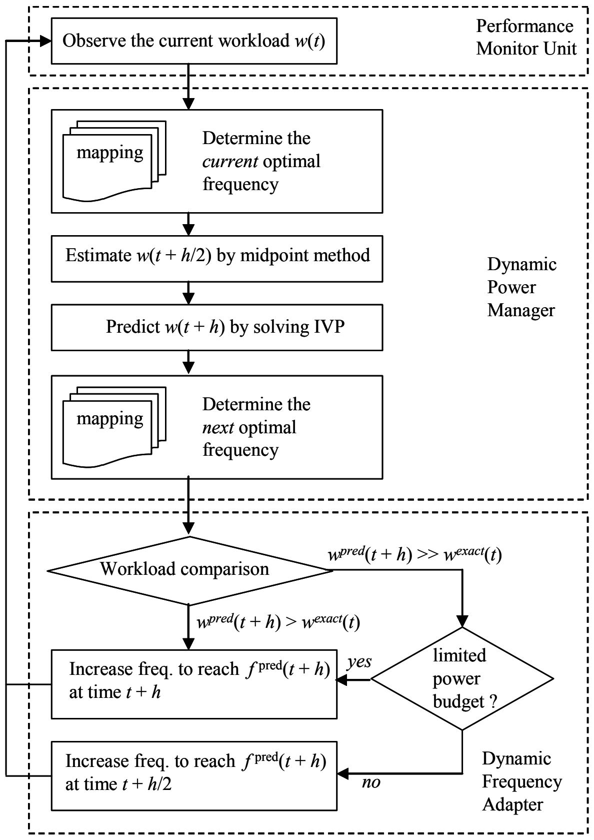
Observe the current workload $w(t)$
Performance Monitor Unit
mapping
Determine the *current* optimal frequency
Estimate $w(t+h/2)$ by midpoint method
Predict $w(t+h)$ by solving IVP
mapping
Determine the *next* optimal frequency
Dynamic Power Manager
Workload comparison
$w^{pred}(t+h) >> w^{exact}(t)$
$w^{pred}(t+h) > w^{exact}(t)$
Increase freq. to reach $f^{\text{pred}}(t+h)$ at time $t+h$
yes
limited power budget ?
Increase freq. to reach $f^{\text{pred}}(t+h)$ at time $t+h/2$
no
Dynamic Frequency Adapter

Fig. 11. The flow of dynamic frequency adjustment method.

V. Analysis and Performance Optimization

In this section, we first analyze the model of PFQ-based system, and then provide energy optimization formulation to find an optimal frequency values under performance constraints, used to implement a workload-frequency mapping table.

A. Model of PFQ-based Functional Block

Generally, network traffic is modeled as a sequence of arrivals of discrete packets, which enables the interarrival times to be treated a random process. In our formulation, the PFQ, which provides a queue mechanism, can be represented by the G/M/1 queuing model, where interarrival times are arbitrarily distributed and service times are exponentially distributed.

A general distribution is assumed for the interarrival times because an exponential distribution would underestimates the occurrence probability of a large request interarrival time and so it does not adequately model the request arrival time in the idle periods [38]. Furthermore, a widely used Poisson process model, whose traffic is characterized by assuming that the packet interarrival times are independent and exponentially distributed with some rate parameter, has limitations in capturing the traffic burstiness which characterizes the data traffic, since traffic burstiness is related to short-term auto-correlations between the interarrival times [39][40]. In particular, the Ethernet traffic exhibits statistically self-similar behavior [41], which is characterized by bursts, where the burstiness of the traffic exists over a wide range of time scales. As illustrated in [42], G/M/1 queuing model can be used to model a network with self-similar arrival times. The service time behavior is captured by a given service time distribution for the functional module when it is in the active modes. Similarly, the input request behavior is modeled by some interarrival time distribution.

Consider a FB with its dedicated PFQ inside the Ethernet controller, where the PFQ follows a first come, first served (FCFS) strategy. Let data packets (or bursts) arrive in the PFQ at time points t_n, for n = 0, 1, …, ∞. The interarrival time of tasks (i.e., packets or bursts), $i_n = t_{n+1} - t_n$ is assumed to be independent and identically distributed (i.i.d.) [43] according to an arbitrary distribution function F_A (density function f_a). Let λ denote the mean arrival rate of tasks. The mean interarrival time is thus equal to $1/\lambda$. We assume that the service times of the FB, T_S, are exponentially distributed with a mean value of $1/\mu$. Evidently, μ, which is the service rate of the FB, is a function of operation frequency of the FB. Then, the state of the G/M/1 model at time t can be described by the pair (x_t, r_t), where x_t is the number of tasks in the PFQ and FB at time t, and r_t is the residual interarrival time (i.e., the expected time remaining for the arrival of next packet). The two-dimensional process $\{(x_t, r_t), t \geq 0\}$ is a Markov chain, which follows the Markovian property [44], but requires complex analysis to compute the transition probabilities in the state space. Therefore, we resort to an embedded semi-Markov chain (SMC) model, which is simpler to analyze and yet sufficient for the purpose of power management technique in the context of PFQ-based FBs, as detailed next.

Definition 1 [45]: If two-dimensional process $\{(x_t, r_t), t \geq 0\}$ is a Markov chain, then $\{x_t, t \geq 0\}$ will be a semi-Markov chain. Note that the time spent in a particular state in the SMC (i.e., the *sojourn time* or the time difference between successive packet arrivals) follows an arbitrary probability distribution, which is a more realistic assumption than an exponential distribution used in the conventional Markov process model [9]. To specify the state probabilities of this SMC, we first consider the probability, $a_n(t)$, that n tasks are served by the FB during the sojourn time,

$$a_n(t) = \int_0^\infty \frac{(\mu t)^n}{n!} e^{-\mu t} f_a(t)dt\ , \qquad n = 0,1,\ldots,\infty \tag{3}$$

where $f_a(t)$ is the probability density function of interarrival time which is arbitrarily distributed. Notice that (3) follows from the fact that the number of service completions by the FB within the sojourn time constitutes a Poisson process since the time between successive services by the FB is exponentially distributed. Then, the equilibrium probability, q_n, of being in a state where there are n tasks in the PFQ and FB just before a new task arrives is calculated as:

$$q_n = (1-\sigma)\sigma^n\ ,\ n = 0,1,...,\infty \tag{4}$$

where $0 < \sigma < 1$ is the unique real solution of one-sided Laplace-Stieltjes transform (LST) of the interarrival time distribution function [32], which is in turn calculated as:

$$\begin{aligned}\sigma &= \sum_{n=0}^{\infty} \sigma^n a_n \\ &= \sum_{n=0}^{\infty} \sigma^n \int_0^\infty \frac{(\mu t)n}{n!} e^{-\mu t} f_a(t)dt \\ &= \int_0^\infty e^{-(1-\sigma)\mu t} f_a(t)dt\end{aligned} \tag{5}$$

Let T_W and T_S represent the *mean waiting time* and the *mean service time* of the tasks in the PFQ and FB, respectively. The *mean response time*, T_R, of the FB is the expected time that the tasks spend waiting in the PFQ plus the time taken for processing in the FB. T_R is calculated as:

$$T_R = \frac{1}{\mu(1-\sigma)} \tag{6a}$$

The time spent waiting in the PFQ is calculated by subtracting the service time T_S from the response time, yielding

$$T_W = T_R - \frac{1}{\mu} = \frac{\sigma}{\mu(1-\sigma)} \tag{6b}$$

With regard to the performance efficiency, we consider the utilization of the FB, i.e., how much of the computational resource provided by the FB is exploited by the application. More precisely the utilization ratio, u, is defined as:

$$u \equiv \frac{E[BP]}{E[BP]+E[IP]} = \frac{\lambda}{\mu} \tag{7}$$

where $E[BP]$ denotes the expected duration of the busy period (when there is at least one task, and thus the FB is busy) of the FB, while $E[IP]$ denotes the expected duration of its idle period (when there are no tasks, and hence, the FB is idle). Without presenting the proof, we simply state the following [32],

$$E[BP]+E[IP] = \frac{1}{\lambda(1-\sigma)} \tag{8}$$

Thus, considering the proportion of idle time, we can calculate

$E[BP]$ and $E[IP]$ as follows:

$$E[BP] = \frac{1}{\mu(1-\sigma)}, \quad E[IP] = \frac{\mu - \lambda}{\lambda\mu(1-\sigma)} \tag{9}$$

TABLE I shows the simulated results for the G/M/1 PFQ model, assuming that the interarrival times are generally distributed with arrival rate ($0 < \lambda < 1$), and $T_S = 1/\mu = 1$ for simplicity.

TABLE I
SIMULATION RESULTS FOR PREDICTIVE-FLOW-QUEUE MODEL

	Arrival rate (λ)						
	0.3	0.4	0.5	0.6	0.7	0.8	0.9
σ	0.056	0.139	0.270	0.416	0.569	0.724	0.868
T_R	1.060	1.162	1.369	1.712	2.320	3.622	7.597
T_W	0.060	0.162	0.369	0.712	1.320	2.622	6.597
$E[BP]$	1.059	1.161	1.369	1.712	2.320	3.623	7.576
$E[IP]$	2.471	1.742	1.369	1.141	0.994	0.906	0.842

B. Energy Optimization Formulation

The proposed framework relies on a workload-frequency mapping table, which will provide the optimum frequency assignment for a FB based on the workload that the FB encounters *online*. To construct this workload-frequency mapping table, we formulate the energy optimization problem as a mathematical program. More precisely, mapping from a workload (i.e., the arrival rate of tasks) to a corresponding optimal frequency value, which affects the service time, T_S, and waiting time, T_W, is performed by solving an *offline* energy optimization problem. Assuming that an operating frequency value, f, is given, the energy dissipation of a FB and corresponding PFQ can be computed as

$$\begin{aligned} ene_{FB-PFQ} &= E[BP]\cdot pow_{A\_FB} + T_W \cdot pow_{A\_PFQ} \\ &\quad + E[IP]\cdot(pow_{I\_FB} + pow_{I\_PFQ}) \\ &= \frac{1}{\mu(1-\sigma)}\cdot pow_{A\_FB} + \frac{\sigma}{\mu(1-\sigma)}\cdot pow_{A\_PFQ} \\ &\quad + \frac{\mu-\lambda}{\lambda\mu(1-\sigma)}\cdot(pow_{I\_FB} + pow_{I\_PFQ}) \end{aligned} \tag{10}$$

Here pow_{A_FB} and pow_{A_PFQ} denote the expected power consumptions for the FB and corresponding PFQ in the active mode, respectively, whereas pow_{I_FB} and pow_{I_PFQ} denote the expected power consumptions of these two components in the idle mode. Note that pow_{A_PFQ} (i.e., memory power) is affected by an operating frequency, besides the arrival rate of tasks, as illustrated in Fig. 12. This figure shows the power consumption of the PFQ (i.e., pow_{A_PFQ}) in the active mode for write/read operations in terms of the normalized arrival rate of packet. In this simulation, we set the packet size to 64bytes, and set the operating supply voltage to 1.20V. For example, when the arrival rate of the traffic is 0.8 (normalized), the operating frequency for PFQ is around 10 times greater than the case of when the arrival rate is 0.08.

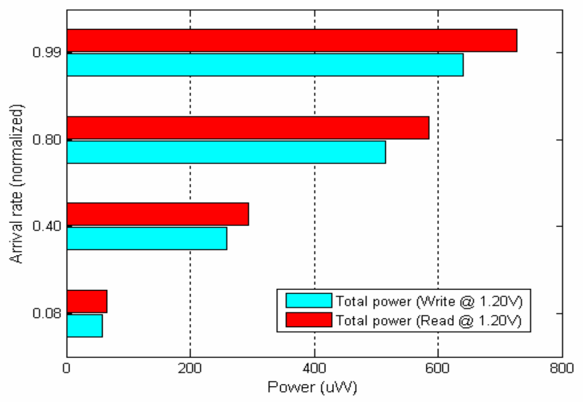

Fig. 12. Power consumption of the predictive-flow-queue in 65nm technology.

After determining the relevant parameters, we set up a mathematical program to solve the performance optimization problem as a linear program. The goal is to minimize energy consumption of the FB and PFQ, given a data packet arrival rate, by choosing an optimal service rate, μ, corresponding to a frequency assignment for the FB. The results are used to fill in various entries of the workload-frequency mapping table.

$$\begin{aligned} &\underset{\mu}{\text{minimize}} \quad ene_{FB-PFQ} \\ &\text{s.t.} \quad T_W + T_S \le T_{UB} \\ &\qquad\; u \ge u_{LB} \end{aligned} \tag{11}$$

Note that T_{UB} is an upper bound on the execution time of tasks going through the FB and its PFQ, and u_{LB} is a lower bound on the utilization of FB, which is provided by the user or application. This linear program is solved by using a standard mathematical program solver (i.e., MOSEK [46]). Although the formulation describes energy optimization for a single FB, system-wide energy minimization can be achieved easily in the same manner.

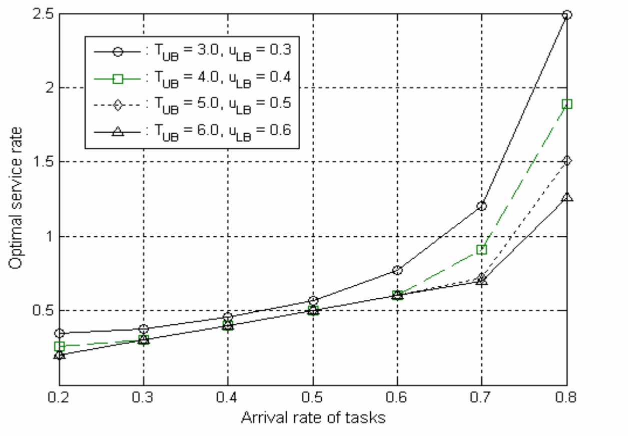

Fig. 13. Optimal service rate as a function of task arrival rates for different combinations of performance constraints.

Fig. 13 shows the results of proposed mathematical programming model with various performance constraints (i.e., T_{UB} and u_{LB}), where the goal is to find an optimal service rate which minimizes the energy dissipation of the FB and corresponding PFQ. For example, based on data given in

TABLE I, for the performance constraints T_{UB} = 3.0 and u_{LB} = 0.3 and task arrival rate of 0.6, the energy-optimal service rate is 0.77. Additional experiments for various scenarios are reported in section VI.

The entries of the workload-frequency mapping table correspond to various combinations of workloads and performance constraints. Fig. 14 illustrates the mapping process from workloads to an optimal operating frequency. In this figure, the mapping table is achieved through extensive offline simulation during design time, considering performance characteristics of each FB provided by the user or application. For example, when a power manager predicts the workload for the near future, an optimal frequency value for the next decision epoch is selected and provided to the dynamic frequency adapter which will continuously change the operating frequency from its present value to the target value. Note that mapping from workload to operating frequencies is achieved by a simple linear function while considering the maximum and minimum operating frequencies that can be applied to the FB in question.

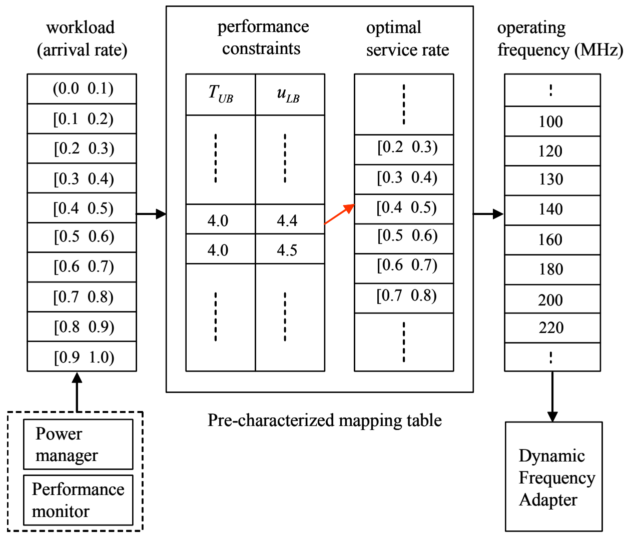

Fig. 14. Mapping of workloads to optimal operating frequency values for each FB.

VI. Experimental Results

In the experimental setup, we applied the proposed PFQ-based power management architecture to a gigabit Ethernet controller, where the Ethernet controller is implemented with TSMC 65nmLP library. Note that, as mentioned before, we considered a part of packet interface between MAC and DMA to capture power-saving opportunities by using the proposed architecture, as shown in Fig. 15.

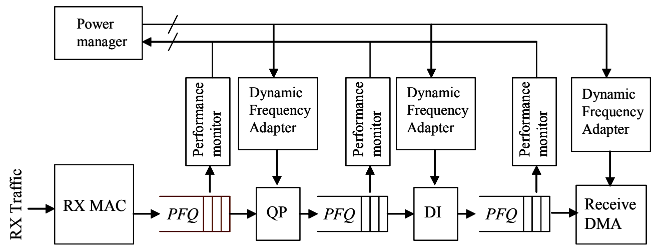

Fig. 15. Adaptation of PFQ-based power management structure with dynamic frequency scaling technique.

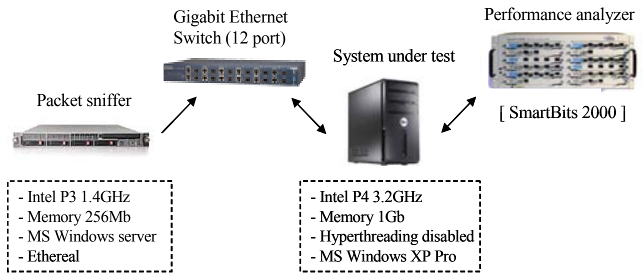

Fig. 16. Performance test configuration.

In the first experiment, we characterize the performance of designed Ethernet controller by obtaining the throughput for data streams with different packet sizes. The performance test setup is configured as shown in Fig. 16. In this setup, the packet sniffer is mainly used for the purpose of collecting traces and debugging, whereas the performance analyzer (SmartBits 2000 [47]) is used to generate various packet streams with the fixed inter-packet gap of 0.096us. TABLE II reports the performance characteristics of the implemented Ethernet controller obtained by measuring the throughput for various data streams.

TABLE II
PERFORMANCE CHARACTERISTICS OF ETHERNET CONTROLLER

Packet size (bytes)	Service rate (pkt/sec)	Inter-arrival time (sec)	Arrival rate (pkt/sec)	Service time (sec)
1518	84819	12.20E-6	81699	11.71E-6
1024	124936	8.28E-6	120656	8.00E-6
512	245100	4.19E-6	238549	4.08E-6
256	317400	2.14E-6	466417	3.15E-6
128	325200	1.12E-6	892857	3.07E-6
64	338000	0.60E-6	1644736	2.95E-6

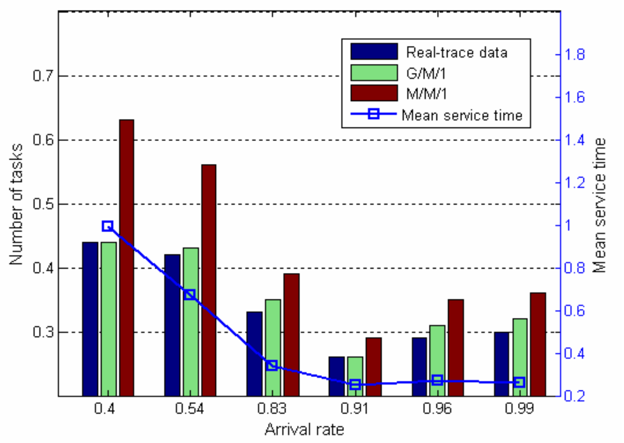

Fig. 17. Evaluation of the G/M/1 model for the predictive flow queue.

The second experiment was designed to evaluate the efficacy of our modeling technique for the PFQ, as represented by a G/M/1 queuing model. We characterize the network traffic in terms of the arrival rate based on our G/M/1 queuing model and compare these results with both the actual trace data from real-application, i.e., obtained data from SmartBits 2000 (see TABLE II), and those achieved by the conventional M/M/1 queuing model (which assumes an exponential distribution for interarrival times). Fig. 17 shows that the G/M/1 model for the PFQ gives more accurate performance results compared to the conventional model. In this figure, all values are normalized to the real-trace data.

TABLE III
NORMALIZED ENERGY DISSIPATION VALUES FOR VARIOUS WORKLOADS UNDER DIFFERENT COMBINATIONS OF SUPPLY AND THRESHOLD VOLTAGES

Arrival rate	Vth	Vdd	QP		DI		DMA	
			E_{Active}	E_{Idle}	E_{Active}	E_{Idle}	E_{Active}	E_{Idle}
0.8	Vth.h	1.00	15.64	4.4E-4	41.87	9.9E-4	64.80	1.9E-4
		1.08	22.46	1.4E-4	48.89	3.3E-4	75.58	4.2E-4
		1.20	28.48	9.5E-5	58.92	5.3E-4	93.20	4.1E-4
	Vth.l	1.00	15.58	9.3E-4	41.81	1.9E-3	64.91	7.1E-3
		1.08	18.46	2.3E-4	48.99	5.2E-4	75.12	1.9E-3
		1.20	22.53	3.3E-4	58.82	8.5E-4	93.14	2.9E-3
0.7	Vth.h	1.00	9.99	4.9E-4	26.72	1.1E-4	41.37	2.2E-3
		1.08	11.82	1.7E-4	31.10	3.7E-4	48.26	4.1E-4
		1.20	14.33	1.1E-4	38.14	5.9E-4	59.45	4.8E-4
	Vth.l	1.00	9.99	1.0E-3	26.72	2.1E-4	41.43	8.2E-3
		1.08	11.77	2.5E-4	31.16	6.7E-4	48.31	2.2E-3
		1.20	14.28	3.7E-4	38.24	9.5E-4	59.45	3.2E-3
0.6	Vth.h	1.00	7.35	6.0E-4	19.75	1.2E-4	30.56	2.4E-3
		1.08	8.71	1.6E-4	23.07	4.2E-4	35.72	4.7E-4
		1.20	10.63	1.2E-4	28.61	6.9E-4	43.95	5.3E-3
	Vth.l	1.00	7.35	1.1E-3	19.86	2.3E-4	30.56	8.2E-3
		1.08	8.66	3.1E-4	23.07	6.7E-4	35.67	2.9E-3
		1.20	10.63	4.4E-4	28.61	9.5E-4	43.85	3.7E-3

In the third experiment, we set the performance constraints on the T_{UB} and u_{LB} (e.g., $T_{UB} = 5$ and $u_{LB} = 0.5$) as in (11) to evaluate the proposed energy optimization technique based on mathematical programming models. The solution of the performance optimization problem produces an optimal frequency level for the FB. Different arrival rates (λ) of traffic are used to generate the multiple rows in TABLE III, which represents the normalized energy dissipation for various FBs (e.g., QP, DI, and DMA) in the active and idle modes, where we use (10) to calculate optimal energy dissipation. In this table, we report energy dissipation values for each FB for each possible combination of supply voltage (e.g., 1.00V, 1.08V, and 1.20V) and (high and low) threshold voltages, available with the TSMC 65nmLP library. To achieve power values, we generated and utilized SAIF (Switching Activity Interchange File) based on RTL simulation of the system with the Synopsys Power Compiler [48].

We next investigate various scenarios corresponding to different workloads for each FB as reported in TABLE IV. We performed static timing analysis with the Design Compiler [48] to assign low threshold voltage transistors in the logic gates on the timing critical paths, while using high threshold voltage transistors in all other logic gates. We fixed the supply voltage to 1.20V. Experimental results in this table (which correspond to the performance constraints T_{UB} =5 and u_{LB} =0.5) demonstrate that the energy optimization technique, achieved by solving mathematical program described in (11), results in up to 19.51% and 56.41% energy savings for active and idle modes, respectively.

TABLE IV
ENERGY SAVINGS BASED ON OPTIMIZATION TECHNIQUE (NORMALIZED)

Workload: arrival rate			Total energy (typical)		Optimal policy		Savings	
QP	DI	DMA	active	idle	active	idle	active	idle
0.8	0.7	0.6	90.26	2.9E-3	72.65	1.3E-3	19.51%	54.72%
0.7	0.6	0.5	64.23	4.0E-3	51.89	1.7E-3	19.21%	56.41%
0.6	0.7	0.8	117.74	4.1E-3	95.21	1.9E-3	19.14%	53.25%
0.5	0.6	0.7	83.07	3.5E-3	67.16	2.1E-3	19.15%	38.10%

Finally, we investigated the energy-efficiency of the proposed PFQ-based interface architecture. In particular for this experiment we apply the proposed technique to QP, DI, DMA and their corresponding PFQs in Fig. 15. We assumed that the workload (i.e., the arrival rate) changes dynamically from 0.1 to 0.9. For comparison purpose, we implemented a couple of power management policies (denoted by PM1 and PM2 and described below) as representatives of the conventional methods, similar to [49][50][51]. We use three set of frequency values to simplify the experimental setup ($F_1 < F_2 < F_3$ in terms of operating frequency values).

PM1: Utilize dynamic frequency scaling technique, while accounting for a 100us power-mode transition overhead; the frequency assignment policy is as follows.

- Use the lowest F_1 value when 0.1 ≤ the arrival rate ≤ 0.3, i.e., low workload.
- Likewise, use F_2 and F_3 values when 0.3 < the arrival rate ≤ 0.6 and 0.6 < the arrival rate ≤ 0.9, respectively.

PM2: The same as PM1 except that frequency change is avoided when the same frequency changes occur consecutively. More precisely, let F^i denote the value of frequency at time *i*.

- Set $F^{i+1} = F^i$, if the predicted F^{i+1} value is the same as F^{i-1} value. (e.g., the sequence of frequency assignments F_3, F_1, F_3 is modified to F_3, F_1, F_1).

Next, we randomly generated dynamic workloads with 50, 100, 500, and 1000 decision epochs at fixed intervals (to simplify the simulation setup) and applied the above-mentioned conventional power management policies and the proposed technique to the PFQ-based architecture. The simulation results in Fig. 18, which corresponds to the case of 50 decision epochs, show that the proposed technique achieves energy savings compared to the conventional methods. Note that the percentages of frequency change suppressions by PM2 are 12%, 11%, 14%, and 12% for the cases of 50, 100, 500, and 1000 decision epochs, respectively.

Results in TABLE V, which also reports the characteristics of the workload distribution (e.g., Low indicates that 0.1 ≤ the arrival rate ≤ 0.3), demonstrate that, compared to the PM1 policy, our approach (column PFQ in the table) achieves power and energy savings of up to 13.8% and 12.2%, respectively.

Note that the overhead of implementing the DFA block inside the Ethernet controller is insignificant in terms of the area and power dissipation because the DFA block consumes less than 1mW of power and requires a small number of gates i.e., fewer than 200 standard cells. To implement the DFA block in hardware, we used a digital PWM architecture similar to that reported in [34]. Furthermore, the overhead of accessing the pre-characterized mapping table to provide the target frequency is also negligible in our specific system in terms of performance and memory size, because the operating frequency is rather low (e.g., below 200MHz), which is enough for the power manager to select the target frequency from the small mapping table and send it to the corresponding dynamic frequency adaptor at each decision epoch.

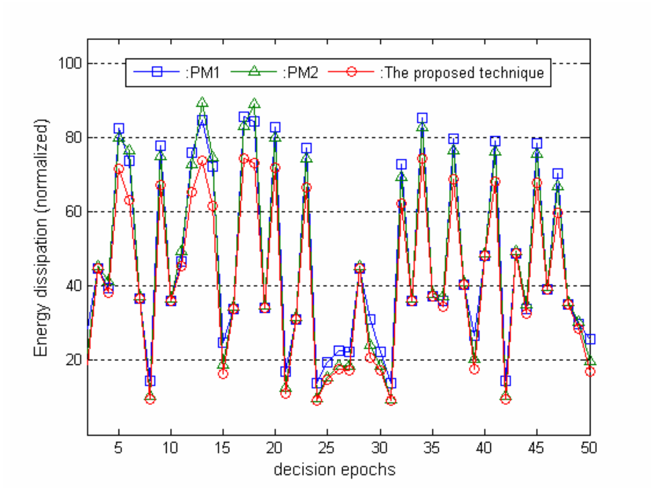

Fig. 18. Evaluation of the proposed energy-efficient PFQ-based architecture.

TABLE V
POWER AND ENERGY SAVINGS OF THE PFQ-BASED ARCHITECTURE

No. of decision epoch	Workload distribution			Average Power (mW)			Power saving over		Energy saving over	
	Low	Mid	High	PM1	PM2	PFQ	PM1	PM2	PM1	PM2
50	17	22	11	12.8	11.2	10.9	13.1%	3.6%	11.8%	9.7%
100	29	47	24	13.2	11.6	11.3	13.6%	2.9%	11.9%	9.5%
500	142	214	144	13.3	11.8	11.6	12.9%	3.2%	11.9%	10.2%
1000	326	376	298	13.4	11.7	11.5	13.8%	3.0%	12.2%	9.8%

We provide in Fig. 19 the functional gate-level simulation results for dynamic frequency adapter (DFA) block (e.g., signal CORE_CLK of control block is dynamically adjusted by DFA), to verify the implementation of the proposed architecture. This figure shows that the control data (e.g., receive buffer descriptors C0002 and C0604) are transferred to the control blocks from the EMAC via PFQs, while performing a packet receive process, whereas the frame data is directly moved to DMA from the EMAC through the temporary memory buffer.

VII. CONCLUSION

A predictive-flow-queue based energy-efficient packet interface architecture for the gigabit Ethernet controller design was presented, where we used a dynamic frequency adjustment technique to minimize the energy dissipation during power-mode transitions. We described a systematic approach for workload prediction and system modeling techniques based on the initial value problem and stochastic processes so as to achieve mathematical programming formulation for optimizing energy dissipation. The proposed architecture achieves high-throughput frame data and low-latency control data processes while conserving the system power consumption. Experimental results with the 65nm gigabit Ethernet controller design have demonstrated that the proposed architecture results in significant energy savings for various workloads under demanding performance constraints.

REFERENCES

[1] ASF (Alert Standard Format) and DMWG (Desktop and Mobile Work Group) document. http://www.dmtf.org.

[2] Sun Blade 8000 Modular System. http://www.sun.com/servers/blades.

[3] P. Willmann, H. Kim, S. Rixner, and V.S. Pai, "An Efficient Programmable 10 Gigabit Ethernet Network Interface Card," *Proc. of Symposium on High-Performance Computer Architecture*, Feb. 2005, pp. 96-107.

[4] Intel 82547 Gigabit Ethernet Controller. http://www.intel.com.

[5] L. Benini, and G. De. Micheli, *Dynamic Power Management: Design Techniques and CAD Tools*, Kluwer Academic Publishers, 1998.

[6] D. Li, Q. Xir, and P. H. Chou, "Scalable Modeling and Optimization of Mode Transitions based Decoupled Power Management Architecture," *Proc. of Design Automation Conference*, Jun. 2003, pp. 119-124.

[7] Y-H, Lu and G. De Micheli, "Comparing System-Level Power Management Policies," *IEEE Design & Test of Computers*, Vol. 18, Issue 2, pp. 10-19, Mar-Apr. 2001.

[8] IEEE 802.3 Tutorials. July 2005. http://www.ieee802.org/802_tutorials.

[9] M.L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Publisher, New York, 1994.

[10] *Mobile AMD Athlon 4 Processor Model 6 CPGA Data Sheet Rev:E.* Tech. Rep. 24319, Advanced Micro Devices. 2001. www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/24319.pdf .

[11] *Intel 80200 Processor Based on Intel XScale Microarchitecture*, http://developer.intel.com/design/iio/manuals/273411.htm .

[12] C. Lichtenau, et al., "PowerTune: Advanced Frequency and Power Scaling on 64b PowerPC Microprocessor," *Int'l Solid-State Circuits Conference Dig. Tech. Papers*, Feb. 2004.

[13] S. Geissler et al., "A Low-Power RISC Microprocessor Using Dual PLLs in a 0.13µm SOI Technology with Copper Interconnect and Low-k BEOL Dielectric," *Int'l Solid-State Circuits Conference Dig. Tech. Papers*, pp. 148-149, Feb. 2002.

[14] C. McNairy and R. Bhatia, "Montecito: A Dual-Core, Dual-Thread Itanium Processor," *IEEE Micro*, Vol. 25, Issue 2, Mar.-Apr. 2005, pp. 10-20.

[15] C. Poirier et al., "Power and Temperature Control on a 90 nm Itanium Architecture Processor," *Int'l Solid State Circuits Conf. Dig. Tech. Papers*, 2005, pp. 304-305.

[16] T. Fischer et al., "A 90 nm Variable Frequency Clock System for a Power-Managed Itanium Architecture Processor," *Int'l Solid State Circuits Conf. Dig. Tech. Papers*, 2005, pp. 294-295.

[17] T. Senanayake, T. Ninomiya, and H. Tohya, "Fast-Response Load Regulation of DC-DC Converter By means of Reactance Switching," *Proc. of Conf. on Power Electronics Specialists*, Jun. 2003, pp.1157-1162.

[18] B. Zhai, D. Blaauw, D. Sylvester, and K. Flaunter, "Theoretical and Practical Limits of Dynamic Voltage Scaling," *Proc. of Design Automation Conference*, Jun. 2004, pp. 863-873.

[19] PCI-Express Document. http://www.pcisig.com/specification.

[20] RDMA Consortium. http://www.rdmaconsortium.org.

[21] The Internet Engineering Task Force. http://www.ietf.org/rfc/rfc2401.

[22] NetXtreme^{TM} Gigabit Ethernet Controller. http://www.broadcom.com.

[23] X. Liu, P. Shenoy, and M. Corner, "Chameleon: Application Level Power Management with Performance Isolation," *Proc. of Int'l Multimedia Conference*, Nov. 2005, pp.839-848.

[24] B. Mochocki, D. Rajan, X.S. Hu, C. Poellabaucer, K. Otten, and T. Chantem, "Network-Aware Dynamic Voltage and Frequency Scaling," *Proc. of Real Time and Embedded Technology and Application Symposium*, Apr. 2007, pp.215-224.

[25] Y. Hotta, M. Sato, H. Kimura, S. Matsuoka, T. Boku, and D. Takahashi, "Profile-based Optimization of Power Performance by using Dynamic Voltage Scaling of a PC cluster," *Proc. of Parallel and Distributed Processing Symposium*, Apr. 2006, pp.8-16.

[26] V. Soteriou, N. Eisley, and L. Peh, "Software-directed power aware interconnection networks," *ACM Trans. on Architecture and Code Optimization*, Vol. 4, Issue 1, Mar., 2007.

[27] F. Li, G. Chen, and M. Kandemir, "Compiler-directed voltage scaling on communication links for reducing power consumption," *Proc of Int'l Conference on Computer-Aided Design,* Nov. 2005, pp.456-460.

[28] M. Najibi, et al., "Dynamic Voltage and Frequency Management Based on Variable Update Intervals for Frequency Setting," *Proc. of Int'l Conference Computer Aided Design*, Nov. 2006, pp. 755-760.

[29] K. Choi, R. Soma, and M. Pedram, "Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times." *IEEE Trans. on Computer Aided Design*, Vol. 24, No. 1, Jan. 2005, pp.18-28.

[30] N. L. Binkert, L. R. Hsu, and A. G. Saidi, "Performance Analysis of System Overheads in TCP/IP Workloads," *Proc. of Conf. on Parallel Architectures and Compilation Techniques*, Sep. 2005, pp.218-230.

[31] C. Isci, M. Martonosi, and A. Buyuktosunoglu, "Long-term Workload Phase: Duration Predictions and Application to DVFS," *IEEE Micro*, Vol. 25, No. 5, pp.39-51, Sep-Oct. 2005.

[32] S. M. Ross, *Introduction to Probability Models*, Academic Press, 8th edition, Dec. 2002.

[33] Allan Liu, "Signal Integrity and Clock System Design," White paper. http://www.idt.com.

[34] A. Syed, E. Ahmed, D. Maksimovic, and E. Alarcon, "Digital Pulse Width Modulator Architectures," *Proc. of Conf. on Annual IEEE Power Electronics Specialists*. Jun. 2004, pp.4689-4695.

[35] Y. Hoskote, et al., "A TCP Offload Accelerator for 10Gb/s Ethernet in 90nm CMOS," *IEEE Journal of Solid-State Circuits*, Vol. 38, No. 11, pp. 258-270, Nov. 2003.

[36] J. R. Dormand, *Numerical Methods for Differential Equations: A Computational Approach*, CRC Publisher, Feb. 1996.

[37] Ian W.C. Lee, and Abraham O. Fapojuwo, "Stochastic processes for computer network traffic modeling," *Journal of Computer Communication*, Vol. 29, Issue 1, pp. 1-23, Dec. 2005.

[38] T. Simunic, L. Benini, P. Glynn, and G. De Micheli, "Event-driven Power Management," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, Vol. 20, Issue 21, pp. 840-857, Jul. 2001.

[39] R. Jain and S. Routhier, "Packet Trains – Measurements and a New Model for Computer Network Traffic," *IEEE Journal on Selected Areas in Communications*, Vol. 6, Issue 6, pp. 986-995, Sep. 1986.

[40] W. Willinger and V. Paxson, "Where Mathematics Meets the Internet," *Notices of the American Mathematical Society*, Vol. 45, No. 8, pp. 961-970, 1998.

[41] E. A. Smith, "Understanding performance issues in IP networks," *BT Technology Journal*, Vol. 24, No. 4, pp. 167-178, Oct. 2006.

[42] Y. G. Kim and P. S. Min, "On the prediction of average queuing delay with self similar traffic," *Proc. of Global Telecommunications Conference*, Dec. 2003, pp. 2987-2991.

[43] S. K. Bose, *Introduction to Queuing Systems*, Kluwer Publishers, 2001.

[44] I. Adan, A. G. Kok, and J. Resing, "A multi-server queuing model with locking," *European Journal of Operational Research*, Vol. 116, pp. 249-258, 1999.

[45] J. Yackel, "A Characterization of Normal Markov Chains," *Proc. of the American Mathematical Society*, Vol. 19, No. 6, pp. 1464-1468, Jul. 1967.

[46] MOSEK Optimization Solver. http://www.mosek.com.

[47] SmartBit2000, Performance Analyzer. http://www.spirentcom.com.

[48] Synopsys Compiler Documents. http://www.synopsys.com.

[49] Q. Wu, P. Juang, M. Martonosi, and D.W. Clark, "Voltage and Frequency Control with Adaptive Reaction Time in Multiple-Clock Domain Processors," *Proc. of 11th Symposium on High-Performance Computer Architecture*, Feb. 2005, pp. 178-189.

[50] P. Choudhary and D. Marculescu., "Hardware Based Frequency/Voltage Control of Voltage Frequency Island Systems," *Proc. of Int'l Conf. on Hardware/Software Co-design and System Synthesis*, Oct. 2006, pp. 34-39.

[51] R. Jejurikar and R. Gupta, "Dynamic Voltage Scaling for System-wide Energy Minimization in Real-time Embedded Systems," *Proc. of International Symposium on Low Power Electronics and Design*, Aug. 2004, pp. 78-81.

[52] Hwisung Jung, Andy Hwang, and Massoud Pedram, "Flow-Through-Queue based Power Management for Gigabit Ethernet Controllers," *Proc. of Asia and South Pacific-Design Automation Conference*, Jan. 2007, pp. 571-576.

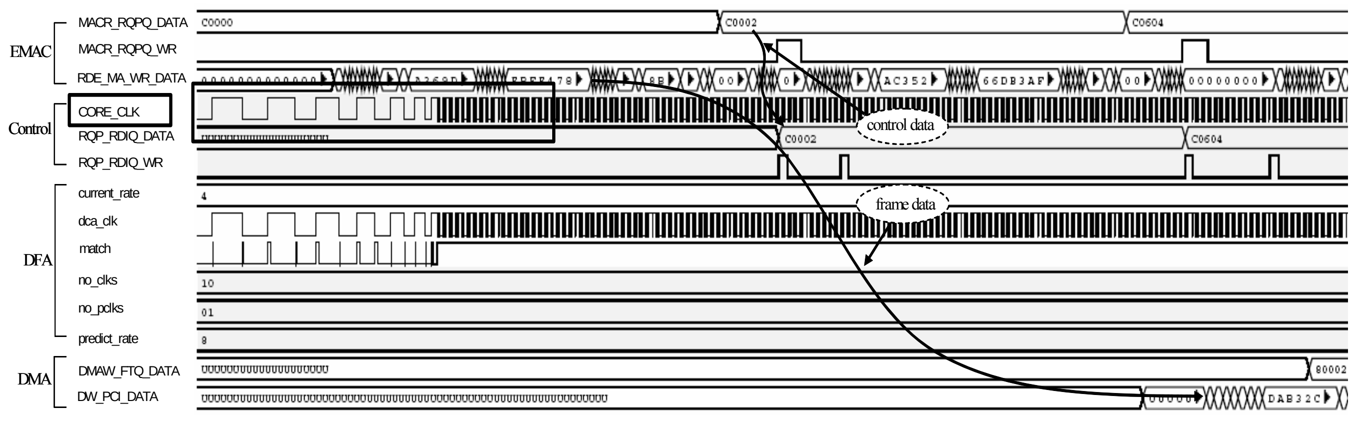

Fig. 19. The simulation result for the proposed DFA which adjusts the clock frequency of the control block.