Energy-aware Mobile Service Overlays: Cooperative Dynamic Power Management in Distributed Mobile Systems

Balasubramanian Seshasayee, Ripal Nathuji, Karsten Schwan
College of Computing
Georgia Institute of Technology
Atlanta, GA 30032
{bala, rnathuji, schwan}@cc.gatech.edu

Abstract

With their increasingly powerful computational resources and high-speed wireless communications, future mobile systems will have the ability to run sophisticated applications on collections of cooperative end devices. Mobility, however, requires dynamic management of these platforms' distributed resources, and such management can also be used to meet application quality requirements and prolong application lifetimes, the latter by best using available energy resources. This paper presents energy-aware Mobile Service Overlays *(MSOs), a set of mechanisms and associated policies for running mobile applications across multiple, cooperating machines while actively performing power management to extend system usability lifetimes. MSO policies manage energy consumption by (i) allocating application components to available nodes based upon their current energy capacities and resource availabilities, (ii) monitoring for, and responding to changes in energy and resource characteristics, and (iii) dynamically exploiting energy-performance tradeoffs in overprovisioned situations. Coupled with mobility, such cooperation enables multiple mobile platforms to bring their joint resources to bear on complex application tasks, providing significant benefits to application lifetimes and performance. This paper evaluates MSOs on a MANET computing testbed indicate an extension in system lifetime of upto 10% for an example application.*

1 Introduction

Distributed applications running on Mobile Adhoc NETworks (MANETs) are being used in many domains, ranging from autonomous robotics, to online gaming, to emergency management. Growth in such applications is encouraged by the proliferation of handhelds and portables and by substantial increases in the combined computational power of the MANET systems on which they run, the latter caused by the increased computational capabilities of these platforms and the improved connectivity afforded by their powerful and diverse communication devices. Also promoted by these trends are cooperative approaches to running applications across sets of participating machines, with early examples of these including robots that collaboratively undertake a search and rescue mission [12], coordinated actions of geographically dispersed agents in an emergency rescue operation [17], distributed surveillance in the battlefield, distributed gaming, and ubiquitous computing environments [18].

Since cooperative mobile applications like those listed above operate in highly dynamic execution environments, their effective execution requires them to adapt, online, to changes in requirements and to dynamically react to changes in underlying platform resources, including reactions to disruptions caused by node mobility or failure [26] Sample disruptions occurring in a team of cooperating robots are (i) the failure of a robot that performs critical processing for the team or (ii) the movement of this robot out of range. The presence of disruptions also implies that centralized solutions are not suitable for managing the application software components distributed amongst nodes. Instead, decentralized methods offer improved fault resilience and scalability [27]. Realizing these facts, the research community has developed a wide variety of decentralized management services, including energy management [20, 18], load balancing [25], and QoS provisioning [32].

This paper focuses on energy management. In particular, for MANET applications, we develop *cooperative energy management strategies* that enhance their energy profiles, (i) by decreasing their energy consumption to the extent permitted by current application performance constraints, and (ii) by extending overall system and application lifetime, by migrating application services that are critical to the application away from energy constrained nodes. Specifically, concerning (i), for each single platform, we reduce its energy consumption by using common techniques for energy management, which are dynamic voltage and frequency scaling (DVFS). The resulting degradation in application execution can be reduced by utilizing memory-bound phases for such scaling [10], or, in our real-time environments where there is a notion of slack, by increasing execution times (thereby reducing energy needs) only to the extent permitted by application deadlines [1]. Similar energy-performance tradeoffs are available for other devices such as memory, peripherals (network and disk interfaces), display, etc. Concerning (ii), we use computational offloading, whereby portions of the mobile workload are dynamically offloaded to nodes with better energy resources [25]. The former set of techniques have a direct effect on energy savings at distinct nodes, whereas the latter helps in longevity by sharing energy resources amongst multiple participants.

Energy-aware management is implemented for realistic mobile applications and systems using the lightweight and efficient Mobile Service Overlays (MSOs) [28]. MSO middleware provides efficient mechanisms for dynamically creating, moving, and reconfiguring computational services across distributed mobile platforms, and in addition, for monitoring underlying platform conditions. This paper uses these facilities to develop and demonstrate novel decentralized management protocols that jointly, extend the lifetimes of distributed MANET applications and systems. These protocols (i) dynamically distribute and re-distribute application components among participating MANET, considering the overlay routes that satisfy the application's latency requirements while at the same time, determining the most energy-efficient allocations, (ii) recover unused portions of resources in an overprovisioned system with little or no impact on application performance, and (iii) use de-centralized online monitoring and reconfiguration to locally and thereby, with low delay and overhead, respond to dynamic changes in application requirements and environment conditions. The management algorithms being used, specifically the algorithm for dynamic resource reclamation, is experimentally demonstrated to track optimality, with low overhead. MSOs using this algorithm – *energy-aware MSOs* – offer notable benefits. On a wireless, multi-hop ad-hoc network of handheld computing platforms, for instance, an energy-aware MSO extends system lifetime up to 10% for a five-node network.

The remainder of this paper is organized as follows: In Section 2, we explore related research, then describe an overview of energy-aware MSOs in Section. 3. This is followed by a description of the energy management techniques used in this MSO in Section 4. Section 5 discusses current trends in power management. We present selected elements of the implementation of energy-aware MSOs and evaluate it experimentally in Section 6. Conclusions and future work appear in Section 7.

2 Related Work

Prior research on energy conservation in ad-hoc networks has mainly focused on energy-aware routing protocols – by improving existing protocols like AODV [24], by factoring the energy levels of each node in the routing cost metric [9], or through novel protocols like probabilistic routing [29]. The former classifies nodes into various classes depending on their energies, and uses this instead of the hop count, to determine the energy-optimal route. The latter uses a similar cost metric by aggregating the energy values of the nodes in each route, then randomly chooses a route with a probability proportional to the cost metric along the route. While these approaches are suitable for network-bound applications and sensor nodes, where the network interface accounts for a significant portion of the power budget, for the applications and platforms considered in our work experimental results that the energy consumed by the network interface is quite small compared to CPU energy. Our research, therefore, primarily leverages prior work on reducing CPU energy consumption, including reducing energy usage by applying dynamic voltage and frequency scaling [11] on multiprocessor systems. [15] uses dynamic slack reclamation in conjunction with DVFS in a real-time setting, on a multiprocessor system. A static schedule is first constructed for periodic tasks, then slack reclaiming is used to save power, yet satisfy real-time constraints. Resource reclaiming in multiprocessor real-time systems has been dealt with in great detail in [30], where the authors describe two algorithms to perform online reclaiming on a static schedule.

Computational offloading has been used extensively for power-aware load balancing ranging from clusters of workstations [25], to embedded devices [31]. The latter performs computational offloading in conjunction with setting the CPU frequencies, to minimize energy consumption. Distributed middleware can be useful to manage computational entities in such environments. [16] surveys the various middleware implementations for a mobile environment. Research efforts to include mobile nodes in Grid technology include [2], which proposes a mobile agent framework to provide/use Grid services at the mobile nodes, so that distributed resources from the Grid can be accessed by such users. DFuse [14] is a cluster of iPAQ handhelds used to perform multimedia and other data fusion tasks. MagnetOS [5] uses a distributed framework to partition a monolithic Java application into its constituent classes for cooperative execution on a MANET. Efficient placement of application components is carried out by monitoring the data traffic among nodes, and using distributed algorithms to shorten the mean path length of data.

3 Mobile Service Overlays – System Overview

Mobile Service Overlays (MSO) [28] is a distributed middleware system designed to execute mobile and pervasive applications. MSO provides mechanisms to dynamically create and manage overlays on a mobile network. It operates under an event-driven computation model [16], where data, in the form of events, is exchanged across distributed computing nodes for processing. MSO models applications as directed flow graphs whose vertices represent processing performed on events arriving via directed edges. Associated with each edge is a format describing the data passing through the edge. Such a graph-based representation implicitly captures the data dependencies among computational entities, and it also simplifies application partitioning for a distributed environment. MSO programs, therefore, consist of code modules running in vertices mapped to overlay nodes, receiving formatted inputs from and generating outputs to other vertices. Similar to an event processing system, the flow graph consists of event sources and sinks.

MSO targets dynamic management of the application flowgraph in unreliable, distributed mobile environments. Correspondingly, its design goals include (1) decentralized resource management and (2) low overhead mechanisms for (re-)deployment. Consequently, no explicit organization is enforced among the participating nodes: all MSO nodes are peers and have identical middleware functionality. Furthermore, to avoid the scalability and reliability issues arising from centralized or global management, the application flow graph is partitioned into its constituent and independently manageable computational chains. Formally, a *chain* is a maximal set of sequential vertices and edges of the flow graph, with a single entry and a single exit. Events enter the first vertex of the chain, the *head*, sequentially pass through and are operated on at each node in the chain, and finally exit at the last node, the *tail*. Many application properties (e.g., end-to-end delay) can be formulated to depend on the corresponding local properties of each chain, and different chains can be managed independently in order to guarantee such properties. In this fashion, chains compartmentalize management to be confined to more "local" portions of the application. Figure 1 shows an example flowgraph decomposed into its constituent chains (A→D, D→E, F→G, G→D, G→I).

Using the partitioning provided by the chain abstraction, MSO provides multiple low level services to higher layers for use in cooperative MANETs, as described next.

Deployment Services: Deployment consists of the instantiation of the application flowgraph onto the mobile network. It has two sub-phases: the *allocation* phase in which each vertex is assigned to a node in the network, and the actual construction of the overlay network by creating the vertices at the assigned nodes and the links connecting them. The *chain*-based decomposition of the flow graph provides a convenient abstraction for distributed allocation. Algorithm 1 details this procedure.

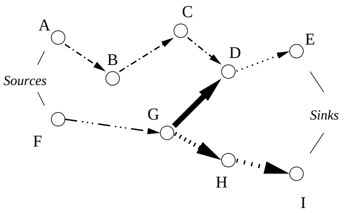


Figure 1. Partitioning the graph into chains

Algorithm 1 Allocation

```
1: ∀ chain $c$, set dependences($c$) to the number of chains leading to it.
2: if dependences($c$) == 0 then
3:     Explore the route taken by a packet to the sink node furthest away, collecting the capacities of all nodes along the route.
4: end if
5: Allocate nodes to vertices of the chain in proportion to the fraction of the overall costs contributed by the vertices (i.e, for chains $c_1$ and $c_2$ such that $c_1$ leads to $c_2$, if the route from head($c_1$) to tail($c_2$) is $r$, the nodes in $r$ are assigned to $c_1$ and $c_2$ in proportion to $\frac{Cost(c_1)}{Cost(c_2)}$)
6: Within a chain, perform a greedy assignment of vertices to nodes.
7: ∀ chain $c'$ : $head(c') = tail(c)$, decrement dependences($c'$) and assign head($c'$) to the same node as $tail(c)$.
8: Repeat the above steps until all chains have been assigned to nodes.
9: Each node that has a chain head mapped to it constructs the overlay network along its chain.
```

In this algorithm, we make use of the decomposition of a potentially large application flowgraph into chains in order to distribute the assignment process and provide scalability. Indeed, after performing the cost-based partitioning of nodes for different chains, the chains themselves independently determine the best set of participants to use, as referred to in Step 6, in a greedy fashion. As we will describe later, a novel contribution of this paper is to integrate energy awareness into this step, to provide global system lifetime benefits.

Algorithm 1 implicitly assumes that the sources and sinks of the flowgraph are already assigned to their nodes. In realistic systems, source and sink vertices can often be assigned only to certain nodes (e.g., those that possess sensors/actuators that produce or consume data). We therefore,

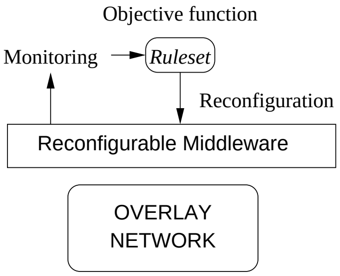


Figure 2. Generic management in MSO

assume the source and sink vertices to be preassigned to particular nodes. If this is not the case, a distributed process can be used by which nodes possessing the capabilities required by the source/sink vertices are chosen according to their abilities and assigned to nodes before deployment.

Monitoring Services: Each MSO node runs a separate monitoring thread that is used to maintain metrics like CPU utilization due to the computation being carried out, the amount of data transferred from/to the node and the expected lifetime of the battery. Additionally, monitoring periodically checks for the liveness of its neighbors (as determined by the routing layer) and for changes in the routing layer. The metrics monitored are available for use by the higher layers for reconfiguration triggers and decisions. Monitored metrics are also available for sharing, as each monitor exposes its data to other nodes through RPC calls, and by piggybacking monitoring information along with events.

Self-Management Services: The generic management framework in MSO involves (i) monitoring for specific changes in observed metrics, and (ii) triggering reconfiguration mechanisms at the appropriate granularity to counter the change, based on a predefined ruleset. As MSO is decentralized, any node that observes a change can trigger a reconfiguration according to predefined rules tailored towards specific objectives (Fig 2). Further, as this is a generic procedure, it can be applied to different management goals like load balancing, mobility & fault tolerance, latency minimization, etc. The complexity of the ruleset used to trigger a reconfiguration dictates the control overheads of the decision process in management. As a result, a simple greedy ruleset can only target local optima – an acceptable provision under dynamic conditions.

Reconfiguration Services: Reconfiguration involves remapping portions of the overlay network to create a different assignment between vertices in the overlay and underlying machines. MSO provides capabilities to perform remapping at various granularities. In response to changing conditions, reconfiguration is performed at the appropriate level, while meeting the application's performance constraints.

- *reconfigure_local*: Intra-chain remapping, where a single assignment (of a vertex to a node) is changed, either by an upstream relocation (a vertex B, that follows another vertex A within a chain (i.e., ..→A→B→..) is relocated to the same node as A), or a downstream one. Such a remapping is of low cost and involves only the participating nodes, hence can be performed frequently, and finds use in continuous minor reconfigurations to optimize the system.

- *reconfigure_single_chain*: Chain remapping, where the existing assignments of vertices in a chain are freed and a fresh assignment performed. This is more expensive than intra-chain remapping, and consequently, used less frequently. Chain remapping is ideal for situations like load-balancing, handling node failure, etc. where only a single chain is affected

- *reconfigure_multiple_chains*: This refers to all remapping procedures that affect more than a single chain, or maybe even the entire flowgraph, and hence require cooperation of a larger set of nodes. Due to its high cost, this is used least frequently – for instance, when detecting a node failure during an ongoing reconfiguration.

4 Techniques for Energy Management

Energy management in a dynamic environment is a continuous process, requiring an energy-aware assignment, followed by online monitoring to trigger actions that shift the system towards optimality. MSO provides three techniques for energy management, viz., energy-aware allocation, reallocation, and dynamic resource reclaiming.

4.1 Energy Aware Allocation

We revisit allocation (Algorithm 1) here, to introduce energy awareness into the assignment process. The chain assignment procedure is modified to address energy management concerns, using the following techniques: (i) modifying route exploration to include *Ad-hoc Route Neighborhoods* and (ii) using the *Global Lifetime Sustainability* heuristic to determine the best assignment of vertices to nodes from among the routes in the neighborhood.

Ad-Hoc Route Neighborhoods: In Algorithm 1, the route explored from the source node to the sink is used to perform the assignment over each chain. However, the route thus found is entirely dependent on the ad-hoc routing protocol employed in the underlying layer. Traditional routing protocols like AODV, DSR, etc. typically use the route with the smallest hopcount. However, recent research has explored a variety of techniques for power management at the network routing layer, including probabilistic routing [29] and multipath routing, and even using multiple wireless interfaces [23]. Hence, prior to allocation, no assumptions

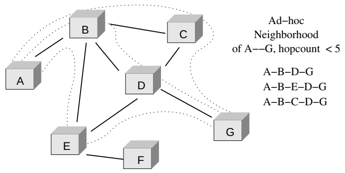


Figure 3. Ad-hoc Route Neighborhood

can be made by MSO about the underlying protocol's behavior. As a result, MSO explores all possible routes that can be taken from the source to the destination (bounded by a maximum hopcount), and chooses the "best" route from among those for the assignment. The set of all the routes discovered in this manner is termed the ad-hoc route neighborhood of the chain. The end-to-end latency requirements of the chain thus limits the maximum hop count for the route, and consequently, the length of each route in the ad-hoc neighborhood, as well as the number of routes thus obtained. Figure 3 illustrates this concept, enumerating the ad-hoc route neighborhood of routes from node A to D in the topology, with hopcount strictly less than 5.

We rely on a distributed protocol to find the route neighborhood of a chain, given the latency constraints (maximum latency and maximum hopcount). The protocol is similar to that used by AODV for route discovery [24], with two differences: (i) the destination node stores all the routes it obtains, and (ii) after the maximum latency period, the source node queries the destination to directly obtain all the routes stored. This protocol enumerates all of the routes satisfying the latency constraints. Since no state is saved in any nodes (except at the destination), its overhead is low. Further, each packet not reaching the destination is eventually dropped, as the monotonically decreasing hopcount reaches zero.

Global Lifetime Sustainability (GLS) Heuristic: The placement of software components during allocation and reallocation can directly affect the lifetime of a MANET scenario. In the simplest case, with two nodes and a single component, energy can be depleted fairly and system lifetime is maximized by migrating the workload to the participant with more battery capacity whenever a reallocation is triggered. When there are multiple components and multiple nodes, all with varying requirements and energy levels, optimal decisions cannot be made within reasonable complexity constraints. Instead, we use a heuristic to determine where to place various components. In particular, we define a GLS metric for a set of candidate nodes and energy levels and utilize a heuristic which attempts to maximize this metric and/or minimize potential decreases to it. this Specifically, for this work, we define the GLS metric to be the product of the remaining energy levels on nodes under consideration when deciding where to place a software component. This approach fits well in the scenarios addressed here since nodes are treated equally and are homogeneous. We note that the GLS metric can also be used more generally, to express variations among nodes in heterogeneous environments and to tune the allocation policies at runtime. We will investigate these issues in our future work.

We now describe how the use of the GLS heuristic is integrated into chain assignment decisions in MSO along with the use of ad-hoc neighborhoods. At a high level, from among all the possible routes discovered (ad-hoc route neighborhood) for each chain, (1) a possible assignment is determined based upon resource availability and the GLS heuristic, and (2) a cost is associated with the route and allocation scheme. The route with the smallest cost is then chosen for the chain. In particular, we define the cost as the projected difference in the GLS metric based upon the allocation over some period of time. The overall approach is described in Algorithm 2.

Algorithm 2 GLS-based assignment for a chain

```
Given the chain $c = v_1, v_2, ..., v_n$ to be assigned to one of the routes $r_1, r_2, ..., r_m$,
Set $Cost(c) = \sum_{v_i} Cost(v_i)$
for all $r_j$ of length $k_j$ do
    Set $l = 1$ {Lower bound}
    for all $v_i$ do
        Find max $u$ s.t. $Cost(c) <= \sum_{p=u}^{k_j} Capacity(n_p)$ {Upper bound}
        for $p = l$ to $u$ do
            Set $GLS_{now}$ as a function of current battery capacities {Current GLS}
            Set $GLS_t$ as a function of estimated future battery capacities after time $t$ when allocating to $n_p$ {GLS after assignment}
            Set $\delta GLS_p = GLS_{now} - GLS_t$
        end for
        Assign $v_i$ to $n_p$ s.t. $\delta GLS_p$ is minimized, call it $\delta GLS_{v_i}$
        Set $l$ to $p$ {Set new lower bound for remaining vertices and nodes}
        Set $Cost(c) = Cost(c) - Cost(v_i)$
        Find max $u$ s.t. $Cost(c) <= \sum_{p=u}^{k_j} Capacity(n_p)$ {New upper bound}
    end for
end for
Determine $\forall$ route $r_j$, $\delta GLS_{r_j} = Aggregate_{v_i}(\delta GLS_{v_i})$
Choose route $r_j$ s.t. $\delta GLS_{r_j}$ is minimized
```

4.2 Energy-Aware Reallocation

Energy-Aware reallocation consists of moving application components in response to changing conditions, and utilizes the monitoring, management, and reconfiguration services provided by MSO (Algorithm 3).

Algorithm 3 Energy-Aware Management

```
Each node $n$, periodically queries expected lifetime of all its neighbors
while $\exists$ node $n'$ s.t. $lifetime(n') - lifetime(n) > threshold$ that is predefined do
    if $\exists$ vertex $v'$ housed at $n'$ s.t. vertex $v$ housed in $n$, and either $v \rightarrow v'$ or $v' \rightarrow v$ then
        MSO.reconfigure_local($v$){Perform the upstream/downstream relocation on $v$}
    else if $\exists v$ housed at $n$ s.t. $v$ is neither the head nor tail of its chain then
        MSO.reconfigure_chain($v$) {Trigger a reallocation of the chain to which $v$ belongs}
    else
        MSO.reconfigure_all($v$) {$v$ is a chain head; Perform a remap on all chains sharing vertex $v$}
    end if
end while
During long periods of idleness, $\forall v$ housed at $n : v$ is a chain head, check for any changes in the ad-hoc neighborhood, and remap the chain if a more optimal neighborhood is found.
```

Algorithm 4 Slack reclamation

```
repeat
    if $\exists v : \forall v' \ and \ v \rightarrow v'$, $v$ has received $slack(v')$, then
        for all $f : f$ is a CPU frequency do
            Set $slack_f(v) = min_{v'}\{slack_f(v')\} - t_f(v)$ , { where $t_f(v)$ is the estimated execution time of the computation of $v$ at frequency $f$}
            $\forall v_p : v_p \rightarrow v \ and \ v_p$ is in the same node as $v$, send $slack(v)$ to $v_p$
        end for
    end if
until all the vertices in the node have been considered
for all $v : v'' \rightarrow v$, and $v''$ is not in this node, do
    $f_{choose}(v) \leftarrow min\{f\} s.t. slack_f(v) \geq 0$
end for
Set the new frequency, $f_{new} = max\{f_{choose}(v)\}$
$\forall v : v'' \rightarrow v$, and $v''$ is not in this node, send $slack_{f_{new}}(v)$ to $v''$
```

4.3 Workload-Aware Dynamic Resource Reclaiming

To conserve energy consumption in overprovisioned nodes, we design a distributed protocol that explores energy-performance tradeoffs in a distributed system through resource reclaiming – i.e., recovering any resource from the system to the extent that it does not affect the performance, and hence the quality, of the application. Many such resources, already studied in standalone systems, can be identified for this purpose, and used in a distributed context. Such resources include peripheral interfaces like storage (via sleep modes), memory (via switching off banks), and CPU (dynamic voltage/frequency scaling). In this paper, we demonstrate this approach with the CPU-based techniques.

The primary constraint in implementing single-platform energy management techniques over a distributed context lies in determining how actions in one node affect others. The distributed protocol used to address this in MSO follows a greedy approach, i.e., each node attempts to reclaim as much of the resources as it can to minimize energy consumption, then distributes the remaining opportunities to other nodes. Event sources in MSO associate a deadline for completely processing each event, and send it along with the event itself (where this is not available/known, the event inter-arrival period is used). As the processed event finally reaches the sink, the slack of the event, i.e., the time difference between the deadline of the event processing and the actual time of completion, is computed. A positive slack value serves as a measure of overprovisioning, in that unnecessary effort was expended in processing the event. This presents an opportunity to 'reclaim' the slack by scaling down the voltage/frequency in some/all of the CPUs involved in the event processing, and save energy in the process. An implicit assumption made here is that the nodes' clocks are synchronized.

Slack reclaiming starts at the sink node, since only it can compute the slack value. Starting from this node, each node, on obtaining the slack values from all its downstream nodes (a node n_1 is downstream to n_2 if $\exists v_1, v_2 : v_1$ is assigned to n_1, v_2 to n_2, and $\exists$ a directed edge from v_2 to v_1), attempts to scale down its own CPU frequency/voltage so as to maximize energy savings. Next, it computes the slack available to each of its upstream nodes and sends them these values. Thus, the algorithm proceeds backwards, starting at the sink node(s), and propagated towards the source(s).

Within each node, Algorithm 4 is used to determine its best CPU frequency. Figure 4 illustrates this procedure, with the slacks along the edges represented by S_x's and the execution time of the computation at each vertex represented by t_x's, and shows how the slack is "consumed" by each node and the remainder passed on to vertices upstream.

Algorithm 4 scales linearly with the number of vertices mapped on that node and the number of frequencies available. Since this is a greedy algorithm, its focus is to quickly reclaim any slack made available, hence it seeks only local optima. As a result, the closer a node is to the sink, the greater will be its energy savings. Fairer methods for slack reclamation would require additional coordination among MSO nodes, thereby introducing additional protocol complexity.

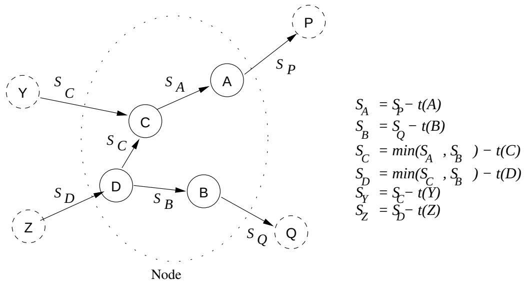


Figure 4. Slack propagation within a node

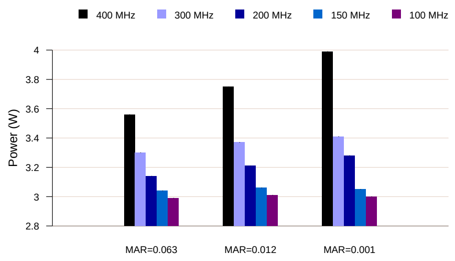


Figure 5. Sitsang Active Platform Power Consumption

The main assumption in this algorithm is that the expected execution time at each frequency for the vertices can be estimated. Research efforts in workload characterization can be used to perform such estimates [4]. Even with accurate application characterization, factors like network jitter, mobility related effects, or even changing application needs can cause changes in slack availability. The response time to such events primarily depends on the frequency at which slack reclamation is initiated. Additionally, since a side effect of the greedy algorithm results in concentrating the bulk of reclaimed slack closer to the sink nodes, this also enables quick reversion of the reclaimed slack, during such conditions.

5 Power Tradeoffs and System Implications

In order to effectively manipulate energy usage amongst distributed nodes, it is essential for MSO to understand the power and performance tradeoffs of the underlying hardware. In this section, we present results from detailed power measurements of our evaluation platform which provide the intuition and tradeoffs that drive the system's power management policies.

The hardware environment used in our experiments is the Intel Sitsang platform, with a PXA255 processor, and 64MB RAM. It runs the Linux-2.4.19 kernel, modified with Xscale and platform specific patches. Each node also has an 802.11b wireless interface, in ad-hoc mode, in addition to a 10Mbps base-T ethernet interface. All power measurements are performed using a Tektronix TDS5104B oscilloscope, Tektronix TCP202 current probes, and Tektronix P6139A voltage probes.

5.1 Platform Power Trends and DVFS

The Sitsang platform is designed around a PXA255 XScale processor. The processor supports frequency and voltage scaling via multiple operating points that vary CPU frequency as well as the frequency to the internal PXA bus, thereby affecting latency to memory and I/O devices. The core frequencies available are 400MHz, 300MHz, 200MHz, 150MHz, and 100MHz. Though multiple bus frequencies are plausible for certain core frequencies, for the purposes of analysis and experiments in this paper, we always utilize the maximum bus speed possible for a given core speed. This results in five operating points with core/bus frequencies of 400/200, 300/100, 200/100, 150/50, and 100/50. The voltages used are 1.3V for 400MHz, 1.1V for 300MHz, and 1.0V for all other frequencies[1].

Power analysis uses a tunable synthetic workload that has characteristics similar to the robotics applications used in our MSO research [28]. Specifically, for these applications, we find that a critial variation in their workloads is memory access behavior. Software components such as Bayesian classifier are CPU bound, where performance scales with frequency, whereas image analysis like blob finding displays increased memory activity due to footprints larger than the 32K cache size on the PXA255 processor. In order to provide a fair comparison across this attribute, we have developed a synthetic workload that can be tuned to vary memory boundedness while maintaining the amount of work (i.e. instruction count) performed. This benchmark is used in subsequent evaluations.

When completely idle, the Sitsang consumes 2.5W-2.64W depending upon the operating point to which it is set. A more significant variation can be observed between the different frequencies when active, as illustrated in Figure 5. The figure provides power data when the platform is one hundred percent utilized and executing workloads with varying memory accesses per instruction (MAR), a parameter that in turn affects the cycles per instruction (CPI) required to execute each application. As expected, we see decreasing power consumption as frequency is reduced, with the difference between 400MHz and 100MHz being as high as 25%. These system-level trends underscore the possibilities of energy savings available via DVFS. We also observe from the figure that the system power is not only a function of frequency, but can also vary significantly based upon workload characteristics. Indeed, at 400MHz, the power varies by as much as 11% between the different applications. This highlights the potential benefits and necessity

[1] The PXA255 electrical specifications prescribe the operating points used, along with the 1V minimum requirement.

of online monitoring in MSO to dynamically tune energy management for application specific behavior.

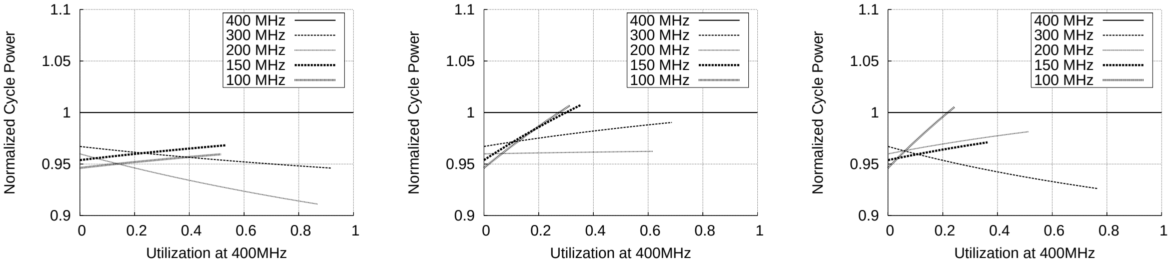
Though it is clear from our results in Figure 5 that reducing the operating point of our platform, when possible from a performance standpoint, can reduce power consumption, the resulting energy savings are not quite as clear. For periodic applications, frequency can be reduced when there is slack without a performance penalty. This reduction increases the active portion of a period while reducing the idle period. As recent work has shown, reducing processor frequency may result in reduced power consumption during active portions of the period, but it can also increase energy consumption after some point of slack reclaiming [19, 11]. The existence of these counterintuitive trends can be affected further when there are other power management schemes with which DVFS must coexist [7, 21]. We consider these trends by obtaining the cycle energy, the combination of the active and idle energy signatures in a period, of the three MAR varying applications across different CPU utilizations in Figure 6.

Figure 6(a) illustrates the tradeoffs of the different operating points across utilization behavior for a memory-bound workload with resultingly high CPI. As the utilization increases, it becomes infeasible to execute at certain frequencies until eventually only the highest operating point can maintain the performance of the application. Since the application is memory-bound, the performance of reduced frequencies can closely match those of higher frequencies, especially when the bus frequency can be maintained. This is exibited between the 300/100 and 200/100 operating points as well as the 150/50 and 100/50 frequencies by the fact that the respective energy curves end at about the same utilization (i.e., the modes become unusable at similar load). The reason for this is that the performance of the application is driven by bus frequency instead of core frequency due to the high memory access rate. We can see from the figure that the optimal operating point is only the smallest one possible for very low utilizations, after which 200MHz is optimal even though 150MHz and 100MHz would be options as well. Similar inflection points can also be observed for lower MARs in Figures 6(b) and 6(c), though in the latter the optimal operating point gets pushed even further to 300MHz at high utilizations. *These trends show that the energy optimal operating point can vary based upon workload characteristics as well as the utilization required to execute the workload.* It should be noted that even simply monitoring MAR is not adequate (our own results show data stalls per cycle should also be monitored), but a full list of required attributes can be obtained via existing workload characterization research [3] and falls outside the scope of this paper. These results just highlight the need for application-specific data, and how MSO can benefit from obtaining this data accurately and dynamically online.

The platform power trends discovered in our experiments directly affect the DVFS-based energy management in MSOs. First, they show that when utilizing DVFS to dynamically tune energy behavior via slack reclaiming of periodic applications, the middleware must monitor resource utilization information for software components so that it can be aware of where the system is located along the utilization curve, thereby determining the minimum operating point to utilize at a particular node. Second, with online monitoring, MSO can also determine the performance scalability of an application by determining metrics such as MAR and the resulting data stalls per cycle. By coupling this information with platform power characteristics, MSO can more readily determine application specific inflection points at runtime than can be done by static policies.

5.2 Offloading and Wireless Communication Overheads

In addition to platform energy savings with respect to utilizing frequency scaling, our MSO approach also exploits cooperative systems by offloading computations to take advantage of remote resources and energy reserves. This type of offloading has been shown to provide significant system lifetime benefits in our previous work [22]. Here, we continue to leverage this type of energy management by considering the possibility of migrating software components in MSOs during reallocations. A question that arises, however, is how the communication energy overheads compare to the benefits of offloading. To obtain insight into this tradeoff, we stream data between two Sitsang platforms over a wireless link. We then monitor the system, CPU, and radio power of one of the systems at different data rates of UDP/IP. These experiments result in the following findings. First, we find that the link becomes saturated at 4Mbps. At this extreme, system power consumption increased by 300mW, while the radio power signature is only elevated by 90mW. The CPU power signature explains this discrepancy, as we observe that the processor is consuming active power during 25% of the time due to packet processing overheads. Therefore, the majority of the power increase can be captured by simply monitoring system utilization. The reason for the minimal increase in radio power consumption even at high link utilizations is that in ad-hoc mode, the radio cannot be placed into a sleep state. Therefore, it is always in a promiscuous read mode, the power signature of which varies little from sending. Since the radio power consumption changes negligibly with use, the overheads of utilizing it, for the sake of our flowgraphs with little communication utilization, can be effectively ignored. Due to these results, in MSO energy management, we only consider the computational overheads of software components when performing energy load balancing.



(a) MAR=0.063

(b) MAR=0.012

(c) MAR=0.001

Figure 6. Platform Cycle Energy vs. CPU Utilization

6 Experimental Evaluation

6.1 Implementation

The approaches described in this paper are implemented in the MSO middleware. This middleware is written entirely in C/C++, and it is implemented using an overlay construction toolkit, termed EVPath [6], a successor of the ECho event-based publish-subscribe system. EVPath allows the creation of entities called '*stones*' that operate on events. Stones can be linked via a variety of network protocols, including UDP, TCP, etc. EVPath also provides a SOAP interface using gSOAP [8] to enable remote management of stones. In addition, MSO allows its own functions to be called remotely via gSOAP-based calls. Event data is described using PBIO, an efficient portable binary format Kernel AODV [13] is used for ad-hoc routing, but MSO is independent of the underlying routing protocol.

Experiments are performed with the Sitsang handheld devices described earlier and with the MobiEmu [33] emulation of mobility on a wireless network. To estimate the system power consumption at each node, we run a daemon that periodically (at 0.1 second intervals) monitors CPU usage, then computes the energy consumed based on the current frequency and with an application dependent power model. The power model is obtained by power measurements performed using the oscilloscopes on the instrumented node.

6.2 Reallocation

Our first set of experiments under reallocation is done on two Sitsang nodes executing a single application, such that only one node is running at any time, with the other node being idle. By monitoring each others' battery levels, the nodes can cooperatively run the application so that they maximize their battery lifetime. The polling frequency for monitoring, as well as the threshold selected for offloading, influence the performance, as seen in Fig 7. As we increase the frequency of reallocation, the difference in energy levels between the nodes decreases, but this also indicates that more time was devoted in performing the reallocations than performing the actual work. The ideal range for our scenario lies at 20-30%, yielding about 16% increase in battery lifetime, when compared to a single node running the entire workload.

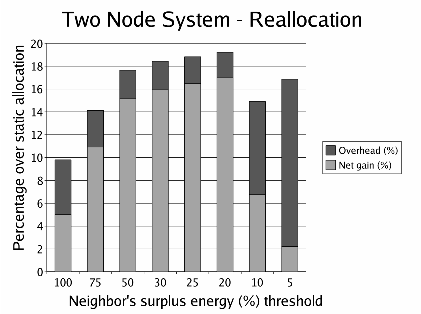


Figure 7. Reallocation in Two Nodes

Next, we study an example on a five node testbed, with the topology shown in Fig 9. The application flow graph is chosen to accommodate all three different kinds of event flow combinations possible, i.e., linear, split, and join flows. Each node runs an instance of MSO, and begins with a fixed energy, thus simulating a battery. Using the insights gained from the previous scenario, we find the ideal threshold for reallocation decisions to be about 50J, and the frequency of polling (for monitoring neighbors' energy levels) is chosen to be 25 sec. All the application components are CPU intensive, and while two of them run at 80% CPU utilization, the other three run at 15%. Such a workload is representative of applications that perform different pipelined processing on data. Events of size 4KB are sent via a single source at the rate of 1 per 1.5 seconds. We discuss the benefits of using MSO along two parameters:- (i) the system lifetime of a cooperating group and (ii) the parity in the lifetimes of the nodes forming the group. An example of the first type of requirement is in collaborative tasks that require participation

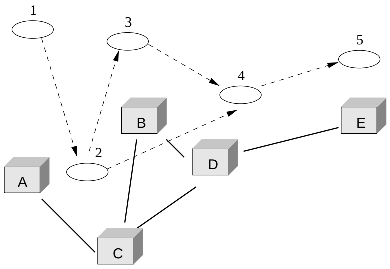


Figure 9. Network topology for experiments

of all mobile nodes. The second rule can be useful in enforcing a uniform policy for all participants in the task. For our purposes, we term the lifetime of the first dying entity in the group as the system lifetime, and quantify lifetime parity by measuring the standard deviation of individual node lifetimes. We observe the trend of these metrics, as the initial energy available with the nodes are varied. The energy values we use here (1kJ-5kJ) represent typical capacities of batteries used in current generation platforms like the Sitsangs. The results are presented in Fig 8, where we compare our reallocation with a static assignment. The difference in the lifetimes afforded by these strategies increase, as the initial energy increases, with the differences being close to 10% at 5kJ. This is a consequence of the fact that the lifetime of the node(s) executing the computationally intensive components of the application flow graph exhibits a linear relationship with the initial energy. However, as reallocation shifts the heavy computation among all the nodes, this effect is mitigated. For the same reason, the lifetime parity with reallocation shows no particular trend with increasing initial energy, as opposed to the widening gap in node lifetimes observed with a static allocation.

6.3 Dynamic Resource Reclaiming

The next set of experiments evaluate the dynamic resource reclaiming algorithm, over a synthetic workload. As discussed previously, we apply this technique to source-defined event slack available at the application sink node. The setup for the experiment consists of five Sitsang nodes in the same configuration as the previous study. Each vertex executes a memory-intensive synthetic application.

The application is run at three different scenarios, corresponding to having a CPU utilization of 40%, 60% and 80%. This is performed by increasing the duration of execution of each event in the application. Events of size 8 KB are sent with a periodicity of 2.5 seconds. The execution time of each event, when run at 80% utilization and the highest frequency (400 MHz) is found to be roughly 2 seconds at each node. This is purposefully chosen so as to exclude noise effects and other sources of error.

The resulting energy consumption of the application, in the presence of our resource reclaiming algorithm is measured. This figure is compared against an optimal choice of frequencies for each possible slack value, that minimizes energy consumption. The energy values are normalized against the default case where all nodes run at the highest frequency. As shown in Figure 10, the algorithm is found to closely track the most optimal settings. In some cases, especially at the top frequencies, our algorithm appears to outperform even the optimal solution, but this is only because of missed deadlines, i.e., the algorithm reclaims more slack than available, thus saving more energy but hurting performance. This is due to errors in predicting the execution time at various frequencies, and other sources of error in event delivery. For the higher utilization workload, as both the slack available, as well as the execution time are high, any errors in estimation can cause large deviations from the optimum frequency settings.

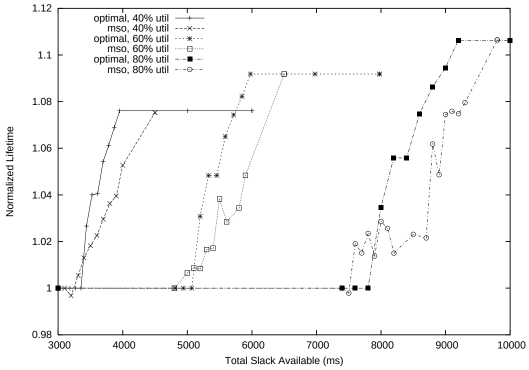
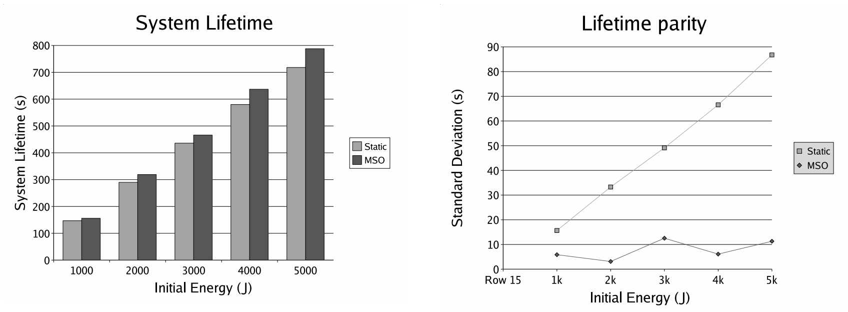


Figure 10. System energy savings with slack reclaiming

Finally, we evaluate the overheads of various strategies that can be employed in resource reclaiming. We consider three strategies in this study: (i) Aggressive, where slack is polled frequently (every 30 sec), and positive, as well as negative slacks are immediately propagated throughout the network, (ii) Conservative, which polls for slack less frequently (60 sec), and (iii) Opportunistic, where positive slacks are propagated at a low rate (60s), and negative slacks at a high rate (30s). The rationale behind opportunistic reclaiming is to conserve energy without a high overhead, but react relatively quickly when performance is hurt. The event traffic was such that all the ranges of slack values were used in the test, over a period of about 15 minutes.The results of these strategies are shown in Table 1.

The aggressive strategy is more prone to mispredictions and overcorrections, than the others, and the conservative approach can let slack available for shorter periods of time go unrecovered. The opportunistic strategy strikes a balance between these extremes, to react quickly during performance critical phases alone. Correspondingly, the number of slack reclaims also lies between these strategies.



(a) System Lifetime

(b) Lifetime Parity

Figure 8. Reallocation

Table 1. Slack Reclaiming – Strategies

Strategy	Mean slack (ms)	Mean abs(slack) (ms)	Reclaims
None	1188	1204	0
Aggressive	-23	305	1012
Conservative	59	245	528
Opportunistic	-45	215	822

7 Conclusions and Future Work

In this paper, we discuss the problem of cooperative energy management in distributed mobile systems, and present decentralized techniques for a distributed application run in a MANET environment. We perform energy aware allocation, followed by reallocation in response to changing energy conditions, and dynamic resource reclaiming in overprovisioned systems. We also evaluate the algorithms on a MANET testbed of handheld computing devices, indicating the feasibility of our approach – the resource reclaiming algorithm is found to track the optimal solution, and our algorithms for reallocation show an increase in battery lifetime of upto 10%. Further, our experiments indicate that the energy consumption of the network interface in handheld devices like the Sitsang platform form a tiny portion of the overall power consumption, underscoring the need to focus on other power hungry components of similar systems.

Future work in this area involves extending MSO to heterogeneous networks with asymmetric nodes, by including mobile phones, laptops and other portable computing devices. We are also currently evaluating MSO with prototype robot devices, with the aim of studying the tradeoffs between quality of service and energy consumption for typical robotic applications. Studying the interplay between policy-enabled energy management in mobile devices, and the resulting quality of service is yet another priority.

References

[1] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Trans. on Computers*, 53(5), May 2004.

[2] D. Bruneo, M. Scarpa, A. Zaia, and A. Puliafito. Communication paradigms for mobile grid users. In *CCGrid*, 2003.

[3] M. Calzarossa and G. Serazzi. Workload characterization: A survey. In *Proc. of the IEEE*, 1993.

[4] Y. Chou, B. Fahs, and S. Abraham. Microarchitecture optimizations for exploiting memory-level parallelism. In *ISCA*, 2004.

[5] H. L. *et al*. Design and implementation of a single system image operating system for ad hoc networks. In *Mobisys*, 2005.

[6] http://http://www.cc.gatech.edu/systems/projects/EVPath/.

[7] X. Fan, C. Ellis, and A. Lebeck. The synergy between power-aware memory systems and processor voltage scaling. In *Proc. of the Workshop on Power-Aware Computer Systems (PACS)*, December 2003.

[8] http://gsoap2.sourceforge.net/.

[9] N. Gupta and S. Das. Energy-aware on-demand routing for mobile ad hoc networks. In *Workshop on Distr. Comp., Mobile and Wireless Comp.*, 2002.

[10] C. Isci and M. Martonosi. Phase characterization for power: Evaluating control-flow-based and event-counter-based techniques. In *HPCA*, February 2006.

[11] R. Jejurikar and R. Gupta. Dynamic voltage scaling for system-wide energy minimization in real-time embedded systems. In *ISLPED*, August 2004.

[12] J. Jennings, G. Whelan, and W. Evans. Cooperative search and rescue with a team of mobile robots. In *ICAR*, 1997.

[13] http://w3.antd.nist.gov/wctg/aodv-kernel/.

[14] R. e. a. Kumar. Dfuse: A framework for distributed data fusion. In *ACM Sensys*, 2003.

[15] J. Luo and N. Jha. Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems. In *ICCAD*, 2000.

[16] C. Mascolo, L. Capra, and W. Emmerich. Mobile computing middleware. *Advanced Lectures on Networking, LNCS*, 2002.

[17] M. *et al*.. Mecella. Workpad: an adaptive peer-to-peer software infrastructure for supporting collaborative work of human operators in emergency/disaster scenarios. In *IEEE CTS*, May 2006.

[18] A. e. a. Messer. Towards a distributed platform for resource-constrained devices. In *ICDCS*, 2002.

[19] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar. Critical power slope: Understanding the runtime effects of frequency scaling. In *Supercomputing*, June 2002.

[20] B. e. Mochocki. Network-aware dynamic voltage and frequency scaling. In *RTAS*, 2007.

[21] R. Nathuji, K. O'Hara, K. Schwan, and T. Balch. Compatpm: Enabling energy efficient multimedia workloads for distributed mobile platforms. In *MMCN*, 2007.

[22] K. O'Hara, R. Nathuji, H. Raj, K. Schwan, and T. Balch. Autopower: Toward energy-aware software systems for distributed mobile robots. In *ICRA*, 2006.

[23] T. e. a. Pering. Coolspots: Reducing the power consumption of wireless mobile devices with multiple radio interfaces. In *Mobisys*, 2006.

[24] C. Perkins and E. Royer. Ad hoc on-demand distance vector routing. In *Proc. IEEE WMCSA*, 1999.

[25] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath. *Dynamic cluster reconfiguration for power and performance*. 2003.

[26] C. Prehofer and C. Bettstetter. Self-organization in communication networks: principles and design paradigms. *IEEE Communications Magazine*, 2005.

[27] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, 2001.

[28] B. Seshasayee and K. Schwan. Mobile service overlays: Reconfigurable middleware for manets. In *MobiShare*, 2006.

[29] R. Shah and J. Rabaey. Energy-aware routing for low energy ad hoc sensor networks. In *WCNC*, 2002.

[30] C. Shen, K. Ramamritham, and J. Stankovic. Resource reclaiming in multiprocessor real-time systems. In *IEEE Trans. Parallel and Distributed Systems*, 1993.

[31] R. Xu, D. Zhu, C. Rusu, R. Melhem, and D. Mosse. Energy efficient policies for embedded clusters. In *LCTES*, 2005.

[32] J. Xue, P. Stuedi, and G. Alonso. Asap: an adaptive qos protocol for mobile ad hoc networks. In *PIMRC*, 2003.

[33] Y. Zhang and W. Li. An integrated environment for testing mobile ad-hoc networks. In *MobiHoc*, 2002.