Triage: An Architecture for Wireless Microservers

Nilanjan Banerjee Jacob Sorber Mark D. Corner Sami Rollins † Deepak Ganesan

Department of Computer Science
University of Massachusetts, Amherst, MA
{nilanb, sorber, mcorner, dganesan}@cs.umass.edu

†Department of Computer Science
Mount Holyoke College, South Hadley, MA
srollins@mtholyoke.edu

Abstract—
The ease of deployment of wireless and mobile systems is pushing the network edge far from powered infrastructures. A primary challenge in building untethered systems is offering properties, such as scalability and security, normally provided by a powered server. *Microservers* are battery-powered in-network nodes that play the same role as a traditional server: processing data from clients, aggregating data, and providing responses to queries. Providing these services can be extremely energy intensive; however, it is crucial that the microserver remain energy efficient, as increased energy consumption translates into a larger battery or shorter lifetime.

This paper presents *Triage*, a tiered hardware and software architecture for microservers. Triage reduces the energy usage of a microserver by combining a high-power resource-rich platform and a low-power resource-constrained platform. The low-power platform can remain always-on to receive, buffer, and filter requests from the network while the high-power platform remains in a power saving mode. To maximize the amount of time the high-power platform remains in a power saving mode, the low-power platform delays execution of requests, caches recent results, and performs tasks locally when possible. We evaluate three services: storage, network routing, and query processing. In a medium-scale video sensor network, our system achieves a battery lifetime *five times* longer than a current, non-tiered design.

I. Introduction

The ease of deployment of wireless and mobile systems is pushing the network edge far from powered infrastructures. Untethered, multi-hop networks support a wide range of applications from wildlife habitat monitoring [23] and security surveillance [21], to space exploration and disaster management [7]. Such applications require the development of highly efficient, long-lived, and low-cost mobile and wireless systems.

A primary challenge in building untethered systems is offering properties, such as scalability, security, and privacy, normally provided by a powered server. A fully autonomous system must rely on in-network components to provide the computation and storage required to support future applications. Moreover, such a system must provide network-wide coordination to ensure energy-efficient communication and operation.

Hierarchical network architectures, such as the example shown in Figure 1, promise to balance functionality and efficiency in wireless systems. A hierarchical system combines both resource-constrained small devices, such as Motes [27] with resource-rich *microservers*. A sensor network, for instance, can incorporate a mix of large nodes to perform advanced processing and storage and small nodes for expanding coverage. Similarly, a combination of small and large devices in mobile networks can yield significant benefits over using only one of the two devices [7].

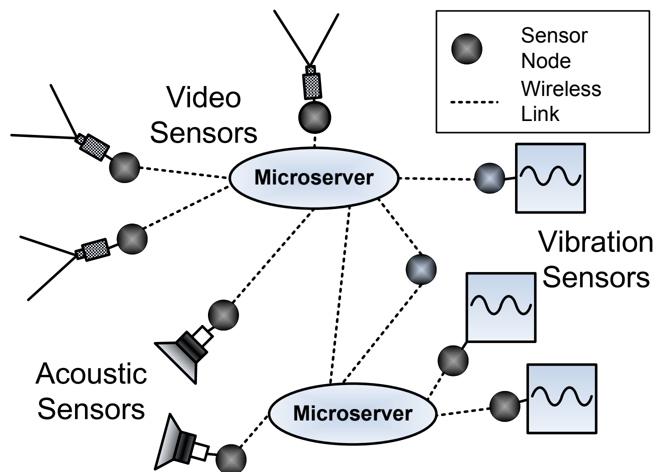

Fig. 1. Microserver Deployment

Energy efficiency is a concern for smaller nodes; however it is even more critical for microservers. Providing services with sufficient computational, networking, and storage resources can be extremely energy intensive. Currently, increasing the lifetime of a microserver requires a larger battery or intrusive solar arrays [23]. The contrast of capabilities and energy consumption found in small and large nodes highlights the inherent tension between sufficient functionality and lifetime.

The goal of this work is to balance functionality and lifetime of microservers by introducing a new architecture,

Triage. Triage reduces the energy usage of a microserver by employing a tiered hardware architecture supported by intelligent software control. By combining a high-power resource-rich platform and a low-power resource-constrained platform we yield a single, tiered device which leverages the advantages of both. The low-power tier, or tier-0, can remain always-on ensuring responsiveness at minimal energy cost. The high-power tier, or tier-1, can remain in a power saving mode until its resources are required for a given service. The software architecture leverages the fact that tier-0 can offer the same services that tier-1 offers and determines how to efficiently utilize the resources available across the tiers to reduce total energy usage.

Triage employs three key techniques to reduce energy usage. First, Triage amortizes the energy and latency costs of waking tier-1 by delaying the execution of tasks that require tier-1 resources. Second, it reduces the amount of work tier-1 must perform by caching recent results at tier-0 and servicing external requests from the cache when possible. Third, it offloads work from tier-1 by using tier-0 to provide services that require only the resources available at tier-0. For each service the microserver provides, tier-0 runs a *surrogate* that receives requests for the service and determines where the request should be executed. Triage inserts and optimizes delayed requests in a *log* that it also uses as a cache to provide low-latency responses. A *dispatcher* monitors the log and determines when it is necessary to wake tier-1.

Triage provides a general mechanism for building microservers suitable for many untethered applications, including sensor applications, mobile networking, and pervasive computing. To support development, we have created a working prototype that targets the common functionality required in such scenarios, including storage, routing, and query processing. We provide three surrogates that implement this common functionality and common insights necessary to construct a larger library of surrogates.

We also show the results of a medium scale deployment of a multi-hop, video sensor network involving six low-power camera nodes, 24 routing nodes, and two Triage microservers. In this scenario Triage reduces the microservers' energy consumption over a current design by 80%, translating into five times the battery lifetime, or an energy source one-fifth the size. We also present several other experiments that highlight the specific advantages of Triage and discuss areas for future improvement.

II. Tiered Hardware

Generally, platforms with more resources (e.g., processing and memory) can complete tasks more quickly as well as complete more resource intensive tasks. However, resources come at the cost of greater power requirements. The Triage tiered hardware platform addresses this dichotomy by combining two or more tightly coupled but independently operating embedded subsystems, as shown in Figure 2. The upper tier is strictly more capable and power-hungry than lower tier. This means that any task the lower-tier system can do the higher-tier system can also perform. Some hardware platforms such as Turducken [34], and the PASTA sensor node [32] employ a similar hierarchical structure, but are not focused on building an energy efficient microserver platform.

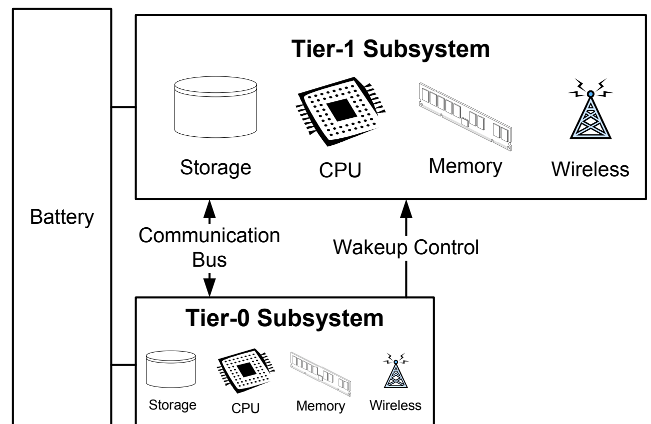

Fig. 2. A Tiered Hardware Platform

In Triage, the two tiers are tightly coupled and directly communicate over a wired link. This enables the lower tier to trigger the wake-up of the higher tier when necessary. Also, while the higher tier is not in use it can be shutdown, suspended, or hibernated to save power. This technique is effective for a platform containing subsystems that are separated in power consumption and capabilities by an order of magnitude or more. Compared to scaling methods, such as DVFS [15], this tiered approach provides a much greater range of useful power states.

One advantage of this technique is that constructing the hardware is straightforward. All that is necessary is to connect together available, low-power, hardware platforms that have the right capabilities. For a long-lived embedded microserver, there are two well-optimized, commercially available, hardware platforms that provide the right mix of power consumption and resources: the Stargate [36] and the TelosB mote [26]. The Stargate is similar to the internal components of a PDA. It contains a 32-bit, 400MHz

PXA255 XScale processor, 64 MB of RAM, 32 MB of internal flash, and a slot for a WiFi card. It draws between 300 and 1800 mW of power, roughly one-tenth the power of currently available laptops. This platform is well suited to a wide range of query processing tasks, including image processing, encryption, and storage management. The TelosB mote is a third-generation sensor platform, containing an 8-bit, 8 MHz microcontroller, 10kB of RAM, 1 MB of external flash, and an 802.15.4 radio. The TelosB consumes between 20 and 120 mW of power, less than one-tenth the power of the Stargate. This platform works well for always-on operation, simple packet processing, and providing low-latency responses.

While exact technology trends are difficult to predict, we expect that embedded platforms will continue to follow the trends found in general computing systems: they will become more efficient, contain more resources, and cost less than current components. However, we also expect that platforms with order-of-magnitude differences in storage and computational resources, will continue to have order-of-magnitude differences in power consumption. This is the fundamental property that the Triage microserver is based upon, not the particular components found in our current choice of tiers.

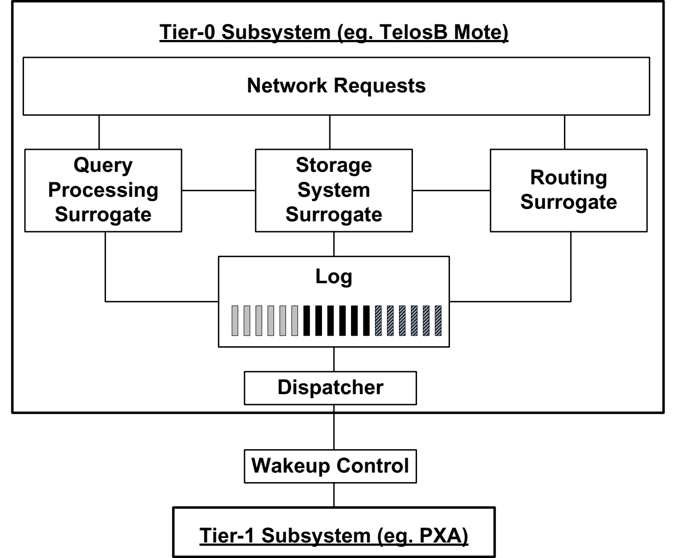

Fig. 3. Microserver Software Architecture

III. Software Architecture

The goal of Triage is to support energy-efficient utilization of a tiered hardware platform. Our design focuses on a platform with two tiers; tier-0 is a very low-power platform and tier-1 is a more capable and higher-power platform. Triage keeps tier-1 in a low-power state whenever possible by using tier-0 to delay execution of requests, service requests using cached results, and route tasks to the most appropriate tier. The locus of control is centered in tier-0 which remains in an always-on mode, maintaining high availability, and receiving requests from client nodes.

Figure 3 illustrates the components of the software architecture. Tier-0 virtualizes resources available on tier-1 using a collection of *surrogates*. While the primary software component for each service is deployed on tier-1, tier-0 surrogates may respond to requests that can be processed locally. However, if a request requires the physical resources of tier-1, the surrogate must insert the request into a *log* maintained in the local storage of tier-0. A *dispatcher* monitors the log and decides when to wake the tier-1 system and dispatch pending requests.

A. Surrogates

Surrogates are small software modules running on tier-0 which provide a network service such as storage or routing. Because tier-0 does not have the resources to execute every request, surrogates also virtualize the resources available at tier-1. For each request, the surrogate may provide the requested service using local resources or delay execution of the request until the tier-1 resources are available.

When a surrogate receives a request, it first determines whether it can execute the request using local resources. Because the surrogate runs on a low-power platform, it is limited in its ability to dynamically analyze the resource requirements of every request. For example, encryption can be done on either platform; however, the length of the encryption and the algorithm used determines where it should be completed—in general this type of dynamic analysis is known to be difficult. Therefore, our current model uses a static set of rules for determining where requests should be processed.

Surrogates rely on the following set of guidelines for request execution rules: (1) execute requests for information already cached at tier-0; (2) execute requests that require a small amount of processing on tier-0, for example a query that only involves metadata; (3) execute requests that require only tier-0 resources, for example requests to route packets destined for receivers reachable by the tier-0 radio; and (4) dispatch any other request to tier-1. It is the job of the developer to specify how the surrogate determines the complexity and requirements of a request; for example, a surrogate may infer this information by the type of the request.

B. Delayed Request Log

Tier-0 keeps a *log* that contains requests that must be dispatched to tier-1. If a surrogate determines that a re-

quest should not be executed at tier-0, it writes it into the log. Each log entry contains an identifier for the tier-1 service that will execute the request, relevant request parameters, and any data accompanying the request. For instance, a routing request would contain an identifier for a tier-1 routing service, the address of the receiver, and the data packet itself.

Efficient management of the log is key to extending the lifetime of the system. When the log fills, tier-0 has no choice but to wake tier-1 and play the log. By reducing the size of the log using cancellation optimizations, tier-0 can defer wakeup of tier-1 and increase overall system lifetime. For example, if the log contains a database insertion and a new insertion overwrites it, the original insertion can be removed from the log.

The log can also be used as a mechanism to virtualize resources available at tier-1, thus enabling the system to keep tier-1 in a low-power state longer. Surrogates running on tier-0 can use space in the log as a cache in order to service requests locally that would otherwise require tier-1. This functionality is particularly useful in storage applications; a read closely following a write to the same data can be serviced from the cache. In order to maximize the amount of cached data, Triage does not erase the tier-0 log when a batch of requests is played at tier-1. Instead, the previously committed log entries and cached results are lazily overwritten by new requests using an LRU eviction policy.

C. Dispatcher

A separate operating system service, the dispatcher, periodically inspects the log to determine if it needs to wake tier-1. There are two cases when the dispatcher must wake tier-1 and dispatch outstanding requests. The first case occurs when the log becomes full. In this case, the dispatcher is automatically invoked by the log storage system; it wakes tier-1 and dispatches each outstanding request to the appropriate service. The second case occurs when a request contains an expired quality of service constraint.

Depending on the storage available at tier-0 and the arrival rate of requests, some requests may be deferred for unacceptably long periods of time. To support time sensitive applications, the dispatcher will wake tier-1 if it discovers a request that contains a quality of service constraint that has expired. Surrogates can write constraints into the log along with the description of the resource request.

Currently, Triage supports one constraint: a maximum queuing time, which is the maximum amount of time that Triage will delay execution for a particular request. While the maximum queuing time does not provide a completion deadline, it provides sufficient guarantees for the applications we envision. Generally, the queuing time at tier-0 is the dominant factor in the response time of the server—tier-0 may delay tasks for up to several minutes, a delay much larger than the processing time for most tasks. Moreover, nothing in our design precludes further enhancements for greater realtime control by predicting task execution time.

D. Surrogate Composition

Many applications require the functionality of several surrogates. For instance, a client may query the microserver for information, and request that the results of the query be sent to another node. This requires a combination of a storage surrogate as well as a routing surrogate. To enable applications to compose the functionality of several surrogates, we provide a surrogate composition mechanism.

The composition is analogous to the design of microkernel operating systems [1]: surrogates communicate with one another using communication primitives provided by the operating system. However, in this case there is no user application space, all services running on the microserver are surrogates, and a composition of surrogates is created by deploying a new surrogate that utilizes the others.

Two types of inter-surrogate communications are necessary. The first is a direct function-call, or event based mechanism. The second is a delayed parameter passing mechanism based on the concept of a Future [19]. To understand the necessity of this mechanism, consider a query processing surrogate that receives a request for all images fitting some description. The query processing surrogate can simply decompose the request into two subrequests, a storage read request and a network routing request, and insert each request into the log. However, the network routing request must include the data which is returned from the storage read. This occurs when the data arrives at tier-0 or when Triage wakes tier-1.

To enable this scenario, a surrogate writes a *future* into the log (e.g., a read request) which acts as a placeholder for the result. Another surrogate (e.g., the network routing surrogate) references the pending result by using a *claim*. When tier-1 reads this sequence of operations from the log, it interprets the future and claim from each subrequest, and completes the entire request. Alternatively, if the data arrives at tier-0 before Triage wakes tier-1, the future can be fulfilled immediately, and the claim can be completed without waking tier-1.

IV. Example Surrogates

The most common, and basic, functions found in servers for sensor networking, mobile networking, and pervasive

computing are routing, storage, and query processing. To this end we present three example surrogates: a storage system surrogate, a network routing surrogate, and a query processing surrogate. As untethered networks mature, this library of surrogates will be expanded, enhanced, and further optimized.

The storage surrogate receives requests for the tier-1 storage system and serves data from the cache when possible. The routing surrogate also receives requests destined for tier-1. However, it provides additional functionality by accepting requests to route data over the tier-0 radio. Finally, the query processing surrogate provides complex processing, such as image processing, and also relies on other surrogates to access and deliver stored data. This demonstrates the use of Triage's Future-based composition method.

A. *Storage System Surrogate*

The storage surrogate enables efficient in-network storage applications [11] by using delayed execution and caching to amortize the energy cost of performing read and write operations on the tier-1 storage system. It provides a simplified file system interface and employs batching and log optimization to reduce the frequency with which tier-1 is woken. In addition, it maintains a log of recent operations and will service requests from this cache when possible.

Because Triage aggressively suspends or powers down tier-1, the full storage system is typically unavailable. Tier-0 maintains a log of recent write requests, read requests, metadata updates, and also a cache of recently read results. The use of logging is similar to a Log Structured File System [30] and the split of the log and full storage system is similar to those found in distributed file systems such as Coda [31].

When a remote node sends a write request to the microserver, it logs the request and data. Read and delete requests are logged in the same way. However, for each read request, the system consults a log index to determine whether or not it can provide the data immediately from the log. If it cannot, the read is delayed until the next time Triage wakes tier-1.

As with each of the surrogates, a client can provide a maximum queuing delay parameter. In the case of reads, this guarantees that the read request will be delayed no longer than the given amount of time. If a read request does not provide this parameter, Triage defers the request until its log fills or until the timeliness constraint of another task forces tier-1 to be woken. For writes, any maximum queuing delay parameter can be ignored. Even though tier-0 does not immediately commit updates to the tier-1 storage system, the system still provides write-read consistency since the written data still exists in the log. Any subsequent reads on the same data will be fulfilled by previous writes in the log.

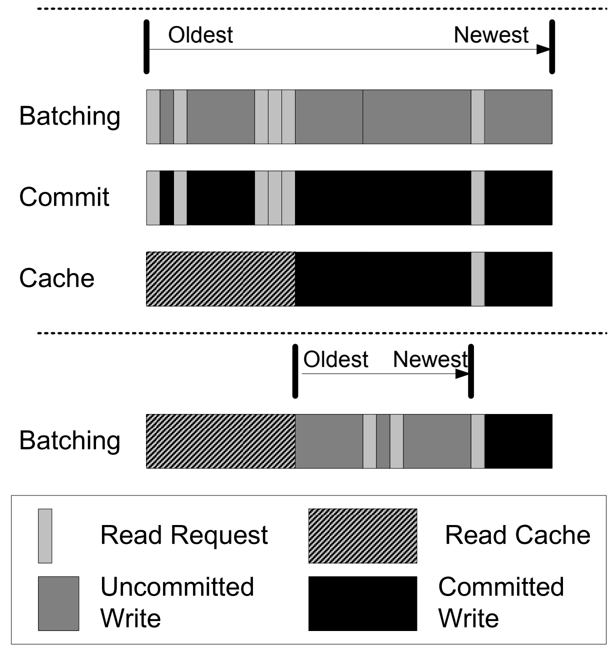

Fig. 4. Storage System Log Management

For each request, the surrogate attempts to optimize the log. A write or delete request for a particular file cancels any previously logged writes for the same file. The results of read requests and recently committed writes remain in the log and serve as a cache for future requests.

This data cache is managed using an approximation of the *least recently used* eviction policy. For writes, the time of use is considered to be when the microserver received the request—writes can be considered completed when Triage writes them to its stable storage. For reads, the choice is less clear: we could chose the time that the read request arrives, or the time it is completed by the system. Without a clear disadvantage, we chose that latter, as it is more convenient for avoiding fragmentation issues. To see why, refer to the process demonstrated in Figure 4, which shows the surrogate's logical view of the log—data from other surrogates may be interspersed with storage system data. During the batching phase, the log accumulates read requests along with uncommitted writes. During the commit phase the dispatcher wakes tier-1 and commits all of the writes. Read results from tier-1 are passed to the surrogate, which writes them into the log. We chose the simpler policy that favors these reads as *most recent*. Read results are therefore written into the cache immediately preceding the most recent batch of writes. As space becomes scarce,

the oldest part of the cache is the first to be overwritten by new requests.

B. Network Routing Surrogate

The network routing surrogate enables efficient routing by delaying execution of tier-1 routing requests and executing routing requests locally, using the tier-0 radio, when possible. A key benefit to using a tiered architecture is the availability of multiple network interfaces each with varying bandwidth, range, latency, reliability, and protocol features. Wireless radios found in less powerful tiers generally have a lower bitrate, shorter range, and simpler communications stacks. For example, variants of the 802.15.4 standard deliver data rates up to 200 kbps and operate with a few tens of milli-watts of power. More powerful tiers incorporate higher bitrate radios capable of providing more advanced software protocol stacks such as TCP/IP, using network interfaces such as WiFi, GPRS, and WiMax; however, their power draw is much higher.

We have designed a network surrogate to operate in a static, multi-hop wireless network. An example network is shown in Figure 5. The surrogate can operate on each packet in two modes: *minimum energy* or *minimum latency* routing. Minimum energy routing is most appropriate for bulk transfers, whereas minimum latency routing is most appropriate for small messages, such as metadata updates, route maintenance, and other management functions.

When a packet arrives at the surrogate, it examines the destination address, consults its routing table, and computes the best radio to use in order to achieve minimum cost with respect to either latency or energy. The surrogate's routing table contains a route and hop count for each destination in the network. The latency to route over the tier-1 interface is computed using an estimated average of the bandwidth of tier-1, the time to wake tier-1, and the time to transfer relevant data from tier-0 to tier-1. The latency to route over the tier-0 interface is computed using an estimated per-hop bandwidth and the number of hops required to reach the destination. The per-hop latency for the radios on tier-0 and tier-1 were measured using a separate experiment described in the evaluation section. Note that routing using tier-0 does not incur the tier-1 start-up and transfer time, but does incur the latency of additional hops. The energy cost to route over the tier-1 interface is computed using the estimated energy cost to wake tier-1 plus the estimated energy cost to transfer the data using the tier-1 interface. The energy cost to route over the tier-0 interface is computed using the estimated energy cost to transfer the data over each hop from the source to the destination. Again, note that tier-0 will incur an energy cost for each node along the path but will not incur the cost of waking tier-1. The per-hop transmit and receive energy for tier-0 and tier-1 radios were measured using experiments which involved sending different amounts of data across a hop using both the radios . Due to the greater efficiencies (bytes/joule) of the tier-1 radio, and the inefficiencies of multi-hop routing, the tier-0 radio is only more efficient for routing relatively small amounts of data—our evaluation section explores this issue in detail.

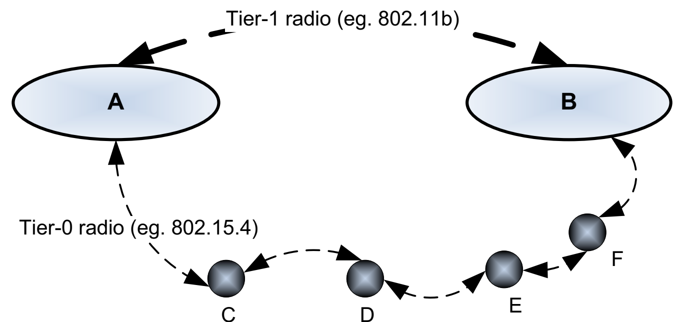

Fig. 5. Multi-hop network example. The latency to transfer data from node A to any node in the network using either radio can be computed from the radio bandwidth, multi-hop delays, number of hops, and the time to wake the necessary tier. The required energy can be computed from the number of hops, the energy cost of that radio, and the energy to wake the tier.

If the surrogate decides to route a packet using the tier-0 interface, it immediately sends the packet. If not, it inserts the packet into the log. The packet may also be accompanied by a maximum queuing time constraint. The packet will remain in the log until the log fills or until the maximum queuing time for a logged request has been reached. When tier-1 is woken, logged packets will be passed up and sent via the tier-1 network interface.

C. Query Processing Surrogate

The query processing surrogate enables efficient data processing by amortizing the cost of performing queries at tier-1, reducing the number of queries that tier-1 must process, and reducing the cost of sending relevant data to remote nodes. Performing computation on the microserver can conserve energy [5], particularly in multi-hop networks [11], or systems with large amounts of in-network data and relatively infrequent queries. The query processing surrogate executes queries over the data it has cached and delays processing of queries that require resources only available on tier-1.

The query processing surrogate provides a database-style query interface for data stored on the microserver. Clients may use simple queries, such as *retrieve all images from the last ten seconds*, or more complex queries, such as

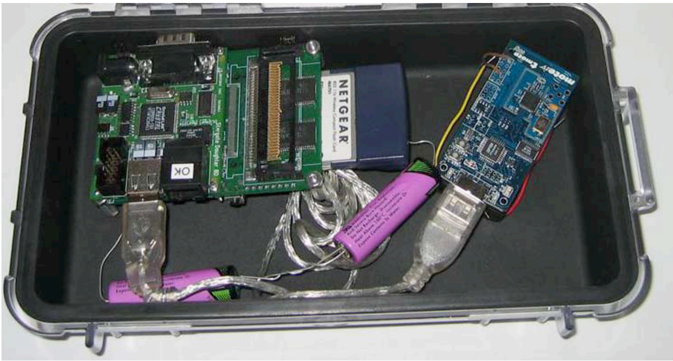

Fig. 6. Prototype Triage System

retrieve all images that contain 2 or more objects and are from a particular geographic region. The query processing surrogate uses the other surrogates to create a complex combination of resources, including processing, routing, and storage. Tier-1 can execute any query since it has access to the powerful radio, the primary storage system, and a powerful processor. However, tier-0 can perform only simple queries. For example, any query that can be performed using simple comparisons of cached metadata can be performed at the tier-0 system. It is also possible for the surrogate to decompose a query into its component operations, for example a file system read and a network routing request.

If the tier-0 surrogate determines that it cannot execute a query, it inserts it into the log. If the surrogate has decomposed the query, it will use the method described in Section III-D to insert a file system read and a network route request into the log. Each request will be accompanied by metadata tagging the request as part of a single query. As described previously, the query may be accompanied by a maximum queuing time and will remain in the log until the log fills or until some request forces a wake of tier-1. When the tier-1 system processes the log, it will compose related entries, perform any necessary processing on the data, and forward the result to the client.

V. Implementation

In order to evaluate our approach we have implemented a working prototype of our Triage architecture shown in Figure 6. This prototype, built using off-the-shelf hardware , consists of a dispatcher and three surrogates.

A. Prototype Hardware

We built our prototype on a hardware platform consisting of a slightly modified Crossbow Stargate (tier-1) [36] and a TelosB mote (tier-0) [26]. These hardware platforms were chosen because they handle the range of workloads that we have targeted, are separated in power consumption by more than an order-of-magnitude (20mW-120mW and 300mW-1800mW), are easily programmable, and are well supported. The Stargate platform runs Linux, making available a broad range of software tools and services. Recall that the Stargate contains a 32-bit, 400MHz PXA255 XScale processor, 64 MB of RAM, 32 MB of internal flash, and a WiFi interface. The TelosB mote contains an 8-bit, 8 MHz microcontroller, 10kB of RAM, 1 MB of external flash, and an 802.15.4 radio.

We made only minor modifications to the hardware including adding a control interface to allow the TelosB to wake the Stargate. Currently, the two platforms are powered by independent batteries, and we are working on a small interface board to link them to the same power source. This use of decoupled, commodity hardware also allows us to rapidly change the tiers to include new platforms. A previous prototype used a MicaZ mote [27], and minimal changes were needed to support the new hardware.

One limitation of our current implementation is the transfer speed from the TelosB mote to the Stargate. The two devices communicate over a USB line which is limited to 230 kbps. However, data needs to be read from the flash and then transfered over the USB which increases the total time required for the transfer. As a result, transferring 1024 KB of batched work along with protocol overhead takes more than 150 seconds and wastes a great deal of energy while blocked on serial I/O. Even with this limitation, the current prototype shows extremely high gains in energy efficiency.

There are several methods for putting the Stargate (tier-1) into a low-power state. It can be suspended (memory refreshed), hibernated (memory written to flash), or shutdown (memory not saved). The choice between these states depends on the length of time tier-1 is expected to be in the low-power state, and we consider this issue orthogonal to our work. When suspended, the Stargate draws roughly 160mW of power as compared to other PXA-based platforms that draw less than 40mW in suspension. This overhead unfortunately makes suspension far too expensive to use. Regardless, Triage is designed to keep tier-1 in a low-power state as much as possible, and except for heavy workloads, full shutdown is a better option.

B. Surrogates, Log, and Dispatcher

As part of this prototype, we implemented a dispatcher and three surrogates: multi-network routing, storage, and query processing. Each of these system components were implemented according to the design described previously and are fully described in Section IV.

Surrogate queues are stored and managed on the

TelosB's flash storage using a custom designed file system. We investigated using standard mote file systems, such as Matchbox [12]; however, Matchbox does not work on the TelosB's flash necessitating a new flash file system. The new file system supports writes, random reads and delete operations. Due to the large erase units of flash storage, we implemented a garbage collector which scans a sector and swaps the used blocks and writes them contiguously into another reserved sector. Subsequently the old sector is deleted. The garbage collector is invoked when the number of dirty blocks exceeds a threshold value. The file system also maintains a cache of directory entries of the recently accessed files. An evaluation of the file system reveals a read throughput of 30 KB/sec and a write throughput of 20 KB/sec. The garbage collector takes on an average 4.5 seconds to clean a sector.

We implemented these components as TinyOS modules written in nesC [13]. The routing, storage, and query processing surrogates comprise 350, 1100, and 250 lines of nesC code respectively, and the file system and dispatcher consist of roughly 1600 and 750 lines of code each [1]. We also implemented an execution engine which runs on the Stargate and executes tasks when they are received from the mote.

VI. Evaluation

Our goal in evaluating Triage is to answer the following questions:

- Does Triage improve overall energy efficiency for battery powered microservers?
- To what extent do delayed execution, caching, and task routing improve efficiency?
- What impact do quality of service constraints have on potential gains?
- What are the main inefficiencies of the Triage system?

We designed several experiments in order to answer these questions. Our first experiment looks at the overall energy efficiency gains of the Triage system in a medium-scale deployment. Additionally, we present the results of several more focused experiments that look at the impact of each of the techniques that Triage employed.

A. Methodology

In each experiment, we measure the power consumed by the system using the setup shown in Figure 7. Current draw is measured using an NI-DAQCard-6036E to record the voltage drop across 1Ω sense resistors. Since both platforms do not yet share a common power supply, individual power consumptions for the mote and the Stargate are measured separately and the results are added to calculate the total power consumption of the system. Each experiment was run for 30 minutes.

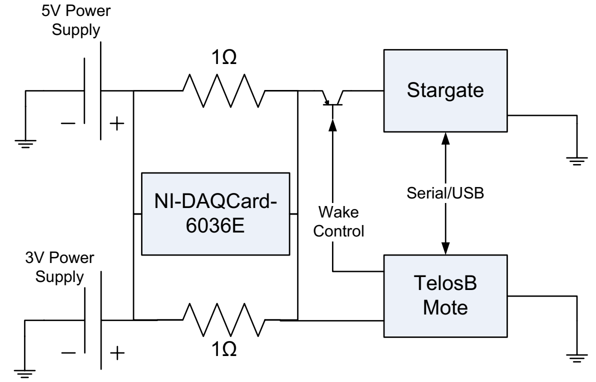

Fig. 7. Experimental Setup

In evaluating Triage, we compare three different approaches: NIC, TRIAGE_BATCH, and TRIAGE. The NIC approach uses tier-0 as a network interface that wakes tier-1 upon arrival of every request. This is a popular approach used in many current microserver applications. The second approach (TRIAGE_BATCH) uses tier-0 to delay the execution of incoming requests until either the flash storage fills or the maximum queuing delay constraint requires tier-1 to wake up and respond to the pending requests. The third approach (TRIAGE) is the full Triage implementation, which uses delayed execution, caching, and execution decisions in order to amortize the cost of waking tier-1.

We conducted experiments on a medium-scale wireless video sensor network. The network comprises data producers, which send data to a microserver for processing, and data consumers, which request data from a microserver. We use Cyclops2C cameras as data producers [28]. These cameras, mounted on MicaZ motes, periodically capture and send 1 KB (8-bit pixel) gray-scale images to the storage surrogate running on the microserver. Due to limited availability of the Cyclops2C cameras, we simulate additional cameras with MicaZ motes that send random images to the microserver. The data stored and produced by the microserver is then consumed by a PC, which periodically injects queries destined for the query processing surrogate of the microserver. Triage uses the

[1] The source code for Triage can be downloaded from the following URL : *http://www.prisms.cs.umass.edu/hpm/triage.tar.gz*

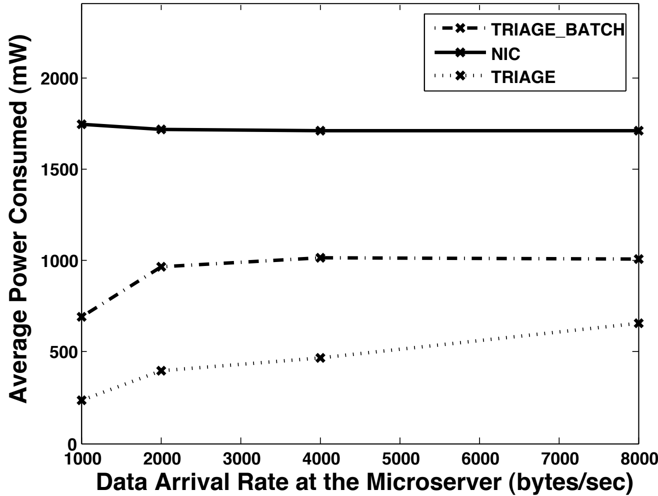

Fig. 8. Microserver power consumption is shown with respect to inbound data rate from a medium-scale video sensor deployment. Delayed execution, caching, and execution decisions in Triage result in a 2.5-5X reduction in power consumption.

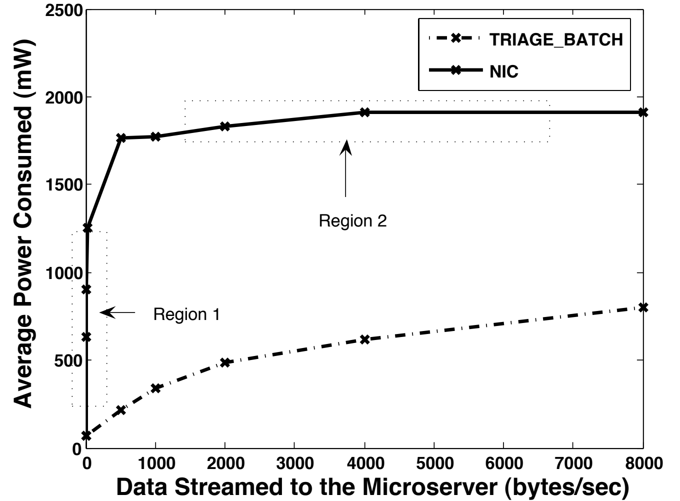

Fig. 9. Power consumption is shown with respect to the rate that data is streamed to the microserver. Delaying task execution to amortize wake up cost results in a 3.5X reduction in power consumption.

network routing surrogate to send responses to the data consumers. All of the nodes communicate over a multi-hop 802.15.4 network, with the exception of microservers which may also use 802.11b radios to more efficiently route packets.

B. Overall Energy Efficiency

Our primary goal is to improve energy efficiency for battery-powered microservers. To evaluate this, we perform an experiment that compares the overall power consumption of NIC, TRIAGE_BATCH, and TRIAGE, in a medium-scale deployment running multiple services. Our deployment spans the second floor of the Computer Science building at the University of Massachusetts at Amherst. It features two microservers, two additional Stargates serving as 802.11 routers, a 24-node multi-hop mote network, 2 data consumers (PCs), and 6 data producers (cameras).

In this experiment we vary the rate that the cameras send image data to the microservers. Queries are injected into the network at a constant rate of every 180 seconds with a 100 second maximum queuing delay. Each query requests a single image taken T seconds ago, where T is selected from an exponential distribution with a mean of 100. While this does not represent a particular application, it does generally represent applications in which newer data is more valuable to the user than older data. The results are routed over either the 802.11b between microservers, or over the 802.15.4 network, based on minimum energy routing.

The results, shown in Figure 8, show a 2-2.5X reduction in power draw by only delaying execution, and a 2.5-5X reduction using the full Triage system. By delaying execution, caching results, and executing tasks on the most appropriate tier, Triage is able to reduce the amount of work performed on tier-1 and amortize the cost of waking tier-1 over many tasks. Our remaining experiments show how each of the techniques used by Triage impact efficiency, and finally identify the inefficiencies in the system in order to motivate future improvements.

C. Delayed Execution

In the next experiment we examine the benefits of delaying task execution in order to amortize the wakeup cost of tier-1. We vary the rate at which images are streamed from 3 cameras to a microserver for storage. There are no queries involved in this experiment. Since our focus is on delayed execution, no quality of service constraints were used, and we only compare the NIC and TRIAGE_BATCH approaches. The effect of quality of service constraints is shown in a later experiment.

Figure 9 shows our results for data rates up to 8KB/sec, which is the maximum rate at which images can be received and batched on the TelosB platform(tier-0). The results for the NIC approach show two distinct behaviors which are identified by rectangular boxes in the figure. The first is seen at low data rates as tier-0 wakes tier-1 with every packet received. The high cost of waking tier-1 results in a steep rise in power consumption as the packet frequency increases. Very quickly, however, the time required to wake tier-1 (10-15s) precludes sleeping tier-1

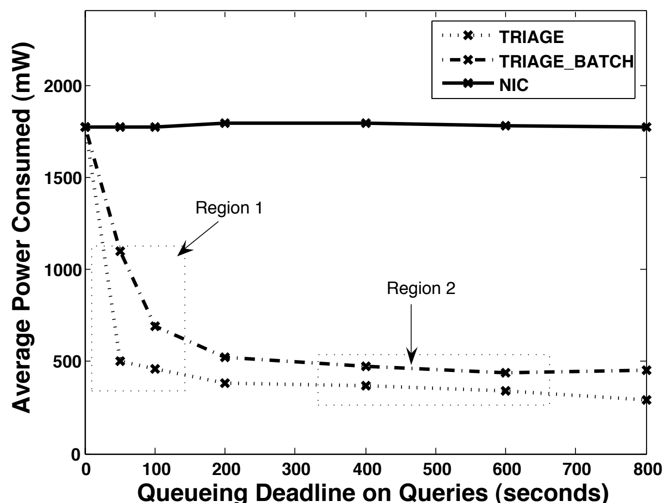

Fig. 10. Average power consumption is shown with respect to maximum queuing time assigned to the queries. Short queuing time requirements limit the efficiency gains that can be achieved using Triage; however, applications that can tolerate a 60 second execution delay achieve an 85% reduction in microserver power consumption from the NIC approach.

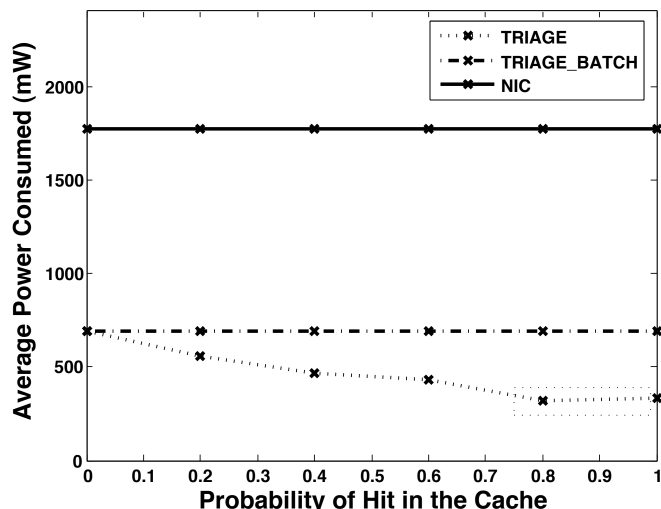

Fig. 11. Average power consumption is shown with respect to the hit rate of queries in tier-0's cache. In applications exhibiting high locality, Triage achieves a 2X improvement over using delayed execution alone.

which must remain awake in order to handle the more frequent requests. This behavior corresponds to the second region on the figure, in which power consumption slowly saturates to the maximum power consumption of the entire system.

In contrast, the results from using delayed execution show a more reasonable increase in power consumption as images arrive more frequently. In this experiment delaying the execution of tasks amortizes the wakeup cost of tier-1 resulting in, on average, a 3.5X power savings.

D. Quality of Service Constraints

The power savings seen from delaying execution are limited by the presence of quality of service constraints in the system. In this experiment we consider the effect of these constraints. We fix the rate at which data is sent to the microserver at 1KB/sec and set queries to arrive every 100 seconds over the mote network. The answers to the queries can be routed back to the client over either of the 802.11b or 802.15.4 networks depending on which radio is more energy-efficient to use. We vary the max queuing delay of the queries from 0 to 800 seconds. Queries are distributed such that they can be answered from tier-0's cache 50% of the time.

The results of this experiment are shown in Figure 10. The NIC approach shows almost constant behavior regardless of the constraints. This is a result of the 1KB/sec data rate, which requires tier-1 to remain on to execute the requests. The other two approaches show more interesting behavior. Both the results for the TRIAGE_BATCH and the TRIAGE systems can be broadly divided into two distinct regions (shown as rectangular boxes in the figure). When the max queuing delay is small, (*Region 1*), tier-1 is required to be on more frequently and reduces the potential for batching. However, with a max queuing time of 60 seconds, TRIAGE achieves a 3.5X reduction in power consumption. The second region of the graph, (*Region 2*), shows nearly constant power consumption for both TRIAGE_BATCH and TRIAGE, as queries can be delayed and the cost to wake tier-1 and store incoming images begins to dominate the total power draw.

E. Surrogate Caching

In addition to quality of service constraints, cache performance also affects the efficiency of the system. In this experiment we examine how system power consumption varies with respect to cache hit rate. As in the previous experiment, the data rate is fixed at 1 KB/sec, Queries arrive every 180 seconds, and each query has a maximum queuing delay of 100 seconds. Throughout this experiment we vary the locality of query request accesses, in order to achieve the desired hit rate.

The results of the experiment, shown in Figure 11, emphasize the importance of caching in the Triage system. If queries exhibit high locality the full TRIAGE approach achieves an additional 2X improvement over using delayed execution alone. Note that as the hit rate approaches 1, the wake up cost incurred by incoming images begins to dominate the cost of cache misses, resulting in a nearly con-

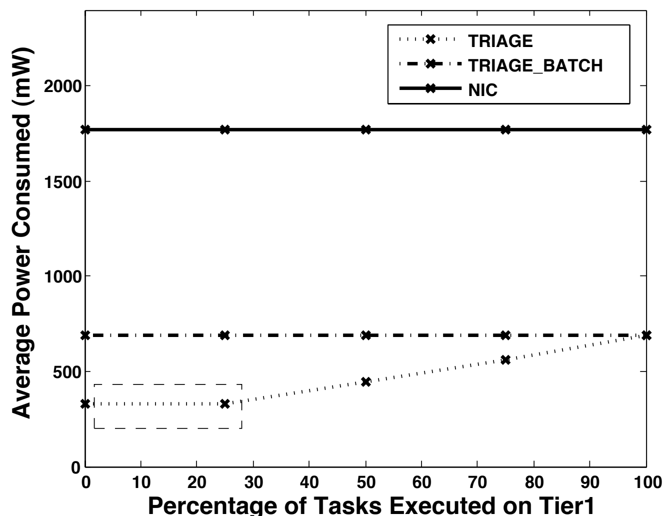

Fig. 12. Average power consumption is shown with respect to percentage of queries that must be answered by tier-1. Triage is able to route simple queries for execution on tier-0, resulting in a potential 2X efficiency improvement over using only delayed execution.

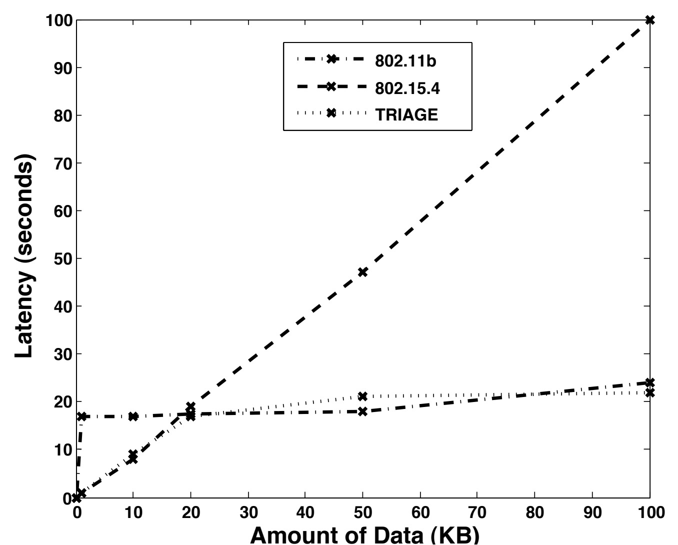

Fig. 13. The incurred latency is shown with respect to the amount of data requested for the 802.11b and 802.15.4 radios. Triage uses the crossover point (20 KB) to determine which radio is most efficient to use.

stant power consumption from 0.8 to 1.0. It is important to note that as in any caching system, the size of the flash available on tier-0, and the caching strategy itself, must be tuned to the intended workload—our experiments demonstrate some of the potential gains.

F. Task Execution

In addition to quality of service constraints, and caching, we also want to examine the benefits of executing tasks on the appropriate tier of the system. For this experiment we send two types of queries: 1) simple queries which can be executed on either tier (e.g. *retrieve the image at camera A with timestamp T*), and 2) more complex queries that involve more complex processing (e.g. *retrieve all images from time T_1 to time T_2 which contain more than 5 objects*). Again data arrives at the microserver at a constant rate of 1 KB/sec. Queries arrive at a fixed rate and have a maximum queuing time of 100 seconds. We vary the percentage of complex queries in the experiment.

The results of this experiment, shown in Figure 12, demonstrate the power savings due to executing tasks on the most appropriate tier. In this experiment our system draws 50% less power draw than delayed execution alone. The trends seen in this set of results, are very similar to the previous experiment. This is not surprising since caching is an implicit form of executing read requests on the appropriate tier based on where the desired data is located.

To demonstrate the network routing surrogates task execution decisions, we performed experiments on Triage's minimum latency routing mode. We first determine the latency to transfer data across a single hop of the 802.15.4 radio. We found that the latency incurred in transfering data using the mote radio varies linearly with the number of hops for our multi-hop routing protocol. Thus, in the general scenario, Triage uses the per hop latency measure and multiplies it with the number of hops to determine which radio should be used.

To validate this, we constructed a 4-hop 802.15.4 network and compared its latency with a single hop 802.11b network. The setup is same as that shown in Figure 5. We vary the amount of data requested and measure the time required to transfer the data over the two networks. We determine the crossover point of the two radios and use it in the microserver to determine when one radio is more efficient to use over the other.

The results of the experiment is shown in Figure 13. Although the 802.11b radio on the Stargate has a much higher bitrate, the requests incur a startup cost before they can be executed. We find that the crossover point for the two curves is around 20 KB of data. Triage uses the mote radio below the crossover point and the 802.11b radio above it and hence achieves better latency performance over the entire range as compared to the two radios separately. Note that as the number of hops in the mote network increases the crossover point shifts to the left of the curve.

G. System Limitations

Our final goal is to identify the limitations and inefficiencies in our current implementation of the Triage system in order to direct future improvements. To do this,

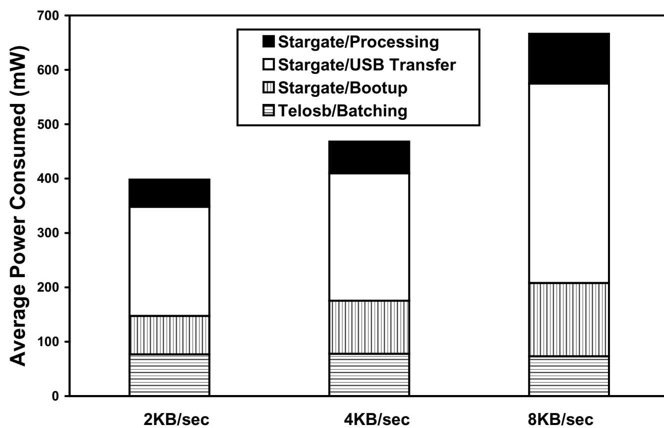

Fig. 14. This figure shows how each activity of the Triage system contributes to the total system power consumption at various data rates. Wakeup and data transfer costs are the dominant sources of inefficiency in the system, especially at high data rates.

we examine how individual system activities contribute to Triage's average power consumption from the first experiment, shown in Figure 8. We show a per-activity comparison in Figure 14 for data rates of 2KB/s, 4KB/s, and 8KB/s.

Our main observation from this comparison is that the power required to boot the Stargate and transfer pending tasks between tiers is the greatest limiting factor in the energy efficiency of the Triage system. We plan to address this problem in a future hardware platform using more efficient transfer mechanisms. This promises to even further improve the already significant efficiency gains achieved by Triage.

VII. Related Work

The design and implementation of Triage draws from several related research areas, which we survey here.

A. Microservers and Clustering

Several sensor network systems utilize a subset of the participating nodes as aggregaters, central processing nodes, or gateways [14]. This work can be classified into algorithms for networks of homogeneous devices and algorithms for networks of heterogeneous devices. In homogeneous systems such as Heed [38], LEACH [16], and the system proposed by Bandyopadhyay and Coyle [4], the leader, or clusterhead, rotates among nodes in the network. The goal is to distribute the extra energy drain incurred by the leader. In heterogeneous systems, larger, more powerful nodes called microservers *herd* other smaller nodes [35]. Our work focuses on the latter scenario and addresses the need for a power-aware software architecture to reduce the energy drain on the resource-rich nodes.

B. Disconnected Systems

Triage is similar to many mobile systems in that parts of the system can become disconnected when suspended, hibernated, or powered down. Several mobile systems have addressed disconnection and lack of availability between participating nodes. Examples include file systems such as Coda [31] and Ficus [29], databases such as Bayou [9] and DBmate [25], remote execution systems, such as Spectra [10] and Chroma [3], and general toolkits such as Rover [18]. Many of the techniques found in these systems, such as logging and caching, strongly influenced our design. In particular, we used the queued RPC mechanism from Rover as a basic building block. However, Triage differs from traditional mobile systems. First, with Triage there is a new element of control: the "client" (the less powerful system) can directly control when connections and disconnections occur to the "server" (the more powerful system). Second, the less powerful system is more resource-constrained than a typical mobile laptop—our lowest tier only contains 10kB of program memory. Additionally, we have integrated the competing concerns of quality of service and energy in our design criteria.

C. File Systems

Logging file systems have been proposed both for hard disk drives, such as the Log-Structured File System [30], as well as flash memory, such as ELF [8], JFFS2 [37], and Matchbox [12]. In our prototype, tier-0 uses a custom file system and tier-1 uses JFFS2. However, any efficient flash file system will work. Our contribution lies in the distribution of the file system over two connected devices.

D. Energy Management

Reducing the power consumption of mobile devices has been the subject of much research. Approaches include scaling the CPU voltage and frequency [15], managing wireless interface usage [2], turning off banks of RAM [17], or employing microsleep [20, 6]. In a larger device, such as a Stargate, these techniques still do not enable a power mode comparable to a mote device. Our architecture is designed to support devices that can operate at power levels separated by an order of magnitude.

Papathanasiou and Scott made an observation similar to ours: batching work, or increasing idle periods, leads to greater energy efficiency [24]. However, the goal of the their work was to increase burstiness in laptop disk drives.

The Wake-on-Wireless project (WoW) [33] proposes a hierarchy of devices for PDAs, including a low-power receiver that can wake the PDA. Our goal is similar to WoW,

to reduce power consumption in battery powered devices. But, we have placed a large amount of functionality in the lowest tier. Our tier-0 system is capable of actually executing some tasks without waking tier-1.

E. Sensor Platforms

Recently, many embedded sensor platforms have emerged. These platforms span a broad spectrum of power requirements and functionality. A popular instance of sensor platforms is the family of motes. These nodes are commercially available, widely used, and include the Crossbow MicaZ and Mica2Dot as well as the Telos node. All of these nodes consume peak power between 10-120mW and are tuned to be highly power efficient.

There are also several more capable but still very power-efficient sensor nodes such as the Yale XYZ [22]. This node has dynamic frequency scaling capability and can operate between 2MHz and 56MHz with a power consumption of up to 3x greater than the mote at comparable clock speeds. Such intermediate platforms can be used as clusterheads in applications that have moderate computation requirements.

Our architecture targets resource-rich but power efficient sensor platforms that combine two processing elements—one small and one large. Two instances of such architectures are currently commercially available. The Stargate platform [36] has been used with a mote, but this has been to provide a gateway to other mote nodes, not to optimize the energy efficiency of the platform. The PASTA node is an architecture that combines a trip-wire board with a DSP processor together with a PXA processor [32]. Other instances of such dual processor systems have been suggested in the literature although they are not commercially available.

VIII. Conclusion

This paper presents the design, implementation, and evaluation of Triage, a system that reduces the energy consumption of an untethered microserver. We detail the design of a hardware architecture that uses off-the shelf hardware to provide a range of energy consumption modes. Additionally, we present the design of a software architecture that uses delayed execution, caching, and local task completion to amortize the cost of servicing requests using a high-power platform. We show the results of experiments conducted on a medium-scale wireless network deployment that bears a 5x reduction in energy consumption.

References

[1] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young. Mach: a new kernel foundation for unix development. In *Proceedings of Summer Usenix*, pages 93–113, July 1986.

[2] M. Anand, E. B. Nightingale, and J. Flinn. Self-tuning wireless network power management. In *Proceedings of the 9th ACM International Conference on Mobile Computing and Networking (MobiCom'03)*, San Diego, CA, September 2003.

[3] R. K. Balan, J. P. Sousa, and M. Satyanarayanan. Tactics-based remote execution for mobile computing. In *Proceedings of The First International Conference on Mobile Systems, Applications, and Services (MobiSys '03)*, San Francisco, CA, May 2003.

[4] S. Bandyopadhyay and E. J. Coyle. An energy efficient hierarchical clustering algorithm for wireless sensor networks. In *Proceedings of IEEE Infocom*, San Francisco, CA, March 2003.

[5] K. Barr and K. Asanovic. Energy aware lossless data compression. In *The First International Conference on Mobile Systems, Applications, and Services (Mobisys)*, pages 231–244, San Francisco, CA, May 2003.

[6] L. S. Brakmo, D. A. Wallach, and M. A. Viredaz. microSleep: A technique for reducing energy consumption in handheld devices. In *Proceedings of the Second International Conference on Mobile Systems, Applications, and Services (MobiSys'04)*, Boston, MA, June 2004.

[7] B. Burns, O. Brock, and B. N. Levine. MV routing and capacity building in disruption tolerant networks. In *Proceedings of IEEE Infocom 2005*, March 2005.

[8] H. Dai, M. Neufeld, and R. Han. ELF: An efficient log-structured flash file system for micro sensor nodes. In *Proceedings of The Second ACM Conference on Embedded Networked Sensor Systems (SenSys '04)*, Baltimore, MD, November 2004.

[9] A. J. Demers, K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and B. B. Welch. The bayou architecture: Support for data sharing among mobile users. In *Proceedings IEEE Workshop on Mobile Computing Systems and Applications*, pages 2–7, Santa Cruz, California, December 1994.

[10] J. Flinn, S. Park, and M. Satyanarayanan. Balancing performance, energy, and quality in pervasive computing. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, Vienna, Austria, July 2002.

[11] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and J. Heidemann. An evaluation of multi-resolution storage for sensor networks. In *Proceedings of The First ACM Conference on Embedded Networked Sensor Systems (SenSys '03)*, Los Angeles, CA, November 2003.

[12] D. Gay. Design of Matchbox, the simple filing system for motes. TinyOS Documentation, August 2003.

[13] D. Gay, P. Levis, R. V. Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. In *Proceedings of Programming Language Design and Implementation (PLDI)*, June 2003.

[14] L. Girod, T. Stathopoulos, N. Ramanathan, J. Elson, D. Estrin, E. Osterweil, and T. Schoellhammer. A system for simulation, emulation, and deployment of heterogeneous sensor networks. In *Proceedings of The Second ACM Conference on Embedded Networked Sensor Systems (SenSys '04)*, Baltimore, MD, November 2004.

[15] K. Govil, E. Chan, and H. Wasserman. Comparing algorithms for dynamic speed-setting of a low-power CPU. In *Proceedings of the First ACM International Conference on Mobile Computing and Networking (MobiCom'95)*, Berkeley, CA, November 1995.

[16] Wendi Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocols for wireless microsensor networks. In *Proceedings of the Hawaiian International Conference on Systems Science*, January 2000.

[17] H. Huang, P. Pillai, and K. G. Shin. Design and implementation of power-aware virtual memory. In *Proceedings of USENIX Technical Conference*, San Antonio, TX, June 2003.

[18] A. D. Joseph and M. F. Kaashoek. Building reliable mobile-aware applications using the Rover toolkit. In *Proceedings of The Second ACM International Conference on Mobile Computing and Networking (MobiCom'96)*, White Plains, NY, November 1996.

[19] R. H. Halstead Jr. MULTILISP: A language for concurrent symbolic computation. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(4), October 1985.

[20] N. Kamijoh, T. Inoue, C. M. Olsen, M. T. Raghunath, and C. Narayanaswami. Energy trade-offs in the IBM wristwatch computer. In *Proceedings Fifth International Symposium on Wearable Computers*, Zurich, Switzerland, October 2001.

[21] P. Kulkarni, D. Ganesan, and P. Shenoy. Senseye: A multi-tier camera sensor network. In *ACM Multimedia*, 2005.

[22] D. Lymberopoulos and A. Savvides. XYZ: A motion-enabled, power aware sensor node platform for distributed sensor network applications. In *Proceedings of Information Processing in Sensor Networks (ISPN)*, Los Angeles, CA, April 2005.

[23] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Workshop on Wireless Sensor Networks and Applications*, Atlanta, GA, September 2002.

[24] A. E. Papathanasiou and M. L. Scott. Energy efficiency through burstiness. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, Monterey, CA, October 2003.

[25] S. Phatak and B. R. Badrinath. Data partitioning for disconnected client-server databases. In *Proceedings of the ACM International Workshop on Data Engineering for Wireless and Mobile Access*, August 1999.

[26] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling ultra-low power wireless research. In *Proceedings of the Fourth International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, April 2005.

[27] J. Polastre, R. Szewczyk, C. Sharp, , and D. Culler. The mote revolution: Low power wireless sensor networks. In *Proceedings of the 16th Symposium on High Performance Chips (HotChips)*, August 2004.

[28] M. Rahimi, R. Baer, O. I. Iroezi, J. C. Garcia, J. Warrior, D. Estrin, and M. Srivastava. Cyclops: In situ image sensing and interpretation in wireless sensor networks. In *Proceedings of the Third ACM Conference on Embedded Networked Sensor Systems (SenSys)*, San Diago, November 2005.

[29] P. Reiher, J. S. Heidemann, D. Ratner, G. Skinner, and G. J. Popek. Resolving file conflicts in the Ficus file system. In *Proceedings of the 1994 Summer USENIX Conference*, Boston, MA, June 1994.

[30] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems*, 10(1):26–52, 1992.

[31] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, 1990.

[32] B. Schott, M. Bajura, J. Czarnaski, J. Flidr, T. Tho, and L. Wang. A modular power-aware microsensor with 1000x dynamic power range. In *Proceedings of Information Processing in Sensor Networks (ISPN)*, Los Angeles, CA, April 2005.

[33] E. Shih, P. Bahl, and M. J. Sinclair. Wake on Wireless: An event driven energy saving strategy for battery operated devices. In *Proceedings of the Eighth ACM Conference on Mobile Computing and Networking*, Atlanta, GA, September 2002.

[34] J. Sorber, N. Banerjee, M. D. Corner, and S. Rollins. Turducken: Hierarchical power management for mobile devices. In *Proceedings of The Third International Conference on Mobile Systems, Applications, and Services (MobiSys '05)*, Seattle, WA, June 2005.

[35] T. Stathopoulos, L. Girod, J. Heidemann, and D. Estrin. System support for coordinated imaging for sensor networks. Unpublished Poster, 2004.

[36] R. Want, T. Pering, G. Danneels, M. Kumar, M. Sundar, and J. Light. The personal server - changing the way we think about ubiquitous computing. In *Proceedings of Ubicomp 2002: 4th International Conference on Ubiquitous Computing*, Goteborg, Sweden, September 2002.

[37] D. Woodhouse. JFFS: The journalling flash file system. In *Ottawa Linux Symposium*, Ottawa, Canada, 2001.

[38] O. Younis and S. Fahmy. HEED: A hybrid, energy-efficient, distributed clustering approach for ad-hoc sensor networks. *IEEE Transactions on Mobile Computing*, 4(4), October 2004.