Energy-Aware Data Transfer Algorithms

Ismail Alan, Engin Arslan, Tevfik Kosar
Department of Computer Science & Engineering
University at Buffalo (SUNY), Buffalo, New York 14260
{ialan,enginars,tkosar}@buffalo.edu

ABSTRACT

The amount of data moved over the Internet per year has already exceeded the Exabyte scale and soon will hit the Zettabyte range. To support this massive amount of data movement across the globe, the networking infrastructure as well as the source and destination nodes consume immense amount of electric power, with an estimated cost measured in billions of dollars. Although considerable amount of research has been done on power management techniques for the networking infrastructure, there has not been much prior work focusing on energy-aware data transfer algorithms for minimizing the power consumed at the end-systems. We introduce novel data transfer algorithms which aim to achieve high data transfer throughput while keeping the energy consumption during the transfers at the minimal levels. Our experimental results show that our energy-aware data transfer algorithms can achieve up to 50% energy savings with the same or higher level of data transfer throughput.

Keywords

Energy-aware data transfers; Energy efficiency; Big-data; Protocol tuning; Power modeling

1. INTRODUCTION

Global Internet traffic will reach 1.1 Zettabytes (one billion Terabytes) per year by 2016 [43]. The annual electricity consumed by these data transfers worldwide is estimated to be 450 Terawatt hours, which translates to around 90 billion U.S. Dollars per year [24, 34, 36, 43]. This fact has resulted in considerable amount of work focusing on power management and energy efficiency in hardware and software systems [12, 14, 16, 17, 25, 27, 31, 38, 39, 46, 50] and more recently on power-aware networking [8, 20, 21, 28, 34].

Most of the existing work on power-aware networking focuses on reducing the power consumption on networking devices such as the hubs, switches, and routers deployed across Internet. Well known techniques in this area include putting idle sub-components such as line cards to sleep mode [23]; adapting the rate at which switches forward packets depending on the traffic rate [35]; developing architectures with programmable switches [22] and switching layers that can incorporate different policies [30]; and new power-aware network protocols for energy-efficiency in network design and routing [13].

Although many studies have been done on power management techniques for the networking infrastructure, there has not been much prior work focusing on saving data transfer energy at the end systems (sender and receiver nodes). There has been studies on Energy Efficient Ethernet [3] for making ethernet cards more energy efficient, but these were very limited in the scope since they were only considering the energy saving of the ethernet card and not the entire end-system (including the CPU, memory, disk etc). Prior studies [4,28,34] show that at least one quarter of the data transfer power consumption happens at the end-systems (sender and receiver nodes) and the rest at the networking infrastructure. Significant amount of data transfer energy savings can be obtained at the end-systems with no or minimal performance penalty. Although network-only approaches are important part of the solution, the end-system power management is another key in optimizing energy efficiency of the data transfers, which should not be ignored.

In our prior work [5], we analyzed various factors that affect the power consumption in end-to-end data transfers with a focus on the end-system energy consumption. Different protocol parameters such as TCP pipelining, parallelism and concurrency levels play a significant role in the achievable network throughput. However, setting the optimal numbers for these parameters is a challenging problem, since not-well-tuned parameters can either cause underutilization of the network, may increase the power consumption drastically, or may overload the network and degrade the performance due to increased packet loss ratio, end-system overhead, and other factors.

In this paper, we introduce three novel data transfer algorithms which consider energy efficiency during data transfers: *i)* a Minimum Energy algorithm (MinE) which tries to minimize the overall energy consumption without any performance concern; *ii)* a High Throughput Energy-Efficient algorithm (HTEE) which tries to maximize the throughput with low energy consumption constraints; and *iii)* an SLA-based Energy-Efficient algorithm (SLAEE) which lets end-users to define their throughput requirements as part of a service-level agreement while keeping the power consumption at the minimum levels to minimize the cost of the service provider.

The rest of this paper is organized as follows. In Section 2, we describe these three energy-aware data transfer algorithms in detail. In Section 3, we present our experimental results and evaluation of these algorithms. In Section 4, we discuss the effect of these algorithms on network power consumption. Section 5 describes the related work in this area, and Section 6 concludes the paper.

2. ENERGY-AWARE TRANSFER ALGORITHMS

In this section, we first describe the application level data transfer parameters we tune in our proposed data transfer algorithms; then we introduce our techniques to model and estimate the energy consumption during data transfers. After giving this background, we present our three novel energy-aware data transfer algorithms.

2.1 Application-layer Parameter Tuning

Three application-layer parameters which can be used both to improve the end-to-end data transfer throughput as well as to control the level of energy consumed during data transfer are: *i)* pipelining; *ii)* parallelism; and *iii)* concurrency. However setting the optimal numbers for these parameters is not an easy task since incorrect tuning of parameters can either cause underutilization of the network or increase the power consumption unnecessarily [5].

Pipelining addresses the problem of transferring a large number of small files. With pipelining, you can transfer multiple files without waiting previous transfer's acknowledgement message in control channel based transfer protocols. It prevents RTT delays between sender and receiver nodes and keeps the transfer channel active. The size of the transferred files should be smaller than the bandwidth-delay product (BDP) to take advantage of pipelining, otherwise using high level of pipelining would increase energy consumption without any benefits to the throughput.

Parallelism refers to transferring different portions of the same file through multiple streams and it can potentially multiply the throughput of a single stream. But, using too many simultaneous streams can cause network congestion and throughput decline. Parallelism is advantageous for large file transfers when the system buffer size is smaller than BDP, which is very common in high-bandwidth long RTT networks.

Concurrency means transferring multiple files at the same time using different data channels. It can result in better throughput especially for transfers in which disk IO is the bottleneck and the end systems have parallel disk systems. In a already fully utilized network, setting concurrency to high levels can cause energy loss because high number of concurrent transfers means increased number of processes (or threads) at the end systems, which can increase the utilization of system resources needlessly.

2.2 Modeling Data Transfer Energy Consumption

Measuring the energy consumption of data transfers is a challenging task due to lack of hardware power monitoring devices on large-scale systems. Even though one can measure power consumption of a specific server using a power meter, it is not always possible to hook up power meters to all servers involved in a particular study, especially when they are remote servers operated by other entities. Thus power models have pivotal role for analyzing and estimating power consumption of servers.

In our previous work [5], we proposed two power models to estimate the server power consumption during data transfers for two different cases of access privileges: *(i)* the fine-grained power model requires access to utilization information of four system components: CPU, memory, disk and NIC; *(ii)* the CPU-based power model only needs utilization information of the CPU in the target system. Our approach resembles Mantis [16] in predicting power consumption of data transfers based on operating system metrics. It is non-intrusive, models the full-system power consumption, and provides real-time power prediction. It requires a one time model building phase to extract power consumption characteristics of the system components. For each system component (i.e. CPU, memory, disk and NIC), we measure the power consumption values for varying load levels. Then, linear regression is applied to derive the coefficients for each component metric. The derived coefficients are used in our power model to predict the total power consumption of the data transfers.

In short, the equation form of our fine-grained power model at time t is as follows:

$$P_t = C_{cpu,n} \times u_{cpu,t} + C_{memory} \times u_{memory,t} + C_{disk} \times u_{disk,t} + C_{nic} \times u_{nic,t} \quad (1)$$

$$C_{cpu,n} = 0.011 \times n^2 - 0.082 \times n + 0.344 \quad (2)$$

P_t refers to predicted power consumption of the data transfer at time t. C_{cpu}, C_{memory}, C_{disk} and C_{nic} are the coefficients of CPU, memory, disk and NIC components respectively; and $\mathrm{u_{cpu}}$, $\mathrm{u_{memory}}$, $\mathrm{u_{disk}}$, and $\mathrm{u_{nic}}$ are the utilization of the same components. Since we used multi-core processor servers, the coefficient of the CPU depends on the number of active cores n during the transfer, as given in Equation 2.

In typical data centers and shared server infrastructures, accessing the disk, memory and network utilization statistics of specific processes is restricted for end-users. Previous studies showed that solely CPU-based power prediction models can perform slightly worse compared to the fine-grained models [39]. According to our regression analysis, the correlation between the CPU utilization and the consumed power is 89.71% for data transfers. Hence, we conclude that CPU-based models can give us accurate enough results where fine-grained models are not applicable.

Moreover, we developed a new extendable power model with the help of the nameplate power value of the CPU component. We used the CPU Thermal Design Power (TDP) value to scale the power drawn by the data transfers at different servers. The CPU based model allows us to predict power consumption of a server solely based on the CPU utilization values and calculating a machine-specific CPU coefficient by using the ratio of the TDP values.

The equation form of the CPU-based power model at time t is as follows:

$$P_t = (C_{cpu,n} \times u_{cpu,t}) \times \frac{TDP_{SR}}{TDP_{SL}} \quad (3)$$

SL is the local server used to define the CPU-based model, and SR is the remote server where we want to extend the CPU-based model. $u_{cpu,t}$ is the CPU utilization of the server SR for a data transfer at time t.

We evaluated the accuracy of our power models on Intel and AMD servers while transferring datasets using various application-layer transfer tools such as scp, rsync, ftp, bbcp and gridftp. The fine-grained model achieves the highest accuracy rate for all tools as expected. The accuracy of model is always above 94% even in the worst case, and the CPU-based model performs close to the fine-grained model when it is tested on the server with similar characteristics. On the other hand, when we extend the CPU-based model to the AMD server by proportioning its TDP value to Intel server's TDP value, the error rate increases by 2-3% compared to fine-grained model. Even though the accuracy of the CPU-based model is higher, it still remains above 95% for ftp, bbcp and gridftp and above 92% for the rest [5].

2.3 Minimum Energy Transfer Algorithm

We developed the Minimum Energy (MinE) algorithm that tunes the above-mentioned transfer parameters to minimize the power consumption during data transfers. Instead of using the same parameter combination for the whole data set, which can increase the power consumption unnecessarily, we initially divide the data sets into three chunks; *Small*, *Medium* and *Large* based on the file sizes and the Bandwidth-Delay-Product (BDP). Then, we check whether each chunk has a sufficient number of files to transfer via the `mergeChunks` subroutine. We merge a chunk with another if it is deemed to be too small to be treated separately. After the dataset is partitioned into chunks, we calculate the best possible parameter combinations for the chunks in which BDP, average file size, and TCP buffer size are taken into consideration. We calculate the pipelining level based on BDP and average file size of the chunk (line 8); the parallelism level based on BDP, average file size and TCP buffer size (line 9). Regarding the value of concurrency, we consider BDP, average file size, and available channel count (line 10) as shown in Algorithm 1. Available channel count is set to maximum channel count and updated at each step as we assign channels to chunks.

Pipelining and concurrency are the most effective parameters at network and system utilization for small file transfers, so it is especially important to choose the best pipelining and concurrency values for such transfers. Pipelining value is calculated by dividing BDP to the average file size of the chunk which returns large values for *Small* chunks. By setting pipelining to relatively high values, we are transferring multiple data packets back-to-back that prevents idleness of the network and system resources, which in turn decreases the energy consumption. Moreover, we assigned most of the available data channels to the *Small* chunk which multiplies the impact of energy saving when combined with pipelining [4]. For the parallelism level, we again consider TCP buffer size, BDP, and average file size. The equation will return small values for *Small* chunks which will avoid creating unnecessarily high number of threads and prevent redundant power consumption.

As the average file size of the chunks increases, pipelining value is set to smaller values as it does not further improve the data transfer throughput. It could even cause performance degradation and redundant power consumption by poorly utilizing the network and system resources. Besides pipelining, MinE also tries to keep the concurrency level of *Large* at minimum since using more concurrent channels for large files causes more power consumption. The parallelism level for *Medium* and *Large* chunks will be high if the system buffer size is insufficient to fill the channel pipe. After setting the best parameter combination for each chunk, the MinE algorithm starts to transfer the chunks concurrently.

Algorithm 1 — Minimum Energy Algorithm

```
1: function MinimumEnergyTransfer(maxChannel)
2:     availChannel = maxChannel
3:     BDP = BW * RTT
4:     files = fetchFilesFromServer()
5:     chunks = partitionFiles(files, BDP)
6:     for each chunk small :: large do
7:         avgFileSize = findAverage(chunk)
8:         pipelining = ⌈BDP / avgFileSize⌉
9:         parallelism = Max(Min(⌈BDP / bufSize⌉, ⌈avgFileSize / bufSize⌉), 1)
10:        concurrency = Min(⌈BDP / avgFileSize⌉, ⌈(availChannel+1) / 2⌉)
11:        availChannel = availChannel − concurrency
12:    end for
13:    startTransfer(chunks)
14: end function
```

2.4 High Throughput Energy-Efficient Transfer Algorithm

The main purpose of the High Throughput Energy-Efficient (HTEE) algorithm is finding the best possible concurrency level to balance the transfer throughput and power consumption levels. The algorithm does not only focus on minimizing the energy consumption or maximizing the throughput, rather it aims to find high performance and low power consumption concurrency levels within defined transfer channel range. While the minimum value for the concurrent number of the transfer channels is 1 for all environments, the maximum value might be different due to variability of end system resource capacities and fairness concerns. Thus, we let the user to be able to decide the maximum acceptable number of concurrent channels for a data transfer. We have chosen concurrency level to tune the throughput, since in our previous work [5,10] we have observed that concurrency is the most influential transfer parameter for all file sizes in most test environments. Even though parallelism also creates multiple channels similar to concurrency, allotting channels to multiple file transfer instead of a single one yields higher disk IO throughput which qualifies concurrency to be the most effective parameter on transfer throughput.

Similar to MinE algorithm, HTEE algorithm partitions datasets into three chunks and calculates the best possible parameter values for pipelining and parallelism using BDP, TCP buffer size and average file size of the chunks (line 5 in Algorithm 2). We also calculated weight for each chunk which is relative to the total size and the number of files in a chunk. Given maximum allowed channel count, weights are used to determine the number of channels to be allocated for each chunk. For example, if we have a dataset dominated by small files, then assigning equal number of channels to chunks would cause sub-optimal transfer throughput since large chunks will be transferred faster than smaller chunks and average transfer throughput will be subjected to small chunk's throughput. To solve this problem, we initially calculate the weights of the chunks (in lines 7 and 8) and allocate channels to chunks accordingly (line 12).

In order to quantify energy efficiency, we introduced throughput/energy ratio as the ratio of transfer throughput to energy consumption. It quantifies how much data can be trans-

Algorithm 2 — High Throughput Energy-Efficient Algorithm

```
1: function HTEE Transfer(maxChannel)
2:     BDP = BW * RTT
3:     files = fetchFilesFromServer()
4:     chunks = partitionFiles(files, BDP)
5:     calculateParameters()
6:     for i = 0; i < chunks.length; i++ do
7:         weights[i] = log(chunks[i].size) * log(chunks[i].fCount)
8:         totalWeight+ = weights[i]
9:     end for
10:    for i = 0; i < chunks.length; i++ do
11:        weights[i] = weights[i]/totalWeight
12:        channelAllocation[i] = ⌊maxChannel * weights[i]⌋
13:    end for
14:    activeChannel = 1
15:    energyEfficiency[];
16:    while activeChannel <= maxChannel do
17:        transfer(activeChannel)
18:        energyConsumption = calculate(SystemMetrics)
19:        throughput = calculateThroughput()
20:        energyEfficiency[i] = throughput / energyConsumption
21:        activeChannel+ = 2
22:    end while
23:    efficentChannelCount = max(energyEfficiency[])
24:    transfer(efficentChannelCount)
25: end function
```

Algorithm 3 — SLA Based Energy-Efficient Transfer Algorithm

```
1: function SLA Transfer(SLA Level, maxChannel)
2:     BDP = BW * RTT
3:     files = fetchFilesFromServer()
4:     chunks = partitionFiles(files, BDP)
5:     calculateParameters()
6:     targetThroughput = maxThroughput * SLALevel
7:     concurrency = 1
8:     transfer(concurrency)
9:     actThroughput = calculateThroughput()
10:    if actThroughput <= targetThroughput then
11:        concurrency = targetThroughput/actThroughput
12:        transfer(concurrency)
13:    end if
14:    while actThroughput <= targetThroughput do
15:        if concurrency < maxChannel then
16:            concurrency++
17:        else
18:            reArrangeChannels()
19:        end if
20:        transfer(concurrency)
21:        actThroughput = calculateThroughput()
22:    end while
23: end function
```

ferred at the cost of unit energy consumption. The HTEE algorithm starts to transfer files with one active channel, and increases the channel count till reaching the user defined maximum channel count. Instead of evaluating the performance of all concurrency levels in the search space, HTEE halves the search space by incrementing the concurrency level by two each time. Each concurrency level is executed for five second time intervals and then the power consumption and throughput of each interval are calculated. Once HTEE examines throughput/energy ratio of all concurrency levels in the search range, it picks the concurrency level with maximum throughput/energy ratio to use it for transferring the rest of the dataset. HTEE differs from MinE in two ways: *(i)* HTEE does not impose any restriction on the number of channels to be allocated to the channels whereas MinE assigns single channel to *Large* chunk regardless of its weight; *(ii)* it evaluates multiple values of concurrency (the number of channel counts) in order to find the one which achieves the highest throughput/energy ratio while MinE transfers whole dataset with a pre-calculated concurrency values.

2.5 SLA Based Energy-Efficient Transfer Algorithm

The rapidly advancing and growing pay-as-you-go and low cost cloud computing services attract more users everyday. As the usage of such systems increases, the guaranteed reliability and quality of the provided services become more important, which requires mutually agreed Service Level Agreements (SLA). Hence, we devised an SLA based Energy Efficient (SLAEE) algorithm. In this algorithm, we seek a concurrency level which satisfies the SLA requirement with the minimal possible energy consumption.

Our algorithm lets the end-users to define their throughput requirements as a percentage of maximum achievable throughput in the transfer environment. The algorithm takes desired throughput value as input and achieves it by tuning the transfer parameters. While keeping the quality of service (transfer throughput in this case) at the desired level, SLAEE uses a technique similar to MinE to keep the power consumption at the minimum possible level. After partitioning the dataset, it calculates the best pipelining and parallelism values using BDP, TCP buffer size and average file size as in the MinE algorithm. It starts with concurrency level one, and if the actual throughput is less than the SLA requirement, it estimates the desired concurrency level based on the current and target throughputs (line 11 in Algorithm 3). If the current throughput is still below the target, then incremental increase is applied to reach the target throughput until it reaches or exceeds the target (line 14-22). When assigning the transfer channels to the chunks, SLAEE gives priority to small chunks similar to the HTEE algorithm due to the energy consumption concerns. While seeking the desired concurrency level, it calculates the throughput in every five seconds and adjusts the concurrency level to reach the throughput level promised in the SLA. While increasing the throughput to the desired level, SLAEE tries to keep the energy consumption at the minimal levels possible. To do this, it follows HTEE algorithm's way of limiting the number of channels of *Large* chunks until either reaching the maximum allowed concurrency level or target throughput. If achieved throughput is still below target throughput as the concurrency level reaches to maximum allowed value, then SLAEE reassigns channels to chunks such that *Large* chunks receive more than one channel (line 18).

3. EXPERIMENTAL RESULTS

The experiments to evaluate our energy-aware data transfer algorithms are conducted on the XSEDE [1], FutureGrid [2], and DIDCLAB [15] testbeds. Location and specifications of the used servers as well as the links between them are presented in Figure 1. In the experiments we transferred different datasets due to different bandwidth capacities of the networks. For 10 Gbps networks, the total size of dataset is 160 GB where file sizes range between 3 MB – 20 GB and for 1 Gbps networks, the total size of experiment dataset is 40 GB where file sizes range between 3 MB – 5 GB. For both datasets, around 20% of the files are smaller than 5MB, 20% between 10MB-100MB, 30% between 100MB-3GB, and the

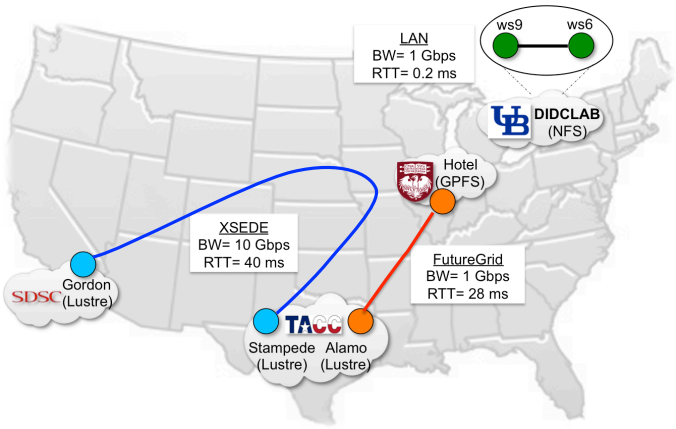

Figure 1: Network map and specifications of test environments.

remaining 30% larger than 3GB.

We compared the performance of our energy-aware MinE and HTEE algorithms with energy-agnostic Single Chunk (SC) and Pro-active Multi Chunk (ProMC) algorithms [10] as well as the popular cloud-hosted data transfer service Globus Online (GO) [6] and the standard Globus GridFTP client (Globus-url-copy). SC and GO algorithms employ divide and transfer approach to transfer a dataset with mixed sized files. Dataset is partitioned into groups (a.k.a chunks) based on file sizes then chunks are transferred one by one using the parameter combination specific to the chunk type. They differ in the partitioning method and the way they decide the values of protocol parameters for each group. While GO uses fixed values to categorize files in to groups (i.e. less than 50MB, larger than 250MB, and in between) and determine values of protocol parameters (e.g. set pipelining level 20 and parallelism level 2 for small files), SC takes network (Bandwidth, RTT) and dataset (average file size, and number of files) characteristics into account. In order to do fair comparison, checksum feature is disabled in GO transfers which is used to enhance data transfer integrity but causes significant slowdowns in average transfer throughput. ProMC algorithm also performs dataset partitioning but instead of scheduling each chunk transfer one by one, multiple chunks are transferred simultaneously in order to alleviate the effect of low transfer throughput of small chunks over whole dataset. Finally, Globus-url-copy (GUC) is a command line GridFTP client that lets user to set protocol parameters for data transfer, however does not allow to use different values of protocol parameters for different files in a dataset. As opposed other algorithms, GUC requires manual tuning of protocol parameters which necessitates expertise on the insights of protocol parameters.

SC, ProMC, HTEE, and MinE use a custom GridFTP client implementation which provides dataset partitioning and better control of data channels such as reallocating a channel to a different chunk and incrementing/decrementing the number of channels (i.e concurrency value) while transfer is running. Moreover, when more than one data transfer channel is intended to be opened, the custom client tries to initiate connections on a single end server even if there are more than one, while GO and GUC distribute channels to multiple servers. Distributing channels over multiple end servers may help to improve reliability and load sharing, however leads to an increase in power consumption due to active CPU utilization on multiple servers.

Although concurrency is the most significant parameter, both SC and ProMC algorithms require user to define the concurrency level. Thus, their performance depends on concurrency value defined by the user. Similarly, MinE algorithm takes concurrency level (represented as *maxChannel* in Algorithm 1) and applies its channel allocation method. Thus, we evaluated the performances of SC, ProMC, and MinE algorithms at different concurrency levels. On the other hand, HTEE algorithm takes upper bound for the concurrency level but finds the optimal level itself during its search phase. Since GO uses fixed value of concurrency, which is 2 for all chunk types, its performance is independent of user-defined maximum value of concurrency. For GUC, we set the value of parallelism, pipelining and concurrency to 1 and consider it as a base performance for a given data transfer. This reflect a use case in which a user without much experience on GridFTP wants to transfer his/her files via GUC. Thus, GUC performance is also independent of the concurrency level.

Figure 2 presents comparison of the algorithms based on their achieved throughput, energy consumption and energy efficiency in the XSEDE network between the Stampede (TACC) and Gordon (SDSC) systems where the network bandwidth is 10 Gbps, the round trip time is 40 ms, and the maximum TCP buffer size is 32 MB. In accordance with their objective, ProMC always achieves highest throughput and MinE achieves lowest energy consumption almost at all concurrency levels. ProMC can reach up to 7.5 Gbps transfer throughput and outperforms all other algorithms in terms of achieved transfer throughput. Interestingly, while MinE and SC yield close transfer throughput in all concurrency levels, SC consumes as much as 20% more energy than MinE as shown in Figure 2 (b). This strengthens the idea that in some cases energy consumption can be decreased even without sacrificing transfer performance. The difference stem from MinE's strategy of assigning single channel to large chunks as large chunks contribute to most of energy consumption in exchange of increased throughput. MinE meets the throughput deficit caused by limiting the number of channels assigned to large chunks by employing "Multi-Chunk" mechanism as used by ProMC. Further, SC and GO achieve very close transfer throughput in concurrency level 2, however, GO consumes around 60% more energy because of the difference in implementation of concurrency which affects the number of servers involved in a data transfer. Since XSEDE systems consist of four data transfer servers, increasing concurrency level from one to two leads to increase in the number of servers taking part in transfer from one to two which then causes more energy consumption as can be seen in Figure 2 (b). Moreover, GUC yields less transfer throughput than SC for concurrency level one which implies that although concurrency is the most influential parameter in transfer throughput, parallelism and pipelining still can make difference in throughput and energy consumption, thus should be tuned reasonably.

While ProMC's throughput increases as concurrency level increases, power consumption follows parabolic pattern and reaches minimum value at concurrency level 4. This is due the fact that data transfer servers on XSEDE have four cores and energy consumption per core decreases as the number of active cores increases [4]. When concurrency level

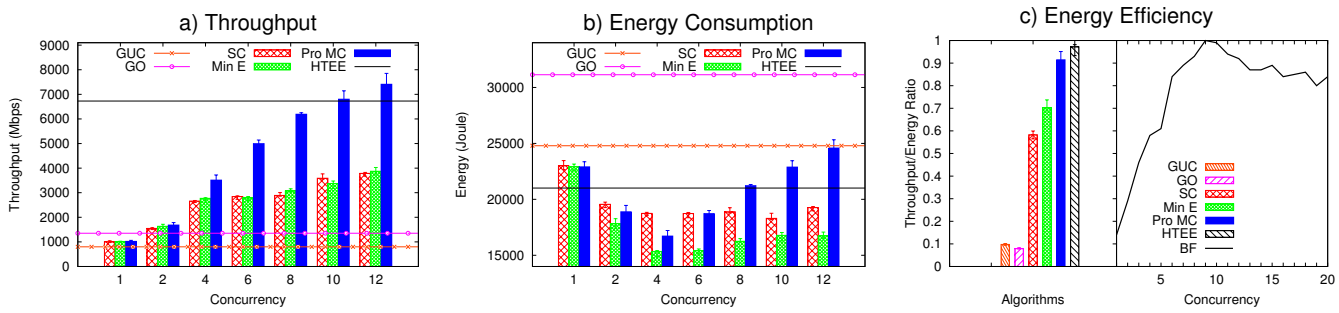

Figure 2: Data transfers between Stampede (TACC) and Gordon (SDSC) @XSEDE.

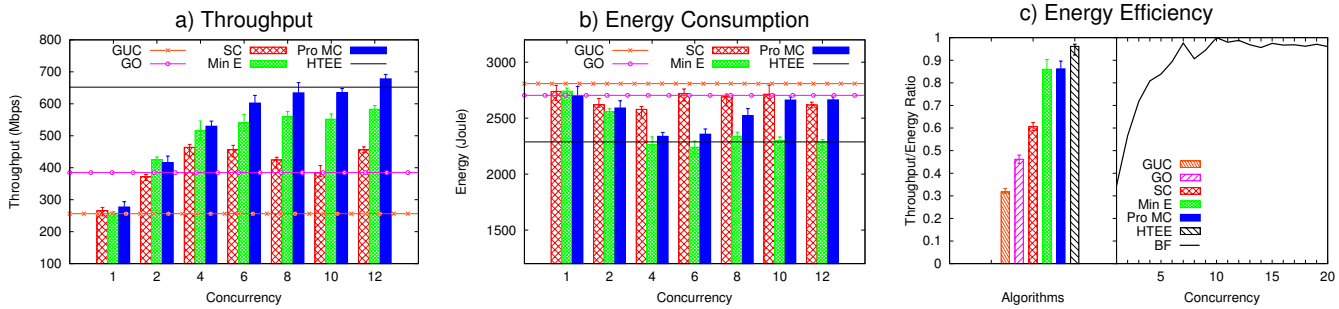

Figure 3: Data transfers between Alamo (TACC) and Hotel (UChicago) @FutureGrid.

goes above 4, then cores start running more data transfer threads which leads to increase in energy consumption per core. HTEE aims to find the sweet spot where the throughput/energy (energy efficiency) ratio is maximized. It searches the concurrency level in the search space, bounded by 1 and *maxChannels*, after which throughput increase is surpassed by energy consumption increase. As given in Algorithm 2, it evaluates the throughput/energy ratio of multiple concurrency levels and picks the concurrency level whose throughput/energy ratio is the highest and runs the rest of the transfer with it. Compared to ProMC, HTEE consumes 17% less energy in trade off 10% less throughput for concurrency level 12 at which ProMC achieves the highest throughput. In concurrency level 8, ProMC consumes similar amount of energy while obtaining 9% less throughput compared to HTEE which can justify the argument of consuming less amount of energy without sacrificing transfer throughput.

In order to compare the performance of the HTEE algorithm to the ideal case, we ran a brute-force search (BF) algorithm to find the concurrency level which maximizes throughput/energy ratio. BF is a revised version of the HTEE algorithm in a way that it skips the search phase and runs the transfer with pre-defined concurrency levels. BF search range is bounded by 20 since the throughput/energy ratio follows a decreasing pattern after around a value of 12 as shown in Figure 2 (c). The concurrency level which yields the highest throughput/energy ratio is considered as the best possible value under these conditions, and the throughput/energy ratio of all other algorithms are compared to this value. As a result, we can see that concurrency level chosen by HTEE can yield as much as 95% throughput/energy efficiency compared to the best possible value obtained by BF. On the other hand, while MinE is successful in consuming the least amount of energy for each concurrency level, it can only reach around 70% of the best possible throughput/energy ratio.

We also tested our energy-aware data transfer algorithms in the FutureGrid testbed between the Alamo (TACC) and Hotel (UChicago) systems where the network bandwidth is 1 Gbps, the round trip time is 28 ms, and the maximum TCP buffer size is 32 MB. GUC again yields the lowest throughput due to lack of parameter tuning. ProMC, MinE, and HTEE algorithms yield comparable data transfer throughput while SC and GO fall behind them. Although the transfer throughput of ProMC, MinE, and HTEE are close to each other, energy consumption values differ considerably. ProMC and MinE consume the least amount of energy when concurrency level is set to 6. As opposed to XSEDE experiments, ProMC and SC achieve their best throughput/energy ratio in comparatively low concurrency levels which are 6 and 4 respectively. This strengthens the motivation behind HTEE as it is designed to capture the network-specific sweet spots in which throughput/energy ratio is maximized. Similar to its XSEDE behavior, MinE reaches its maximum throughput/energy ratio at concurrency level 12 which is because of its channel allocation method. As opposed to ProMC and SC, MinE assigns single channel to the *large* chunk regardless of the maximum channel count and shares the rest of the available channels between *medium* and *small* chunks. Hence, it is able to benefit from increased number of data channels while keeping energy consumption low which paves the way for higher energy/throughput ratio.

Finally, we tested HTEE and MinE algorithms in a local area network as presented in Figure 4. Unlike the XSEDE and FutureGrid experiments, increasing the concurrency level in the local area degrades the transfer throughput and increases the energy consumption. This is due to having single disk storage subsystem whose IO speed decreases when the number of concurrent accesses increases. Since ProMC ag-

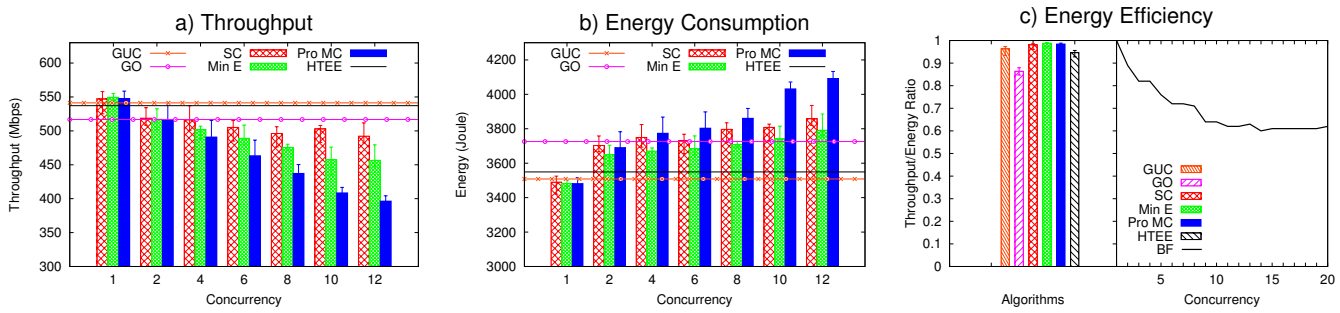

Figure 4: Data transfers between WS9 and WS6 @DIDCLAB.

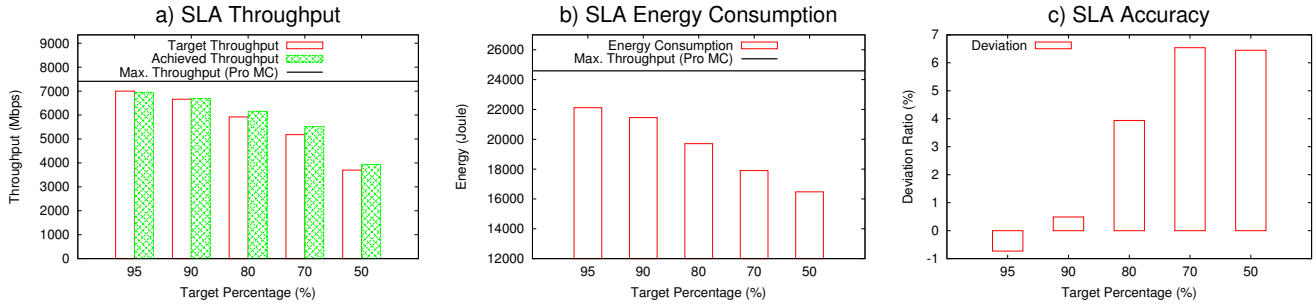

Figure 5: SLA transfers between Stampede (TACC) and Gordon (SDSC) @XSEDE.

gressively uses the available data transfer channels oblivious to the type disk subsystem, its throughput performance suffered the most. All algorithms achieve their best throughput/energy ratio at concurrency level 1 in the local area as expected. While HTEE yields significant benefit in XSEDE and FutureGrid, it performs little worse than other algorithms in the local area since it spends some time in large concurrency levels during its search phase. However, it is worth to mention that all other algorithms perform better than HTEE if they are run at concurrency level one. This requires the user to be able to tune concurrency level manually while HTEE can find optimal concurrency level with the cost of a reasonable overhead. All algorithms except GO are able to achieve above 90% energy efficiency since there is not a way to considerably improve transfer throughput by setting protocol parameters in local area network with single storage subsystem. Since concurrency value is not tunable in GO, its performance suffers as increasing concurrency level causes less throughput and high energy consumption.

We tested our SLA-based Energy-Efficient (SLAEE) transfer algorithm with different levels of throughput targets proportioned to the maximum throughput achieved by ProMC algorithm at concurrency levels 12, 12, and 1 for XSEDE, FutureGrid and DIDCLAB experiments respectively. In this algorithm, x% target percentage means SLAEE tries to achieve transfer throughput more than x% of the maximum throughput possible (i.e. with at most 100% – x% performance loss). For example, maximum throughput achieved by ProMC in XSEDE network is around 7500 Mbps, so 95% target percentage corresponds to 7125 Mbps or more transfer rate.

SLAEE is able to deliver all SLA throughput requests except 95% target throughput percentage at the XSEDE network since SLAEE is unable to reach the requested throughput on this network even after reaching the maximum level of concurrency. As presented in Figure 2 (b), after concurrency level four, energy consumption starts to increase. Thus, achieving SLA target throughput with minimum possible concurrency level will minimize energy consumption. SLAEE is able to achieve all SLA expectations within 7% deviation rate as shown in Figure 5 (c). Since the effect of each new channel creation on throughput is higher when the total number of channels is small, accuracy decreases as SLAEE is expected to provide low transfer throughput. Figure 5 (b) depicts energy consumption comparison between ProMC and SLAEE. SLAEE can deliver requested throughput while decreasing the energy consumption by up to 30%. Hence, if customers are flexible in transferring their data with some reasonable delay, SLAEE algorithm helps the service providers to cut from the energy consumption considerably. Finally, the service providers can possibly offer low-cost data transfer options to their customers in return for delayed transfers.

Since the maximum achievable throughput of the FutureGrid and DIDCLAB testbeds is 1 Gbps, the accuracy of SLAEE differs in these testbeds. While SLAEE can deliver requested throughput with as low as 5% deviation ratio for most cases in FutureGrid, the deviation ratio jumps to 25% when requested throughput is 50% of the maximum throughput as shown in Figure 6 (c). This is because of achieving more than 50% of the maximum throughput with the minimum value on concurrency level which is 1. Hence, SLAEE does its best and picks the concurrency value 1 to transfer the dataset. On the other hand, SLAEE obviates redundant energy consumption compared to ProMC by keeping concurrency level low. The saving in energy consumption ranges between 11% to 19% as the requested throughput ranges between 95% and 50% of the maximum throughput. For the LAN experiments @DIDCLAB, neither throughput nor energy consumption can be improved with useful reason as concurrency level one is the optimal level both for

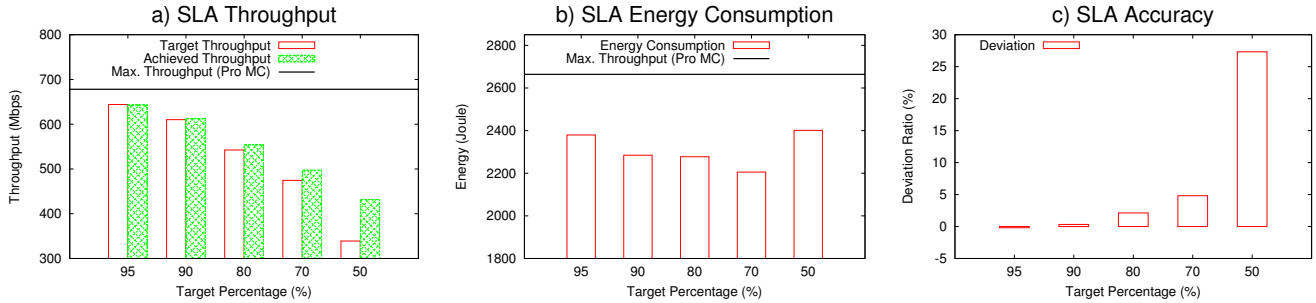

Figure 6: SLA transfers between Alamo (TACC) and Hotel (UChicago) @FutureGrid.

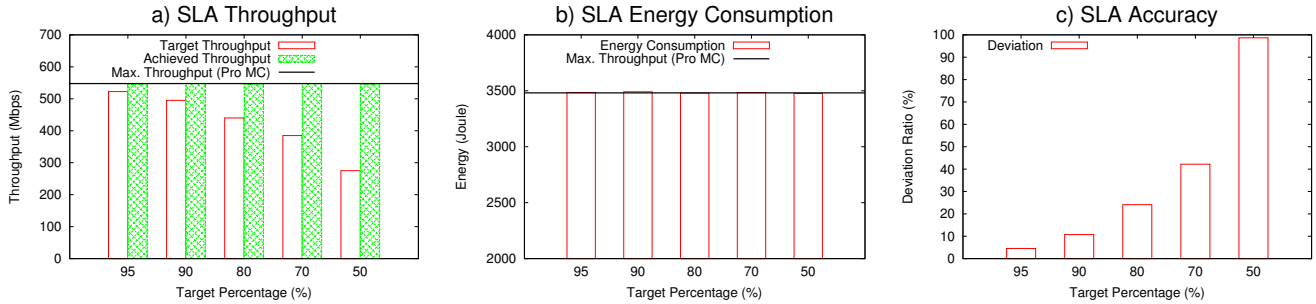

Figure 7: SLA transfers between WS9 and WS6 @DIDCLAB.

throughput and energy consumption. That's also why deviation ratio reaches up to 100% as shown in Figure 7 (c).

4. EFFECT ON NETWORK ENERGY CONSUMPTION

Besides the energy consumption at the end-systems, the data transfers also cause considerable amount of energy consumption at the networking infrastructure (i.e. hubs, switches, routers etc). Although our proposed algorithms primarily aim to reduce the energy consumption at the end-systems during data transfers, it is vital to analyze the effect of these algorithms on the energy consumption of the networking infrastructure as well. Our algorithms simply tune three application-layer transfer parameters (parallelism, pipelining, and concurrency) and do not cause any changes either in the networking configuration, the end-to-end routing, the number of switches and routers involved in the data transfer, or the number of active ports on the switches. In our case, the only cause of possible power consumption variation in the networking infrastructure could be as a results of changing the instant port utilization at certain switches, since our algorithms change the rate at which the data is pushed over the network.

Despite the increasing attention on power-aware networking, the energy consumption of specific network devices at varying level of port utilization is still not clearly defined by the vendors. Network device vendors generally list power consumption of a network device when port utilization is 50% and 100%. However, neither power consumption values for the rest of the utilization values nor a formula to derive power consumption at certain utilization level are provided. A research by Mahadevan et al. [34] claims there is a significant difference between the reported maximum power consumption by the vendors and the actual measured

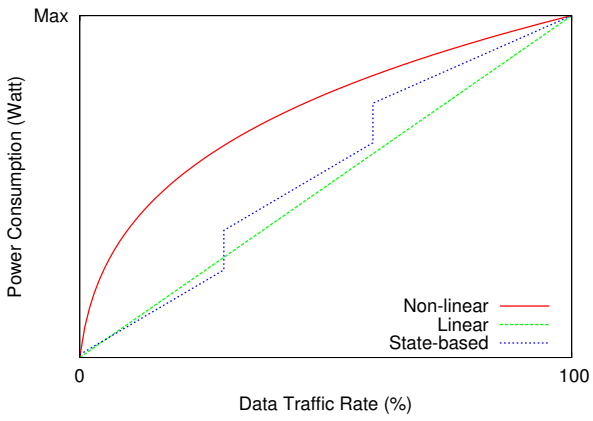

Figure 8: The relation between the data transfer rate vs. network device power consumption based on three different models.

power consumption. Hence, it is hard to find exact power consumption values for different network traffic rates unless directly measuring via internal sensors or external power meters. Taking these into consideration, we evaluated the power consumption of network devices under different utilizations through three different approaches: *(i)* non-linear approach in which the power consumption and port utilization follow a non-linear relation similar to a study by Mahadevan et al. in which they measured the power consumption of edge switches under different traffic rates [34]; *(ii)* linear approach in which the power consumption and the data traffic relation is formulated by linear equations [45]; and *(iii)* state-based approach in which the power consumption increases only at certain throughput rates [9, 41]. High level representation of these cases are depicted in Figure 8.

Network devices have both static (a.k.a idle) and dynamic

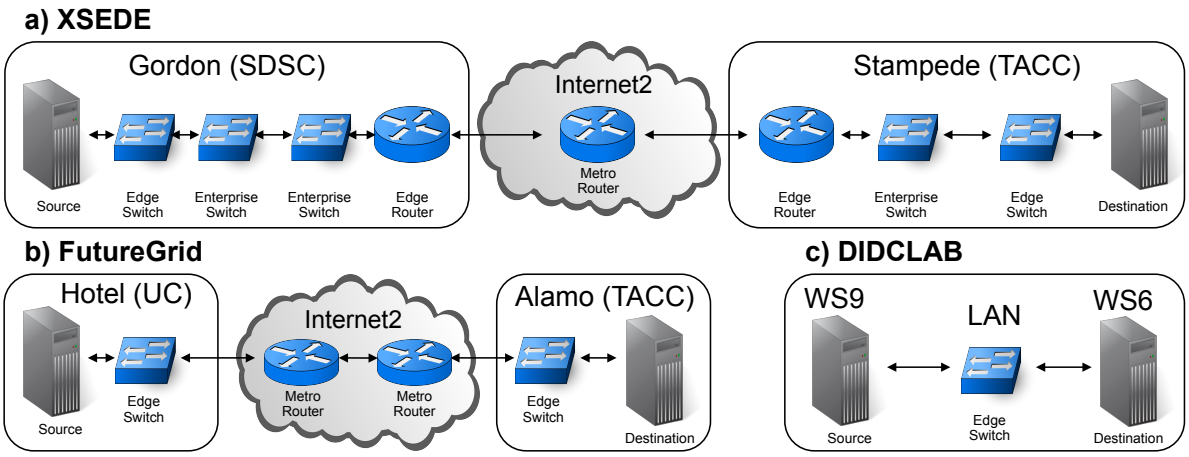

Figure 9: Overview of the networking infrastructure of XSEDE, FutureGrid and DIDCLAB testbeds respectively.

power consumption components. Static (idle) power consumption is the power consumption of a network device when there is no data flow over the device. Dynamic power consumption refers to the increase in the power usage by means of increasing the data rate on the device. Finally, the total energy consumption E_T of a network device over a period of time T is defined as:

$$E_T = P_i T + P_d T_d \tag{4}$$

where P_i and P_d refer to the idle and dynamic power consumption respectively. P_d is calculated by subtracting the idle power from the total instantaneous power, which measures the net increase in the power consumption. Finally, T_d refers to the time period during which the network device is operated at a data transfer rate d.

While the idle power consumption of the network devices will be the same for all three estimation approaches, dynamic power consumption will differ. Then, the difference of the power consumption calculation with three methods boils down to the difference in dynamic power consumption. In the non-linear case, power consumption and data transfer rate follow a sub-linear relation as presented in Figure 8 in which the data transfer rate grows faster than the power consumption. If we assume a network device consumes p power per second for data transfer rate d, then $p\frac{D}{d}$ joule energy will be consumed to transfer a dataset with total size D. On the other hand, when the data transfer rate is increased to $4d$, instantaneous power consumption becomes $2d$ (in a square root relation between the power consumption and the data rate). Since the transfer time reduces to $\frac{D}{4d}$, the total energy consumption becomes $2p\{\frac{D}{4d}\}$ which is half of the base case.

In the linear approach, let say network device consumes p power per second when data transfer rate is d. For a dataset with size D, it will take $\frac{D}{d}$ seconds to complete the transfer. Then, dynamic energy consumption will be $p\frac{D}{d}$ joule. When data transfer rate is quadrupled ($4d$) by means of protocol parameters optimization, dynamic power consumption will increase to $4p$ because of linear relationship between the data rate and the power consumption as shown in Figure8. Transferring dataset with size D at $4d$ throughput rate will take $\frac{D}{4d}$ seconds. Energy consumption, then, will be $4p\{\frac{D}{4d}\}$ which is equal to the case where data rate is d. Hence, in the linear approach, total energy consumption will be same regardless of transfer rate. The state-based approach follows same behavior with linear case as fitted regression line of state-based case is also linear.

Device	$P_p(nW)$	$P_{s-f}(pW)$
Enterprise Ethernet Switch	40	0.42
Edge Ethernet Switch	1571	14.1
Metro IP Router	1375	21.6
Edge IP Router	1707	15.3

Table 1: Per-packet power consumption coefficients of networking devices for load dependent operations.

In our study, we also quantified the power consumption of network devices using recently developed power models by Vishwanath et al. [45] which uses linear regression to model various switches and routers. They formulated the power consumption of networking devices as follows:

$$P = P_{idle} + packetCount \times (P_p + P_{s-f}) \tag{5}$$

where P_{idle} refers to the base power consumption of the device when there is no load. P_p is the per-packet processing power consumption and P_{s-f} is the per-packet store-forward power consumption. In our study, we just considered load-dependent part of this equation since the idle power consumption is independent of the algorithm type.

We calculated the energy consumption of networking devices using the coefficients listed in Table 1. Figure 9 shows the high level overview of the network configurations of the testbeds used in the experiments. As shown in the figure, there are different number of switches and routers between the source and destination nodes in each testbed. The number and type of the network devices on the transfer route affects the total energy consumption of the networking devices as well as the amount of processed data. Figure 10 presents the decomposition of the total data transfer energy consumption into its two high level components which are the end-system consumption and the network infrastructure consumption for the HTEE algorithm. At all testbeds, the end-systems consume much more power than the network infrastructure when only load-dependent energy consumption is taken into consideration. As the number of metro routers — which consume the most power as given in Table 1 — in the path increases, the proportion of the network infrastruc-

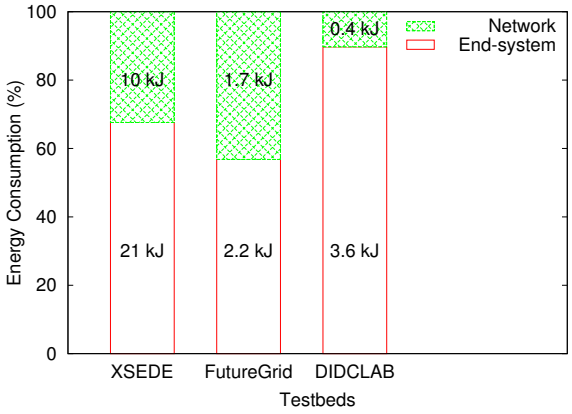

Figure 10: Power consumption of end-systems vs network devices.

ture energy consumption increases too as it can be seen in the FutureGrid case. In DIDCLAB, since there is only one switch between the source and destination servers, the network power consumption is much less than the end-systems power consumption. Due to the load-agnostic design of most of the network devices, change in the power consumption of a through data transfer is fractional. The idle power consumption constitutes as high as 70-80% [42,44] of their total power consumption.

In conclusion, our energy-aware data transfer algorithms will not only result in a decrease in energy consumption at the end systems, but there will also be additional energy consumption decrease at the network infrastructure if dynamic power consumption follows a sub-linear relation with the data transfer rate. On the other hand, if the dynamic power consumption follows a linear relation with the data transfer rate, then the total power consumption at the networking infrastructure will neither increase nor decrease. Since the protocol parameter optimization at the end-systems will save energy at the end-systems in the either case, we will still be saving energy when end-to-end system is considered.

5. RELATED WORK

To the best of our knowledge, this is the first work focusing on energy-aware data transfer algorithms for minimizing the energy consumption at the end-systems. On the other hand, there has been other studies in the literature focusing on network throughput optimization and reducing the power consumption on networking devices (i.e. ethernet cards, switches,routers etc.).

The work on network throughput optimization focuses on tuning transfer parameters such as parallelism, pipelining, concurrency and buffer size. The first attempts to improve the data transfer throughput at the application layer were made through buffer size tuning. Various dynamic and static methods were proposed to optimize the buffer size [29,37,40]. However, Lu et al. [33] showed that parallel streams can achieve a better throughput than buffer size tuning and then several others [7,26,49] proposed throughput optimization solutions by means of tuning parallel streams. Another transfer parameter used for throughput optimization was pipelining, which helped in improving the performance of transferring large number of small files [11, 18, 19]. Liu et al. [32] optimized network throughput by concurrently opening multiple transfer sessions and transferring multiple files concurrently. They proposed increasing the number of concurrent data transfer channels until the network performance degrades. Globus Online [6] offers fire-and-forget file transfers through thin clients over the Internet. It partition files based on file size and transfer each partition using partition-specific protocol parameters. However, the protocol tuning Globus Online performs is non-adaptive; it does not change depending on network conditions and transfer performance.

The work on power-aware networking focuses on saving energy at the networking devices. Gupta et al. [24] were among the earliest researchers to advocate conserving energy in networks. They suggested different techniques such as putting idle sub-components (i.e. line cards, etc.) to sleep [23], which were later extended by other researchers. S. Nedevshi et al. [35] proposed adapting the rate at which switches forward packets depending on the traffic load. IEEE Energy Efficient Ethernet task force proposed the 802.3az standards [3] for making ethernet cards more energy efficient. They defined a new power state called low power idle (LPI) that puts ethernet card to low power mode when there is no network traffic. Other related research in power-aware networking has focused on architectures with programmable switches [22] and switching layers that can incorporate different policies [30]. Barford et al. proposed power-aware network protocols for energy-efficiency in network design and routing [13].

We argue that although network-only approaches are part of the solution, the end-system power management is a key in optimizing energy efficiency of the data transfers, which has been long ignored by the community. In or previous work [48], we proposed network-aware data transfer optimization by automatically detecting performance bottlenecks and improving throughput by utilizing network and end-system parallelism. PCP [47] algorithm is proposed to find optimal values for transfer parameters such as pipelining, concurrency and parallelism. In another study [10], we presented four application level dynamic data transfer algorithms that tune the transfer parameters heuristically for reaching optimum network throughput but we did not consider the power consumption of these algorithms. In [4], we analyzed how each data transfer parameter may affect both the throughput and the power consumption in the end-to-end data transfers.

6. CONCLUSION

In this paper, we introduced three novel data transfer algorithms which consider energy efficiency during data transfers: *i)* a minimum energy algorithm which tries to minimize the energy consumption without any performance concern; *ii)* a high throughput energy-efficient algorithm which tries to maximize the throughput with low energy consumption constraints; and *iii)* an SLA-based energy-efficient algorithm which lets end-users to define their throughput requirements while keeping the power consumption at minimum levels. These algorithms provide the ability to intelligently and dynamically select the ideal set of transfer parameters for high performance, low cost (in terms of energy) data transfers. Our experimental results show that our energy-aware data transfer algorithms can achieve up to 50% energy savings with the same or higher level of data transfer throughput. Considering the vast amount of electric power consumed

during world-wide data transfer every year, these energy-aware data transfer algorithms would be of crucial importance to decrease this cost.

Acknowledgment

This project is partially supported by National Science Foundation (NSF) under award number CNS-1131889 (CAREER).

7. REFERENCES

[1] The extreme science and engineering discovery environment (xsede). `https://www.xsede.org/`.

[2] Futuregrid testbed. http://www.futuregrid.org.

[3] IEEE energy efficient ethernet standards. 10.1109/IEEESTD.2010.5621025, Oct. 2010.

[4] I. Alan, E. Arslan, and T. Kosar. Energy-aware data transfer tuning. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 626–634. IEEE, 2014.

[5] I. "Alan, E. Arslan, and T. Kosar. "energy-performance trade-offs in data transfer tuning at the end-systems". *Sustainable Computing: Informatics and Systems*, 2014.

[6] B. Allen, J. Bresnahan, L. Childers, I. Foster, G. Kandaswamy, R. Kettimuthu, J. Kordas, M. Link, S. Martin, K. Pickett, and S. Tuecke. Software as a service for data scientists. *Communications of the ACM*, 55:2:81–88, 2012.

[7] E. Altman and D. Barman. Parallel tcp sockets: Simple model, throughput and validation. In *Proceedings of IEEE INFOCOM*, 2006.

[8] G. Ananthanarayanan and R. Katz. Greening the switch. In *In Proceedings of HotPower, December 2008*.

[9] S. Antonakopoulos, S. Fortune, and L. Zhang. Power-aware routing with rate-adaptive network elements. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 1428–1432. IEEE, 2010.

[10] E. Arslan, B. Ross, and T. Kosar. Dynamic protocol tuning algorithms for high performance data transfers. In *Euro-Par 2013 Parallel Processing*, pages 725–736. Springer, 2013.

[11] J. Bresnahan, M. Link, R. Kettimuthu, D. Fraser, and I. Foster. Gridftp pipelining. In *Proceedings of TeraGrid*, 2007.

[12] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th annual international symposium on Computer architecture*, ISCA '00, pages 83–94, New York, NY, USA, 2000. ACM.

[13] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright. Power awareness in network design and routing. In *In Proceedings of IEEE INFOCOM, April, 2008*.

[14] G. Contreras and M. Martonosi. Power prediction for intel xscale® processors using performance monitoring unit events. In *ISLPED'05*, pages 221–226, 2005.

[15] DIDCLab. Data intensive distributed computing laboratoy. http://www.didclab.org.

[16] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan. Full-system power analysis and modeling for server environments. In *Proc. of Workshop on Modeling, Benchmarking, and Simulation*, 2006.

[17] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. *ACM SIGARCH Computer Architecture News*, 35(2):13–23, 2007.

[18] K. Farkas, P. Huang, B. Krishnamurthy, Y. Zhang, and J. Padhye. Impact of tcp variants on http performance. *Proceedings of High Speed Networking*, 2, 2002.

[19] N. Freed. SMTP service extension for command pipelining. http://tools.ietf.org/html/rfc2920.

[20] E. Goma, M. C. A. L. Toledo, N. Laoutaris, D. Kosti, P. Rodriguez, R. Stanojev, and P. Y. Valentin. Insomnia in the access or how to curb access network related energy consumption. In *In Proceedings of ACM SIGCOMM 2011*.

[21] A. Greenberg, J. Hamilton, D. Maltz, and P. Patel. The cost of a cloud: Research problems in data center networks. In *In ACM SIGCOMM CCR, January 2009*.

[22] A. Greenberg, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Towards a next generation data center architecture: Scalability and commoditization. In *In ACM PRESTO, pages 57ÃŘ62, 2008*.

[23] M. Gupta and S. Singh. Energy conservation with low power modes in ethernet lan environments. In *IEEE INFOCOM (MiniSymposium) 2007*.

[24] M. Gupta and S. Singh. Greening of the internet. In *ACM SIGCOMM, pages 19ÃŘ26, 2003*.

[25] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, and M. Kandemir. Using complete machine simulation for software power estimation: The softwatt approach. In *Prpc. of 8th High-Performance Computer Architecture Symp.*, pages 141–150, 2002.

[26] T. J. Hacker, B. D. Noble, and B. D. Atley. Adaptive data block scheduling for parallel streams. In *Proceedings of HPDC '05*, pages 265–275. ACM/IEEE, July 2005.

[27] K. Hasebe, T. Niwa, A. Sugiki, and K. Kato. Power-saving in large-scale storage systems with data migration. In *IEEE CloudCom 2010*.

[28] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. Elastictree: Saving energy in data center networks. In *Proceedings of NSDI 2010*.

[29] M. Jain, R. S. Prasad, and C. Dovrolis. The tcp bandwidth-delay product revisited: network buffering, cross traffic, and socket buffer auto-sizing. 2003.

[30] D. A. Joseph, A. Tavakoli, and I. Stoica. A policy-aware switching layer for data centers. In *SIGCOMM CCR 38(4):51ÃŘ62, 2008*.

[31] R. Koller, A. Verma, and A. Neogi. Wattapp: an application aware power meter for shared data centers. In *Proceedings of the 7th international conference on Autonomic computing*, pages 31–40. ACM, 2010.

[32] W. Liu, B. Tieman, R. Kettimuthu, and I. Foster. A data transfer framework for large-scale science experiments. In *Proceedings of DIDC Workshop*, 2010.

[33] D. Lu, Y. Qiao, P. A. Dinda, and F. E. Bustamante.

Modeling and taming parallel tcp on the wide area network. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 68b–68b. IEEE, 2005.

[34] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan. A power benchmarking framework for network devices. In *In Proceedings of IFIP Networking, May 2009*.

[35] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wether-all. Reducing network energy consumption via rate-adaptation and sleeping. In *Proceedings Of NSDI, April 2008*.

[36] U. of Minnesota. Minnesota internet traffic studies (mints), 2012.

[37] R. S. Prasad, M. Jain, and C. Dovrolis. Socket buffer auto-sizing for high-performance data transfers. *Journal of GRID computing*, 1(4):361–376, 2003.

[38] F. Rawson and I. Austin. Mempower: A simple memory power analysis tool set. *IBM Austin Research Laboratory*, 2004.

[39] S. Rivoire, P. Ranganathan, and C. Kozyrakis. A comparison of high-level full-system power models. *HotPower*, 8:3–3, 2008.

[40] J. Semke, J. Mahdavi, and M. Mathis. Automatic tcp buffer tuning. *ACM SIGCOMM Computer Communication Review*, 28(4):315–323, 1998.

[41] L. Shang, L.-S. Peh, and N. K. Jha. Dynamic voltage scaling with links for power optimization of interconnection networks. In *High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings. The Ninth International Symposium on*, pages 91–102. IEEE, 2003.

[42] Y. Shang, D. Li, and M. Xu. A comparison study of energy proportionality of data center network architectures. In *Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on*, pages 1–7. IEEE, 2012.

[43] C. Systems. Visual networking index: Forecast and methodology, 20130̣2018, June 2014.

[44] W. Van Heddeghem, F. Idzikowski, E. Le Rouzic, J. Y. Mazeas, H. Poignant, S. Salaun, B. Lannoo, and D. Colle. Evaluation of power rating of core network equipment in practical deployments. In *Online Conference on Green Communications (GreenCom), 2012 IEEE*, pages 126–132. IEEE, 2012.

[45] A. Vishwanath, K. Hinton, R. Ayre, and R. Tucker. Modelling energy consumption in high-capacity routers and switches. 2014.

[46] S. V. Vrbsky, M. Galloway, R. Carr, R. Nori, and D. Grubic. Decreasing power consumption with energy efficient data aware strategies. *FGCS*, 29(5):1152–1163, 2013.

[47] E. Yildirim, J. Kim, and T. Kosar. How gridftp pipelining, parallelism and concurrency work: A guide for optimizing large dataset transfers. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 506–515. IEEE, 2012.

[48] E. Yildirim and T. Kosar. Network-aware end-to-end data throughput optimization. In *Proceedings of Network-Aware Data Management Workshop (NDM 2011)*.

[49] E. Yildirim, D. Yin, and T. Kosar. Balancing tcp buffer vs parallel streams in application level throughput optimization. In *Proceedings of DADC Workshop*, 2009.

[50] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R. Y. Wang. Modeling hard-disk power consumption. In *FAST 2003*.