Sustainable Computing: Informatics and Systems 22 (2019) 1–12

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

Coordinated Power Management in Data Center Networks

Joyanta Biswas*, Madhurima Ray, Sanjeev Sondur, Amitangshu Pal, Krishna Kant

Computer and Information Sciences, Temple University, Philadelphia, PA 19122, United States

ARTICLE INFO

Article history:
Received 8 April 2018
Received in revised form 7 October 2018
Accepted 31 January 2019
Available online 8 February 2019

Keywords:
LPI(low power idle)
Network simulator NS3
Data center network
Energy management

ABSTRACT

In this paper, we present a coordinated power management technique for data center networks (DCN) which we have implemented in the popular network simulator NS3. The proposed mechanism includes three key entities: (a) a local controller (LC) that manages traffic at each DCN node (switches and routers), (b) a global controller (GC) that has a global view of the DCN and provides hints to the LC's based on this knowledge, and (c) a topology aware user request assignment controller (RAC) that controls placement of the external requests on the endpoint hosts based on hints from the GC. The main goal of the coordination from the energy management perspective is to properly direct and consolidate network traffic to maximize low power (or "sleep") opportunities for the network interfaces while avoiding link congestion. In the process of testing out our mechanisms, we have enhanced the popular NS3 package for network modeling. The key enhancements are in the accounting of energy consumption at various levels and enhanced traffic routing mechanisms. We show that these mechanisms can reduce the power consumption by up to ∼40% in the common fat-tree based DCNs using the low power idle (LPI) feature of the Ethernet. We also apply our mechanism to the Hypercube network and show that it too can save a significant amount of energy.

1. Introduction and motivation

With increasing size and energy footprint of data centers, their effective energy management is crucial. The major power consumers of the IT infrastructure in a data center are compute/memory subsystem, storage subsystem, and data center network (DCN) [1,2]. In this paper, our focus is the DCN, which consumes an increasing percentage of power as the network speeds and connectivity rise due to network upgrade. For example, the power consumption of a 10 Gb/sec Ethernet can be anywhere between 2–10 times the power consumption of 1 Gb/s Ethernet, depending on the number of ports and the technology used [3]. The power consumption goes up with the number of powered-up ports even when they are idle since the underlying link Phy is synchronous and constantly consumes power to keep transmit and receive sides in sync. Furthermore, the increasing speeds generally result in much lower network utilization, since the deployment of higher speed links is mostly motivated by the technological availability, latency considerations, and ability to handle highly bursty workloads, and much less by sustained high bandwidth demands. This is particularly true in HPC data centers with dense interconnects (e.g., hypercube, toroid) [4] and HPC workloads that may go through multiple steps of collective communication (high network traffic) and parallel computation (very little network traffic). Such situations make network energy management techniques even more important. Another type of data center infrastructure is multi-tenant colocation data center or a colo center. Such a center rents out servers and associated networks and storage infrastructure to multiple tenants. The colo operator is mainly responsible for power, cooling, security, hardware maintenance, etc. Therefore, it is crucial to improve both the energy efficiency and sustainability of such data centers [5].

Our underlying model of data center is one that serves *external* requests coming in through a user request assignment controller (RAC). Each such request lands on a server, and in turn could generate *internal* flows across various servers using the DCN. For example, an HTTP request may initiate some application and database traffic across the servers. Since our focus is on DCN, we avoid the issues of application deployments and the details of application level interactions. Instead, we only consider source and destination assignments for individual flows.

As in any energy management context, there are three potential ways in which energy consumption of DCNs can be reduced: (a) Shape the workload at the source by techniques like batching of requests or proper admission control policies, (b) Reduce the link speeds commensurate with the loading – a kind of dynamic volt-

* Corresponding author.
E-mail address: tug54519@temple.edu (J. Biswas).

https://doi.org/10.1016/j.suscom.2019.01.021
2210-5379/

age/frequency switching (DVFS) control, and (c) Assign the source, destination and network path for each request so that the traffic is largely concentrated on fewest links and thus maximizes opportunity for other links to sleep.

Workload shaping [6] is a well studied subject and we do not consider it here, although it could help in more effective use of sleep modes that we do explore here. Link level DVFS is also thoroughly explored, but it is practical only at a very large time-scale and coarse granularity level. This is because most networks provide only a few speeds (e.g., 40, 10 and 1 Gb/s) and the switching is essentially amounts to PHY switching and can be very slow. Although IEEE introduced the Rapid PHY Selection (RPS) mechanism to allow for dynamic speed changes, such a mechanism is still too slow and may cause packet drops and other problems [7]. Therefore our focus is primarily on (c) and we use the link sleep mechanisms defined at PHY and higher layers, as detailed in Section 4. In particular, we use the Ethernet specific sleep mechanism called Low Power Idle (LPI) in addition to the generic PHY level mechanisms.

The energy efficient Ethernet standard approved by IEEE in 2010, improves the Ethernet energy proportionality by defining a link sleep mode known as Low Power Idle (LPI). Although the standard defined the low-level mechanisms for switching to low power mode, the EEE standard does not define the strategy for deciding when to enter and leave the low-power mode. The right configuration of EEE is critical for maximum energy saving and low performance overhead [8] and best setting depends on the distribution of the traffic on the network. Currently all these parameters are set by the vendor at default values without any knowledge of the workload. This often results in LPI not working well and perhaps the cause the general perception that LPI does not work and should be turned off. Previous studies show that the energy savings depends on the traffic pattern and network load [9,10]. Some of the recent research has tried to understand the impact of EEE for different types of applications such as MapReduce [11], video streaming, scientific computing, etc. For example, Map-Reduce has a very specific traffic pattern during different phases that one could exploit for network energy management without introducing substantial latency. In our work the routing mechanism dynamically consolidates the traffic to enhance possibilities for using low power mode for as many links as possible with a loose coordination mechanism.

Flow path selection and consolidation to maximize sleep opportunities can be quite challenging in a large data center network because a workable scheme must simultaneously consider availability of endpoint (or server) resources (CPU, memory, etc.), local bandwidth demands at the switches, and end-to-end blocking/delay on the network paths. In a small data center, this can be accomplished via a central controller that has global network and endpoint visibility, however, such a scheme does not scale. In this paper instead, we design a low-overhead semi-distributed scheme that involves three cooperating mechanisms: (a) an energy aware distribution of requests to the endpoints by the user request assignment controller (RAC), (b) an intelligent assignment of a new flow at each switch by a local controller (LC), and (c) a lightweight global controller (GC) that monitors network traffic and provides hints to LC's and RAC for better placement of flows at endpoints.

The most prevalent network topology in data centers is "fat-tree", and this is likely to remain so in the foreseeable future. Therefore, we largely use this topology; however, our underlying mechanisms are general and can work for any regular network topology. In particular, we also study the hypercube topology, which is most relevant in the context of HPC data centers. However, good mechanisms to handle the energy savings vs. congestion avoidance trade-off do depend on the specifics of the topology. Also, while this paper considers Ethernet based interconnect, the precise MAC layer is unimportant; what matters is the available energy management capabilities. Thus our results can be applied to the highly popular Infiniband based data centers as well assuming similar sleep mechanisms.

The rest of the paper is organized as follows. Section 2 provides a broad overview of our scheme along with the key contributions. Section 3 reviews the related works on network energy management. Section 4 discusses power management mechanisms for a network interface including the low power idle (LPI) mechanism for Ethernet. Section 6 discusses the enhancement of the NS3 frameworks. Section 5 discusses various controllers, their interaction, and how they are implemented. Section 7 presents details on our simulation based evaluation of the mechanisms as well as the simulation results. Finally, section 8 concludes the discussion and lays out the future work.

2. Overview and contributions

An intelligent choice of flow paths through the DCN and their potential reorganization requires global visibility into the network that is not present in traditional networks. Although a software defined network (SDN) provides the same kind of visibility, still there is a matter of cost and other deployment issues associated with it. So our proposal evolves from the existing management layer, residing inside switches. We assume that management layer on each switch monitors the bandwidth usage on all the interfaces. Periodically this information is passed to a global network coordinator i.e. Global Controller (GC) that runs on some server. GC can use this information for flow placement and/or reshuffling; however, doing so will make our solution unscalable. Thus in our approach, GC plays only an advisory role and provides hints to the local controllers. Following the instructions, the local controller can undertake some measurements to alleviate network congestion in the event of an approaching congestion. Those measurements are for example – the probability of choosing a path, changing the priority of the outgoing links, etc. Those have been discussed in Sections 5, and 7. The whole intent of our work is to keep the GC both lightweight and non-critical – the LCs can continue to function uninterrupted but with degraded performance even if the GC fails and is restarted. There is a question of contradiction between the course of actions, taken by the LC and GC. But this not likely to occur since the LC only follows the recommendation provided by the GC, which is invoked infrequently. LC is only responsible for (1) sending the bandwidth information and (2) changing the local parameters. The above two actions are responsible for both energy efficient and congestion free communication.

Properly directing the incoming request to the appropriate server and choice of destination server for the flow is crucial for ensuring consolidation of the workload at the hosts without creating any overload. From the perspective of DCN, traffic consolidation is an important aspect of the energy management. But aggressive consolidation of requests at the host level may lead to congestion at the network links if the consolidation decision being made independently without considering the network capabilities. Likewise, greedily packing flows into the network links can also create network congestion. Thus, the network switches must ensure that consolidation on outgoing links does not lead to network congestion at any time. We accomplish the goal of energy management without creating congestion by having the GC provide hints both to the RAC and the LCs. By doing that GC ensures necessary adaptation of LCs and RAC to the perturbation in a lightweight and non-critical manner.

It is important to note that the energy management can be handled – simultaneously, if needed – at multiple time granularities. For example, built-in hardware mechanisms often operate transparently at fine time scales (e.g., C1-C6 CPU states), whereas sophisticated traffic consolidation schemes based on solving the

optimization problems and reconfiguration can operate at long time granularities (10's more minutes or more). Most techniques explored in the literature fall in the latter category [12]. In contrast, our primary interest is in medium grain mechanisms that can be implemented in software but must be relatively lightweight and non-disruptive.

We emphasize on the placement of the new incoming flows intelligently along the network to avoid rerouting of any ongoing flow. Since rerouting of an ongoing flow can cause variable packet delay, packet reordering, packet drops, etc. due to its critical hand-off process. Additionally, it can also increase the energy consumption under different circumstances. By extensive simulations, we show that our scheme can significantly *reduce the overall network power consumption* (*by up to* ∼40% *compared to the DCN without any power saving scheme*). *This is especially true during low traffic periods which become increasingly prevalent in DCNs as the link speeds increase*, *as discussed in Section* 1. *The power savings comes at the cost of increased latencies*, *but that increase becomes significant only at high network utilization*, *where it is not even clear whether we can save any energy by applying the aforementioned techniques*.

We have used the network simulator NS3 for the implementation of our techniques. While working with the simulators, we find out that NS3 currently has little to offer in terms of energy management. It merely defines some parameters on energy consumption. To remove this deficiency, we have implemented an energy model for NS3 based on currently available network device energy management features in both inside-the-box fabrics (e.g., PCI-E) and outside-the-box fabrics (e.g., Ethernet). We have enhanced different NS3 packages to capture the real time matrices like link utilization, idle time, sleep time, active time, etc. at the net-devices. Besides, we have enhanced the simple deterministic routing model provided by NS3 into a hint based probabilistic routing model. In this paper, we describe these implementations and demonstrate the test results for the two common data center interconnect networks, fat-tree and hyper-cube.

3. Related works

3.1. Low power idle

Mostowfi [13] studied EEE LPI operations that put the device into two states – deep sleep and fast wake. Thaenchaikun et al. [14] proposed energy saving model using a control plane that utilizes an energy aware routing protocol. They showed that a combined strategy for routing protocol and energy aware path augmenting solution can provide good energy savings. Nedevschi et al. [15] proposed a buffer and burst scheme that shapes traffic to alternate active and idle periods, thereby providing increased opportunities to sleep. The authors also proposed a scheme where rate of operation of network links is dynamically adapted to the arrival rate of packets. Abts, Dennis and Marty [16] discussed link adaptation to dynamically reduce the link speed for less energy consumption using a central controller. They do so use adaptive link rate (ALR), which is a predecessor to LPI mechanism for EEE that attempts to switch PHY(s) at run time (as opposed to simply the initialization time). It turns out that even with Rapid PHY Selection (RPS) it takes time to change the speed of the link and there is a huge overhead of changing the link speed.

3.2. Traffic consolidation

The better outcome through LPI is possible with minimization of the number of active links and switches, in certain cases. The works in [17,18] have tried to implement different intelligent mechanisms to achieve the goal of traffic consolidation. All the solutions consist of a centralized optimizer, which tracks the traffic statistics and redirects the traffic according to the optimization solution. For example, Wang [18] discusses the correlation of peak workloads among different flows while consolidating. In our work, centralized optimizer uses topology statistics for both the traffic merging and request aggregation in a coordinated manner to avoid the conflicts in agenda.

3.3. Endpoint/VM consolidation

To save power at the endpoints, VM consolidation has also been considered as an efficient method in literature. Most of them [19], [20] have considered mainly VM placement and migration as optimization problems to reduce the server energy consumption without any non-local network side considerations. Thus, the server energy savings may be accompanied with increased packet drop and jitter.

3.4. Request assignment controller

In [21], the authors present an energy aware request assignment controller that distributes the tasks on VMs based on their speed and energy efficiency. In their model the migration decisions are based on the vCPU units demanded by an application and the available capacity of the host and of the other servers in the cluster. Authors in [22] schedule the tasks on VMs that are normalized based on system and application level resource management. Their scheduling and migration of VMs is based on the vCPU units demanded by an application and the available capacity of the host and of the other servers in the cluster.

3.5. Coordinated energy management

A coordination between VM consolidator and network is essential to address the increased packet drop and jitter problem, as we have explored in this paper. In our work, we perform the endpoint consolidation by aggregating the external request to the target servers. There are several previous works on server-network coordination like [23–26]. The primary concern of the [25] and [26] is to place VMs close to each other, those having more mutual communication. That reduces the path length, in other word the delay and the bandwidth requirement at the higher layer of the data center. [23] has extended the work of [18], by considering VM utilization and correlation analysis for both the VMs and flows - in a coordinated way. In [24], the optimization problem of VM placement and flow routing has been represented as a unified optimization problem and tries to solve it. Doing all these approaches, both the probability of network congestion and energy consumption of the overall system (includes both end points and DCN) can be reduced significantly.

Since VM migration can be expensive both in terms of latency and network traffic [19,27], we do not depend mainly on VM migration in our work. Instead, our main goal is to provide hints to local controller and the Request Assignment Controller to work in sync. Besides that, we work on the minimal assumptions: the affinity of VMs or the correlation of loads between mutual flows and VMs are unknown in our case. In reality, these metrics would be very uncertain, so VM consolidation/migration and flow routing based on that statistics might not bring expected results.

3.6. Network energy model simulator

The existing work on energy modeling in NS3 is quite limited. Hu [28] presented the first framework for incorporating the energy model in NS3. Their study computes topology specific energy consumption via a framework shown in Fig. 1 that does not link the

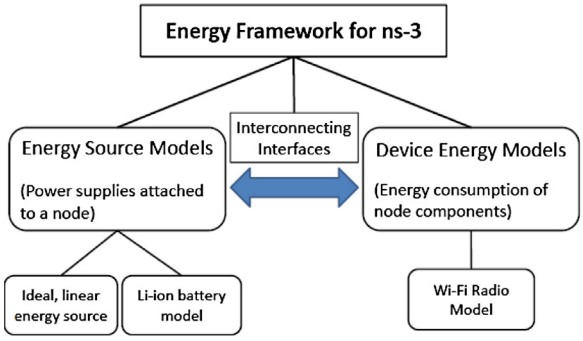

Fig. 1. Existing NS3 energy model.

network device to the energy model for capturing the work done by the device, i.e. the actual work of packet transmission. But per packet transmission energy is not realistic for synchronous links. Tapparello [29] used the same model from Hu to design an energy harvesting framework. Our model is built over these basic frameworks while enhancing the energy metrics captured and granularity of data collected. We believe that these granular data and enhanced energy metrics will be of interest for future research studies and aid in understanding the underlying mechanism of device energy consumption.

4. Ethernet power management

4.1. Operating states of the network links

Nearly all of the network links are currently based on the serial links that use differential signaling, because such a technology can easily handle noise and does not suffer from cross-talk and clock skew issues. The links are then built using one or more such serial interfaces called "lanes". This has led to the notion of repurposable Phy layer, i.e. the same basic Phy module that can support very different higher layer technologies including PCI-Express, Ethernet, Infiniband, Fiber Channel, etc. At a very low level, all the links possess three operating states:

L0: This is the normal operational state with highest power consumption, say, P_{L0}. P_{L0} does not have much dependence on the utilization since "filler" Idle messages are constantly exchanged whenever the link is idle even for very short periods. In other words, the link is always 100% busy.

L0s: This is a sleep state with entry and exit penalty typically in 10's to 100's ns range. In L0s, the power level P_{L0s} is around 40–50% of P_{L0} depending on the design [9]. There is usually a trade-off between transition latency and sleep power. L0s control applies independently to both sides of a bidirectional link. The link is kept "trained" during L0s and thus a part of the interface must stay awake.

L1: This is a much higher latency non-operational state with exit latencies in 10's of μs range, but generally with a very low power consumption (e.g., 10% of P_{L0}) [9]. This state requires a handshake between transmit and receive sides and link retraining upon exit from low power mode. If either side refuses to go into L1, L1 will not be entered. The training symbols are not exchanged (like L0s) during L1 and thus link retraining is required on wake up, which makes the exit latency quite high.

The L0s and L1 states are defined at PHY level but may not be exposed at higher level. In case of Ethernet, IEEE has defined an enhanced version of L1 called Low Power Idle (LPI) in the 802.3az-2010 [9] standard. Instead of transmitting continuous idle signals LPI sends periodic refresh signals to maintain the synchronization. Wake up from LPI involves a significant exit latency, since the transmitter needs to wake up the receiver before transmitting anything. There are also new emerging energy management standards for 40 and 100 Gb/s Ethernet links for example, 802.3bj [30] (deep sleep mode) and 802.3bm (shallow sleep with fast wakeup). The deep sleep mode is essentially same as LPI described here, but the shallow sleep allows for faster wakeup.

4.2. LPI models

In this paper we describe two types of LPI models: Transmitter Only Sleep (TOS) model and (ii) Transmitter Receiver Sleep (TRS) model as described below. Basically, TOS is based on the availability of hardware level L0s sleep state and TRS is based on handshaking based L1 state, as stated earlier.

4.2.1. TOS model

The TOS model is mainly driven by the hardware, hence it can apply energy management method at fine time grain for example gaps between successive packets on a link. Low power consumption of L0s state is offset by the very low exit latency to move from sleep to active state and resume transmission. The packet-transmit diagram for TOS is given in Fig. 2. During active transmission periods, the device is in active state S_{active} and hence consumes active power. During the idle (no transmission) period, the link steps down to sleep state after some "runway" interval, as discussed by Kant in [31]. We have followed the same "runway" concept both in case of TOS and TRS, to avoid unnecessary delay introduced due to transition to sleep states. Dynamic runway to adapt to the network state is left for future studies.

4.2.2. TRS model

Compared to the TOS, the TRS model works at a coarser time grain and uses L1 sleep state. The runway for TRS is kept larger than

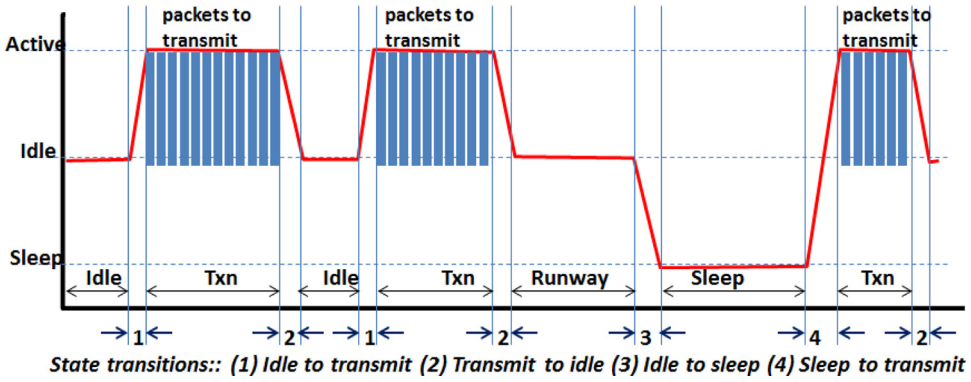

Fig. 2. TOS packet transmit model.

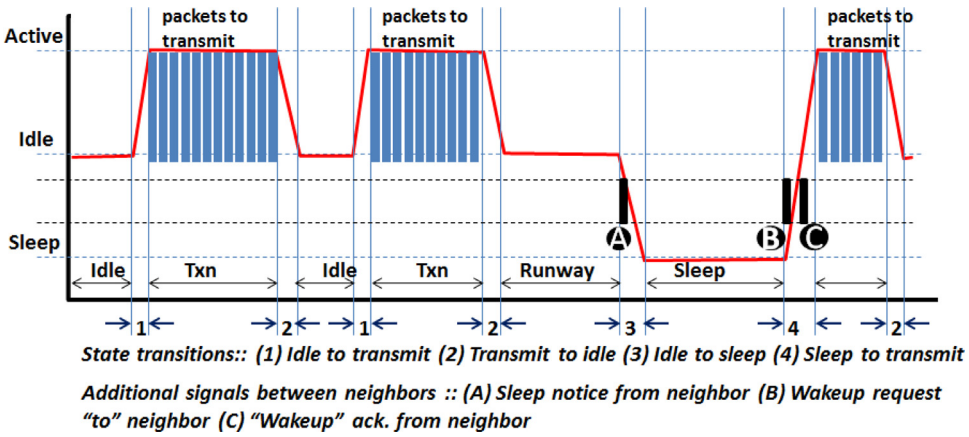

Fig. 3. TRS packet transmit model.

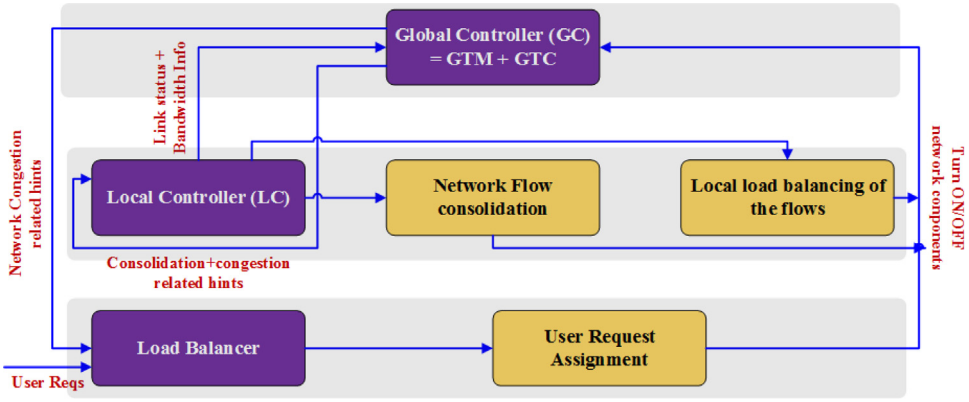

Fig. 4. Overall Diagram.

TOS because of its higher exit latency. In the TRS model, each device is aware of its neighboring devices' energy state. Before switching to deep sleep state, an idle device has to inform its neighbor of its transition to sleep state. Likewise, if the neighbor of a device is in sleep state, the device has to wake up the sleeping neighbor, before forwarding the network traffic. This adds a level of management burden on both the transmit and the neighboring receive devices. This overhead involves – additional queue depth to buffer packets until the neighbor wakes up and acknowledges its active status, and an inter-device handshaking through packets exchange (sleep packet and wake up packet). The TRS state transition model and the packet transmission modes with the overlay of inter-device wake-up packets is shown in Fig. 3. We study this model along with its effect on latency, queue depth and opportunities to extend the sleep state. This mechanism can save more energy than TOS when the link utilization is low, as we have discussed in the results.

5. Controllers and coordination

The energy efficiency achieved by applying LPI can best be attained by the mean of different controllers and the coordination. In our design, the network operation and traffic management is controlled by three interconnected controllers: Global Controller (GC), Local Controller (LC) and the topology aware Request Assignment Controller (RAC). We have further divided the functionality of GC into two entities, namely, Global Traffic Monitor (GTM) and the Global Traffic Consolidator (GTC).

When multiple controllers are involved, it is crucial to build a consistent strategy that needs to be followed by the controllers to avoid any contradicting actions [32]. GC is responsible for network monitoring and providing hints to LCs and RAC for traffic consolidation and external request placements respectively. The main functionality of GTC is to build *consistent strategies* for the LCs in order to maximize sleep opportunities and to avoid network congestion. Migrating an ongoing flow from one path to another can be disruptive and may reorder packets; therefore, any migration should be done very infrequently. This requires that the GTM is reasonably active in collecting up to date information, so that GTM can provide timely hints to LC's and LB about flow/request placements for enough flow consolidation and at the same time avoid the network congestion (Fig. 4).

Each LC periodically collects low-level network statistics (For example, link utilization, number of active flows, etc.) and sends it to GTM. GTM builds a global view of the network based on those accumulated intelligence. Depending on the global state of the network, the GTM sends back very little hints that can help LCs to make routing decisions. By adopting this simple information exchange mechanism, it is possible to keep the GTM scalable even with reasonably large networks.

LCs are responsible for placement of a new incoming flow and can only do local consolidation of the flows. For example, in the fat-tree network shown in Fig. 5, an edge switch has two links towards the aggregation switch. From a power management perspective, it is better to concentrate traffic on one of them (if possible), so that the other link can have longer idle duration and hence better chance to save power by going into the sleep mode. Similar consolidation principle can be applied to other interconnection typologies like hypercube, as shown in Fig. 6. It is clear that if each LC greedily focuses traffic on one of the links, this may result in congestion or suboptimal traffic distribution at higher levels. The purpose of

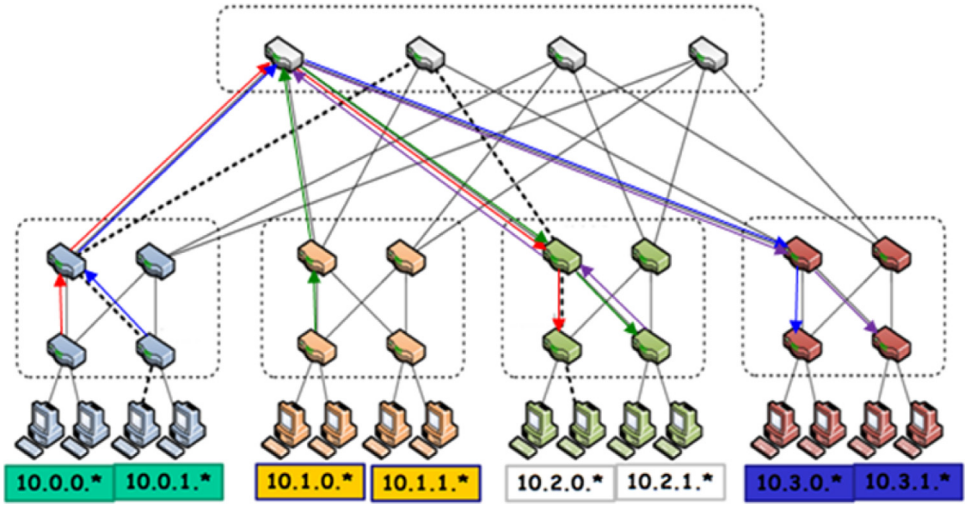

Fig. 5. An illustration of fat-tree network.

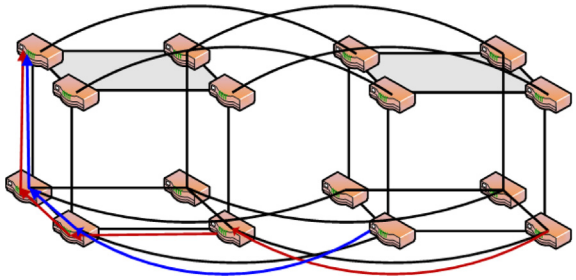

Fig. 6. An illustration of hypercube network.

GTM is to provide hints to LCs to avoid this situation. However, if this hinting is inadequate and congestion or poor traffic distribution does occur further up in the network, GTC can handle it by migrating some of the flows in such a way that they can still reach their destination. The GTC must ensure that it (1) does not move too many flows within the network, and (2) a flow that has already been moved from its original path, is not moved yet again in its lifetime. The later restriction is adequate in most commercial workload environments serving customer queries, since the likelihood of having many extremely long lasting flows is negligible. However, in HPC environments with long running workflows, this aspect may need to be studied further.

Besides providing hints to the LC, at the same time GTM provides these topology aware hints to the RAC. Doing so, the LC and LB can work collaboratively. These hints help the LB to place an external request in such a way, so that LCs get the opportunity to consolidate the traffic without much degradation in quality of services. In the following, we discuss the optimization achieved by a coordination between these various controllers.

5.1. Local controller

As mentioned before LCs work at switching node level. LC operates based on probability factor assigned to the outgoing links, and a *customized routing table* which records all the currently active flows. When any packet comes to a switch, LC first matches the flow id and the destination associated with the packet. If it finds any matches with any existing entry then it simply forwards packet to the path associated with the entry. If not, the packet is forwarded based on the probability factor assigned by the GTC.

In case the LC at a switch is not able to place a flow in its first attempt (due to congestion at some links), it simply drops the flow. We call such incidents "flow blocking". The amount of flow blocking is used as in indicator to balance our twin objectives of flow consolidation and congestion avoidance through load balancing. Notice that instead of dropping these flows, the LCs can try these flows through different links. However, we ignore this case for simplicity.

5.2. Global controller

As stated earlier GC gives hints to LCs to consolidate the traffic over a small number of links, where others mostly remain idle and or more likely to have gaps between packets transmission and can easily move to sleep mode. GC does this by assigning a *probability factor*, which the LCs refer to while forwarding the flows. Notice that in a k-ary fat tree each edge and aggregate switch has $\frac{k}{2}$ candidate links for forwarding the traffic over other Pods. The candidate links are given ranks or priorities that are consistent throughout the network. Without any loss of generality, we assume that the candidate links can be ranked from left-to-right, i.e. the flows are mostly consolidated to the leftmost links of the switches.

For example in Fig. 5, the LCs assign the flows at the leftmost links which ensures better flow consolidation and thus energy savings. The probability factor determines which fraction of the incoming traffic is forwarded to each one of the $\frac{k}{2}$ candidate links.

We now explain calculation of the probability factor for the candidate links for $k = 4$, which can be generalized for any other k. There are two candidate links at the ith switch, which are denoted from left-to-right as $\{l_{i1}, l_{i2}\}$ and their probability factors are $\{P_1, P_2\}$ respectively. The key idea is to ensure that the flows are mostly consolidated at the leftmost links at the time of low-traffic hours, whereas are gradually spread across the network at the high-traffic hours to avoid unnecessary flow blocking. We define the overall network *utilization factor* as the ratio of cumulative bandwidth of all the inter-rack flows, and the total capacity of the edge-level links. This utilization factor is used as a nob to tune the values of P_1 and P_2 to consolidate or spread the traffic over the network. The overall scheme can be described by 4 states, as described below.

1. State1 (S1): The system stays in this state at the lull hours. This happens when there is no flow blocking in the last N windows (each window consists of n number of flows) and the utilization is less than a threshold τ_1. In this case, P_1 is incremented by Δ_i (whereas P_2 is decreased by the same amount) and N becomes 1. At this state P_1 is quickly incremented, so that the flows are more consolidated at the leftmost links.

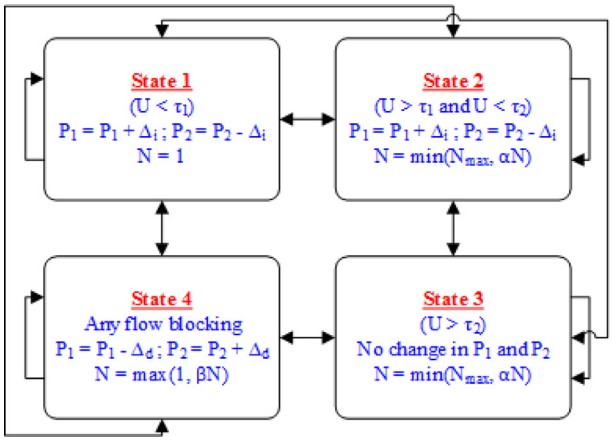

Fig. 7. Proposed state diagram for assigning P_1 and P_2 to the links. U denotes the utilization factor.

2. State2 (S2): The network switches from S1 to S2 if the utilization grows from τ_1 to τ_2 ($\tau_2 > \tau_2$), without experiencing any flow blocking in the last N windows. In this case, P_1 is incremented by Δ_i and N is incremented by a factor of α ($\alpha > 1$). Thus, in this state the probability of forwarding the flows to the leftmost links increases, but at a slower rate.
3. State3 (S3): The system transitions from S2 to S3, when the utilization grows beyond τ_2. In this state the probability factors remain unchanged to avoid overburdening the leftmost links further.
4. State4 (S4): The system goes to S4 whenever it experiences any flow blocking. In this case, P_1 is decremented by Δ_d, whereas N is decremented by a factor of β ($\beta < 1$). Thus if there are more flow blocking in the successive windows, P_1 is decremented quickly to shift more flows to the rightmost links. The entire state diagram is depicted in Fig. 7, where $N_{\max}$ is assumed to be the maximum value of N beyond which N is not increased.

We adjust the τ_1 value by adopting a simple learning approach. When there is no flow blocking for $N_{\max}$ windows with an utilization of U_{NB}, then the value of τ_1 is set to $U_{NB}(1 - \eta\Delta_d)$, where $\eta = 0, 1, 2, \ldots$ is a predefined constant. The objective of this is to keep τ_1 low enough to avoid flow blocking, and at the same time keep it sufficiently high for effective consolidation.

In addition to that the GTC also provides hints to the request assignment controller to assign the user requests for better energy efficiency as mentioned in Section 5.3.

In general for a switch with $n+1$ possible links, the probability factors are calculated as follows. Assume that the probability factors for the $n+1$ links are $P_1, P_2, \ldots, P_{n+1}$ respectively from left to right. Thus if P_1 is incremented (or decremented) by some amount, then probability factors for all the other corresponding links are calculated by solving the above expressions:

$$\begin{aligned} &P_2 + P_3 + \ldots + P_{n+1} = 1 - P_1 \\ &P_i = \zeta^{i-1}(1 - P_1) \quad \forall 2 \le i \le n+1, \quad 0 \le \zeta \le 1 \end{aligned} \tag{1}$$

where ζ is a system constant. The idea is that the probability factors of the links decrease exponentially from left to right. Thus the traffic is mostly consolidated at the left-most links, which provides enough opportunity for the links towards the right to sleep.

5.3. Request assignment controller

Based on the hints provided by GC, the RAC assigns the incoming requests to the servers. It also takes into account the host capacity (I/O per second, number of VMs, etc) while assigning the request. There are multiple policies for the RAC [33]. Some of the policies can be taken independently by the RAC, whereas other need network information provided by the GC. The detailed evaluations of different policies of RAC is beyond the scope of this paper, instead we mainly evaluated our performances to show the interactions between GC and LC.

5.4. Scalability of control mechanism

The amount of information exchanged between LCs and GC increases both with the size of the data center and the dynamism of the traffic in terms of congestion episodes. It is important to note that each switch handles the congestion locally in its outgoing links; only the congestion in the down-links requires exchange of hints between LCs and GC. However, the congestion in the downlink means that a lot of data headed to the corresponding rack. If this happens frequently or persistently, it means that there is deficiency in the hints are provided to RAC by GC, rather than a scalability issue. In this regard we note that the congestion notification to the GC is an event driven process, rather than a periodic one. In the fat-tree topology, as the size of network (i.e. the parameter k) increases, so does the number of alternative links, which is likely to reduce congestion episodes. Nevertheless, as the network size increases, the GC must keep track of more information and provide hints to more switches.

Suppose that at some point in time, the fraction X links have exceeded the higher threshold. The corresponding LCs will then continue sending the congestion notification until the congestion reaches down to the lower threshold. From the definition of Fat-tree, in a K-array Fat-tree, we have $K^2/4$ core switches, each connecting to all K pods. This gives $K^3/4$ downlinks from core switches. Each of the K pods has $K/2$ aggregate switches, each with $K/2$ downlinks, or a total of $K^2/4$ downlinks per pod from aggregate switches. The number of edge switches is same as aggregate switches, each connecting to $K/2$ "hosts", which yields another $K^3/4$ downlinks. Thus we have a total of $3.K^3/4$ downlinks.

In real data centers, the "host" of a pure fat-tree structure can be considered as a rack connected via "top-of-the-rack" (ToR) switch to the edge switch on one side and the rack servers on the other. Thus a pure fat-tree structure beyond the rack level has $N = K^3/4$ racks and $3N$ downlinks. A standard rack can typically hold 42 1U servers, and as many as 96 blade servers. A rack typically also has two uplinks to load balancing and reliability, but let's ignore that for simplicity. We also do not consider congestion on the links going to individual servers in order to keep focus on the fat-tree which lives above the rack.

Now to get some insight into the request handling capability required by the GC, we need to choose the fraction of congested links, say X, realistically. In a pathological scenario, it is possible that $X = 1$, i.e. every link is congested. This is completely unreasonable since real data center networks are designed with enough capacity to make packet drops and congestion a very infrequent occurrence. We assume that it is adequate to be prepared to handle, on the average, one congested link in every downlink set of a switch, i.e. $X = 2/K$. (Note that this assumption correctly accounts for the fact that congestion becomes less likely in larger network due to more alternate links). Thus, the required request handling capability required by the GC, say R, is given by:

$$R \le (3.K^3/4) * X = 3.K^2/2 = 6N/K \tag{2}$$

Now consider a case of $K = 32$, which yields $N = K^3/4 = 8K$. With 40 servers per-rack, this corresponds to a very large data center with 320K servers. For this, $R = 1.5K$. Suppose that the GC recomputes hints every 100 ms and send them out to individual switches to mitigate congestion. Such responsiveness should suffice for most interactive applications. Now, receiving, processing, and respond-

ing to 1.5K congestion notification requests in 100 ms should be possible for high end server with 16 cores. The networking requirements are also very modest – 15K packets/s ingress and outgress, each perhaps only 100–200 bytes in size and should not stress the minimum 10 Gbps capability that one would expect to have here. As to the storage requirements, maintaining 3*N* or 24K different congestion entries is quite small and could well be mostly residing in processor caches (L1-L3).

A truly large data center may require breaking up GC into per-pod GCs coordinated by a global GC, but we do not consider such extreme cases here.

6. Implementation of enhanced energy modeling framework in NS3

The NS3 network devices implemented two states: BUSY and IDLE. Since we have a concept of "runway", to simulate our idea, we have enhanced the net devices by including low power reactive(LRR) state. The duration of this state can be set as user defined parameter. It is reasonable to assume that the power consumption during entry to and exit from LPR is the same as IDLE power. During the LRR state, the power is same the active power. In this section, we briefly discuss the enhancement we made over existing NS3 framework. Implementation details have been described here [34].

6.1. Network devices (NS3::PointToPointNetDevice)

It tracks the device queue, and implements MAC & PHY layer logic for P2P connections, in a network topology. We overload the methods Receive, Send, TransmitStart and TransmitComplete to support our TOS and TRS mechanism. Whenever the device is in SLEEP state (deep or shallow), and first packet arrives, the packet is scheduled at time $t_0 + t_{wakeUp}$; where t_0 = current time and t_{wakeUp} = wake up latency. This additional time is accounted for transmission time.

6.2. Energy models (NS3:SimpleDeviceEnergyModel)

The energy model "models" the behavior of the device w.r.t packet transmission and device states. Our enhanced energy model includes both the active current and the leakage current. We designed our enhEnergyModel to represent the energy consumption behavior of the network device (enhPointToPointNetDevice). The leakage current was used to model the behaviour of the SLEEP state. Besides we provided flexibility to supplement the supply voltage (as active voltage) and another low power state voltage (to be used during L0s state).

6.3. Energy consumption of backplane/fabric (NS3:node)

In the basic switch fabrics, all the incoming and outgoing net-devices are connected through the device backplane as shown in Fig. 9. The switch backplane consumes more power compared to the ports, connected to it. A NS3 node can be basic representation of the switch back-plane. We derived our extension enhNode from basic NS3 node. We designed the energy management model on the node fabric by implementing a fabric state model and monitoring logic shown in Fig. 8. The node fabric periodically checks the activity of it's connected network devices. Based on the inactivity interval, the fabrics can switch to low power (SLEEP) state. While in SLEEP state, the penalty for processing the first packet would be t_0 + t_{wakeup}, where t_{wakeup} is the wakeup penalty and in the order of 100 s of μs(C6). Because of the high cost of fabric exit latency, the fabric wake time suppresses the net device wake up time.

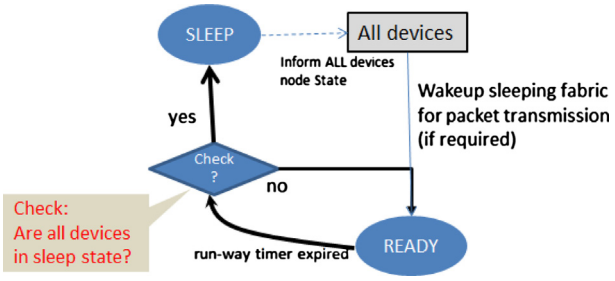

Fig. 8. Fabric state model.

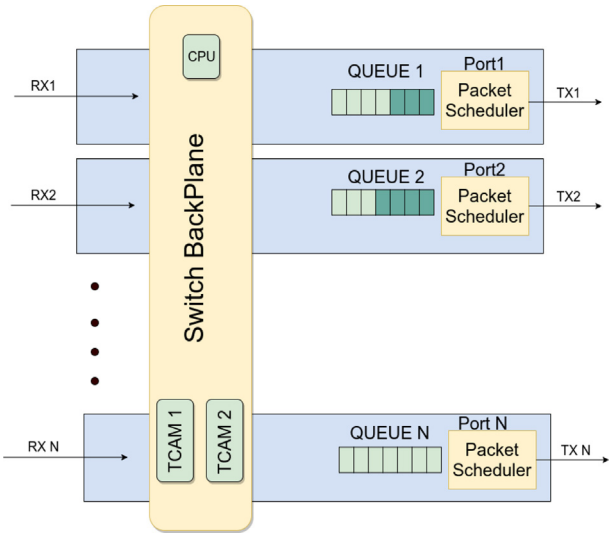

Fig. 9. Switch fabrics.

6.4. Energy source (NS3:BasicEnergySource)

BasicEnergySource is responsible to calculate the power consumption to the entire node. Since we have linked device models directly to the network device, we need to modify the energy consumption calculation of this module (The *UpdateEnergySource* and *CalculateRemainingEnergy* methods). We define the overall node power consumption as:

$$Power(backplane) + \sum_{n=1}^{TotalPorts} Power(port_n)$$

Our energy source contains an active state voltage and low power state voltage[1] required to calculate energy consumption at various device states.

6.5. Enhanced energy efficient routing mechanism (NS3:Ipv4GlobalRouting)

We have enhanced the default routing module of NS3 by introducing deterministic and probabilistic routing. The base change for both type of routing is at the packet attributes. We include flow id attribute with each packets, to differentiate the packets from different flows. Besides the default routing table structure, we derived one dummy routing table structure. This includes additional attributes like putting the default routing table intact. The custom routing table includes the information like bandwidth usage, flow path, current system constant, etc. Like LC

[1] defined as per sleep state.

Table 1
Simulation configuration (parameters used in experiment).

Parameters	Values
Active power	14.85 W
Idle state power	14.85 W (Sync Link)
TOS sleep state power	5.94 W
TOS sleep to active penalty	0.1 μs
TOS runway	5 μs
TRS sleep state power	1.85 W
TRS sleep to active penalty	4.8 μs
TRS runway	23 μs
Backplane active power	60 W
Backplane sleep power	12W
Backplane sleep to active penalty	200μs
Backplane runway	500μs

(discussed earlier), this enhanced module send periodically the dummy/custom routing table information to the GC. This frequency of information exchange can be varied by tuning user defined parameter. The implementation of the GC is done on *GlobalRouteManagerImpl* class, which by default is the controller of all the *Ipv4GlobalRouting* instances.

The goal of both the deterministic and probabilistic routing is consolidation. While consolidation GC should put the network congestion into consideration. The decision (assign probability/priority) is based on the bandwidth usage data sent by the derived *Ipv4GlobalRouting* instances. In case of probabilistic routing, GC method of *GlobalRouteManagerImpl* class only assign one *system constant* value to the *Ipv4GlobalRouting* instances. The task of the LC would be to find out the probability distribution from that system constant value (as discussed in Section 5). The interactions of different enhanced NS3-modules is shown in Appendix A.

7. Evaluation of coordinated power management technique

7.1. Experimental setup

We used our developed simulator to show the effect of traffic consolidation on different network performance metrics. Unless otherwise mentioned, we assume a fat-tree topology with $k = 4$ that accommodates 16 host machines. We assume that the link bandwidths of all levels (i.e. edge, aggregate and core levels) are identical and equal to 10 Gb/s. We study the following network parameters for comparison purpose: average end-to-end latency, power consumption and flow blocking percentage. We compare the advantage of power savings by adopting the TOS and TRS, considering both probabilistic and deterministic routing.

We have used synthetic data load to test the new functionality. Flow duration in the network follows Pareto distribution with shape equal to 3.5 and mean of 36 ms, ranging from 22 ms to 375 ms. The bandwidth of each flow follows uniform distribution with mean 100 Mbps and within a range 50 Mbps to 150 Mbps. To support the varying level of burst traffic, we implemented two state Markov Modulated Poisson Process (MMPP) for a flow. In this model, the two states have different (Poisson) flow generation rates based the given burst-rate and average rate parameters.

The parameters used during our experiment are listed at Table 1. Active power for 10 Gbps NIC card is considered as 14.85 W (Chelsio N210 10GbE Server Adapter[2]). According to the IEEE 802.3 standard, the deep sleep and shallow sleep state power consumption is 10% amd 40% of the active power consumption respectively [9]. So, sleep state power of TOS and TRS is 5.94 W and 1.485 W respectively. While applying LPI, the values of Wake up time of 10 Gbps link is 4.8 μs [9]. The wake up time for TOS is 0.1 μs, since this is a hardware mechanism and does not depend on the transmitter and receiver message exchange. So the penalty is negligible compared to the TRS.

[2] http://www.ieee802.org/802_tutorials/05-July/Tutorial-July-Nordman.pdf.

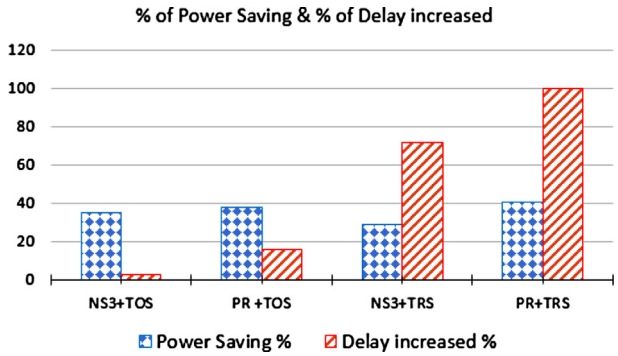

Fig. 10. Comparison of power and delay with different energy management techniques.

7.2. Comparison between different power saving strategies

In our model we have used broadly four different power saving techniques;

(a) No power management at all,
(b) Transmit only sleep, henceforth called TOS for each port
(c) Transmit-Receive sleep, henceforth called TRS for each port and
(d) Additional switch backplane power saving mechanism with port based TOS and TRS discussed earlier.

We first compare the probabilistic routing proposed in Section 5 and the NS3 provided ECMP routing. In case of NS3 ECMP routing, each packet is routed uniformly randomly over one of the shortest paths to the destination. So, ECMP does not provide good opportunities for power savings. For the same reason, the delay is smaller as compared to the consolidated routing (less queue depth). The result of the comparison is shown in Fig. 10. Here the baseline condition for comparison is the NS3 default routing with no power management.

It is clear that the energy management techniques like TOS and TRS can save a significant percentage of power, regardless of routing techniques (probabilistic routing and ECMP). The best power saving can be achieved by probabilistic routing with TRS. The power savings with PR + TRS as compared to the baseline is 40%, although they come at the cost of increased delay. The PR + TOS mechanism also provides great power savings, but with a smaller increase in delay (16 μs). While the delay increase is significant in percentage terms, it is important to keep in mind that the end to end network delays are in 100 of μs range, and the impact of additional 16 μs delay on the performance depends on the nature of the application. While some HPC applications can be very latency sensitive, most are unlikely to be affected by a small increase in networking delays. The additional delays are a result of additional queuing delay due to traffic consolidation and is unavoidable. Also TOS provides better power and performance trade-off than TRS in case of ECMP. The percentage of power saving and delay increase with ns3 ECMP routing and TOS (ns3+TOS) is 35.1% and 2.8%, whereas with TRS the values are 29% and 71.9% respectively. The reason behind higher energy consumption and higher delay in TRS is the frequent transition between active and deep sleep state. TOS does not show the same trend, because of smaller wake up delay and no association of request-acknowledgment message exchange mechanism (Section 4).

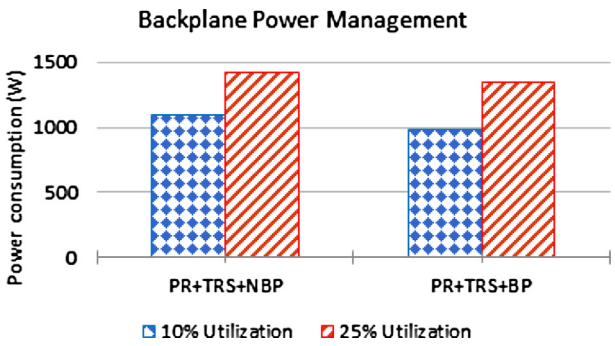

Fig. 11. Power with BP management.

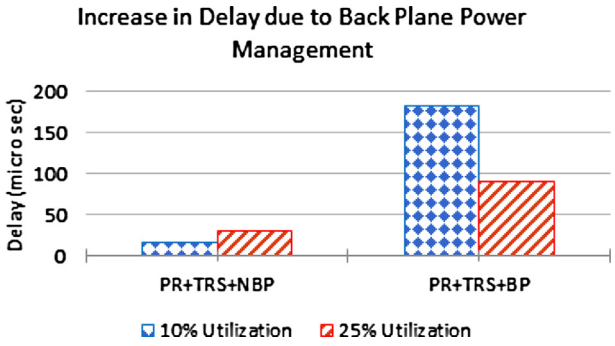

Fig. 12. Delay with BP management.

7.3. Effects of backplane management

Figs. 11 and 12 show effect of backplane (BP) management, on power consumption and network delay. From Fig. 11 we can observe that with 10% utilization (10 %), TRS+PR can reduce the power consumption by 10.3% using the backplane power management. However, the delay due to BP increases significantly as shown in Fig. 12, especially in case of low utilization. This is because the back plane gets much opportunity to switch back to sleep state, while causing more flow delay. In case of higher utilized network, the black plane gets less opportunity to switch into deep sleep state. So, the delay impact is less severe. From Fig. 12 we can observe that with backplane management, the average delay in case of 25% utilization(117.111 μs) is less than the average delay associated at 10% utilization (181.811 μs).

7.4. Comparison between probabilistic and deterministic routing

We next compare the probabilistic routing (PR) with a deterministic routing(DR) scheme. DR is similar to greedy approach proposed by the Elastic Tree [35], except that the priority of the links are changed dynamically. The GC provides congestion hints to the LC and based on that the priority is changed. In DR routing scheme, traffic is forwarded to the highest priority links up to some threshold. When this threshold exceeds, the remaining new flows are forwarded to the next highest priority link, and so on. In our experiment, we assume the maximum threshold of DR to be 0.8.

Figs. 13–15 show the comparison between PR and DR. From Fig. 13 we can observe that the power consumption is reduced significantly in case of DR. This is because of the more aggressive consolidation of the traffic in case of DR. This comes at the cost of higher end-to-end delay and flow blocking as shown in Figs. 14 and 15. From Fig. 14 we can observe that, when the network load is low (10%), the difference in average delay is not so significant. At the lower utilization, there is no flow blocking (drop) as well (see Fig. 15). However, as the network utilization become

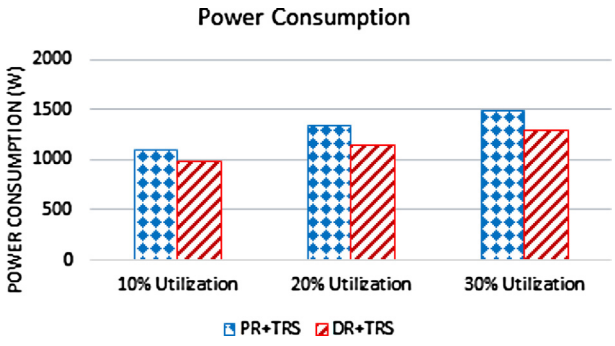

Fig. 13. Power consumption with PR and DR.

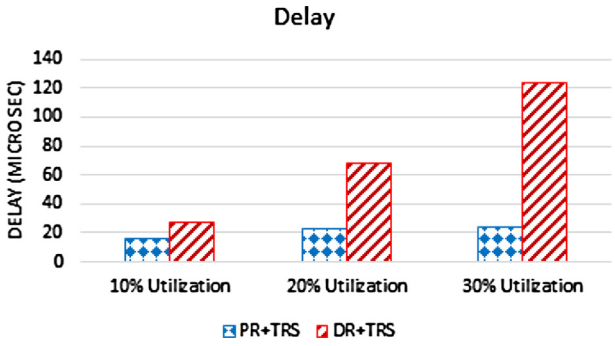

Fig. 14. Delay with PR and DR.

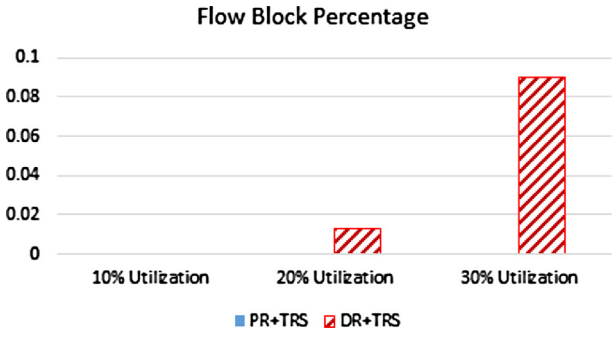

Fig. 15. Flow blocking with PR and DR.

moderate (25%) or high (40%), both the delay and flow blocking start increasing in case of DR. For example in 25% and 40% utilization, the percentage of flow drop becomes 0.013% and 0.09% of the total flows, respectively. The delay is also higher significantly compared to PR. In a moderate load (20%), the average delay has increased by almost 203%. The delay increase could be problematic at high utilization, but packet drop surely is a problem in data center environments.

7.5. Comparison between fat-tree and hypercube network topology

Although fat-tree is the most popular data center network topology, other topologies are also used, particularly in HPC data centers. These include hypercube, toroid and related topologies. Here we compare hypercube and fat-tree with respect to our probabilistic routing (PR) mechanism. To make the hypercube and fat tree infrastructures comparable, we equate the bisection bandwidth of the two while selecting the k for fat-tree and the dimension d for hyper-

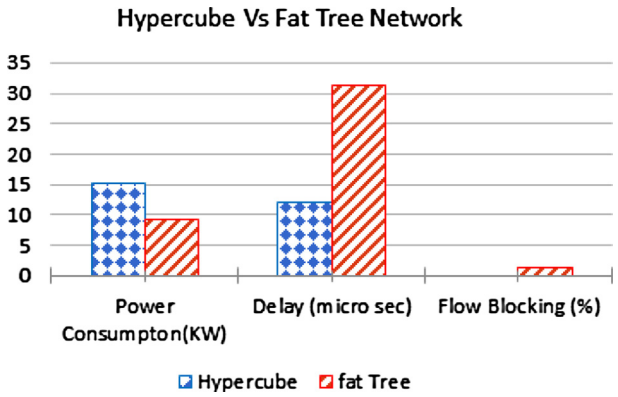

Fig. 16. Comparison of fat tree and hyper-cube.

cube. The bisection bandwidth for fat tree is $(k^3)/8$. The bisection bandwidth for d-dimensional hypercube is 2^{d-1}. Therefore,

$$(k^3/8) = 2^{d-1} \Rightarrow k = \sqrt[3]{2^{d+2}} \quad (3)$$

Based on the above equation, we create a 7-dimension hypercube topology with one host per switch, which results in an interconnection network of 128 hosts. We also use $k=8$ for fat-tree to accommodate 128 hosts.

Fig. 16 compares and power, delay, and blocking probability for the two networks. It is clear that hypercube yields higher power than the fat-tree but also has correspondingly lower delay. It also provides better congestion management and flow blocking. Hypercube consumes 72% more power than the fat tree topology, but with a significantly lower average delay. The average delay with the fat-tree is almost 263% higher, and the flow blocking is 4.6%. With detailed observation we found that for fat-tree topology, the congestion is dominant in the link from the core to the aggregate switch, which is the key reason for higher flow blocking.

8. Conclusions

In this paper, we extend the basic energy management capabilities of networking devices in the context of the popular NS3 network simulation tool. We also propose a coordinated energy management strategy using local (switch-level) and global controllers. We show that a significant amount of power reduction can be achieved by adopting an intelligent, lightweight, hints-based coordination scheme across the controllers. We apply the mechanism to two popular network topologies – the fat-tree and the hypercube and show fat-tree can provide lower power consumption but at the cost of higher delay. There are further opportunities for further reduction of power without creating network congestion by more explicit congestion notification that we are currently exploring.

Appendix A. Interaction between different enhanced modules

Please refer to Fig. A.17

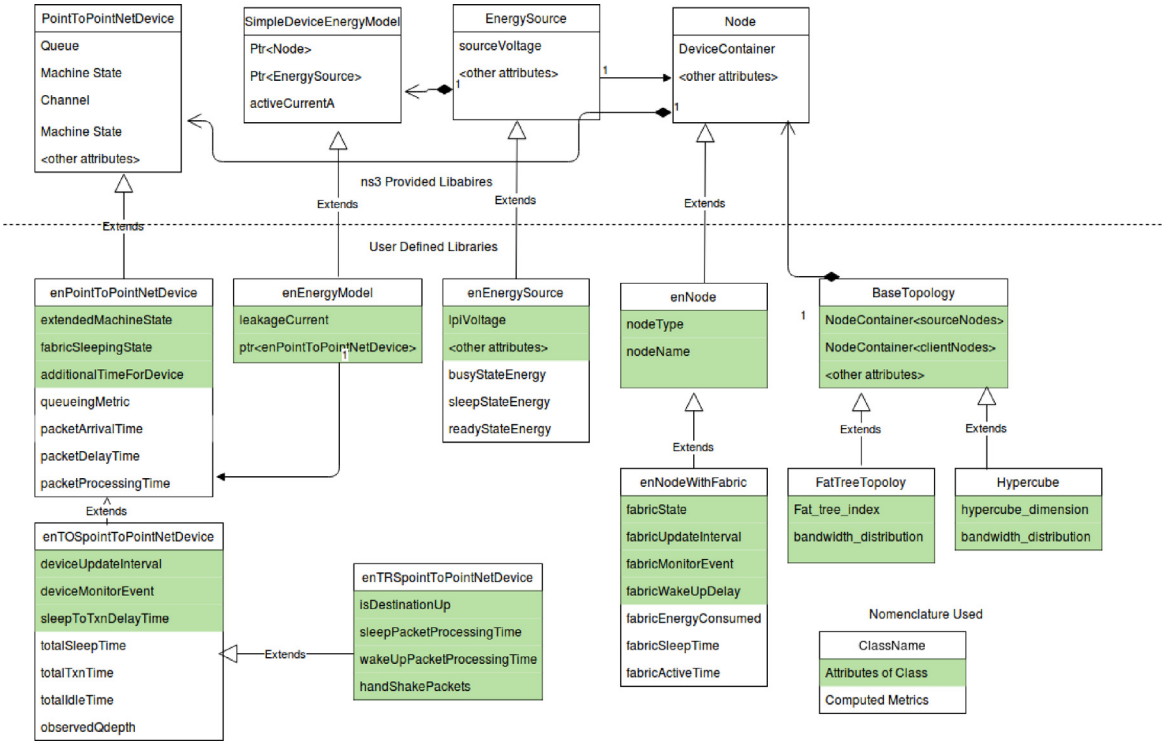

Fig. A.17. Interactions of different modules in NS3.

References

[1] R.E. Brown, R. Brown, E. Masanet, B. Nordman, B. Tschudi, A. Shehabi, J. Stanley, J. Koomey, D. Sartor, P. Chan, E.I.U.E.P.A, et al., With Alliance to Save Energy, ICF Incorporated, Report to Congress on Server and Data Center Energy Efficiency: Public Law 109-431, 2007.

[2] Greenberg, et al., The cost of a cloud: research problems in data center networks, SIGCOMM Comput. Commun. Rev. 39 (1) (2008) 68–73.

[3] Sohan, et al., Characterizing 10 gbps network interface energy consumption, IEEE LCN (2010) 268–271.

[4] M. Murugan, et al., On the interconnect energy efficiency of high end computing systems, Sustainable Computing: Informatics and Systems.

[5] Y. Guo, M. Pan, Y. Gong, Y. Fang, Dynamic multi-tenant coordination for sustainable colocation data centers, IEEE Transactions on Cloud Computing.

[6] Erickson, et al., Using Network Knowledge to Improve Workload Performance in Virtualized Data Centers, Ph.D. Thesis, Stanford University, 2013.

[7] Blanquicet, et al., An initial performance evaluation of rapid Phy selection (rps) for energy efficient ethernet, IEEE LCN (2007) 223–225.

[8] S. Herrera-Alonso, M. Rodrguez-Prez, M. Fernndez-Veiga, C. Lpez-Garca, Optimal configuration of energy-efficient ethernet, Computer Networks.

[9] Christensen, et al. IEEE 802.3 az: the road to energy efficient ethernet, IEEE Communications Magazine 48 (11).

[10] P. Reviriego, J. Hernadez, D. Larrabeiti, J. A. Maestro, Burst transmission for energy-efficient ethernet, IEEE Internet Computing.

[11] R. F. e Silva, P. M. Carpenter, Energy efficient ethernet on mapreduce clusters: packet coalescing to improve 10gbe links, IEEE/ACM Transactions on Networking.

[12] P. Mahadevan, et al., On energy efficiency for enterprise and data center networks, IEEE Commun. Mag. 49 (8) (2011) 94–100.

[13] Mostowfi, et al., A simulation study of energy-efficient ethernet with two modes of low-power operation, IEEE Commun. Lett. 19 (10) (2015) 1702–1705.

[14] C. Thaenchaikun, et al., Augmenting the energy-saving impact of IEEE 802.3az via the control plane, IEEE ICCW (2015) 2843–2849.

[15] Nedevschi, et al., Reducing network energy consumption via sleeping and rate-adaptation, USENIX NSDI (2008) 323–336.

[16] Abts, et al., Energy proportional datacenter networks, SIGARCH Comput. Archit. News 38 (3) (2010) 338–347.

[17] Al-Fares, et al., Hedera: dynamic flow scheduling for data center networks, USENIX NSDI, 19–19 (2010).

[18] X. Wang, et al., Carpo: correlation-aware power optimization in data center networks, IEEE INFOCOM (2012) 1125–1133.

[19] Verma, et al., pmapper: power and migration cost aware application placement in virtualized systems, ACM/IFIP/USENIX International Conference on Middleware (2008) 243–264.

[20] Liu, et al., Greencloud: a new architecture for green data center, ACM ICAC-INDST (2009) 29–38.

[21] Gao, et al., Energy-aware load balancing in heterogeneous cloud data centers, ACM ICMSS (2017) 80–84.

[22] Paya, et al., Energy-aware load balancing and application scaling for the cloud ecosystem, IEEE Transactions on Cloud Computing.

[23] K. Zheng, et al., Joint power optimization of data center network and servers with correlation analysis, IEEE INFOCOM (2014) 2598–2606.

[24] H. Jin, et al., Joint host-network optimization for energy-efficient data center networking, IEEE IPDPS (2013) 623–634.

[25] Meng, et al., Improving the scalability of data center networks with traffic-aware virtual machine placement, IEEE INFOCOM (2010) 1154–1162.

[26] D. Li, et al., Joint power optimization through vm placement and flow scheduling in data centers, IEEE IPCCC (2014) 1–8.

[27] Travostino, et al., Seamless live migration of virtual machines over the man/wan, Future Gener. Comput. Syst. 22 (8) (2006) 901–907.

[28] H. Wu, S. Nabar, R. Poovendran, An energy framework for the network simulator 3, SimuTools (2011).

[29] C. Tapparello, H. Ayatollahi, W.B. Heinzelman, Energy harvesting framework for network simulator 3, ENSsys@SenSys (2014).

[30] Mostowfi, et al., An analytical model for the power consumption of dual-mode eee, Electron. Lett. 52 (15) (2016) 1308–1310.

[31] K. Kant, Multistate power management of communications links, Proc. of COMSNET (2011).

[32] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, X. Zhu, No “power” struggles: coordinated multi-level power management for the data center, SIGARCH Comput. Archit. News.

[33] Ray, et al., Opportunistic power savings with coordinated control in data center networks, ACM ICDCN (2018), 48:1-48:10.

[34] S. Sondur, M. Ray, J. Biswas, K. Kant, Implementing data center network energy management capabilities in ns3, 2017 Eighth International Green and Sustainable Computing Conference (IGSC) (2017) 1–8.

[35] Heller, et al., Elastictree: saving energy in data center networks, USENIX Conference on Networked Systems Design and Implementation (2010).