CapNet: A Real-Time Wireless Management Network for Data Center Power Capping

Abusayeed Saifullah*, Sriram Sankar†, Jie Liu‡, Chenyang Lu*, Ranveer Chandra‡, and Bodhi Priyantha‡
Washington University in St louis, St louis, Missouri 63130, USA*
Microsoft Corporation, Redmond, Washington 98052, USA†
Microsoft Research, Redmond, Washington 98052, USA‡

***Abstract*—Data center management (DCM) is increasingly becoming a significant challenge for enterprises hosting large scale online and cloud services. Machines need to be monitored, and the scale of operations mandates an automated management with high reliability and real-time performance. Existing wired networking solutions for DCM come with high cost. In this paper, we propose a wireless sensor network as a cost-effective networking solution for DCM while satisfying the reliability and latency performance requirements of DCM. We have developed CapNet, a real-time wireless sensor network for power capping, a time-critical DCM function for power management in a cluster of servers. CapNet employs an efficient event-driven protocol that triggers data collection only upon the detection of a potential power capping event. We deploy and evaluate CapNet in a data center. Using server power traces, our experimental results on a cluster of 480 servers inside the data center show that CapNet can meet the real-time requirements of power capping. CapNet demonstrates the feasibility and efficacy of wireless sensor networks for time-critical DCM applications.**

I. Introduction

The continuous, low-cost, and efficient operation of a datacenter heavily depends on its management network and system. A typical data center management (DCM) system handles physical layer functionality such as powering on/off a server, motherboard sensor telemetry, cooling management, and power management. Higher level management capabilities such as system re-imaging, network configuration, (virtual) machine assignments, and server health monitoring [1], [2] depend on DCM to work correctly. DCM is expected to function even when the servers do not have a working OS or the data network is not configured correctly [3].

Today's DCM is typically designed in parallel to the production data network (in other words, *out of band*), with a combination of Ethernet and *serial connections* for increased redundancy. There is a cluster controller for a rack or a group of racks, which are connected through Ethernet to a central management server. Within the clusters, each server has a motherboard microcontroller (BMC - Baseboard Management Controller) that is connected to the cluster controller via point-to-point serial connections. For redundancy reasons, every server is typically connected to two independent controllers on two different fault domains, so there is at least one way to reach the server under any single point of failure. Unfortunately, this architecture does not scale. The overall cost of management network increases super-linearly with the number of servers in a data center. At the same time, massive cabling across racks increases the chance for human errors and prolongs the server deployment latency.

This paper presents a different approach to data center management network at the rack granularity, by replacing serial cable connections with low cost wireless links. Low power wireless sensor network technology such as IEEE 802.15.4 has intrinsic advantages in this application.

- *Cost:* Low-power radios (i.e., IEEE 802.15.4) are cheaper individually than wired alternatives and the cost scales linearly with the number of servers.
- *Embedded*: These radios can be physically small and be integrated onto motherboard to save precious rack space.
- *Reconfigurability:* Wireless sensor networks can be self-configuring and self-repairing with the broadcast media to prevent human cabling error.
- *Low power:* With a small on-board battery, the DCM based on wireless can continue to function on batteries providing monitoring capabilities even when the rack experiences a power supply failure.

However, whether a wireless DCM can meet the high reliability requirement for data center operation is not obvious for several reasons. The amount of sheet metals, electronics, and cables may completely shield RF signal propagation within racks. Furthermore, although typical traffic on a DCM is low, emergency situations might need to be handled in real time, which could require the design of new protocols.

Power capping is an example of emergency event that imposes real-time requirements. Today, data center operators commonly oversubscribe the power infrastructure by installing more servers to an electric circuit than it is rated. The rationale is that servers seldom reach their peak at the same time. By over-subscription, the same data center infrastructure can host more servers than otherwise. In the rare event when the aggregate power consumption of all servers exceeds the circuit's power capacity, some servers must be slowed down (i.e. power capped), through dynamic frequency and voltage scaling (DVFS) or CPU throttling, to prevent the circuit breaker from tripping. Every magnitude of oversubscription is associated with a trip time which is a *deadline* by which power capping must be performed to avoid circuit breaker tripping.

This paper studies the feasibility and advantages of using low-power wireless for DCM. In two data centers, we empirically evaluate IEEE 802.15.4 link qualities in server racks to

1052-8725/14 $31.00 © 2014 IEEE
DOI 10.1109/RTSS.2014.35

show that the overall packet reception rate is high. We further dive into the power capping scenario and design **CapNet**, a wireless **Net**work for power **Cap**ping, that employs an event-driven real-time control protocol for power capping over wireless DCM. The protocol uses distributed event detection to reduce the overhead of regularly polling all nodes in the network. Hence, the network throughput can be used by other management tasks when there is no emergency. When a potential power surge is detected, the controller uses a sliding window and collision avoidance approach to gather power measurements from all servers, and then issues power capping commands to a subset of them. We deployed and evaluated CapNet in a data center. Using server power traces, our experimental results on a cluster of 480 servers in the data center show that CapNet can meet the real-time requirements of power capping. It demonstrates the feasibility and efficacy in power capping like wired DCM with a fraction of the cost..

II. The Case for Wireless DCM (CapNet)

Typical wired DCM solutions in data centers scale poorly with increase in number of servers. The serial-line based point-to-point topology incurs additional costs as we connect more of them together. Here, we compare the costs of the wired DCM to our proposed wireless based solution (CapNet) by considering the cost of the management network, and by measuring the quality of in-rack wireless links.

A. Cost Comparison with Wired DCM

To compare the hardware cost, we consider the cost of the DiGi switches ($3917/48port [4]), controller cost (approx. $500/rack [5]), cable cost ($2/cable [6]) and additional management network switches ($3000/48port on average [7]). We do not include the labor or management costs for cabling for simplicity of costing model, but note that these costs are also significant with wired DCMs. We assume that there are 48 servers per rack, and there can be up to 100,000 servers that need to be managed, which are typical for large data centers. For the wireless DCM based CapNet solution, we assume IEEE 802.15.4 (ZigBee) technologies for its low cost benefits. The cost of network switches at the top level layer stays, but the cost of DiGi can be significantly reduced. We assume $10 per wireless controller, which is essentially an Ethernet to ZigBee relay. For wireless receivers on the motherboard, we assume $5 per server for the RF chip and antenna as the motherboard controller is already in place [8].

# of servers	Wired-N	Wired-2N	CapNet-N	CapNet-2N
10	7450	14900	3060	6070
100	16560	33120	3530	6560
1000	98820	197640	8210	11420
10000	980780	1961560	79090	108180
100000	9772280	19544560	781840	1063680

TABLE I
System cost (in US Dollar) comparison and scalability

We develop a simple cost model based on these individual costs and compute the total devices needed for implementing management over number of servers ranging from 10 to 100,000 (in order to capture how cost scales with the number of servers). We consider solutions across two dimensions 1) Wired vs Wireless, and 2) N-redundant vs 2N-redundant (A 2N redundant system consists of two independent switches, DiGis and paths through the management system). Table I shows the cost comparison across these solutions. We see that a wired N-redundant DCM solution (Wired-N) for 100,000 servers is $12.5\times$ the cost of a wireless N-redundant DCM solution (CapNet-N). If we increase the redundancy of the management network to 2N, the cost of a wired solution (between Wired-2N and Wired-N) doubles. In contrast, the cost of a wireless solution increases only by 36% (due to 2N controllers and 2N switches at the top level). The resulting cost of Wired-2N is $18.4\times$ that of CapNet-2N. Given the significant cost difference between wired DCM and CapNet, we next explore whether wireless is feasible for communication within racks.

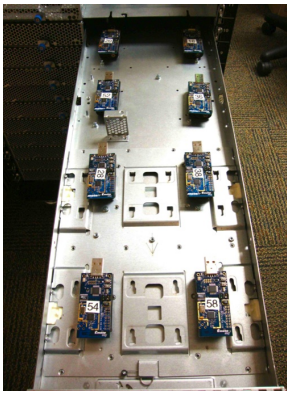


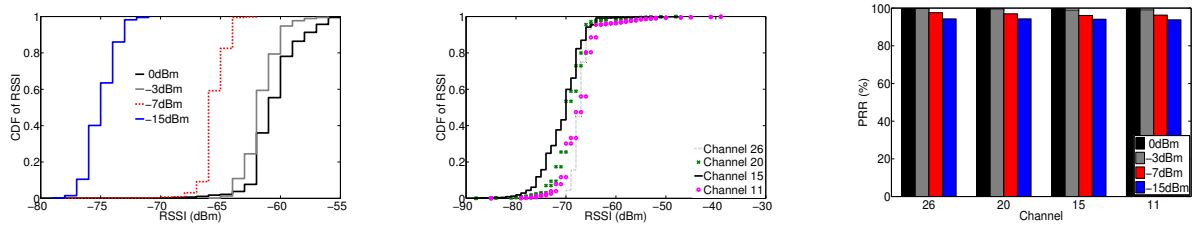
Fig. 1. Mote placed in bottom sled

B. Choice of Wireless - IEEE 802.15.4

We are particularly interested in low bandwidth wireless like IEEE 802.15.4 instead of IEEE 802.11 for a number of reasons. First, the payload size for data center management is small and hence a ZigBee (IEEE 802.15.4) network bandwidth is sufficient for control plane traffic. Second, in WiFi (IEEE 802.11) there is a limit on how many nodes an access point can support in the infrastructure mode since it has to maintain an IP stack for every connection, and this impacts scalability in a dense deployment. Third, to support management features, the data center management system should still work when the rack is unpowered. A small backup battery can power ZigBee longer at much higher energy efficiency. Finally, ZigBee communication stack is simpler than WiFi so the motherboard (BMC controller) microcontroller can remain simple. Although we do not rule out other wireless technologies, we chose to prototype with ZigBee in this paper.

C. Radio Environment inside Racks

We did not find any previous study that evaluated the signal strength within the racks through servers and sheet metal. The sheet metals inside the enclosure are known to weaken radio signal, giving a harsh environment for radio propagation inside racks. RACNet [9] studied wireless characteristics in data centers, but only across racks when all radios are mounted



(a) RSSI when Tx power varies (channel 26) (b) RSSI on various channels (Tx power -3dBm) (c) PRR at a receiver

Fig. 2. Downward signal strength and PRR in bottom sled

at the top of the rack. Therefore, we first perform an in-depth 802.15.4 link layer measurement study based on in-rack radio propagation inside a data center of Microsoft Corporation.

Setup. The data center used for measurement study has racks that consist of multiple chassis in which servers are housed. A chassis is organized into two columns of sleds. In all experiments, one TelosB mote is placed on top of the rack (ToR), inside the rack enclosure. The other motes are placed in different places in a chassis in different experiments. Figure 1 shows the placement of 8 motes inside a bottom sled (which is open in the figure but was closed during the experiment). While measuring the downward link quality, the node on ToR is the sender and the nodes in the chassis receive. Then we reverse the sender and the receiver to measure the upward link quality. In each setup, the sender transmits packets at 4Hz. The payload size of each packet is 29 bytes. Through a week-long test capturing the long-term variability of links, we collected signal strengths and packet reception rate (PRR).

Results. Figure 2(a) shows the cumulative distribution function (CDF) of Received Signal Strength Indicator (RSSI) values at a receiver inside the bottom sled for 1000 transmissions from the node on ToR for different transmission (Tx) power using IEEE 802.15.4 channel 26. For -7dBm or higher Tx power, RSSI is greater than -70dBm in 100% cases. RSSI values in ZigBee receivers are in the range $[-100, 0]$. Previous study [10] on ZigBee shows that when the RSSI is above -87dBm (approx.), PRR is at least 85%. As a result, we see that signal strength at the receiver in bottom sled is quite strong. Figure 2(b) shows the CDF of RSSI values at the same receiver for 1000 transmissions from the node on ToR on different channels at Tx power of -3dBm. Both figures indicate a strong signal strength, and in each experiment the PRR was at least 94% (Figure 2(c)). We observed similar results in all other setups of the measurement study, and omit those results.

The measurement study reveals that low-power wireless, such as IEEE 802.15.4, is viable for communication within data center racks and can be reliable for telemetry purpose. We now focus on the power capping scenario and CapNet design for real-time power capping over wireless DCM.

III. CapNet Design Overview

Power infrastructure bears huge capital investment for a data center, up to 40% of the total cost of a large data center that can cost hundreds of millions of US Dollars [11]. Hence, it is desirable to use the provisioned infrastructure to

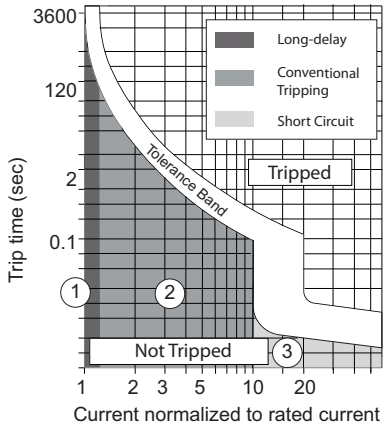


Fig. 3. The trip curve of Rockwell Allen-Bradley 1489-A circuit breaker at 40°C [16]. X-axis is oversubscription magnitude. Y-axis is trip time.

its maximum rated capacity. The capacity of a branch circuit is provisioned during design time, based on upstream transformer capacity during normal operation or UPS/Generator capacity when running on backup power. To improve data center utilization, a common practice in enterprise data centers is to do *oversubscription* [12]–[15]. This method allocates servers in a circuit exceeding the rated capacity (i.e. *cap*), since not all servers reach their maximum power consumption at the same time. Hence, there is a circuit breaker (CB) that trips to protect expensive equipment. The peak power consumption above the cap has a specified time limit, called a *trip time*, depending on the magnitude of over-subscription (as shown in Figure 3 for Rockwell Allen-Bradley 1489-A circuit breaker). If the over-subscription continues for longer than the trip time, the CB will trip and cause undesired server shutdowns and power outages disrupting data center operation. *Power capping* is the mechanism to bring the aggregate power consumption back to the cap. An overload condition under practical current draw trips the CB on a time scale from several hundred milliseconds to hours, depending on the magnitude of the overload [16]. These trip times are the *deadlines* for the corresponding oversubscription magnitudes within which power capping must be done to prevent CB tripping to avoid power loss or damage to expensive equipment.

A. The Power Capping Problem

To enable power capping for a rack or cluster, a *power capping manager* (also called *controller*) collects all servers' power consumption and determines the cluster-level aggregate

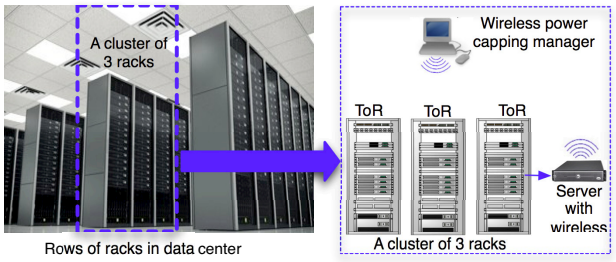


Fig. 4. Wireless DCM architecture

power consumption. If the aggregate consumption is over the cap, the manager generates control messages asking a subset of the servers to reduce their power consumptions through CPU frequency modulation (and voltage if using DVFS) or utilization throttling. The application level quality of service may require different servers to be capped at different levels. So the central controller needs all individual server readings. In some graceful throttling policies, the control messages are delivered by the BMC Controller to the host OS or VMs, which introduce additional latency due to OS stack [14], [17]. To avoid abrupt changes to application performance, the controller may change the power consumption incrementally and require multiple iterations of the feedback control loop before the cluster settles down to below the power cap [14], [18]. These control policies have been studied extensively by previous work and are out of the scope of this paper.

B. Power Capping over Wireless DCM

Servers in a data center are stacked and organized into racks. One or more racks can comprise a power management unit, called a *cluster*. Figure 4 shows the wireless DCM architecture inside a data center. All servers in a cluster incorporate a wireless transceiver that connects to the BMC microcontroller. Each server is capable of measuring its own power consumption. A cluster power capping manager can either directly measure the total power consumption using a power meter, or, to achieve fine-grained power control, aggregates the power consumption from individual servers. We focus on the second case due to its flexibility. When the aggregate power consumption approaches the circuit capacity, the manager issues capping commands over wireless links to individual servers. The main difference compared to a wired DCM is the broadcast wireless media and challenge of scheduling communication to meet the real-time demands.

To reduce extra coordination and to enable spatial spectrum reuse, we assume a single IEEE 802.15.4 channel for communication inside a cluster. Using multiple channels, multiple clusters can run in parallel. Channel allocation can be done using existing protocols that minimize inter-cluster interference (e.g. [19]), and is not the focus of our paper. For protocol design, we focus on a single cluster of n servers.

C. A Naive Periodic Protocol

A naive approach for a fine-grained power capping policy is to always monitor the servers by periodically collecting the power consumption readings from individual servers. The manager periodically computes the aggregate power. Whenever the aggregate power exceeds the cap, it generates a control message. Upon finishing the aggregation and control in η iterations, it resumes the periodic aggregation again.

D. Event-Driven CapNet

Oversubscribing data centers may provision for the 95-th (or more) percentile of the peak power, and require capping for 5% (or less) of the time, which may be an acceptable hit on performance in relation to cost savings [17]. Thus power capping is a rare event, and the naive periodic protocol is an overkill as it saturates the wireless media by always preparing for the worst case. Other delay-tolerant telemetry messages cannot get enough network resources. An ideal wireless protocol should generate significant traffic only when a significant power surge occurs. Therefore, CapNet employs an event-driven policy that is designed to trigger power capping control operation only when a potential power capping event is predicted. Due to the rareness and emergency nature of power surge, the network can suspend other activities to handle power capping. It provides real-time performance and a sustainable degree of reliability without consuming much network resource. The details of the protocol is explained in the next section.

IV. Power Capping Protocol

We design a distributed event detection policy, where we assign local caps to each individual server from their global (cluster-level) cap. When a server observes a local power surge based on its own power reading, it can trigger the collection of the power consumption of all the servers to detect a potential surge in the aggregate power consumption of cluster. If a cluster-level power surge is detected, the system initiates a power capping action. As many servers can simultaneously exceed their local caps, a standard CSMA/CA protocol can suffer from significant packet loss due to excessive contention and collisions. Similarly, a slot stealing TDMA (Time Division Multiple Access) protocol such as Z-MAC [20] would suffer from the same problem as those servers will try to steal slot simultaneously. Furthermore, pure TDMA based protocols do not fit well for our problem since they need to have a predefined communication schedule for all nodes. Finally, as power aggregate consumption can be quite dynamic, it may be infeasible to predict an upcoming power peak based on historical readings. This observation leads us to avoid a predictive protocol that proactively schedule data collection based on historical power readings.

While a global detection is possible by just monitoring at the branch circuit level, say using a power meter, it cannot support fine-grained and flexible power capping policies such as those based on individual server-priority or reducing powers of individual servers based on their power consumptions. Also, a centralized measurement introduces a single point of failure. That is, if the power meter fails, power oversubscription will fail also. In contrast, our distributed approach is more resilient to failure. If individual measurement fails, the system can

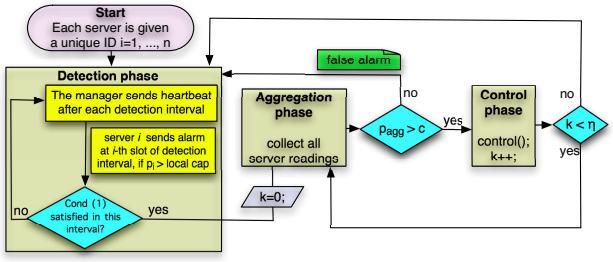


Fig. 5. CapNet's event-driven protocol flow diagram

always assume a maximum power consumption at that server and keep the whole cluster going.

The event-driven protocol runs in 3 phases as illustrated in Figure 5: **detection**, **aggregation**, and **control**. The event detection phase generates alarms based on local power surges. Upon detecting a potential event, CapNet runs the second phase which invokes a power aggregation protocol. False detection may happen when some servers generate alarms exceeding the local caps, but the aggregate value is still under the cap. This is corrected in the aggregation phase, where the controller determines the aggregate power consumption. The impact of a false positive case is that the system runs into the aggregation phase which incurs additional wireless traffic. The control phase is executed only if the alarms are true.

We normalize each server's power consumption value between 0 and 1 by dividing its instantaneous power consumption by the maximum power consumption of an individual server. This normalized power consumption value of server i is denoted by p_i, where $0 \leq p_i \leq 1$, and is used in this paper as a server's power consumption. The cap of a cluster of n servers is denoted by c, and the total power consumption of n servers is considered as the aggregate power consumption and is denoted by p_{agg}.

Assigning local cap. If $p_{agg} > c$, a necessary condition is that some servers' (at least one) individual power consumption values locally exceed the value $\frac{c}{n}$. Therefore, a possible way is to assign $\frac{c}{n}$ as each server's local cap. However, there can be situations where only one server exceeds $\frac{c}{n}$ while all other servers are under $\frac{c}{n}$, thereby triggering an aggregation phase upon a single server's alarm. As a result, this policy will generate many false alarms. Therefore, to suppress false alarms, we assign a slightly smaller local cap, and consider alarms from multiple servers before aggregation phase. Thus we use a value $0 < \alpha \leq 1$ close to 1 and assign $\frac{\alpha c}{n}$ as the local cap for each server. A server i reports alarm if $p_i > \frac{\alpha c}{n}$.

Each server is assigned a unique ID i, where $i = 1, 2, \cdots, n$. The manager broadcasts a *heartbeat* packet at every h time units called *detection interval*. The detection interval of length h is slotted among n slots, with each slot length being $\left\lfloor \frac{h}{n} \right\rfloor$. The value of h is selected in a way so that a slot is long enough to accommodate one transmission and its acknowledgement. After receiving the heartbeat message, the server clocks are synchronized.

A. Detection Phase

Each node i, $1 \leq i \leq n$, takes its sample (i.e., power consumption value p_i) at the i-th slot in the detection phase. If its reading is over the cap i.e. $p_i > \frac{\alpha c}{n}$, it generates an alarm and sends the reading (p_i) to the manager as an acknowledgement of the heartbeat message. Otherwise, it ignores the heartbeat message, and does nothing. If an alarm is received at the s-th slot, the manager determines, based on whether the network is reliable or not, whether an aggregation phase has to be started. Let the servers who have sent alarms in the current detection window so far be denoted by A.

Reliable Network. Let an alarm be generated in the s-th slot of a detection interval. Considering a reliable network we can consider that no server message was lost. Therefore, each of the other $s - |A|$ servers among the first s servers has a power consumption reading of at most $\frac{\alpha c}{n}$ as it has not generated an alarm. Each of the remaining $n - s$ servers can have a power consumption value of at most 1. Thus based on the alarm at s-th slot, the manager can estimate an aggregate power of $\sum_{j \in A} p_j + (s - |A|)\frac{\alpha c}{n} + (n - s)$. Hence, if an alarm is generated at the s-th slot, the manager will start aggregation phase if

$$\sum_{j \in A} p_j + (s - |A|)\frac{\alpha c}{n} + (n - s) > c \tag{1}$$

Unreliable Network. Now we consider a scenario where some server alarms were lost. As a result, if an alarm is generated in the s-th slot of a detection window, each of the other $s - |A|$ servers among the first s servers may have a power consumption reading of at most 1 as its alarm is assumed to be lost. Therefore, each of the $n - |A|$ servers can have power consumption of at most 1, making an estimated aggregate power of $\sum_{j \in A} p_j + (n - |A|)$. Thus, if an alarm is generated in the s-th slot, the manager will start aggregation phase if

$$\sum_{j \in A} p_j + (n - |A|) > c \tag{2}$$

If there are no alarms in the detection phase or all alarm messages were lost due to transmission failure, the controller resumes the next detection phase (to detect the surges again using the same mechanism) when the current phase is over.

B. Aggregation Phase

To minimize aggregation latency, CapNet adopts a sliding window based protocol to determine aggregate power consumption denoted by p_{agg}. The controller uses a window of size ω. At anytime, it selects ω servers (or, if there are fewer than ω servers whose readings are not yet collected, then selects all of them) in a round-robin fashion who will send their readings consecutively in the next window. These ω server IDs are ordered in a message. In the beginning of the window, the controller broadcasts this message, and starts a timer of length $\tau_d + \omega\tau_u$ after the broadcast, where τ_d denotes the maximum downward communication time (i.e., the maximum time required for a controller's packet to be delivered to a server) and τ_u denotes the maximum upward

communication time (server to controller). Upon receiving the broadcast message, any server whose ID is in order i, $1 \leq i \leq \omega$, in the message transmits its reading after $(i-1)\tau_u$ time. Other servers ignore the message. If the timer fires or packets from all ω nodes are received, the controller creates the next window of ω servers that are yet to be scheduled or whose packets were missed (in the previous window). A server is scheduled in at most γ consecutive windows to handle transmission failures, where γ is the worst-case ETX (expected number of transmissions for a successful delivery) in the network. The procedure continues until all server readings are collected or there is no server that was retried γ times.

C. Control Phase

Upon finishing the aggregation phase, if $p_{agg} > c$, where c is the cap, it starts the control phase. The control phase generates a capping control command using a control algorithm, and then the controller broadcasts the message requesting a subset of the servers to be capped. To handle broadcast failures, it repeats the broadcast γ times (since the broadcast is not acknowledged). The servers react to the capping messages by DVFS or CPU throttling that incurs an operating system (OS) level latency as well as a hardware-induced delay [17]. If the control algorithm requires η-iteration, then after the capping control command is executed in the first round, the controller will again run the aggregation phase to reconfirm that capping was done correctly. The procedure iterates up to $(\eta-1)$ more iterations. Upon finishing the control, or after the aggregation phase upon a false alarm, it resumes the detection phase.

D. Latency Analysis

Given the time criticality for power capping, it is important for CapNet to achieve bounded latency. Here, we provide an analytical latency upper bound for CapNet's power capping latency that consists of detection phase latency, aggregation latency, OS level latency, and hardware latency. In practice, the actual latency is usually lower than the bound. The analysis can be used by system administrators to configure the cluster to ensure power capping meets the timing constraints.

Aggregation latency. For n servers in the cluster, the total aggregation delay L_{agg} under no transmission failure can be upper bounded as follows. Note that each window of ω transmissions can take at most $(\tau_u\omega+\tau_d)$ time units. There can be at most $\lfloor \frac{n}{\omega} \rfloor$ windows where in each window ω servers transmit. Then, the last window will take only $(n \bmod \omega + \tau_d)$ time to accommodate the remaining $(n \bmod \omega)$ servers. Hence,

$$L_{agg} \leq (\tau_u\omega + \tau_d)\left\lfloor \frac{n}{\omega} \right\rfloor + (n \bmod \omega + \tau_d)$$

Considering γ as the worst-case ETX in the network,

$$L_{agg} \leq \left((\tau_u\omega + \tau_d)\left\lfloor \frac{n}{\omega} \right\rfloor + (n \bmod \omega + \tau_d)\right)\gamma \qquad (3)$$

The above value is only an analytical upper bound, and in practice the latency can be a lot shorter.

Latency in detection phase. The time spent in the detection phase is denoted by L_{det}. In a detection window the protocol never will need the readings from the last $\lfloor c \rfloor - 1$ servers as an aggregation phase must start before this should a power capping needed (assuming that not all alarms were lost). Therefore the alarms generated within the first $(n - \lfloor c \rfloor + 1)$ slots must trigger aggregation phase. Hence,

$$L_{det} \leq \left\lfloor \frac{h}{n} \right\rfloor (n - \lfloor c \rfloor + 1) \qquad (4)$$

Total power capping latency. To handle a power capping event, a detection phase and an aggregation phase are followed by a control message that is broadcasted γ times and takes $\tau_d\gamma$ time. In addition, once the control message reaches a server, there is an operating system level latency, and after processor frequency changes, there is a hardware-induced delay. Let the OS level latency and the hardware level latency in the worst case be denoted by L_{os} and L_{hw}, respectively. Thus, the total power capping latency in one iteration, denoted by L_{cap}, is bounded as

$$L_{cap} \leq L_{det} + L_{agg} + \tau_d\gamma + L_{os} + L_{hw}$$

A η-iteration control means that once power capping command is executed, the controller will again need to collect all readings from servers, and reconfirm that capping was done correctly in $(\eta-1)$ more iterations. Therefore, for η-iteration control, the above bound is given by

$$L_{cap} \leq L_{det} + (L_{agg} + \tau_c\gamma + L_{os} + L_{hw})\eta \qquad (5)$$

V. Experiments

In this section, we present the experimental results of CapNet. The objective is to evaluate the effectiveness and robustness of CapNet in meeting the real-time requirements of power capping under data center realistic settings.

A. Implementation

The wireless communication side of CapNet is implemented in NesC on TinyOS [21] platform. To comply with realistic data center practices, we have implemented the control management at the power capping manager side. In our current implementation, wireless devices are plugged to the servers directly through their serial interface.

B. Workload Traces

We use workload demand traces from multiple geo-distributed data centers run by a global corporation over a period of six consecutive months. Each cluster consists of several hundreds of servers that span multiple chassis and racks. These clusters run a variety of workloads including Web-Search, Email, Map-Reduce jobs, and cloud applications, catering to millions of users around the world. Each cluster uses homogeneous hardware, though there could be differences across clusters. We use workload traces of 2 representative server clusters: C1 and C2. In both clusters each individual server has CPU utilization data of 6 consecutive months in every 2 minutes interval. While we recognize that full system power is composed of storage, memory and other components, in addition to CPUs, several previous works show that a server's

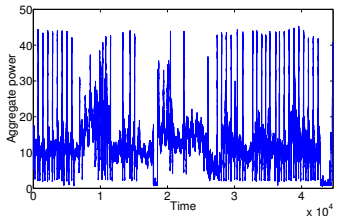
utilization is roughly linear to its power consumption [22]–[25]. Hence, we use server's CPU utilization as a proxy for power consumption in all experiments.

C. Experimental Setup

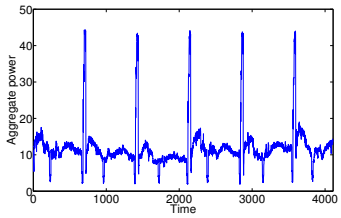
1) Experimental Methodology: We experiment with CapNet using TelosB motes for wireless communication. First we deployed 81 motes (1 for manager, 80 for servers) in Microsoft's data center in Redmond, WA. When we experiment with more than 80 servers to test scalability, one mote emulates multiple servers and communicates for them. For example, when we experiment for 480 servers, mote 1 works for first 6 servers, then mote 2 works for next 6 servers, and so on. We place all 80 motes in racks. The manager node is placed on ToR and connected through its serial interface to a PC that works as the manager. No mote in the rack has direct line of sight with the manager. Using the workload demand traces, CapNet is run in a trace-driven fashion. For every server the reading at a time stamp sent from its corresponding wireless mote is taken from these traces at the same time stamp. While the data traces are of 6-month long, our experiment does not run for actual 6-month. When we take a subset of those traces, say for 4 weeks, the protocols skip the long time intervals where there is no peak. For example, when we know (looking ahead into the traces) there is no peak between time t_1 and t_2, the protocols skip the times between t_1 and t_2. Thus our experiments finish in several days instead of 4 weeks.

2) Oversubscription and Trip Time: We use the trip times from Figure 3 as the basis, in order to determine the different caps required in various experiments. X-axis shows the ratio of current draw to the rated current and is the *magnitude of oversubscription*. Y-axis shows the corresponding trip time. The trip curve is shown as a tolerance band. The upper curve of the band indicates *upper bound (UB) trip times* above which is the tripped area, meaning that the circuit breaker will trip if the duration of the current is longer than the UB trip time. The lower curve of the band indicates *lower bound (LB) trip times* under which is the not-tripped area. This band between 2 curves is the area where it is non-deterministic if the circuit breaker will trip. LB trip time is a very conservative bound. In our experiments we use both LB and UB of conventional trip times to verify the robustness of CapNet.

3) CapNet Parameters: For all experiments, we use channel 26 and Tx power of -3dBm. The payload size of each packet sent from the server nodes is 8 bytes, which is enough for sending power consumption reading. The maximum payload size of each packet sent from the manager is 29 bytes, the maximum default size in IEEE 802.15.4 radio stack for TelosB motes. This payload size is set large to contain the schedules as well as control information. For aggregation protocol, window size ω is set to 8. A larger window size can reduce aggregation latency, but requires the payload size of the manager's message to be larger (since the packet contains ω node IDs indicating the schedule for next window). In the aggregation protocol both τ_d and τ_u were set to 25ms. The manager sets its timeout using these values. These values are relatively larger compared



(a) Aggregate power (2 Months)



(b) Aggregate power (1st 6 days zoomed-in)

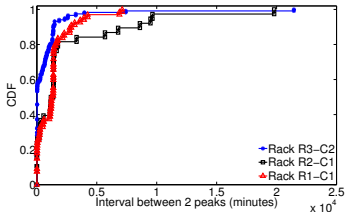
Fig. 6. 60 Servers on Rack R1 in Cluster C1

to the maximum transmission time between two wireless devices. The time required for communication between two wireless devices is in the range of several milliseconds. But in our design the manager node is connected through its serial interface to a PC. The TelosB's serial interface does not always incur a fixed latency for communication between PC and the mote through serial. Upon experimenting and observing a wide variation of this time, we have set τ_d and τ_u to 25ms.

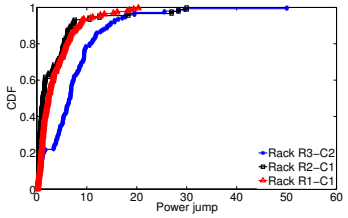
4) Control Emulation: In our experiments, we emulate the final control action since we use workload traces. We assume that one packet is enough to contain the entire control message. To handle control broadcast failure, we repeat control broadcast $\gamma = 2$ times. Our extensive measurement study through data center racks indicated that this is also the maximum ETX for any link between two wireless motes. Upon receiving the control broadcast message, the nodes generate an OS level latency and hardware level latency. We use the maximum and minimum OS level and hardware level time required for power capping experimented on three servers with different processors: Intel Xeon L5520 (frequency 2.27GHz, 4 cores), Intel Xeon L5640 (frequency 2.27GHz, dual socket, 12 cores with hyper-threading), and an AMD Opteron 2373EE (frequency 2.10GHz, 8 cores with hyper-threading), each running Windows Server 2008 R2 [17]. The ranges of OS level and hardware level latencies are in the range of 10-50ms and 100-300ms, respectively [17]. We generate OS and hardware level latencies using a uniform distribution in this range.

D. Power Peak Analysis of Data Centers

We first analyze whether CapNet protocol is consistent with the data center power behavior leveraging our data traces. For brevity, we present the trace analysis results of 3 racks: Racks R1 and R2 from Cluster C1, and Rack R3 from Cluster C2. To give an idea on how power consumption varies over time in a data center, Figure 6(a) shows the aggregate power of 60 servers on RACK R1 in cluster C1 for 2 consecutive months which is zoomed in for 6 consecutive days in Figure 6(b). For



(a) Time interval between 2 peaks



(b) Power jump

Fig. 7. Power characteristics (2 month data)

each rack, we use the 95-th percentile of aggregate power over 2 consecutive months as the power cap.

We first explore the power dynamics of the servers and the unpredictability of power capping events. Using 2-month long data, Figure 7 shows that the time intervals between two consecutive peaks can range between few minutes to several hundred hours. We define *power jump* as the difference between the power that exceeds the cap and the preceding measurement that is below the cap. As Figure 7(b) shows that power jumps can vary between 0 to 51 for 60 servers in each rack (while their aggregate power is in range $[0, 60]$). This result shows the motivation for an event-driven protocol.

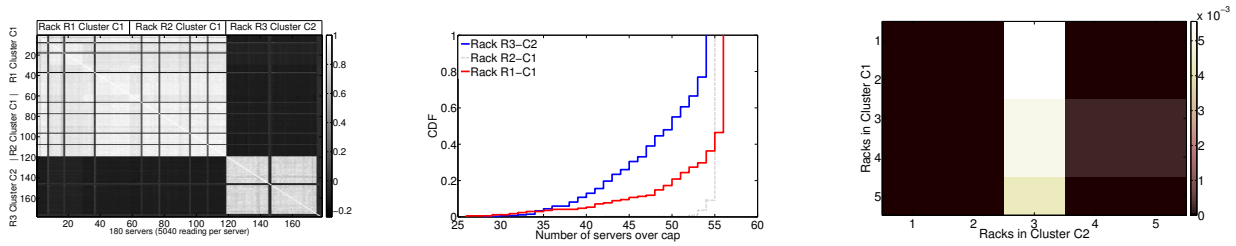
Figure 8 illustrates the correlations across 180 servers from different racks and clusters using their raw power consumption data over 1 week. The image is a visualization of a 180×180 matrix, indexed by the server number. That is, the entry indexed at $[i, j]$ in this matrix is the correlation coefficient of the values (5040 samples) between the i-th and the j-th server. We can clearly see that the servers in the same rack are strongly positively correlated, and those in the same cluster are also positively correlated. But the servers between clusters are less or negatively correlated. This usually happens because the servers in the same cluster hosts similar workloads leading to synchronous power characteristics [25]. We further assume a local cap of $\frac{c}{60}$ (considering $\alpha = 1$) for each individual server, and show in Figure 8(b) the CDF of the number of servers that exceed local caps when the cluster level aggregate power exceeds cap c. The figure shows that in 80% cases when the rack level aggregate power exceeds cap c, the numbers of servers (among 60 servers per rack) that are over the local cap are 43, 55, and 50 for Rack R3, R1, and R2, respectively. The strong *intra-cluster synchrony* in power surge suggests the feasibility of detecting a cluster-level power surge based on local server-level measurements. Figure 8(c) shows probabilities of different racks in 2 clusters to be at peak simultaneously. The entry indexed at $[i, j]$ in this 2D matrix is the probability that the i-th rack in cluster 1 and the j-th rack in cluster 2 are at peak simultaneously. The probabilities were found in the range $[0, 0.0056]$. This strong *inter-cluster asynchrony* implies that using an event-driven protocol (that performs wireless communications only upon detecting an event) significantly minimizes inter cluster interference caused by transmissions generated by the event-driven CapNet in different clusters.

We observe strong synchrony in power behavior among the servers in the same cluster and strong asynchrony among between different clusters. The major implication of the trace analysis is that CapNet protocol is consistent with real data center power behavior. As the intra-cluster synchrony suggests the potential efficacy of a local event detection policy, our protocol is particularly effective in the presence of strong intra-cluster synchrony that exists in enterprise data centers as observed in our trace analysis. However, in absence of intra-cluster synchrony in power peaks, CapNet will not cause unnecessary power capping control or more wireless traffic than a periodic protocol. The synchrony only enhances CapNet's performance.

E. Power Capping Results

Now we present our experimental results with CapNet's event-driven protocol. First we compare its performance with the periodic protocol and a representative CSMA/CA protocol. We then analyze its scalability in terms of number of servers. First we experiment only for the simple case, where a single iteration of control loop can settle to a sustained power level, and then we also analyze scalability in terms of number of control iterations, where multiple iterations are needed to settle to a sustained power level. We have also experimented it under different caps and in presence of interfering clusters. In all experiments, detection phase length, h, was set to $100 * n$ ms, where n is the number of servers. We set this value because this makes each slot in the detection phase equal to 100ms, which is enough for receiving one alarm as well as for sending a message from the manager to the servers. Setting a larger value reduces the number of cycles of detection phase, but reduces the granularity of monitoring. For assigning a local cap of $\frac{\alpha c}{n}$ to the servers, we first experiment with $\alpha = 1$. Later, we experiment under different values of α. Condition 1 is used for detection and starting an aggregation phase. In the results, ***slack*** is defined as the difference between the trip time (i.e. deadline) and the total latency required for power capping. That is, a negative value of slack implies a deadline miss. We use *LB slack* and *UB slack* to define the slack calculated considering LB trip time and UB trip time, respectively. In our results, in cases timing requirement can be loose, while there are cases where these are very tight, and the results are shown for all cases. We particularly care for tight deadlines, and want to avoid any deadline misses.

1) Performance Comparison with Base Lines: Figure 9 presents the results using 60 servers on one rack for single-iteration control loop. We used 4 weeks long data traces for this rack. We set the 95-th percentile of all aggregate powers

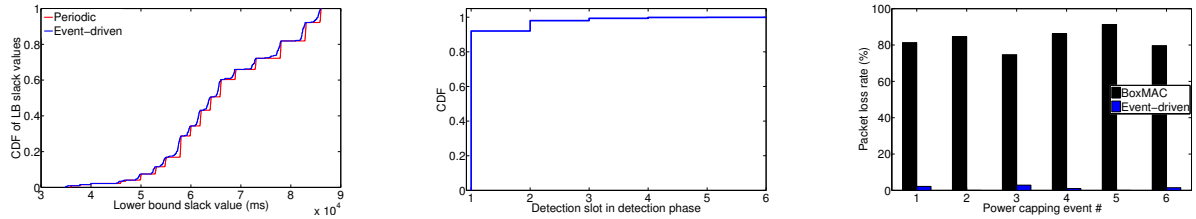


(a) Correlations among 180*180 server pairs in 3 racks in 2 clusters

(b) Total number (out of 60) of servers that exceed local cap $\frac{c}{60}$

(c) Probability of simultaneous peak between two different clusters

Fig. 8. Correlations among servers, racks, and clusters



(a) CDF of lower bound slack

(b) CDF of detection slots in detection phase

(c) Packet Loss Rate

Fig. 9. Performance of Event-Driven protocol on 60 servers (4 weeks)

values of all data points in every 2-minute interval as its cap c. For assigning local cap we use $\alpha = 1$. In running the protocols using these traces, the protocols observe all peaks. The upper bound of aggregation latency (L_{agg}) given in (3) was set as the period of the periodic protocol. Figure 9(a) shows the LB slacks for both the event-driven protocol and the periodic one. The figure only plots the CDF for the cases where the magnitude of oversubscription was above 1.5 for better resolution as the slack was too big for a smaller magnitudes (which are not of interest). Since UB trip times are easily met, we also omit those results. The non-negative LB slack values for each protocol indicate that it easily meets the trip times. Hence there is no benefit in using non-stop communications (i.e., the naive periodic protocol).

While the slacks in event-driven protocol are shorter than those in the periodic protocol because the former spends some time in the detection phase, in 80% cases event-driven protocol can provide a slack of more than 57.15s while the periodic protocol provides 57.88s. The difference is not significant because as shown in Figure 9(b) in 90% cases among all power capping events the detection happened in the first slot of the detection cycle. Only in 10% cases, it was after the first slot of the detection phase, and all detection happened within the 6-th slot, although the phase had a total of 60 slots (for 60 servers, one slot per server). These results indicate that CapNet's local detection policy can quickly determine the events. This is also an implication that experimental values of power capping latencies are quite different (or shorter) from the pessimistic analytical values derived in (5). Also, in this experiment, 94.16% of the total detection phases did not have any transmission from the servers. Therefore, if we compare with the periodic protocol that needs to continue communication always in the network, the event-driven protocol

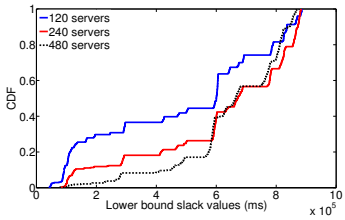


Fig. 10. CDF of LB slack under various numbers of servers (4 weeks)

suppresses transmissions at least by 94.16% while the real-time performance of two protocols are similar.

We also evaluate the performance when BoxMAC (the default CSMA/CA based protocol in TinyOS [21]) is used for power capping communication for up to first 6 capping events in the data traces. Figure 9(c) shows that it experiences packet loss rate over 74% while performing communication for a power capping event. This happens because all 60 nodes try to send at the same time, and the back-off period in 802.15.4 CSMA/CA under default setting is too short, which leads to frequent repeated collisions. Since we lose most of the packets, we do not consider latency under CSMA/CA. Increasing the back-off period reduces collisions but results in long communication delays. In subsequent experiments, we exclude CSMA/CA as it does not fit for power capping.

2) Scalability in Terms of Number of Servers: In our data traces each rack has at most 60 active servers. To test with more servers, we combine multiple racks in the same cluster since they have similar pattern of power consumption (as we have already discussed in Subsection V-D. For sake of experimentation time, in all subsequent experiments we set cap at 98-th percentile (that would result in a smaller number of capping events). The lower bound slack distribution are

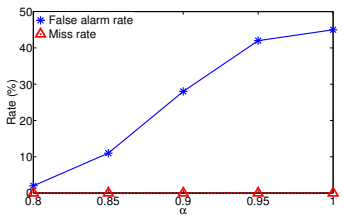


Fig. 11. Deadline (trip time) miss rate and false alarm rate under varying α

shown in Figure 10 for 120, 240, and 480 servers by merging 2, 4, and 8 racks, respectively (for single iteration capping). Hence, for single iteration, the deadlines are easily met for even 480 servers (since in each setup, 100% of all slack values are positive).

3) Experiments under Varying α: Now we experiment with different values of α for assigning a local cap of $\frac{\alpha c}{n}$ to the servers using 480 servers. The results in Figure 11 show the tradeoff between false alarm rate and power capping latency under varying α. As we decrease the value of α from 1 to 0.80, the false alarm rate decreases from 45% to 2%. This happens because with decreased value of α, CapNet considers multiple alarms before detecting a potential event. Note that this alarm rate is very small compared to the whole time window since power capping happens in at most 5% cases. Therefore alarms are also generated rarely. Since waiting for multiple alarms increases the latency in detection, the total power capping latency increases as the value of α decreases. However, as this latency increase happens only in the detection phase which is negligible compared to the total capping latency, there is hardly any impact on deadline miss rates. The figure shows a deadline miss rate of 0 under varying α.

4) Scalability in Number of Control Iterations: Now we consider a conservative case where multiple iterations of control loop are required to settle to a sustained power level [14], [17], [18]. The number of iterations required for the rack-level loop as experimented in [18] can be up to 16 in the worst case (which happens very rarely). Hence, we now conduct experiments considering multiple numbers of control iterations (up to 16 assuming a pessimistic scenario). We plot the results in Figure 12 for various numbers of servers under various number of iterations. As shown in Figure 12(a), for 120 servers under 16-iteration case, we have 13% cases with negative slack meaning that the LB trip times were missed. However, the UB trip times were met in 100% cases. Note that we have considered a quite pessimistic set up here because using 16-iteration as well as trying to meet the lower bound of trip times are both very conservative considerations. For 120 servers under 8 iterations, in 0.13% cases slacks were negative. However, in 80% cases the slacks were 92.492s, 66.694s, and 22.238s for 4, 8, and 16 iterations, respectively indicating that the trip times were easily met, and the system could oversubscribe safely. For 4-iteration, the minimum slack was 23.2s. To preserve figure resolution, we do not show the UB slacks since they were all positive. For 480 servers (Figures 12(b), 12(c)), 98.95%, 97.86%, 94.93%, and 67.2% LB trip times were met for 2, 4, 8, and 16 iterations, respectively. For 240 nodes, we miss deadlines in 5% cases under 8-iteration and 13.94% cases under 16-iteration.

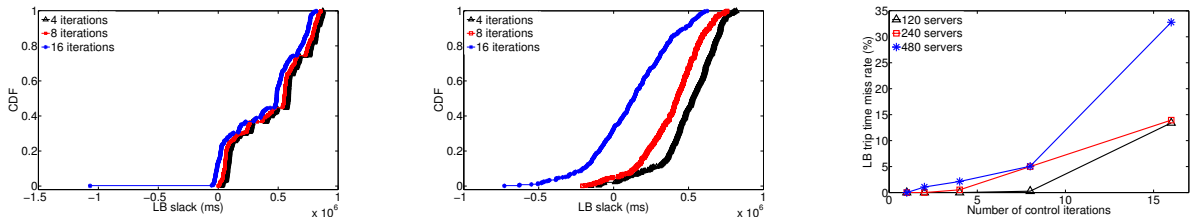
For all cases we met UB trip times in 100% cases. Note that assuming 16-iteration and considering the LB trip times are very conservative assumption as it can rarely happen. Hence, the above results show that, even for 480 servers, the latencies incurred in CapNet for power capping remain within even the conservative latency requirements in most cases.

5) Experiments under Varying Caps: In all experiments we have performed so far, CapNet was able to meet UB trip times. Now we make some setup changes to encounter some scenario where UB trip times can be smaller, by making oversubscription magnitude higher. For this purpose, we now decrease the cap to decrease the trip times so as to make scenarios to miss upper bound trip times to see the robustness of the protocol. Now again we set the 95-th percentile of aggregate power as the cap. This would give the previous capping events shorter deadlines since a smaller cap implies a larger magnitude of oversubscription. For the sake of experiment time, we only tested with 120 servers and their 4 week data traces. Figure 13 shows that we now miss more LB trip times and miss some UB trip times as well since the deadlines now become shorter. However, UB trip times are missed only in 0.11% and 1.02% cases under 8 and 16 iterations, respectively, while LB deadlines were missed in 2.14%, 6.84%, and 26.56% cases under 4, 8, and 16 iterations, respectively. All deadlines were met for up to 3 iterations (and not shown in the figures). We have shown the results only for higher number of iterations that rarely happen. These results demonstrate the robustness for larger magnitude of oversubscription in that even when we use 16-iteration only 1.02% UB trip times are missed.

6) Experiments in Presence of Multiple Clusters: We have shown through data center trace analysis in Figure 8(c) that the probability that two clusters are over the cap simultaneously is no greater than 0.0056. Yet, in this section we perform some experiment from a pessimistic point of view. In particular, we perform an experiment and see the performance of CapNet under an interfering cluster.

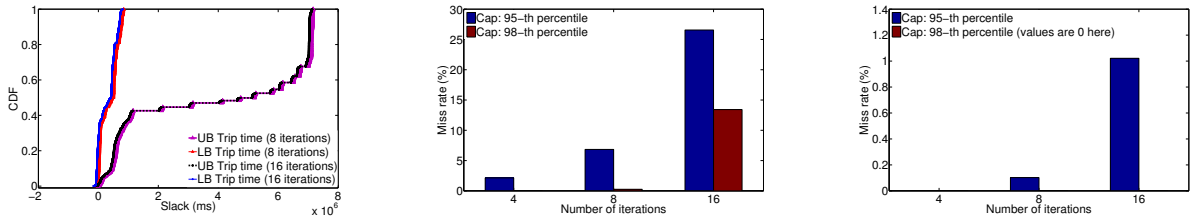
We mimic an interfering cluster of 480 servers in the following way. We select a nearby cluster and place a pair of motes in the rack: one at the ToR and the other inside the rack. We set their Tx power at maximum (0dBm). The mote at the ToR represents its manager and carries on a pattern of communication like a real manager to control 480 servers. The mote inside the rack responds as if it were connected to each of 480 servers. Specifically, the manager executes a detection phase of $100 * 480$ms, and the node in the rack randomly selects a slot between 1 and 480. On that slot, it generates an alarm with probability 5% since capping happens in no more than 5% cases. Whenever the manager receives the alarm, it generates a burst of communication in the pattern like what it would have done for 480 servers. After finishing this pattern of communication it resumes the detection phase.

We run the main cluster (system used for experiment) using 4 weeks data traces, and plot the results in Figure 14.



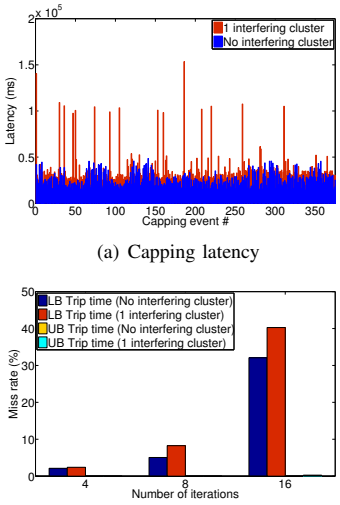
(a) LB slack for 120 servers (b) LB slack for 480 servers (c) Miss rate (LB trip time)

Fig. 12. Multi-iteration capping under event-driven protocol (4 weeks)



(a) CDF of slack values (Cap: 95-th percentile) (b) LB trip time miss rate (c) UB trip time miss rate

Fig. 13. Capping under different caps on 120 servers (4 weeks)



(a) Capping latency

(b) Miss rate

Fig. 14. Capping for 480 servers under interfering cluster

Figure 14(a) shows the latencies for different capping events in 4 weeks data both under interference and without interference (when there was no other cluster). Under interfering cluster, the delays mostly increase. This happens because the event-driven protocol experiences packet loss and uses retransmission for those, thereby increasing network delays. While the maximum increase was 124.63s, in 80% cases the increase was less than 15.089s. We noticed that such big increase happened due to the loss of alarms in a detection phase that resulted in a detection in the next phase (i.e., while the phase length is 48s). Still power capping was successful in all cases but those when the control broadcast was lost. Among 375 events, 4 broadcasts were lost at some server even after 2 repeatations, resulting in control failure in 1.06% cases. This value became 0 in multi-iteration cases. For multi-iteration cases, at least one control broadcasts was successful that resulted in no capping failure for control message loss. However, as the delay due to transmission failure and recovery increased in detection phase, we experienced capping failure. For 16-iteration, we missed the upper bound of trip time in 40.27% cases and lower bound of trip times in 32.08% cases. However, we use a conservative assumption here. For 4 iteration miss rate was 5.06% and 8.26% only. And for 2-iteration they are only 2.13% and 2.4% which are very marginal. The result indicates that even under interference, CapNet demonstrates robustness in meeting the real-time requirements of power capping.

VI. Discussions and Future Work

While our paper addresses feasibility, protocol design and implementation, several engineering challenges such as security, EMI and fault tolerance needs to be addressed.

Fault Tolerance. One important challenge is handling the failure of power capping manager in a cluster. To address this, power capping managers can be connected among themselves either through a different band or through a wired backbone. As a result, when some manager fails, a nearby one can take over its servers. This paper focuses on communication within a single cluster. DCM fault detection, isolation, and node migration need to be studied in future work.

Security. Another challenge is the security of the management system itself. Since the system relies on wireless control, someone might be able to maliciously tap into the wireless network and take control of the data center. There are two typical approaches to handle this security issue: First, the signal itself should be attenuated by the time it reaches outside the building. We can identify secure locations inside the data center from which the controller can communicate, and identify a signature for the controllers which would be known to the server machines. Second, it is possible to encrypt wireless messages, for example, using MoteAODV (+AES) [26]. We

can also use shielding within the data center to keep the RF signals contained within the enclosed region.

EMI & Compliance. While less emphasized in research studies, a practical concern of introducing wireless communications in data centers is that they do not adversely impact other devices. There are FCC certified IEEE 802.15.4 circuit design available(e.g. [27]). Previous work has also used WiFi and ZigBee in live data centers for monitoring purposes [9].

VII. Related Work

In order to reduce the capital spending on data centers, enterprise data centers use an over-subscription approach as studied in [12]–[15], which is similar to over-booking in airline reservations. Server vendors and data center solutions providers have started to offer power capping solutions [28], [29]. Power capping using feedback control algorithms [30] has been studied for individual servers. In contrast, the study of this paper concentrates to coordinated power capping which is more desirable in data centers as it allows servers to exploit power left unused by other servers. While such power capping has been studied before [14], [18], [31]–[34], all existing solutions rely on wired network for controller-server communication. In contrast, we focus on wireless networking for power capping. We have outlined the advantages of wireless management in Section II.

Previous work on using wireless network in data centers exists on applications to high bandwidth (e.g. with 60GHz radio) production data network [35]. In contrast, CapNet is targeted at data management functions that have much lower bandwidth requirement while demanding real-time communication through racks. RACNet [9] is a passive monitoring solution in the data center that monitors temperature or humidity across racks where all radios are mounted at the top of the rack. Our solution enables active control and requires communication through racks and server enclosures, and hence encounters fundamentally different challenges. Also, RACNet also does not have real-time features, while CapNet is designed to meet the real-time requirements in power capping.

VIII. Conclusion

Power capping is a time-critical management operation for data centers that commonly oversubscribe power infrastructure for cost savings. In this paper, we have designed CapNet, a low-cost, real-time wireless management network for data centers and validated its feasibility for power capping. We deployed and evaluated CapNet in an enterprise data center. Using server power traces, our experimental results on a cluster of 480 servers inside the data center show that CapNet can meet the real-time requirements of power capping. CapNet represents a promising step towards applying low power wireless networks to time-critical, close-loop control in DCM.

Acknowledgment

This work was supported, in part, by Microsoft Research and NSF through grants CNS-1320921 (NeTS) and 1144552 (NeTS).

References

[1] M. Isard, "Autopilot: automatic data center management," *Operating Systems Review*, vol. 41, pp. 60–67, 2007.
[2] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *SIGCOMM '08*.
[3] Private communication with data center operators.
[4] www.cdwg.com/shop/products/Digi-Passport-48-console-server/1317701.aspx.
[5] http://www.cdwg.com/shop/search/Servers-Server-Management/Servers/x86-Based-Servers/result.aspx?w=S62&pCurrent=1&p=200008&a1520=002200.
[6] http://www.cdwg.com.
[7] http://www.cdwg.com/shop/search/Networking-Products/Ethernet-Switches/Fixed-Managed-Switches/result.aspx?w=N11&MaxRecords=25&SortBy=TopSellers.
[8] http://www.digikey.com/us/en/techzone/wireless/resources/articles/comparing-low-power-wireless.html.
[9] C.-J. M. Liang, J. Liu, L. Luo, A. Terzis, and F. Zhao, "RACNet: a high-fidelity data center sensing network," in *SenSys '09*, 2009.
[10] K. Srinivasan and P. Levis, "RSSI is under appreciated," in *EmNets '06*.
[11] Hamilton, 2008, http://perspectives.mvdirona.com.
[12] S. Pelley, D. Meisner, P. Zandevakili, T. F. Wenisch, and J. Underwood, "Power routing: Dynamic power provisioning in the data center," in *ASPLOS '10*.
[13] X. Fu, X. Wang, and C. Lefurgy, "How much power oversubscription is safe and allowed in data centers?" in *ICAC '11*.
[14] H. Lim, A. Kansal, and J. Liu, "Power budgeting for virtualized data centers," in *USENIXATC '11*.
[15] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ISCA '07*.
[16] http://literature.rockwellautomation.com/idc/groups/literature/documents/sg/1489-sg001_-en-p.pdf.
[17] A. Bhattacharya, D. Culler, A. Kansal, S. Govindan, and S. Sankar, "The need for speed and stability in data center power capping," in *IGCC '12*.
[18] X. Wang, M. Chen, C. Lefurgy, and T. Keller, "SHIP: A scalable hierarchical power control architecture for large-scale data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, 2012.
[19] A. Saifullah, Y. Xu, C. Lu, and Y. Chen, "Distributed channel allocation protocols for wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*.
[20] I. Rhee, A. Warrier, M. Aia, and J. Min, "Z-MAC: a hybrid MAC for wireless sensor networks," in *SenSys '05*, 2005.
[21] "TinyOS Community Forum," http://www.tinyos.net/.
[22] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ISCA '07*, 2007.
[23] J. Choi, S. Govindan, B. Urgaonkar, and A. Sivasubramaniam, "Profiling, prediction, and capping of power consumption in consolidated environments," in *MASCOTS '08*.
[24] P. Ranganathan, P. Leech, D. Irwin, J. Chase, and H. Packard, "Ensemble-level Power Management for Dense Blade Servers," in *ISCA '06*.
[25] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, "Energy-aware server provisioning and load dispatching for connection-intensive internet services," in *NSDI '08*.
[26] W. Backes and J. Cordasco, "Moteaodv – an aodv implementation for tinyos 2.0," in *WISTP '10*.
[27] http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en535967.
[28] http://www.intel.com/content/dam/doc/case-study/data-center-efficiency-xeon-baidu-case-study.pdf.
[29] http://h20000.www2.hp.com/bc/docs/support/SupportManual/c01549455/c01549455.pdf.
[30] Z. Wang, C. McCarthy, X. Zhu, P. Ranganathan, and Talwar, "Feedback control algorithm for power management of servers," in *ASPLOS '08*.
[31] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No power struggles: coordinated multi-level power management for the data center," in *ASPLOS '08*.
[32] M. E. Femal and V. W. Freeh, "Boosting data center performance through non-uniform power allocation," in *ICAC '05*.
[33] V. Kontorinisy, Zhangy, Aksanliy, and Sampson, "Managing distributed UPS energy for effective power capping in data centers," in *ISCA '12*.
[34] Y. Zhang, Y. Wang, and X. Wang, "Capping the electricity cost of cloud-scale data centers with impacts on power markets," in *HPDC '11*.
[35] X. Zhou, Z. Zhang, Y. Zhu, Y. Li, S. Kumar, A. Vahdat, B. Y. Zhao, and H. Zheng, "Mirror mirror on the ceiling: flexible wireless links for data centers," in *SIGCOMM '12*.