Multi-state Power Management of Communication Links

Krishna Kant, Intel Corporation

***Abstract*—The continuing speed increases of communication links at all levels from intra-server interconnects all the way to WAN results an increasing amount of network power consumption. Thus an effective, multi-level and multi-scale power management of communication links is increasingly important. In this paper we propose algorithms that can reduce link power consumption by exploiting three key attributes: (a) available non-operational low power (or sleep) states, (b) link width modulation, and (c) link speed modulation. Although shutting down inactive links at coarse time scales is well known, our emphasis is primarily on finer-time granularities and can be used in addition to coarse grain methods. The paper explores considerations in designing such algorithms and provides some specific algorithms and their parameterization. We show that the link width modulation is the most attractive alternative when available, but other controls are also useful. The techniques explored can be applied to a wide variety of links in modern systems.**

Keywords: Idle power control, low power states, exponential smoothing, width modulation, performance modeling

I. Introduction

Recently, the focus in computer systems has shifted from purely performance to good performance at lowest possible power consumption. This has resulted in intensive efforts to reduce both the idle and active power consumption of all major subsystems including servers, networking infrastructure and storage. Although traditionally the highest power consumption is associated with the compute and storage subsystems, increasing connectivity, a proliferation of networking devices, and a move to ever higher operational speeds means an increasing fraction of the power consumption due to the networking infrastructure.

Although the proliferation of networking and its increasing power consumption are quite obvious at the WAN and LAN level, the same is true in many other contexts as well, particularly in data centers. For example, the increasing speed of storage devices (including the emerging solid-state storage technology) require faster storage area networks (SAN) and faster mainstream networks to support distributed storage. Even the interconnects inside a server or on server backplane have mostly switched to the message passing paradigm and those links are becoming increasingly power hungry. For example, PCI-Express (PCI-E) links continue to get "fatter" (i.e., more lanes) and progressing from 2.0Gb/s per lane (Gen1) to 4.0GB/s (Gen2) to 10 Gb/s (Gen3). Numerous other platform interconnects such as those between CPU and memory and between processing cores of a CPU have followed a similar path. The extremely high speeds coupled with shrinking feature sizes lead to enormous power densities for low level links, thereby making power management crucial for them.

A networking device such as a router or switch has 3 planes: (a) data plane for forwarding packets which includes the individual link interfaces (or ports) and the forwarding backplane, all typically implemented in hardware, (b) control plane for specialized processing of packets and forwarding table maintenance, and (c) management plane for configuration, diagnostics, etc. In this paper, our concern is primarily with the port power consumption. Significant additional power savings are possible if the entire switch can be idled by diverting traffic along alternate paths. It is also possible to shut-down individual ports and redirect traffic over remaining ports. Such techniques obviously must work over rather coarse time granularities, and while useful, they are well studied in the literature[1] and not the subject of this paper. Instead, we focus on techniques that operate on finer time grain and essentially exploit the idle periods (or "gaps") in the traffic on each port.

To put the communication link power consumption in perspective, let us consider Ethernet as an example. While the power consumption of a 1 Gb/s Ethernet is only 1-2 watts, the power consumption of 10 Gb/s Ethernet can easily exceed 10 watts. In the next several years, Ethernet speeds are expected to scale up to 40 or 100 Gb/s, thereby making the power consumption of Ethernet interface (and associated switching infrastructure) very substantial. At the same time, the *average* link utilization – already quite low – is only expected to go down as the link speeds increase. This coupled with the fact that the link power consumption is relatively insensitive to traffic intensity implies that without aggressive power control links, most links will burn near 100% of their rated power while doing very little.

Although Ethernet remains the dominant networking fabric in general, it is by no means the only one to focus upon. Infiniband has made substantial strides in the data center and is already available at speeds of 48 Gb/s [7]. Similarly, PCI-Express has become a universal inside-the-box interconnect and is moving towards GEN-3. Fibre channel is the de facto storage interconnect and already available at 20 Gb/sec. Of course, there are other niche high speed fabrics such as Myrinet, QsNet that are also deployed in some HPC environments.

Communication links can support three ways of doing power control: (1) one or more (non-operational) low power or sleep states, (2) multiple "lanes" or parallel bits which can

978-1-4244-8953-4/11/$26.00

be individually put into a low-power mode, and (3) multiple operational speeds. However, depending on the link type, only some of these options may be available or practical. In this paper, we consider design of an integrated power control algorithm that can exploit one or more of these aspects for minimizing link power consumption. We shall show that different power management mechanisms have differing characteristics, and need to be combined carefully into a single integrated algorithm. Furthermore, since any power management scheme will increase link transfer latencies and hence affect QoS properties, it is necessary to examine power-latency tradeoffs as well.

Section II discusses the related work. Section III of the paper introduces essential characteristics of power control algorithms and performance metrics. Section IV discusses design of power state control algorithms and proposes some new algorithms. Section V discusses design issues for link width control and design of a specific algorithm. Section VI does the same for link speed control. Section VII then discusses issues concerning the combination of multiple controls. Section VIII discusses the evaluation methodology and shows detailed performance comparison of various algorithms. Finally section IX concludes the paper and discusses future work.

II. Related Work

Power management has been an integral part of wireless communications and has been studied thoroughly; however, the issues for the wired platform links are quite different. In particular, effective but simple HW implementable power control algorithms are a crucial issue. To our knowledge, a thorough treatment of link power control algorithms is new, although many of the underlying ideas have been explored elsewhere.

References [5] and [4] make the case for shutting down idle links and switch/router ports in order to save energy until certain number of packets have accumulated. Dynamic voltage frequency scaling (DVFS) – a popular scheme for CPU power state control – has been explored for links [16] and can be exploited in designing PHYs that switch between multiple modes. Simultaneous DVFS control for both processors and links is considered in [13]. Reference [14] proposes redesigning OS to increase the burstiness of the workload and thereby elongate low power periods. References [15] provides a survey of techniques for energy-efficient on-chip communication. Reference [1] describes a shadow-port mechanism for coarse-grain power management of switches.

III. Power Control Basics

In this section, we briefly discuss some basics of the 3 types of link power management mechanisms mentioned above.

A. Link Power State Control

The most important aspect in terms of idle power management is the existence of non-operational, low-power (or sleep) states that can be entered whenever the resource in question is idle. In case of communication links, various states are referred to as Lx states, and there are generally three of them:

1) L0: This is the normal operational state with highest power consumption, say, P_{L0}. For many links, P_{L0} has little dependence on the utilization level, i.e., an idle link dissipates almost as much power as one bursting data at full rate.
2) L0s: This is a sleep state with entry and exit times typically in 10's to 100's ns range but lower power P_{L0s} than P_{L0}. L0s control applies independently to both sides of a bidirectional link.
3) L1: This a much higher latency non-operational state with exit latencies in several μs range (entry latencies may still be of the same order as for L0s) but generally with a very low power consumption (e.g., 10% of P_{L0}). L1 requires a handshake between the two directions of a bi-directional link. If either side refuses to go into L1, L1 will not be entered.

For simplicity in exposition, we shall develop our algorithms assuming a single bi-directional low power state, namely L0s. Section IV-C considers extensions of power control to the case where multiple sleep states exist.

The PCI-E specification explicitly includes the definition of L0s and L1 states for each lane. With PHY unification, L0s and L1 states (and the link width control discussed below) should be feasible even for other link types; however, the current availability is spotty at best. For example, IEEE is currently engaged in an effort called *Energy Efficient Ethernet* or EEE that has examined link state control [6]. However, the currently defined sleep state corresponds to L1 (bidirectional control) and the focus is more towards slow changes so that the interfaces can be placed in sleep state during long-idle periods. Consequently, the issue of minimizing exit latencies has not been addressed adequately at this point. Power management is being pursued more aggressively in the links internal to a system, such as CPU-memory interconnects.

B. Link Width Control

In the past, most multi-conductor interconnects (e.g., twisted pair Ethernet, PCI bus, processor-memory bus, etc.) used a simple parallel interface. Unfortunately, with the increase in link speeds, the parallel interface became untenable because of a host of problems including excessive cross-talk between bit lines, high capacitance, skew between bit timings, reflections, etc. As a result, links started a slow transition to so called *bit-serial* technologies using differential signaling where the signal and its complement are sent along a pair of wires and the receiver detects their difference. This technology improves signal quality substantially and *nearly all of the multi-Gigabit links have transitioned to this technology.* In fact, many phys are already designed to be "re-purposable", so that a link port can be configured to act as a PCI-E, Ethernet, or Fiber-Channel depending on the needs.

With serial technology, link BW scaling is done by running multiple "lanes". The 10 Gb/s Ethernet copper implementations (CX-4 and KX-4) need to use four lanes when using Gen1 technology (running at 2.5Gb/s per lane). Although Gen3 technology can provide a single 10 Gb/s lane, copper implementations of that are very difficult. Looking forward,

40 and 100 Gb/s Ethernet versions have little choice but to go with more lanes.

Serial links admit dynamic width changes by putting certain lanes in sleep mode. (Note that this is not possible in a traditional parallel link.) The main attraction of link width control is that at moderately low traffic rates when one or more lane is operational, the sleep state entry/exit latency will have less of an impact for width control vs. power state control.

The motivations for multi-lane links are somewhat similar to those driving us to multi-core CPUs. So, the width-control algorithms are relevant to multi-core CPUs as well (in terms of so called C-state controls); although the details are different and not addressed here.

C. Link Speed Control

When a resource is lightly utilized, it may be possible to run it at a lower clock rate w/o adversely affecting the performance. The lower power consumption comes not only from the lower clock rate per se but more significantly from the lower operating voltage consistent with the clock rate. This follows from the fact that for the same level of reliability, a lower clock rate admits a lower operating voltage. This dynamic voltage frequency scaling (DVFS) has been the mainstay of CPU power control for some time, and has been explored for links as well [16], [13]. However, the algorithms considered in the past dealt with adapting link voltage/frequency to longer-term traffic trends, rather than short-term fluctuations. As with power state and link width control, our focus here is primarily with shorter term variations, which have somewhat different requirements.

Link speed changes impact both directions of a bidirectional link and require negotiation between the two ends. Auto-negotiation of speeds is already available in Ethernet and several other technology implementations; however, this feature is intended for configuration rather than dynamic changes. Thus, the phy speed selection is typically done only on boot-up. A HW based speed negotiation can eliminate driver delays but due to the need for handshake, it will be slow – at best of the order to L1 power state entry/exit latencies. Also, speed bumps may not be simply a matter of changing clock rate. For example, in case of Ethernet, the original 10 Mb/s version evolved through 100 Mb/s, 1 Gb/s and then 10 Gb/s with significant changes in terms of collision detection/avoidance and retry. Thus Ethernet speed change really amounts to PHY switching, which can be very slow. The Energy Efficient Ethernet (EEE) project has considered a rapid PHY switching (RPS) scheme that is more suited for dynamic speed changes, but it too is designed for dealing with low traffic over longish intervals (at least a few seconds) [2].

D. Power Control Effectiveness

In this section we briefly address the all important question what makes a power control algorithm desirable? Here we need to address both implementation and performance issues. For ease of reference, we shall henceforth speak of 3 power states: BSY (busy transmitting), IDLE (idle in normal power mode), and LPR (low power mode).

On the implementation side, it is critical that the algorithms be extremely simple, be easily implementable in HW and have a negligible power consumption. These attributes are forced by the already multi-Gb/sec and increasing link speeds. Furthermore, the implementation can only use the information that is already available to the interface (e.g., number of waiting requests), instead of something that needs to be fetched from elsewhere. These considerations rule out sophisticated estimation methods; in fact, even performing integer multiplication/division could be a problem.

In terms of performance, two basic metrics are *average efficiency* (F) and *average additional latency* (L_p) introduced by the power control. The efficiency F is defined as the fraction of idle period (or "gap") duration for which an algorithm is able to keep the resource in LPR state. We note that the power consumption during state transition is typically same as for IDLE state, therefore, the entry + exit period, henceforth denoted as η, is excluded from LPR duration. Efficiency directly translates into the power savings. That is, $\mathcal{P}'_{\rm idle}$, the average power with power control, can be related to idle and LPR mode power consumption (P_{IDLE} & P_{LPR}) by the trivial equation:

$$\mathcal{P}'_{\rm idle} = [1 - F(U_{\rm rsrc})]P_{IDLE} + F(U_{\rm rsrc})P_{LPR} \qquad (1)$$

The power control latency L_p can be anywhere between 0 and entry + exit cost η. The upper bound is realized when the traffic arrives just when the resource begins to enter the LPR state. Generally, there is no way to short-circuit the process – the resource must enter and then immediately exit LPR in this case.

Power management algorithms need to contend with the fact that increasing efficiency invariably results in increased latency. Thus, we need a metric that includes both power savings and latency. That is done easily, however, the impact of latency is very much application dependent. In some cases, the increase in latency or latency jitter may itself be undesirable, whereas in others, the relevant quantity is the performance impact of latency. For the latter case, an appropriate metric is performance per watt (PpW), where the performance can be defined as an appropriate throughput measure.

The relationship between latency and workload performance can be quite complex. For example, for a reliable transport protocol such as TCP, higher latency directly lowers the throughput. In case of internal links within a server, a higher latency could result in increased CPU stalls and hence lower application throughput. This impact is readily estimated by assuming that only some fixed fraction of the latency is visible to the CPU core. The net result is the following equation for the workload throughput λ in terms of data access latency L_a [8]:

$$\lambda = \frac{C\,U_{cpu}}{1 + \zeta L_a} \qquad (2)$$

where C and ζ are appropriate constants that depend on the workload and server parameters.

In view of varying impacts of latency on performance depending on the workload, we limit our subsequent discussion mostly to latency. So long as the algorithms provide a "knob"

to vary the latency, it is possible to choose the appropriate parameters for the situation of interest.

IV. Power State Control Algorithms

In this section, we consider various types of power state control algorithms and their parameterization. As already stated, effective implementations at multi Gb/sec links must necessarily be done using simple HW and local information.

A. Reactive vs. Proactive Control

A power state control algorithm has to make two crucial decisions: (a) when to go into a sleep state, and (b) when to exit it [15]. Because of the finite entry/exit time to/from sleep state, it is undesirable to go into sleep state for small gaps (or idle periods). Since the gap duration is not known in advance, the general technique is to monitor it for some period, henceforth called *runway*, and if the resource is still idle, start transition into sleep state. The mechanism to set runway is one aspect of designing power state algorithms.

A reactive exit refers to exit from LPR in response to arrival of a packet at the interface. In contrast, a *proactive exit* is self-triggered with the objective of completing the LPR→IDLE transition just in time to avoid additional latency penalty of exit and yet maximize the LPR duration. This requires an accurate prediction of the idle period (or gap). An under-prediction wastes power, whereas and over-prediction essentially turns the operation into reactive exit.

The accuracy of the prediction is a function of two attributes: (a) sophistication of the prediction algorithm, and (b) the inherent predictability of successive gaps. Given the requirement of very high speed operation, the prediction must necessarily be very simple. More important, however, is the fact that predictability is inherently constrained by the variability of gap sizes and correlation between successive gaps. In spite of these difficulties, a proactive approach can be useful at moderately low utilization levels.

Normally, it is assumed that each link direction is being controlled independently. It is however possible to coordinate the two sides and get somewhat better performance. For example, for most links a request going one way triggers a response coming down the other way. In this case, if one side of the link exits L0s state while the other side is in L0s, it is beneficial to start the exit of the other side as well. This is very simple to implement but a very helpful technique when exit latencies are large. It is possible to use a similar strategy across multiple links as well.

B. Power State Control Parameters

For a reactive algorithm, the only parameter to choose is the runway, and for proactive algorithms we need both runway and gap size estimation (since gap size determines proactive delay). These are discussed below.

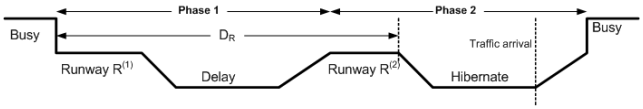

Fig 1. Illustration of ESA algorithm

1) Gap Size Estimation: The gap size estimation must necessarily depend on the history. If the successive gaps show certain deterministic patterns or trends, there is a rich set of statistical techniques to expose these patterns and techniques [3]. However, such patterns and trends typically occur at very coarse time scales (e.g., daily traffic patterns). At the level of gaps between successive bursts on a link, the behavior is generally quite random, although there may be significant correlations between successive gaps. In the interests of easy HW implementation, we propose to use a simple exponential smoothed averaging, which works almost as well as more complex auto-regressive methods. We call this genre of algorithms as ESA (exponential smoothing algorithm).

Let G_n denote the gap estimate at nth step, and g_n the current gap. Then with $0 < \alpha < 1$ as the exponential smoothing constant, we have:

$$G_n = (1 - \alpha)\, G_{n-1} + \alpha\, g_n \tag{3}$$

where α determines the rate of convergence towards the true average. The HW implementation of this estimator is completely trivial and can be done with only shifts, adds and subtracts if α is chosen as a negative power of 2. The avoidance of significant truncation errors does require that the calculations use a few more bits than for the desired final result. We found that 4 additional bits is adequate for this purpose.

One issue with a pure proactive algorithm is unnecessary exits when the traffic intensity is very low (i.e., long gaps between arrivals). Long gaps are particularly likely with heavy-tail inter-arrival time distributions or self-similar traffic that is often encountered in practice. To address this, we use a 2-phase operation shown in Fig. 1 where the first phase is proactive. After the proactive exit, the algorithm waits for second runway $R^{(2)}$ to see if any traffic will arrive. If not, it transitions to LPR for an eventual reactive exit.

Such an algorithm can be parameterized so that it behaves like a pure reactive algorithm at low utilizations where reactive is optimal, and becomes more and more proactive at higher utilizations. This is done by simply making the duration D_R in Fig. 1 (i.e., time from the beginning of the gap to go into LPR for reactive exit) as the control parameter instead of the runway R_2. With this change, if the *timer* D_R expires before a proactive exit has started, the resource will stay put in LPR state. However, for the purposes of this paper we want to keep the distinction between reactive and proactive algorithms, and thus do not show results for this hybrid algorithm.

2) Runway Estimation: In the simplest case, the runway can be constant – set large enough to avoid entry to LPR – on the average – above a certain resource utilization. However, we can do better. The idea is to shrink the runway when gaps are large and increase it when they are small. This way, the

algorithm will become more aggressive as the gaps increase and thereby increase the efficiency.

A direct implementation of this idea is to make the runway proportional to the current resource utilization U_n, and we call this as ESA-U algorithm. Let B_n denote the exponentially smoothed estimate of the nth *busy period*. B_n is estimated using a equation similar to that for G_n, i.e.,

$$B_n = (1-\alpha)B_{n-1} + \alpha\, b_n \tag{4}$$

Then, the current link utilization U_n can be estimated as

$$U_n = B_n/(B_n + G_n) \tag{5}$$

The main problem with this estimation is the need for a division. However, a precise calculation of U_n is unnecessary and rounding the denominator (B_n+G_n) to the next power of 2 turns out to be quite adequate. In fact, we consider an even simpler version that does a bang-bang control of runway, i.e., normally the runway is kept at some minimum value $R_{\min}$, and if the estimated proactive LPR time becomes too small, it is increased to some maximum value. We call this version as ESA-S (simple ESA) and it varies the first runway $R^{(1)}$ as

$$R_n^{(1)} = \begin{cases} R_{\min} & \text{if } G_n > (R_{\min} + \eta_0) \\ R_{\min}(1 + \text{Limit}/2) & \text{otherwise} \end{cases} \tag{6}$$

where Limit (a power of 2) is a control parameter. Choosing $R_{\min}$ as 1/2 of entry + exit cost (η) provides a good balance between latency and efficiency at moderate utilizations. As for the second runway $R_n^{(2)}$, it suffices to choose it large all the time, i.e., $R_n^{(2)} = \eta\,\text{Limit}$.

C. Multiple Sleep States

In this section, we briefly discuss a number of issues concerning extension of the above algorithms to deal with more than one LPR state. For generality, we allow for possibilities beyond the traditional L0s and L1 link states; i.e., the existence of more than 2 sleep states of which the first few are bi-directional and others are unidirectional. More than 2-sleep states are quite common and may be found in CPUs (e.g., C1, C3, C6, C7) and memory systems (e.g., slow-CKE, fast-CKE, self-refresh, etc.). We henceforth denote the extry + exit delay for kth sleep state as η_k (where $k = 1$ is the shallowest sleep state).

A simple and popular mechanism for transition among multiple sleep states is via time based *promotion* and *demotion*. In case of links, this means that the link initially always enters L0s according to the algorithm discussed here, and if no traffic arrives for certain amount of time, it is *promoted* to L1. The purpose of promotion is to save more power over very long idle periods. On the other end, if it is possible to predict that traffic arrival is imminent, it may help to *demote* a link from L1 to L0s so as to minimize exit latencies when the traffic does arrive. Demotion is rarely used in practice because of the difficulties of accurate prediction. Timed promotion can be extended to more than 2 states in an obvious way, i.e., via a systematic walk-through to deeper and deeper sleep states with increasing promotion time. Since an unnecessary promotion from state $k-1$ to k could cost up to η_k time in latency, the promotion time is often chosen as some large enough multiple (e.g., 10x) of η_k.

A somewhat different approach is to enter a deeper sleep state directly based on the recent history. In particular, if the estimated gap size G_n is at least $m\eta_k$ (but less than $m\eta_{k+1}$), we enter the kth sleep state. Here m is an integer parameter that can be chosen to achieve the desired latency-power tradeoff. Typically, the appropriate values of m are in low single digits. Notice that if the sleep state selected this way is not the deepest possible state, we still need to allow for timed promotion. Thus, the up-front selection is merely a refinement of time-promotion, rather than an alternative.

There are two potential advantages for considering up-front selection: (a) at lower utilizations, the deeper sleep state will be entererd directly thus giving more power savings, and (b) change between sleep states often requires waking up to active state, and hence a switchover from shallower to deeper state itself wastes energy. On the down side, an incorrect decision to go into a deeper state can introduce a lot of latency; therefore, an up-front selection may not work well for uncorrelated traffic streams (i.e., when the autocorrelation coefficient is small). In short, whether up-front selection is advantageous or not depends on many factors and needs to be evaluated for each situation.

V. Link Width Control

Link width control was presented in [10], however, we include its description here since many of the ideas are relevant both for designing speed control algorithm and also for contrasting the two. However, our description will be somewhat abbreviated.

Any link width control algorithm involves 4 decisions: (a) when to increase link width, (b) when to decrease it, (c) how much to increase, and (d) how much to decrease. Although the width increase/decrease can be done in arbitrary units, a few discrete and well defined steps not only simplify the implementation but also turn out to be quite adequate. For example, for a 10-lane (or x10) link, the possible widths are x1 through x10, but in reality the supported widths may be only x10, x4, x2 and x1. For convenience in description, we assume that successive steps are represented by multiplicative factors between adjacent widths. (In an actual implementation, additive factors may be more convenient.) We denote the decrease and increase factors by the vectors δ and Δ respectively. For example, for the set [x1, x2, x4, x10], we have $\Delta = \{2.0, 2.0, 2.5, 1.0\}$ and $\delta = \{1.0, 0.5, 0.5, 0.4\}$ where the 1.0 entry really means that no change is possible. Obviously, $\Delta_i = 1/\delta_{i+1}$. Often, it is convenient to talk about a delta value for a given width W, we shall denote this as $\delta(W)$ and $\Delta(W)$ respectively.

There are two important parameters for deciding link width change. The first is the link utilization U_n in eqn 5 and the second is the current *backlog*, which may be measured in packets or bytes, as appropriate. We shall call both as "queue length" (denoted Q_n) with appropriate interpretation being implementation dependent.

Let W_n denote the *fractional* width of the link at nth step, i.e., link width divided by the maximum width. Let

W_{min} denote the fractional minimum width. Note that $W_{\min}$ excludes the case where the entire link is shut down; therefore, it is a non-zero fraction. Then the condition for width decrease is given by:

$$W_n > W_{min} \ \& \ G_n > \gamma_1 B_n \tag{7}$$

where γ_1 is an integer constant. The first part of the condition is obvious and the second part requires the current link utilization to be sufficiently small, i.e., $U_n < 1/(1+\gamma_1)$. Notice that eqn(7) advocates a single utilization threshold for decreasing the link width even though link width reductions may occur multiple times, e.g., from x4 to x2 to x1. This is not anomalous because a width reduction will increase the utilization proportionately. For example, a move from x4 to x2 doubles the utilization, and further downward movement to x1 will happen only when the utilization once again dips below the *same* threshold.

For link width increase, we need an emergency provision – if the queue length crosses a high (Q_{HT}), an immediate increase is started. Otherwise, the increase is triggered if (a) queue length exceeds a low threshold (Q_{LT})], and (b) utilization increases beyond the point where decrease was done. The reason for using Q_{LT} is to avoid unnecessary changes due to errors in utilization estimate.

$$W_n < 1 \ \& \ \left\{ Q > Q_{HT} W_n \ || \ Q > Q_{LT} W_n \ \& \ G_n < \gamma_2 B_n \right\} \tag{8}$$

where γ_2 is another integer constant. Note that the equation does not use the queue length thresholds Q_{HT} and Q_{LT} directly; instead it lowers them by the current fractional width so as to avoid the accumulation of too many packets for a slow link. The net effect of this scaling is to maintain the same latency threshold as the link width goes down (e.g., 1/2 width link means double the packet service time but then halving the queue length threshold means no net impact).

Now, let's address the question of by *how much* should the width of the link be changed each time. In general, a change could be either gradual (i.e., move to next higher or lower width), or drastic (move to maximum or minimum width). Overall, this provides 4 different possibilities for the algorithm. We found from experiments that drastic changes do not really provide any advantage over gradual changes – when examined over a wide variety of workload characteristics. Therefore, we shall stick with gradual changes only.

With gradual change, the parameters γ_2 and γ_1 can be related as follows: Suppose that we are right at the point where $G_n/B_n = \gamma_1$. Now, if the prevailing utilization U_n dips ever so slightly, we decrease the width by the factor of $\delta(W)$ (i.e., the δ value at the current width W) thereby bumping up the utilization by $1/\delta(W)$. Thus if there is no further change, the ratio G_n/B_n – after both G_n and B_n have adjusted to the change – no longer equals γ_1, but something else, say γ_1'. It is easy to see:

$$\gamma_1' = \frac{1 - U/\delta(W)}{U/\delta(W)}, \text{ where } U = \frac{1}{1+\gamma_1} \tag{9}$$

Simplifying, we get $\gamma_1' = \delta(W)(1+\gamma_1) - 1$. Now, we can choose $\gamma_2 = h\gamma_1'$ where $h < 1$ is the hysteresis parameter that ensures that the link will not flip-flop.

As an example, suppose that $\gamma_1 = 4$ and $\delta(W) = 0.5$. Thus when the link utilization hits $1/(1+\gamma_1) = 20\%$ on its way down, we halve its width. With no material change in traffic, the link utilization will now increase to 40%. In this case, $\gamma_1' = 1.5$ and with $h = 0.5$, the link width will increase only when $G_n > 0.75B_n$, or $U_n > 1/(1+0.75) = 57\%$. If the width does increase, the resulting utilization would be 28.5% and the utilization would again have to go down to 20% for a downshift to occur. Thus no link flip-flop can occur. Although this safety margin may seem somewhat low, the queue length part of the condition further protects against flip-flops.

Note that a width decrease can happen almost immediately after the decrease decision has been made (since the multiplexer reconfiguration takes only a few ns), but the actual increase happens only after the sleeping lanes have exited the low power state. This delay could allow for multiple width changes in progress, unless we disallow them. For example, before an actual increase has taken place, the conditions could trigger yet another width increase or even a decrease. In most situations, there is little to be gained by allowing multiple concurrent changes, but the hardware becomes lot more complicated. Therefore, we mandate a single change at a time.

The multiplication/division implicit in the equations above can be avoided by judicious choice of parameters such as γ_1, h, $Q_{HT}/W_{\max}$ and $Q_{LT}/W_{\max}$. Also, the implementation may be further specialized to the available widths, rather than kept general and have to deal with vectors such as δ.

VI. Link Speed Control

Link speed change can be treated similar to link width change since a half-speed link is equivalent to a half-width link in performance terms. However, there are many differences between the two and thus imply quite different approaches for them:

1) Because of the bidirectional nature of link speed change it is necessary to consider traffic in both directions of the link in deciding speed changes.
2) The latencies involved in switching over to a different speed are significantly higher than for width change because of the need for two-way handshake. In some cases, such as Ethernet, a speed change really involves a change in phy, rather than simply a clock speed change.
3) Unlike link width change, no transmission is possible on any of the lanes until the rather slow process of link speed change has completed. In this sense, link speed change behaves more like power state change.

The required bi-directional agreement can be done in a routine manner: each direction first decides on the speed change locally and sends its "proposal" to the other side. The other side then acknowledges with a speed that is the maximum of its local decision and the remote proposition. The use of maximum function avoids unnecessary constriction of bandwidth and is stable with respect to concurrently generated proposals from both sides of the link.

We are now ready to state conditions for speed changes, assuming for now, that link speed change is the only control

in effect. Let us define a suitable monitoring interval M and estimate the link utilization U_n over such an interval. As usual, here the subscript n refers to the estimate in the n interval. We estimate U_n by counting the number of bytes transmitted and dividing that by the maximum byte count. That is,

$$U_n = B_n/(M \times f) \tag{10}$$

where B_n is the number of bytes transmitted over the interval M and f is the link BW in bytes/s. In order to avoid explicit division, the equations that follow can be expressed in terms of the byte count B_n, but we shall use the utilization measure for greater clarity. The monitoring interval M should be large compared with the link speed change latency – at least 10 times larger – in order to ensure that monitored utilization is not overly impacted by link speed change interval (during which no traffic is served).

Let S_n and S_{min}, denote, respectively the current and minimum link speeds (actual, not fractional). Let U_{min} and U_{max} denote, respectively, the utilization thresholds for effecting a speed decrease and increase. The spacing between U_{min} and U_{max} provides the necessary hysteresis. Then the speed change conditions can be modeled after width change equations (7) and (8). However, the long latencies associated with link speed changes require some changes. In particular, the speed decrease condition can be simplified to:

$$S_n > S_{min} \;\&\; U_n < U_{min} \;\&\; \tau_{chg} \geq M \tag{11}$$

where we have taken out the low threshold on queue length (Q_{LT}) since it is unnecessary given the long monitoring period. Instead, to ensure that we don't have multiple link speed changes during a monitoring interval, we have added an additional condition which says that the time since last change – denoted τ_{chg} – must be at least M. Now for the increase, we have

$$S_n < S_{max} \;\&\; \tau_{chg} \geq M \;\&\; \{Q > Q_{HT} \;||\; U_n > U_{max}\} \tag{12}$$

The high threshold Q_{HT} is left in, but without any scaling based on current speed. This is again a result of slow change – a reduced Q_{HT} at lower speed will do nothing to reduce the latency of power management; in fact, it would only lead to unnecessary speed increases and thus actually *raise* the latency. In fact, Q_{HT} should be chosen sufficiently high and is truly intended to be a "relief valve" instead of participating actively in speed changes.

It is clear that the complexity and power consumption of HW implementation of the above algorithm is similar to that for the width change algorithm, and thus will stay quite small.

VII. Multi-State Link Power Control

We now consider development of an integrated algorithm that can take advantage of all 3 modes of power management: sleep states, width controls and speed controls. As stated earlier, not every link provides all three mechanisms, and in some cases certain mechanisms may not even be practical (e.g., speed change doable only via BIOS). However, we would like to design an algorithm that can automatically take advantage of the available modes and either select among them or combine them in useful ways. We call such an algorithm as multi-state link power control (MLPC) regime. Since we have already described the low level details of each type of control, the purpose of MLPC is merely to put them together and address any issues that arise from combining multiple mechanisms.

A. Combining Power State and Link Width Controls

Since the availability of low-power (or sleep) states is essential for link width control as well, the latter can be seen as a refinement of power state control in that it allows individual lanes to be placed in sleep mode. However, there are two situations that call for link power state control to be combined with link width control:

(a) Putting all lanes in sleep state. If only one lane is currently active, and the traffic simply stops for a rather long period, it is desirable to place the last remaining lane also in the sleep state. This situation is not just an extreme case of link width control, since putting all lanes into sleep state implies that much of the link interface infrastructure can be shut-down. That is, entire link being in a sleep state is different from each lane individually being in the same sleep state.

(b) Inadequate granularity for link width control. If the feasible link widths are very few, width control does not provide adequate granularity. For example, the width control algorithm may oscillate the link width between 8 and 16 lanes in order to carry traffic that on the average requires 12 lanes. In such cases, it is desirable to change link width more slowly and allow power state control of the set of currently active lanes.

We consider (a) as an integral part of the width control algorithm. Part (b) is a part of the combined algorithm but could remain mostly unengaged (by choosing a large runway) on links with sufficient width granularity. Nevertheless, we design MLPC for the general case.

Simultaneous width and power state control could lead to some race conditions that must be avoided. Towards this end, we mandate a specific order for link width change and power state change. When a "gap" begins, we first check conditions to see if we need to reduce the link width, and if so, start LPR (low-power state) entry on the desired lanes. Following this, we consider power state control of the lanes that are still up (e.g., start their runway timer), however, with a larger timer value than for pure power state management (i.e., $R_{\min} = 2.0\eta$).

The exit from LPR happens either within a gap (proactive exit) or at the end of a gap (i.e., reactive exit). In contrast, link width increase condition needs to be tested at the end of each packet transmission.[1] Note that if only one lane is currently active, its LPR entry automatically triggers a sort of "promotion" so that the entire link interface can go into the low power state. In addition, following the normal promotion mechanism, a very idle link may advance from L0s to L1.

[1]For a dedicated HW solution, there is no overhead issue with this; in fact, a less frequent checking becomes more complex!

B. Combined Link Speed and Power State Changes

When both link speed and power state management are available, there may be a question of whether to use only one of them or combine them in some way. It is important to note in this regard that the link speed change is not useful if the lower speeds do not accompany lower voltages. This follows from the fact that the power savings from the lower speed are cancelled out by the longer packet transmit times. However, if the lower link speed can also provide a significant reduction in the operating voltage, it may be preferable to choose speed reduction over power state management. This is shown in a somewhat different context in [11]. Furthermore, if the link speed change is rather slow – as is normally the case – speed control at coarser granularity along with width or power state control at finer granularity may be an attractive approach. We henceforth consider only this case. Speed change requires that the link finish any current transmission and the entire link is not in sleep mode, since it requires handshake. However, it is possible that during the change some lanes are in sleep mode.

The link speed control algorithm described in section VI required the estimate of current utilization U_n. If we were to measure this utilization as the fraction of total time (including time spent in low power mode) is link is busy, the measured value will be affected by the link width control that may be in effect, but not by the power state management. For proper functioning, we do not want the either form of power management to affect the estimated utilization. The measure suggested in eqn(10) automatically discounts for any fine grain power measurement, assuming that the bandwidth f is the full link BW with all lanes operating. Thus, the equations in section VI remain valid.

VIII. Power Control Performance

For a comprehensive evaluation of various power control algorithms, we designed a detailed simulation model representing two platforms connected via a communication link. Each of these systems has an abstract model of CPU with HW threads, memory channels and memory ranks [12]. The model – originally designed for more detailed platform performance studies – does represent the impact of latencies on CPU stalls; however, this aspect is not important for the results shown here. The model could be driven both using analytically generated traffic as well as actual link traces.

Given that we are dealing with 3 different basic power management algorithms, each with its own control parameters, the number of combinations to consider for performance evaluation is too large to accommodate in this paper. Therefore, we only give a flavor of the performance of various algorithms and their combinations. Consequently, we shall mostly show results with suitably chosen parameter values rather than a sweep across the parameters.

We consider a 10-lane 40 Gb/s (bi-directional) link driven by continuous non-overlapping requests for data (request size: 64 B, response size: 400 B) from both systems. This results in minimum inter-request time (IRT) of 80 ns and maximum link utilization of 58%. We also consider successively lower utilization levels by a factor of 2 up to 8 octaves (i.e., maximum IRT of 10.0 us).

The results shown here use a renewal process for request arrivals with IRT distribution represented as Zipf with complementary distribution function as $P(N > n) \propto n^{1.25}$ (with an underlying "slot-size" as the basis for discretization used by the simulator). The reason for using Zipf distribution with a heavy tail is to consider situations with high traffic variability that is often seen in real traffic. Also, we chose a renewal process (instead of one with significant autocorrelation), in order to keep the power savings results conservative. We note that real-traffic often shows significant correlations (self-similar traffic being the extreme case) [9]. Experiments with self-similar traffic (not shown) do confirm greater power savings for all algorithms.

We start with a simple comparison of three basic power state management algorithms mentioned in section IV: (a) Basic reactive control with fixed runway (B-REA), (b) Simple ESA (ESA-S), and (c) Precise utilization estimation based ESA (ESA-U). For ESA-S and ESA-U, we choose $R_{\min} = 0.5$ and $\mathrm{Limit} = 4.0$, and for B-REA, a runway size that gives B-REA performance similar to that for ESA. The exit latency from L0s state is assumed to be 1us and entry latency is 16 ns. The power consumption per lane is representative of current serial link technologies, although the precise numbers are undoubtedly implementation specific and could vary significantly.

Figs 2 and 3 show respectively, the total link latency and power consumption as a function of IRT. The general trends of these graphs (and others shown later) are as expected: as IRT increases, power management is engaged more frequently and hence the power consumption goes down, but at the expense of increasing latency. An exception to this occurs in the initial portion of the curves, where the utilization is high enough that the latency is primarily governed by the queuing delays since the power management algorithms do not really get engaged. This explains the initial latency dip. Another general observation is that an algorithm that yields more power savings, generally does so at the expense of higher latency. This is not an deficiency of the algorithm, but rather an essential feature since short of a perfect predictability, a more aggressive use of low power mode would cause more delays to the packets.

In terms of more detailed results, it is seen that all 3 algorithms perform well. Proactive algorithms show two desirable characteristics: (a) a marginally better behavior at moderate latencies, and (b) a less sharp increase in latency at moderate utilizations.

We now consider the latency and power consumption for the link width control, as shown in Figs. 4 and 5. For this, we chose $\gamma_1 = 7$, $Q_{LT} = 4$, and $Q_{HT} = 16$. In the figures, we show 3 cases: (a) Fixed width with "variable" power state (meaning power state control only), (b) Variable width and fixed power state, and (c) Variable width and variable power state. We use a consistent notation for these (and the speed control results to come), as in [F/V]S-[F/V]W-[F/V]P where F means fixed, V means variable, S is for speed, W for width, and P for power state.

It is seen that the width control provides substantially lower power consumption than power state control *while* providing a similar level of latency. (Of course, the precise latency-power

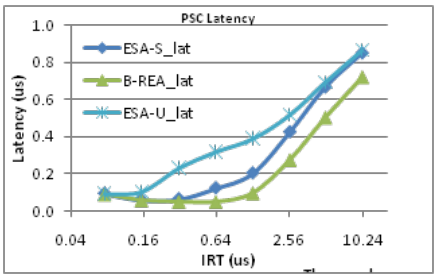

Fig 2. Latency of Power State Control

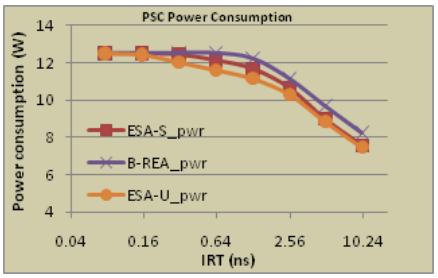

Fig 3. Power Consumption of Power State Control

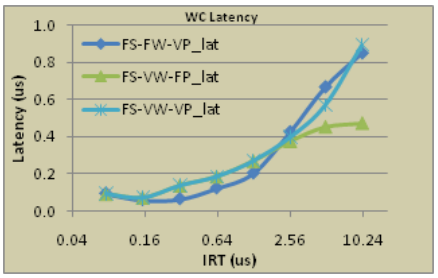

Fig 4. Latency of Link Width Control

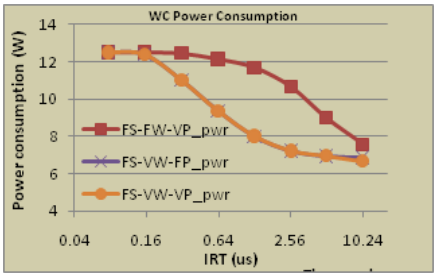

Fig 5. Power Consumption of Link Width Control

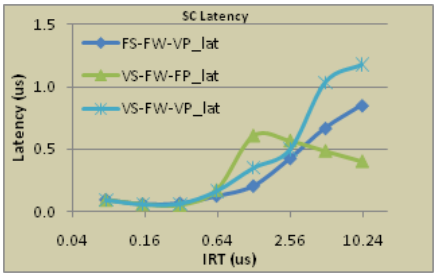

Fig 6. Latency of Link Speed Control

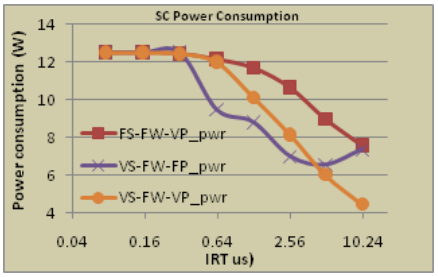

Fig 7. Power Consumption of Link Speed Control

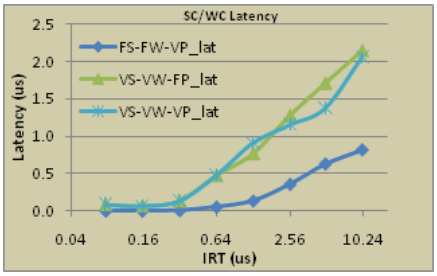

Fig 8. Latency of Link Speed & Width Control

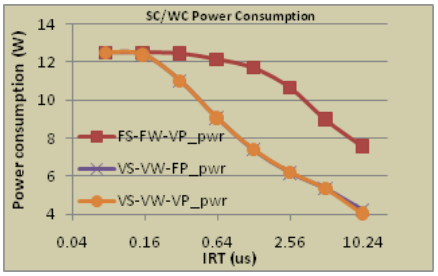

Fig 9. Power Consumption of Link Speed & Width Control

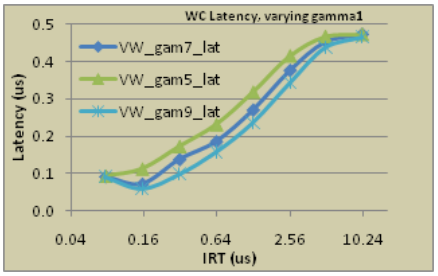

Fig 10. Latency of Link Width Control with different γ_1 values

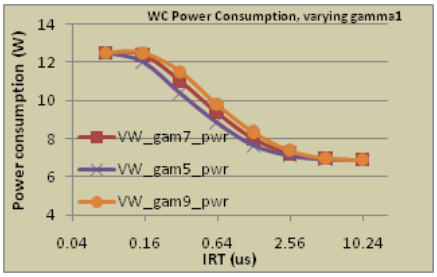

Fig 11. Power Consumption of Width Control with different γ_1 values

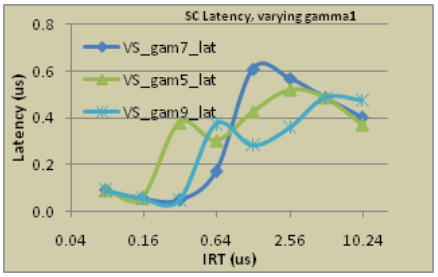

Fig 12. Latency of Link Speed Control with different γ_1 values

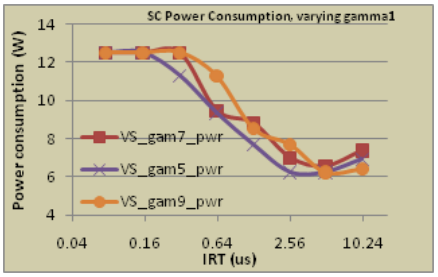

Fig 13. Power Consumption of Speed Control with different γ_1 values

tradeoff depends on the parameters chosen – it is possible to make the latency of width control lower while providing same level of power savings as the power state control.) Notice that the addition of power state control to width control only makes a difference at low utilizations and would provide slightly lower power at really low utilizations. The main reason why link width control works better than power state control is simply that a pure width control always keeps at least one lane up. Now, given the extremely fast link, even one lane can handle low utilization traffic with very little latency while at the same time saving power. In contrast, in case of power state control, any packet that arrives while the link is in low power mode, must suffer the rather large exit latency. It follows that link width control is highly desirable whenever available.

Figs 6 and 7 show the impact of link speed control. Real implementations of link speed control can be really slow: here we optimistically assume that the link speed change is quite fast – 20 us per change. This is still about 20-times slower

than the power state (and hence the link width control as well). The results shown here will be significantly worse if the speed change takes 10's of ms. As for other parameters, we choose $\gamma_1 = 7$, $Q_{HT} = 64$, and $M = 400$ us. Note that M has been chosen as 20 times that of link speed change time after some experimentation.

As in case of link width control, we compare 3 cases: (a) Fixed speed with variable power state, (b) Variable speed and fixed power state, and (c) Variable speed and variable power state. (These are denoted, respectively, as FS-FW-VP, VS-FW-FP, and VS-FW-VP.)

It is seen that the latency behavior of pure link speed change is quite complex and shows several phases. At moderate to high utilizations, link speed change does not take effect, and the latency decreases mildly with increasing IRT (or decreasing utilization). This is because of the decreasing queuing delays. At lower utilizations, the speed change does take effect which shoots up the latency because of repeated back and forth changes in speed. Finally, as the average utilization becomes very low, the latency decreases because the link simply stays in one state. At intermediate utilizations, latency behavior can be erratic due to very large latencies of speed change and a rather bursty traffic.

The addition of power state change moderates the above behavior and makes the latency closer to monotonic increase (except for initial dip due to decreasing queuing delays). In this sense, power state control can be considered a helpful companion of link speed control. This also follows from the fact that link speed operates at a much coarser granularity than power state management, and hence the combination is beneficial.

Let us now examine the power consumption shown in Fig. 7. In modeling the power, we assumed that the voltage level is also reduced consistent with the speed. This is what gives link speed control a definite advantage in terms of power consumption. At very low utilizations, link speed control still does not perform very well because of the need to keep the entire interface alive all the time. Instead, combining it with power state control improves even the power consumption behavior.

Finally, we consider what happens if link speed and width control are combined together by our MLPC algorithm. Figs 8 and 9 show the latency and power consumption for this case. Once again, we compare variable speed, variable width with and without power state control, and also against the baseline power state control only.

It is seen that just like the power state control, width control also moderates the peculiar behavior of the link speed control. In fact, the power consumption curve here is noticeably smoother than for the case of link speed with power state control. The overall power consumption – though not obvious in the graph – is the lowest in this case. Unfortunately, the additional watt or so of power consumption is achieved at the cost of a huge increase in latency. Consequently, combining link speed and width control may not be very desirable.

Although the space limitations do not allow a comprehensive set of results to be displayed, it is important to note that all algorithms discussed here can be "tuned" for the appropriate latency vs. power savings tradeoff. In particular, Figs 10 and 11 show the link width control characteristics as a function of the parameter γ_1. As expected, a larger γ_1 reduces latency but at the cost of lower power savings. Figs 12 and 13 do the same for link speed control. As discussed earlier, the behavior here is much more complex and is driven by the large transition latency and bursty traffic characteristics.

IX. Conclusions and Future Work

In this paper we studied multi-state power control of communication links. In particular, we examined algorithms for the control of link power state, link width and link speed. We also examined issues relating to the combined use of these controls. We showed that link width control is highly desirable and should be preferred whenever possible over pure link power state control. Link speed control is useful at a much coarser grain and needs to be used carefully.

In the future, we plan to examine coordinated power management of multiple links (and high level entities such as line cards and entire switches/routers) in the context of a data center.

Acknowledgements: The author would like to acknowledge the help of Alexander Jimbo on the design and implementation of various algorithms.

References

[1] G. Ananthanarayanan, R.H. Katz, "Greening the switch.", Proc. of 2008 conf. on Power Aware Computing & Systems (San Diego, California).

[2] F. Blanquicet & K. Christensen, "An Initial Performance Evaluation of Rapid PHY Selection (RPS) for Energy Efficient Ethernet", IEEE Conf on local computer networks, Oct. 2007, pp223-225

[3] P.J. Brockwell and R.A. Davis, *Introduction to Time Series and Forecasting*, Springer-Verlag, 1996.

[4] M. Gupta and S. Singh, " Dynamic Ethernet Link Shutdown for Power Conservation on Ethernet Links", Proc. of IEEE Intl Conf on Communications 2007, June 2007.

[5] M. Gupta, S. Grover and S. Singh, " A Feasibility Study for Power Management in LAN Switches", Proc of 12th IEEE ICNP, Oct 2004.

[6] IEEE task group 802.3.az, "Energy Efficienct Ethernet", www.ieee802.org/3/az/public/nov07/hays_1_1107.pdf.

[7] Infiniband Trade Association, "Infiniband Architecture Specification 1.2.1", Vols 1 & 2. Available at www.infinibandta.org/specs/

[8] K. Kant and Y. Won, "Server Capacity Planning for Web Traffic Workload", IEEE trans. on knowledge and data engineering, Oct 1999, pp 731-747.

[9] K. Kant and M. Venkatachalam, "Modeling traffic nonstationarity in e-commerce servers", Proc. of SPECTS 2002, San Deigo, CA, July 2002, pp 949-956.

[10] K. Kant, "Power Control of High Speed Network Interconnects in Data Centers", Proc. of 2009 INFOCOM High speed networks symposium

[11] K. Kant, "Distributed Energy Adaptive Computing", Proc. of International Conf. on Communications (ICC), May 2010.

[12] K. Kant, "LMPOWER – A Comprehensive Link-Memory Power Management Simulator", available at www.kkant.net.

[13] J. Luo, N. Jha, Li-S Peh, "Simultaneous dynamic voltage scaling of processors & communication links in real-time distributed embedded systems", IEEE Trans on VLSI, 15, 4, April 2007.

[14] A. Papathanasiou and M. Scott, "Energy Efficiency through Burstiness", Proc of the 5th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'03), pp. 44-53, Oct 2003.

[15] V. Raghunathan, M. B. Srivastava, and R. K. Gupta. A survey of techniques for energy efficient on-chip communication. In Proc. the 40th Conference on Design Automation, 2003.

[16] L. Shang, Li-S Peh, N. Jha, "Dynamic voltage scaling with links for power optimization of interconnection networks", Proc of HPCA 2003.