# The Impact of Distributed Programming Abstractions on Application Energy Consumption

Young-Woo Kwon and Eli Tilevich

*Dept. of Computer Science*
*Virginia Tech*
*Blacksburg, VA 24060*
*Email: {ywkwon,tilevich}@cs.vt.edu*

**Abstract**

With battery capacities remaining a key physical constraint for mobile devices, energy efficiency has become an important software design consideration. Distributed programming abstractions (e.g., sockets, RPC, messages, etc.) are an essential component of modern software, but their energy consumption characteristics are poorly understood. The programmer has few practical guidelines to choose the right abstraction for energy-constrained scenarios. In this article, we report on the findings of a systematic study we conducted to compare and contrast major distributed programming abstractions in terms of their energy consumption patterns. By varying the abstractions with the rest of the functionality fixed, we measure and analyze the impact of distributed programming abstractions on application energy consumption. Based on our findings, we present a set of practical guidelines for the programmer to select an abstraction that satisfies the energy consumption constraints in place. Our other guidelines can steer future efforts in creating energy efficient distributed programming abstractions.

*Keywords:* energy-efficiency, programming abstraction, distributed system, measurement, software design pattern

## 1. Introduction

Mobile devices have been surpassing stationary computers as the primary means of utilizing computing [6]. As a result, several software design assumptions need to be fundamentally reconsidered to produce applications that use

the limited resources of a mobile device optimally. One such resource is energy, provided by constantly improving but always limited batteries. Indeed, energy efficiency has become an important software design constraint [21].

Network communication constitutes one of the largest sources of energy consumption in a distributed application [1]. To streamline the implementation of its network-related functionality, a distributed application can take advantage of programming abstractions (sometimes referred to as *middleware*), which can influence its energy consumption profile. To implement the same application functionality, a software designer can select from a range of distributed programming abstractions, a decision that can significantly impact how much energy the application will consume. Unfortunately, this impact on energy efficiency when choosing one distributed programming abstraction over another has not been studied systematically, thus leaving software designers no choice but to rely on their intuition when reasoning about application energy efficiency.

Unfortunately, relying on one's gut feeling [4] about the energy consumption characteristics of distributed programming abstractions can lead to suboptimal designs that may compromise the application's overall business utility. Although some of the results presented here may seem "obvious," the main contribution of this work is empirical evidence that can support or discredit these commonly held beliefs about application energy consumption.

To equip the software designer with an informed understanding of application energy consumption, we have focused on the Open Service Gateway Initiative (OSGi), an industry standard for deploying software in multiple domains, including mobile platforms. In particular, we have studied how OSGi bundles (coarse-grained software components) communicate with each other by means of distributed programming abstractions, having considered eight abstractions that differ on two axes: network communication footprint and level of abstraction (See Figure 1).

In terms of network communication footprints, we have considered platforms that transfer data in binary and in XML-based formats. In terms of the level of abstraction, we have considered socket-, remote method call-, and message-based platforms. Specifically, we have studied TCP sockets; Message Oriented Middleware (MOM); J2EE RMI; XML-RPC; and OSGi-based remote methods (synchronous and asynchronous), our own Remote Batch Invocation (RBI), and SOAP-based services.

For each considered distributed programming abstraction, our experiments assessed the energy consumption (1) of passing varying volumes of

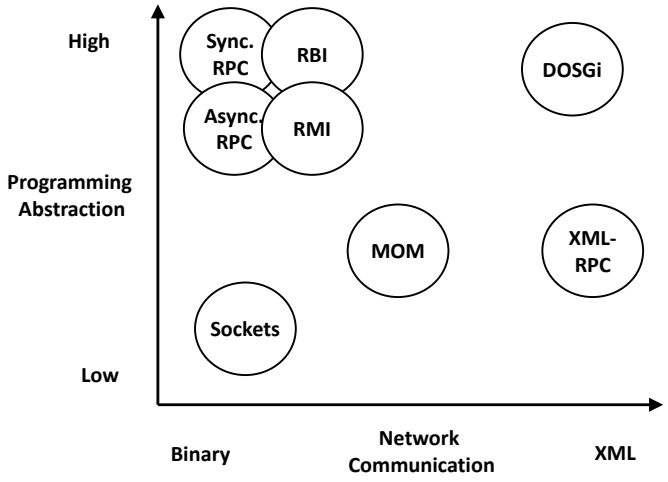



Figure 1: Considered DPAs and their classification.

data over networks with different latency/bandwidth characteristics, (2) of marshaling/unmarshaling complex data, and (3) of staying idle. The main contributions of this article include:

- **A systematic study of energy consumption in distributed programming abstraction mechanisms:** We have systematically compared and contrasted the energy consumption of eight distributed programming abstraction mechanisms; by varying the mechanisms while keeping the rest of functionality fixed, we were able to accurately estimate the impact of programming abstractions on the overall application energy consumption.
- **Energy consumption profiles for the aforementioned abstractions:** By analyzing the results of our study, we ranked the abstraction mechanisms by their energy consumption profiles, thus informing programmers needing to choose between different mechanisms.
- **Guidelines for energy-efficient and -aware distributed programming abstractions:** We put forward several guidelines that can guide software engineering researchers, who strive to innovate in the distributed programming abstraction space with energy efficiency in mind.

The rest of this article is structured as follows. Section 2 introduces the studied distributed programming abstractions. Section 3 describes our experimental study, while Section 4 analyzes the results. Section 5 infers energy consumption patterns and proposes new guidelines for both programmers and distributed system designers. Section 6 compares this work to the existing state of the art. Finally, Section 7 presents concluding remarks and future research directions.

## 2. Background

We first describe the distributed programming abstractions we have evaluated and then introduce the issue of measuring energy consumption in software systems.

### *2.1. Distributed Programming Abstractions (DPAs)*

Distributed computing coordinates the execution of multiple remote processes. Distributed programming abstractions (DPAs) provide programming and runtime support for one process to execute functionality in a different process. In other words, by eliminating the need for low-level network programming, programming abstractions offer convenient building blocks for constructing distributed systems. Major, widely used DPAs include sockets, messaging, remote procedure/method calls, and remote services.

#### *2.1.1. Remote Procedure Calls*

Remote Procedure Calls (RPC) serve as a foundation for a wide range of implementations. In this model, the programmer expresses functionality to be invoked in a different process as a regular procedure. However, when such a procedure is invoked, the runtime executes it in a remote process, transferring the parameters and returning the results. RPC has been extended to support object-oriented programming through Remote Method Invocation (RMI); object proxies forward invocations across processes. Representative RMI implementations are Java RMI and XML-RPC.

#### *2.1.2. Message Oriented Middleware*

To communicate through messages, remote processes can take advantage of Message Oriented Middleware (MOM). MOM commonly supports synchronous and asynchronous interactions through two primary message topologies: point-to-point and publish/subscribe. With point-to-point, a

sender delivers messages to a particular client by depositing them onto a message queue. With publish/subscribe, a sender publishes messages for multiple clients through intelligent broadcasting, called a message topic.

To communicate through messages, Java programs can use the standardized API of the Java Message Service (JMS) [20]. In this article, we use a popular JMS-compliant MOM infrastructure called ActiveMQ [30].

#### *2.1.3. Remote Services*

Service Oriented Architectures (SOA) provide uniform access to a variety of computing resources in multiple application domains. In SOA, software components are provided as services, self-encapsulated units of functionality accessed through a public interface. Services can access each other only via each other's public interfaces.

Remote OSGi (R-OSGi) [25] is an RPC-based DPA for OSGi. The R-OSGi distribution infrastructure allows accessing OSGi services remotely through a proxy-based approach, with proxies exposed as standard OSGi bundles. R-OSGi is based on RPC, but both synchronous and asynchronous. The OSGi R4.2 specification codifies the discovery and usage of remote services [23], with Apache CXF DOSGi [31] implementing this specification as SOAP-based Web services.

#### *2.1.4. Remote Batch Invocation*

As an alternative to RPC, whose unit of distribution is a single procedure call, we introduced Remote Batch Invocation (RBI) [14], whose unit of distribution is a block of code. RBI partitions blocks of code into remote and local parts, while performing all communications in bulk. Batches are specified using a *batch* statement, with the body of a batch statement combining remote and local computation. A batch block looks like a collection of remote method calls but is executed using *remote evaluation* [28], in which all the remote calls are transmitted in a single, compiler-constructed *batch script*.

### *2.2. Measuring Energy Consumption*

To measure energy consumption, two primary approaches have been proposed in the literature. One approach leverages specialized hardware (e.g.,

ACPI[1] or IPMI[2]). These hardware solutions can measure energy consumption quite precisely, but they do not map the consumed energy to the specific application functions or execution phases.

Another approach leverages energy models. For example, Seo at el. [27] put forward a model that divides the total energy consumed by an application into the *functions* of computation, communication, and infrastructure (e.g, JVM garbage collection, implicit OS routines, etc.). Kansal at el. [15] put forward an alternate model that instead focuses on the *phases* of waiting, execution, and idling.

Our measurement model amalgamates features of both of these models. Specifically, we focus on both application *functions* and *phases* by distinguishing between computation and communication, while also differentiating between the phases at which the energy is consumed. It is the amalgamated model that makes it possible for us to infer application energy consumption patterns. By flexibly adjusting our model for the measurement scenario at hand, we are able to infer general energy consumption patterns while ignoring the irrelevant factors. For example, our model omits the energy consumed by the infrastructure (i.e., it assumes that software design does not directly affect low-level infrastructure functions such as garbage collection and OS calls).

## 3. Measuring Energy Consumption of Distributed Programming Abstractions

In devising our approach to measuring energy consumption of DPAs, we wanted to be able:

1. to understand which components of DPA mechanisms mainly affect their overall energy consumption.
2. to identify temporal patterns in how DPA mechanisms consume energy; these patterns can guide the programmer in search of an abstraction delivering an application-specific energy consumption profile.
3. to infer opportunities for improving the energy efficiency of emerging abstractions.

[1]Advanced Configuration and Power Interface: `http://www.acpi.info`

[2]Intelligent Platform Management Interface: `http://www.intel.com/design/servers/ipmi/index.htm`

Next, we first present our energy consumption measurement model. Then we describe our experimental measurements. And finally, discuss the results.

*3.1. Energy Consumption Model*

We estimate total energy consumption by computing the workload incurred by each major piece of functionality. Specifically, the total energy consumption comprises two components—application and DPA:

$$E_{total} = E_{application} + E_{DPA}$$

Then, each energy consumption component is computed as follows:

$$E_{application} = E_{CPU} + E_{mem} + E_{disk} + E_{comm}$$

where $E_{applicaion}$ is the application-specific energy consumption, which includes $E_{CPU}$—energy consumed by CPU processing, $E_{mem}$—energy consumed by memory access, $E_{disk}$—energy consumed by I/O operations, $E_{comm}$—energy consumed by network communication.

The DPA-specific energy consumption is computed as follows:

$$\begin{aligned} E_{DPA} &= E_{CPU} + E_{mem} + E_{disk} + E_{comm} \\ &:= E_{CPU} + E_{comm} \end{aligned}$$

where $E_{DPA}$ is the DPA-specific energy consumption, which includes the energy consumed by CPU processing, memory/disk access, and network communication. For our experiments, we have excluded both the disk and memory access components from our measurements. The measured DPAs do not use disk I/O, and one cannot reliably distinguish between the energy consumption incurred by accessing application vs. DPA-specific memory without specialized hardware. Thus, our model considers the energy consumed by a DPA during CPU processing and network communication.

Furthermore, our measurements are confined to the client side of all distributed interactions; we assume a client/server communication model, in

which server computation and communication do not exhaust battery power. This assumption makes this work inapplicable to energy-conscious server environments or peer-to-peer setups, with mobile devices communicating with each other directly. We plan to extend our measurement model to a broader set of scenarios as a future work direction.

When an application executes, it goes through several phases: initialization, execution, idling, and termination, with the resulting energy consumption divided into four processes:

$$E_{total} = E_{init} + E_{exe} + E_{idle} + E_{term}$$

where the energy consumption for each of these phases is denoted as $E_{init}$, $E_{exe}$, $E_{idle}$, and $E_{term}$, respectively. For systematic evaluation, one must not only measure the energy consumption of a running application, but also the energy consumption during the application's initialization, idling, and termination phases.

Software systems that we are targeting in this article follow the traditional server-client model. Our energy consumption model does not consider energy consumption of server part.

### *3.2. Experimental Setup*

Our experimental setup comprised a client and a server. The server machine: 3.0 GHz Intel Dual-Core CPU, 2 GB RAM, Windows 7, and JVM 1.6.0 13 (build 1.6.0_13-b03); the client machine: 2.53 GHz Intel i3 CPU (dual-core), 4 GB RAM, Windows 7, and JVM 1.6.0 16 (build 1.6.0_23-b05). The client and server were connected via a wireless LAN. To create a controlled networking environment with delay and bandwidth limitation, we have used `Network Emulator for Windows Toolkit` [18], a popular network emulator. To measure energy consumption, we have used `pTopW` [5], a process-level power profiling tool that measures energy consumption at the kernel level.

An important goal of this work is to ensure that our results are applicable to distributed applications running on a broad range of mobile computing devices, ranging from laptops to phones. That is why although a laptop is a mobile device, whose energy consumption is an essential issue, we chose our client machine's setup (the CPU, OS, VM) to be as close as possible to the latest models of smartphones and tablets. The somewhat high amount

of RAM makes it possible to run the emulator and profiling tools without causing memory paging. Without the RAM taken by our measurement infrastructure, the client machine has about 1 GB left available for running applications, a typical setup for a modern hand-held device.

Because our goal is to determine how the choice of a DPA affects application energy consumption, our measurements focus on application-level energy-consumption patterns rather than on the underlying systems stack (e.g., OS and hardware). We also chose to perform our measurements over a Wi-Fi connection rather than a cellular network such as 3G. The reason is the increasing prominence of Wi-Fi networking, even for hand-held mobile devices. According to CISCO, Wi-Fi networks occupy 36% of the Internet traffic, while cellular networks deliver less than 10 % of traffic [2]. In fact, major US cities, including San Francisco, Washington D.C., Los Angeles, and New York City, have started to provide municipal wireless access through Wi-Fi networks [9]. Therefore, our experimental environment is typical for executing a substantial class of modern distributed applications.

Finally, since mobile devices run on a variety of platforms in fluctuating execution environments, the execution environment we set up for our benchmarks is unlikely to reflect all possible real-world scenarios. To create a meaningful approximation, we thus isolate CPU processing and network communication from the rest of the functionality, so that the specific hardware characteristic of the client device should not affect the energy consumption patterns of these two parts of system execution. As a result, in this article, we focus on inferring patterns and trends rather than on obtaining specific numbers.

### *3.3. Measurement Methodology*

#### *3.3.1. Benchmark Applications and DPAs*

We have based our test suite on the benchmarks originally proposed by the JavaParty project [12], which is used widely in benchmarking middleware platforms. These benchmarks comprise remote invocations with varying parameter sizes and types. Similarly, our test suite assumes that a client needs to execute some server methods, each of which takes different parameters. Because the executed server methods are empty, one can reasonably attribute the measured energy consumption to the underlying DPAs.

We have implemented eight versions of the same benchmark that have the same functionality but communicate through different abstractions. The

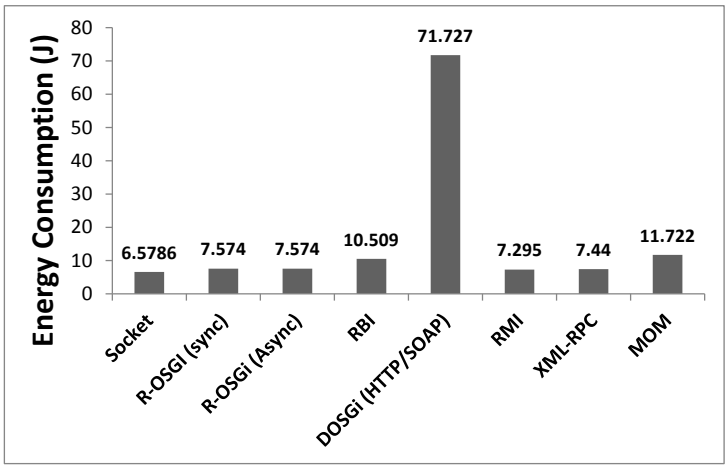



Figure 2: Energy consumption—initializing DPA mechanisms.

client and server parts of each version are OSGi bundles. We do not measure the energy consumed by the server bundle.

*3.3.2. Network Condition*

We have experimented with three emulated network conditions that have the following respective round trip time (RTT) and bandwidth characteristics: 2 ms and 50 Mbps, typical for a high-end mobile network; 30 ms and 1 Mbps, typical for a medium-end mobile network; and 30ms and 300 Kbps, typical for a low-end or congested mobile network [7, 29].

*3.4. Benchmarks*

*3.4.1. Energy Consumed by Initialization*

Figure 2 shows how much energy is consumed by initializing each DPA mechanism, a phase that also includes the initialization of the OSGi framework. DOSGi incurs the highest initialization costs; sockets, R-OSGi, RMI, and XML-RPC all initialize more energy efficiently than either RBI or MOM. DOSGi's high initialization cost are due to its dependence of a high number of third-party OSGi services.[3] The same explanation applies to RBI and MOM, even though their reliance on third-party OSGi bundles is not as significant as that of DOSGi.

[3]In case of Apache CXF Distributed OSGi, it loads 52 bundles.

*3.4.2. Scenario 1—Energy Costs of Invoking Remote Functionality*

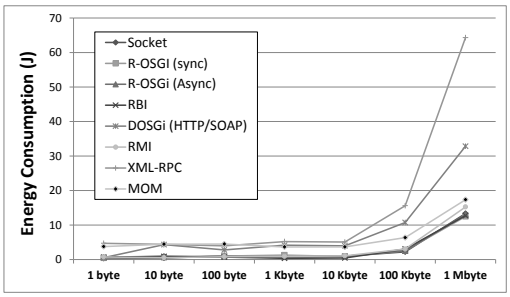
In this experiment, we isolate the energy costs of initiating the execution (i.e., invoking) of various remote methods. I.e., $E_{invoke} = E_{CPU} + E_{comm} - E_{init}$.

We measured the aggregate energy consumption of invoking the server method **void** ping(**byte**[]) 100 times in a loop; each experiment was repeated 10 times with the results averaged. Since the goal of our work is to provide programmers with practical guidelines they can follow to create energy-efficient applications, we chose not to report the standard deviation or extreme values. To inform the programmer, we focus on identifying the overall energy consumption patterns.

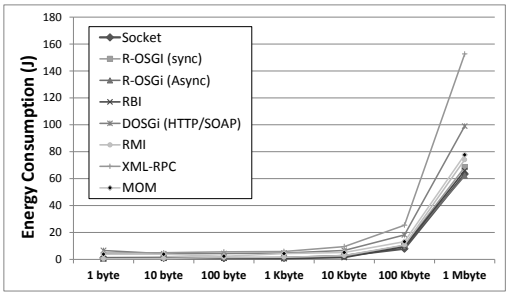
Figure 3 shows how much energy was consumed by each DPA mechanism. For all platforms, the energy consumption is directly proportional to the increases in latency and transferred data sizes. For example, under the emulated high-end mobile network (i.e., 2 ms latency and 50 Mbps bandwidth), all distributed programming abstractions consume little energy up until the arguments' size reaches 100 Kbytes. Beyond this argument size, the energy consumption begins to increase linearly. Similarly, once the latency goes up to 30 ms and the bandwidth goes down to 300 Kbps, the energy increases significantly. The effect is particularly pronounced for DOSGi and XML-RPC, due to their high bandwidth requirements for transferring XML.

Figure 4 shows the energy consumed by the CPU and network communication portions. pTopW makes it possible to ascertain how the total energy consumption is split into these two portions. Since we compare the respective levels of energy consumption of these two parts of system execution while changing network conditions, a line graph can faithfully represent such energy consumption trends. The left three figures depict each DPA mechanism's overall CPU energy consumption. The CPU energy consumption is directly proportional to the size of the transferred data. Specifically, when the latency increases and the bandwidth decreases, the energy consumed by CPU processing remains constant. However, the energy consumed by network processing increases significantly, particularly for XML-based DPAs.

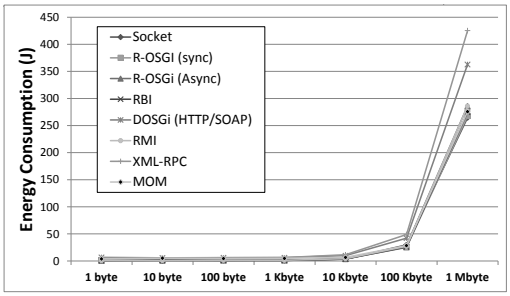
A surprising result is that asynchronous processing, be it in asynchronous R-OSGi, MOM, or sockets, does not affect the CPU energy consumption. This could be due to the fact that idle CPU cores still consume energy. These results indicate that the network characteristics with respect to the size of the transferred data can significantly influence the overall energy consumption.




(a) 2 ms latency and 50 Mbps bandwidth




(b) 30 ms latency and 1 Mbps bandwidth




(c) 30 ms latency and 300 Kbps bandwidth

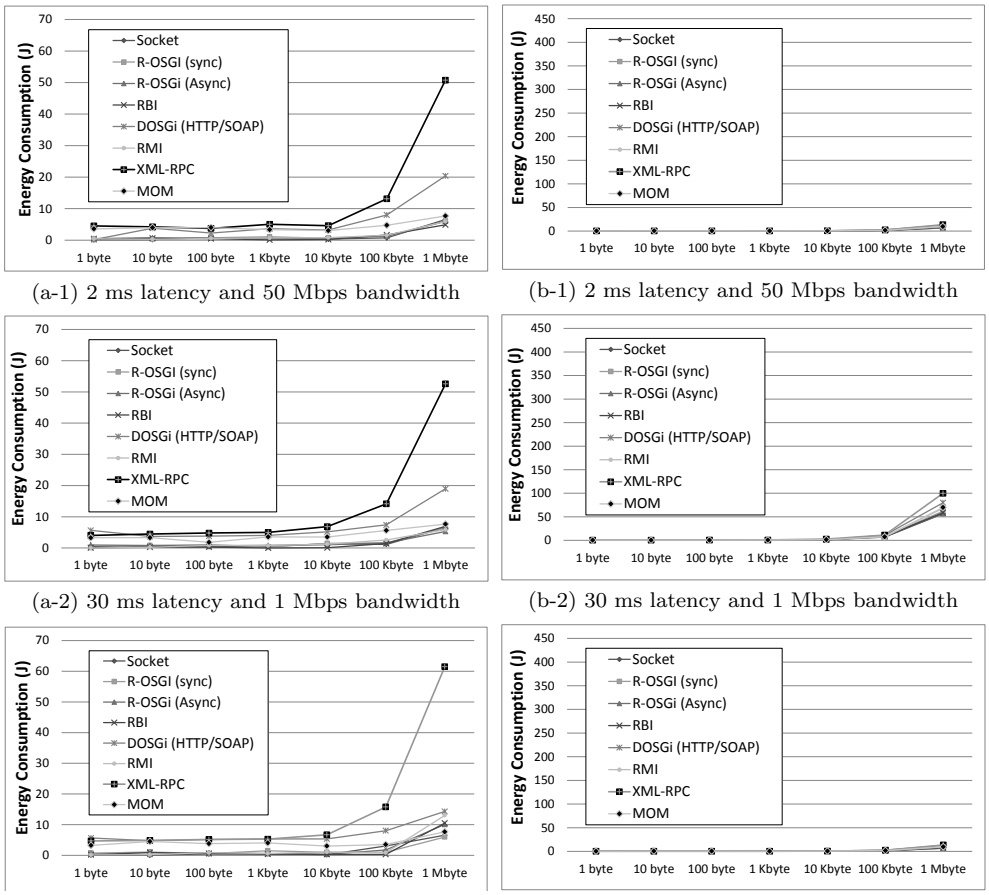
Figure 3: Energy consumption—invoking remote functionality.

### *3.4.3. Scenario 2—Data marshaling/unmarshaling*

In this experiment, we isolate the energy costs of marshaling/unmarshaling the data sent as parameters to the invoked remote methods. We have modified the remote methods instead of taking byte buffers to take arguments of different types that the DPA mechanism in place has to marshal/unmarshal. Specifically, we measured the energy consumed by passing (1) an object containing 32 **int** fields, (2) 1 non-primitive object which has two other non-primitive objects[4], or (3) a binary tree of 100 nodes, each holding an **int** value and two child recursive references. Each case's transferred data size is as follows: (1) $32 \times 4$ bytes $= 128$ bytes, (2) overall objects sizes are estimated as approximately 420 bytes, and (3) $(1 \times 4$ bytes $+ 2 \times 32$ bytes$) \times 100 = 680$ bytes.

Figure 5 shows the energy consumed by each DPA configuration. Each

[4]This test case is widely used for assessing the efficiency of Java serialization mechanisms. We used revision r128 of JVM serialization benchmark: `http://code.google.com/p/thrift-protobuf-compare/source/detail?r=128`




(a-1) 2 ms latency and 50 Mbps bandwidth

(b-1) 2 ms latency and 50 Mbps bandwidth

(a-2) 30 ms latency and 1 Mbps bandwidth

(b-2) 30 ms latency and 1 Mbps bandwidth

(a-3) 30 ms latency and 300 Kbps bandwidth

(b-3) 30 ms latency and 300 Kbps bandwidth

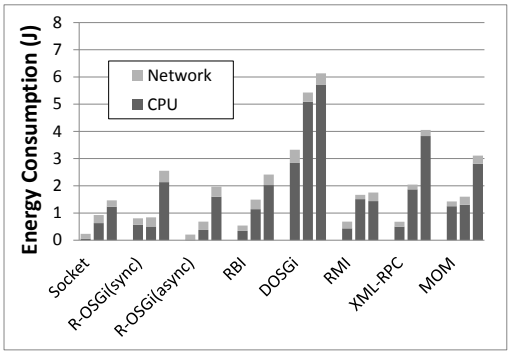
(a) Energy consumption—CPU.

(b) Energy consumption—Network.

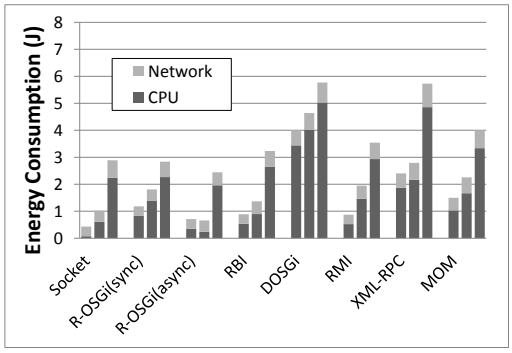
Figure 4: Energy consumption—CPU and network communication.

bar is the sum of the energy consumed by CPU and network processing. As expected, XML-based DPA mechanisms consume more energy than those that use either native Java serialization (i.e., RMI, Sockets, and MOM) or optimized serialization mechanisms (i.e., RBI and R-OSGi). Although the inefficiency of Java serialization is well known [24], our experiments did not indicate it to consume significantly more energy than the optimized serialization mechanisms. At least, the difference was not nowhere near as large as that between XML-based and binary serialization formats.

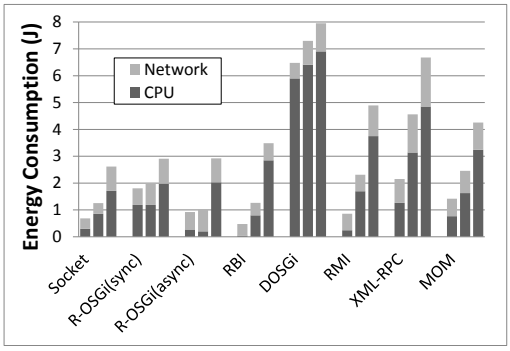
When breaking down the consumed energy into CPU and network processing, XML-based DPA mechanisms are particularly vulnerable to limited




(a) 2 ms latency and 50 Mbps bandwidth




(b) 30 ms latency and 1 Mbps bandwidth



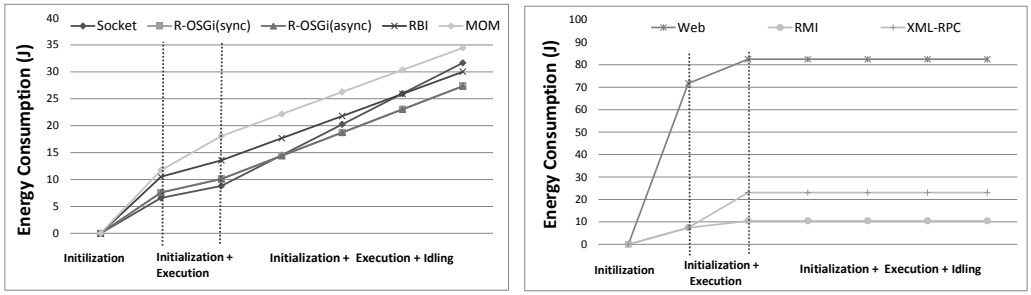

(c) 30 ms latency and 300 Kbps bandwidth

Figure 5: Energy consumption of three serialization cases.

network conditions, as transferring bulky XML-encoded data can quickly increase the amount of required network transmissions, thereby raising up the energy costs of network communication. Thus, energy-sensitive mobile applications should prefer DPA mechanisms that encode data in binary.

*3.4.4. Scenario 3—Energy Consumption per Execution Phases*

In this experiment, we measured the aggregate energy consumption of invoking the server method **void** ping (**byte**[100KB]) 100 times in a loop over the emulated network with 2 ms latency and 50 Mbps bandwidth. Because we could not observe any significant variability in the energy consumption patterns occurring under different network conditions, in this section, we only discuss the results tested for a high-end mobile network. The benchmark initializes, executes its functionality, idles for one minute, and then exits. We measured how much energy is consumed by each of these phases, with Figure 6 showing the results. Phase boundaries are determined by means




(a) Socket, R-OSGi(sync/async), RBI, and MOM (b) DOSGi (HTTP/SOAP), RMI, and XMP-RPC

Figure 6: Energy consumption per execution phases.

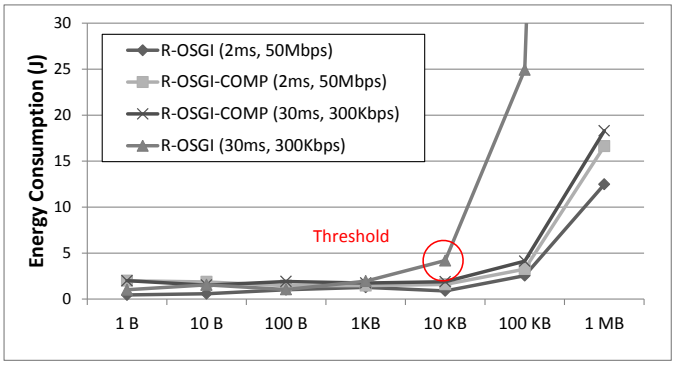
of simple log entries inserted by our reference implementation. The X-axis represents logical time and the Y-axis means cumulative energy consumption. The dotted line means the end of each execution phase.

Despite its name, the idling phase is important: when an application is long-running, the energy consumed when idling may constitute a significant percentage of the application's energy budget. In fact, our measurements indicate that some DPA mechanisms may consume more energy when idling than when executing remotely, for some application patterns. During idling, it is the open network connections that consume energy. In other words, keeping a network connection open indefinitely (i.e., until the application exits or the connection is interrupted) consumes energy at a constant rate. As we have determined, however, RMI, DOSGi, and XML-RPC consume no energy when idling. Using these DPAs will save energy for long running applications that experience prolonged idle periods.

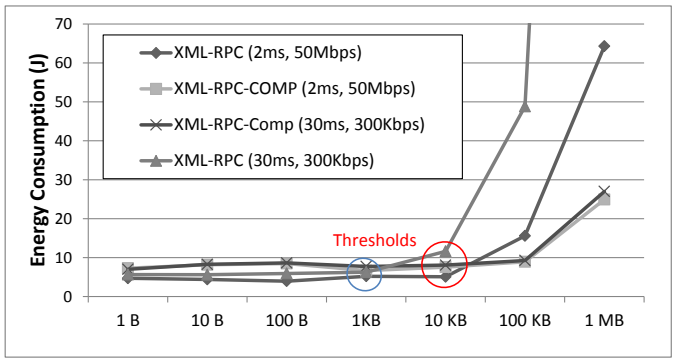
*3.4.5. Scenario 4—Impact of Energy Optimization Techniques*

In our last experiment, we measured how much energy can be saved by compressing the data transferred across the network. To quantify how much energy this commonly used optimization could potentially save, we measured the respective energy consumption rates when using a binary-based DPA of R-OSGi and a text-based DPA of XML-RPC to transfer compressed and uncompressed data. We emulated two distinct network environments with the following round trip time (RTT) and bandwidth characteristics—a high-end mobile network (2 ms latency, 50 Mbps bandwidth) and a low-end mobile network (30 ms latency, 300 Kbps bandwidth).

Figure 7 shows the impact of compressing the transferred data on energy




(a) Binary-based DPA (R-OSGi)




(b) Text-based DPA (XML-RPC)

Figure 7: Impact of compressing transferred data on energy consumption.

consumption for both of these emulated network environments. For the low-end mobile network, the binary-based DPA transferring uncompressed data consumes insignificant amounts of energy when transferring compressed data, until the transferred data's volume reaches 10 KB. Beyond this threshold, the energy consumed transferring compressed data is lower than that consumed transferring uncompressed data. Therefore, for the low-end mobile network, the 10 KB threshold is a point at which one can expect to see the benefits of compression. On the other hand, for the high-end mobile network, no such threshold has been observed. Therefore, for high-end mobile networks, compression only consumes additional energy, suggesting the need to consider an alternative energy optimization strategy.

The thresholds at which compression starts to save energy in XML-RPC are 1 KB and 10 KB for the low-end and high-end network environments, respectively. Thus, because the network environment in place determines

the threshold at which compression should be engaged to reduce energy consumption, the DPA runtime system should be able to turn this optimization on and off as needed.

### *3.5. Threats to Validity*

The measurements above are subject to both internal and external validity threats. The internal validity is threatened by the way in which we chose to use the different DPAs to implement our benchmarks. In our day-to-day programming practices, we do not regularly use all of the measured DPA mechanisms. Therefore, the way we integrated them into our benchmarks may not be fully optimal, both in terms of the APIs used and the configuration options specified. It is likely that a programmer deeply experienced in any of the measured DPA may implement the same functionality exhibiting the energy consumption patterns differing from our observations. However, the energy consumption patterns we have discerned are mainly due to the physical properties of the measured setups, and as such are unlikely to change drastically when accessed through different APIs or configurations.

The external validity is threatened by our measurement infrastructure. Rather than measuring the physical consumed energy directly, the pTopW profiler estimates the consumed energy based on the actual resource usage information. As a result, the reported energy numbers are likely to be less precise than those that would be measured through specialized hardware. In addition, we had no choice but to execute empty remote methods to work around the limitations of process-level energy consumption measurement. Nevertheless, our goal is to infer energy consumption patterns, whose identification is tolerant to approximated energy consumption numbers, as long as they remain proportional to each other.

## 4. Result Analysis

We next analyze the results obtained from the experiments above. We first present and discuss the *Energy Consumption/Performance ratio* of the benchmarked DPA mechanisms. Then, we report on the correlation between network characteristics, the amount of transferred data, and energy consumption.

### *4.1. Energy Consumption/Performance Ratio*

One can interpret the amount of energy consumed by a DPA mechanism as *the cost* of performing some useful activity. An important goal for a

DPA mechanism is performance—the aggregate latency of executing a remote operation. To compare the DPAs more comprehensively, we computed the *Energy Consumption/Performance ratio* (EPR) for the studied platforms, where the price component is total energy consumption.

Figure 8 presents the performance numbers for each DPA, defined as the total roundtrip time to execute the **void** ping (**byte**[]) method. We executed this method over an emulated network with 2 ms latency and 50 Mbps bandwidth 100 times with the results averaged. XML-RPC and DOSGi take the longest time to execute the benchmark method. In general, asynchronous platforms and RBI are more efficient than synchronous platforms, such as synchronous R-OSGi, DOSGi, RMI, and XML-RPC.

To correlate performance and energy consumption, we define the performance-energy consumption ratio as:

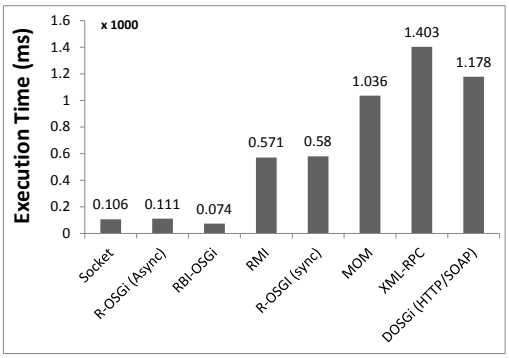
$$EPR(x) = \frac{P_x/MAX(P_0,...,P_n)}{E_x/MIN(E_0,...,E_n)} \times 100$$

where, $P$ is the performance represented by the total execution time and $E$ is the energy consumption.

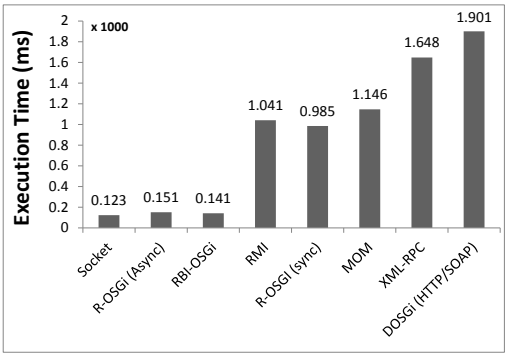
The energy consumption numbers came from Section 3.4.2. Figure 9 shows calculated *Energy Consumption/Performance ratio* numbers. Not surprisingly, raw sockets have the highest EPR for any transferred data size. Asynchronous RPC has the next highest EPR. Batching remote calls in RBI also yields a high EPR. Synchronous RPC has a low EPR when transferring small data volumes. XML-based DPA mechanisms have the worst EPR for any data size. MOM's EPR is somewhere in between, increasing proportionally to the transferred data size.

## 5. Energy Consumption Patterns and Guidelines

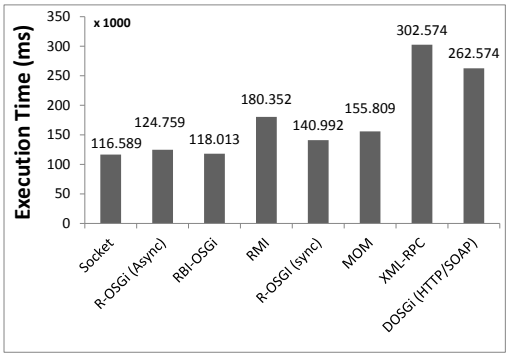
Based on the results obtained from the experiments above, we next attempt to infer some general energy consumption patterns in DPA mechanisms. Even though we infer the following patterns by analyzing the results obtained from benchmarking the eight DPAs above, we express these patterns in general terms, making them applicable to a wide variety of DPA mechanisms. These patterns should inform software designers charged with the challenges of choosing the right DPA for energy-constrained application scenarios.




(a) 1 Byte




(b) 1 KByte




(c) 1 MByte

Figure 8: Total execution time.

Because the same application behavior can be implemented by using any of the equivalent DPAs, being aware of the application's energy consumption patterns becomes an important decision support aid for software designers. By matching these patterns with the intended application behavior, a software designer can make an informed choice when deciding which of the available DPA mechanisms should be applied to a given application scenario.

*5.1. Energy Consumption Patterns*

- Transferring increasing volumes of data over limited networks (characterized by diminishing bandwidths and growing latencies) causes a direct increase in energy consumption. Therefore, when an application is likely to be executed over a limited network, software designers should favor binary-based DPA mechanisms over XML-based ones, as the former ones encode the transferred data more concisely, which reduces the bandwidth requirements.

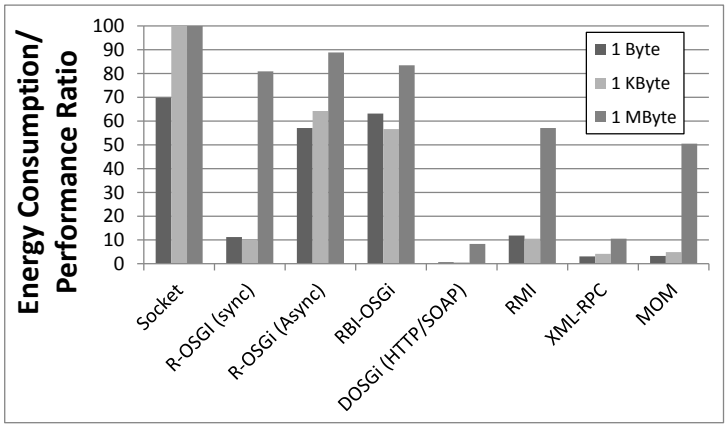



Figure 9: Energy Consumption/Performance ratio.

- In the case of high latency networks, asynchronous DPA mechanisms should be used to avoid having to block remote communications while waiting for a response. Our results indicate that asynchrony does not incur additional energy costs, and as such constitute a viable software building block in the presence of high latency.

- Even though marshaling/unmarshaling can be computationally complex, these functionalities tend to consume more energy on network communication than on CPU processing. Transferring large, complex object graphs across a network requires high bandwidth. At the same time, encoding objects into concise binary representations requires substantial CPU processing. Therefore, for high-throughput networks, simple serialization protocols can yield an acceptable energy consumption, as it would reduce CPU processing and would transfer larger volumes of data without causing an energy consumption spike due to insufficient bandwidth. However, if the underlying network is limited (high latency and low bandwidth as in a congested network), designs that employ CPU-intensive routines to serialize the transferred data concisely should be preferred, as they would reduce the energy consumed by network processing, a dominant energy consumption ingredient for these types of network.

- Applications with long idle periods should prefer DPA mechanisms that do not consume energy when idling. Sophisticated DPA features,

such as issuing heartbeat messages and alternate service discovery, do consume energy even if no core DPA-related functionality is utilized. Therefore, for applications with prolonged idle cycles the energy costs of idling DPA mechanisms should be taken into account. For example, using DPA mechanisms that follow stateless communication protocols (e.g., Web services) can reduce the overall energy consumption, as these mechanisms do not maintain any state between remote interactions.

- When both high performance and low energy consumption are equally at stake, no high-level DPA can outperform raw sockets, which has the highest energy consumption/performance ratio. However, asynchronous and batched RPC are a close second, while offering convenient programming abstractions to the programmer. When large data volumes are to be transferred across the network, binary DPA mechanisms offer a higher ratio than XML-based ones.

### *5.2. Designing Energy-Efficient DPAs*

Based on our results, we next present several guidelines for designing energy efficient DPAs.

#### *5.2.1.* Minimize network interactions and transferred data size

As network communication incurs the largest costs in the overall energy budget, an energy efficient DPA mechanism should strive to transmit small data volumes over high-throughput networks. Because limited network conditions are hard to avoid, system designs should aim at minimizing the frequency of network interactions and reducing the transferred data size. To achieve the first objective, energy-sensitive designs should minimize state exchange messages (e.g., service discovery message, heartbeat, etc.) to an absolute minimum. To achieve the second objective, binary protocols should be favored over XML-based ones, data compression and advanced serialization (e.g., kryo, protobuf, etc)[5] should be used, and delta should be applied whenever possible. Because algorithmically intensive data compression can increase the CPU energy consumption, the right trade-offs should be sought between transferring smaller data and encoding it into compressed formats.

[5]Various versions of serialization tools have been tested and discussed here: https://github.com/eishay/jvm-serializers/wiki/

*5.2.2.* Share core DPA components across different applications

If more than one mobile application can share the same DPA mechanism, the device's aggregate energy consumption may be reduced for two reasons: (1) the initialization phase in a DPA mechanism can consume substantial energy and should be amortized across multiple applications whenever possible; (2) because when idle, a DPA mechanism can still incur energy costs, sharing the infrastructure across applications will reduce its idling time. In the OSGi framework, multiple applications can share common components, realizing the benefits outlined above.

*5.2.3.* Monitor energy consumption levels and handle outliers

For controlling fine-grained energy consumption at the application level, DPA mechanisms should provide energy-monitoring APIs. Such APIs can monitor energy consumption and report potential usage outliers (e.g., attempting to send a large data volume over a limited network). The programmer can then implement functionality to handle such abnormalities by either postponing the remote interaction or even replacing it with some local computation. Energy monitoring should not, however, consume additional energy. Estimating energy consumption rather than measuring it directly provides a pragmatic trade-off.

*5.2.4.* Provide different connection management mechanisms

Connection management policies differ: a connection can be terminated after each remote interaction or reused a varying number of times. Reusing connections can both waste and save energy, if managed flexibly. To provide such flexibility, an API can provide methods to select an appropriate connection management policy for a given application's characteristics. For example, for an application rarely invoking remote methods, establishing a connection at every request saves energy. However, for an application frequently invoking remote methods, it is reusing a connection that saves energy. The programmer should have the flexibility to specify the desired policy on a per-application basis.

*5.2.5.* Flexibly adapt at runtime

Based on our measurements, the underlying network environment determines whether and after which threshold the transferred data should be compressed to save energy. A DPA tuned for a particular network through a set of static optimizations is unlikely to consume an optimal amount of

energy when operating over networks with fluctuating bandwidth/latency characteristics. An effective energy consumption behavior requires that the DPA switch optimizations on and off dynamically in response to such fluctuations. Although it is the application's business logic that determines what data needs to be transferred across the network, the DPA in place can cluster, encode, and compress the transferred data by means of adaptive optimization.

### *5.3. Discussion*

The benchmark results presented above gave rise to the following two insights: 1) the latency/bandwidth characteristics of mobile networks can heavily affect the energy consumption of a mobile application and 2) adapting the execution behavior of a DPA in response to changes in latency/bandwidth can reduce the overall energy consumption. Based on these two basic insights and the presented guidelines, DPA designers should be able to create novel, energy-aware DPAs. In the following discussion, we give a concrete example of how such an energy-aware DPA can handle adaptive energy optimization.

*Example: Adaptive Data Marshaling*

Data marshaling refers to the process of encoding program data into a format that can be transferred across the network. For example, an integer value can be encoded as a byte buffer. The unmarshaling process reverses the marshaling encodings. Multiple marshaling strategies can be applied to the same program data. With respect to energy consumption, one can consider the trade-off between CPU processing and network transfer. Marshaling the data into a smaller byte buffer will reduce network transfer, but will be more computationally intensive, thus requiring additional CPU processing. Marshalling the data into a larger byte buffer will result in transferring more data over the network, but it will require less CPU processing. Which of the strategies will consume less energy depends on the runtime conditions in place.

For example, if the underlying network is limited (high latency and low bandwidth as in a congested network), transferring data concisely should be preferred, as it would reduce the energy consumed by network processing. In case of high-throughput networks, simple serialization protocols can reduce the overall energy consumption, as it would require less CPU processing while transferring larger volumes of data without the energy consumption spikes due to insufficient bandwidth. Therefore, the right trade-offs can only

be determined at runtime, as it depends on the current network conditions. Whether to compress the transferred data presents another trade-off between data size vs. processing overhead. Similar to basic marshaling, algorithmically intensive data compression or delta calculation is computationally intensive, while reducing the amount of data transferred across the network.

In summary, the discussion above presented several high-level guidelines that can be applied to designing energy-aware DPAs. A key insight is that because each mobile application has different execution patterns and environments, applying a single energy-optimization strategy to all execution patterns for all network conditions is ill-advised. Thus, mobile application programmers must understand the execution patterns of mobile applications to be able to implement and configure application-specific, dynamic energy optimization strategies.

## 6. Related Work

To the best of our knowledge, this work is the first attempt to assess the energy consumption characteristics of DPA mechanisms. However, several prior efforts have informed and inspired this work. These efforts fall into three major categories: studies assessing different properties of DPAs, measuring software energy consumption, and energy saving strategies for various computer system layers.

### *6.1. Studies of DPA mechanisms*

Different properties of distributed programming abstractions have been assessed, including performance, scalability, reliability, and programming effort. Gokhale et al. [10] assess how the abstraction level of a distributed programming abstraction affects its performance. Other efforts focused on evaluating MOM and JMS implementations in terms of their respective performance, scalability, and reliability [32, 26]. Our prior work [17] compares DPA mechanisms in terms of performance, reliability, and programming effort. This work complements these studies by assessing the energy consumption of major DPA mechanisms.

### *6.2. Energy Consumption Measurement*

Because energy efficiency has become an important consideration in software design, several recent research efforts have focused on creating effective approaches to measuring energy consumption. Three primary approaches

have been described in the literature: at the architecture, network, and application levels. An example of an architecture level energy measuring approach is PowerPack [8], which physically connects to the CPU, disk, memory, and mother board component to measure and analyze the energy consumption of high-performance applications. Then, it maps the measured energy consumption patterns to the application's source code, making it possible to analyze energy consumption both at the hardware and source code levels. An example of a network level energy measuring approach is described in reference [1], which measures energy consumption of the general network activity for 3G, GSM, and WiFi networks. Examples of an application level measurement approach are described in reference [11], which measures how VoIP applications consume energy, and in reference [33], which measures how video streaming applications consume energy; both of these focus on mobile phones as their execution environment. JALEN monitors runtime energy consumption by injecting the monitoring code into Java bytecode [22]. As compared to the architecture, network, and application levels measurements, the focus of this work is on DPAs or middleware, a software layer that is situated in between hardware and software layers. Nevertheless, our measurement methodology is an example of an application level approach.

### *6.3. Energy-Saving Techniques*

Extending the battery life of mobile devices by reducing the energy consumption of mobile applications has been the focus of multiple complimentary research efforts: energy-efficient design patterns and programming languages [21]), offloading energy-intensive functions to a remote server [16], using specialized-network protocols [1], or switching different algorithms according to pre-defined energy consumption scenarios [13].

While the majority of these efforts focused on one particular system layer (i.e., mainly the network), advanced techniques have been proposed to utilize multiple levels of system information, a technique called a cross-layer approach. A cross-layer approach can effectively control energy consumption by leveraging the information provided by multiple system layers. DYNAMO [19] is a middleware platform that adapts power optimization strategies across various system layers, including applications, middleware, OS, network, and hardware, to optimize both performance and energy. The focus of DYNAMO is on reducing energy consumption for video streaming applications. Our application energy consumption patterns can provide empirical results to efforts such as DYNAMO, which can interpret and apply

them to specific application domains.

A recent language-based approach to energy-aware programming is ET [3], a new object-oriented programming language that enables the programmer to write energy-aware code by specifying *phases*, which represent distinct program workloads, and *modes*, which represent required energy states, such as high and low energy consumption. A language like ET can be a useful tool for distributed application programmers who want to take advantage of our DPA energy consumption patterns.

## 7. Conclusions and Future Work

As mobile devices are rapidly replacing stationary computers as the primary means of accessing computing resources, battery lives now play a critical role in the end-user experiences. As a result, energy efficiency has come to the forefront of system design. Because distributed programming abstractions are a mainstay for the majority of distributed applications, the energy efficiency of DPA mechanisms can have a huge impact on the overall energy budget. This article closes the gap in our understanding of energy consumption in DPA mechanisms. By systematically measuring and analyzing the constituent parts of major DPA mechanisms, we have identified their energy consumption patterns. These patterns will inform both software designers and distributed system researchers.

As a future work, we plan to conduct more systematic measurements of other DPA mechanisms under a wider range of networks. We may also experiment with obtaining our energy consumption numbers from physical devices and compare their accuracy. As a result, our energy consumption model may need to be adjusted. We also plan to create a new DPA mechanism that takes our findings into consideration. By combining energy-efficient features from the mainstream DPAs, we hope to obtain a new platform that offers the same abstractions energy-efficiently by considering both the runtime environment and application semantics. This new platform will benefit all distributed applications that are mindful of their energy consumption.

## Acknowledgments

This research is supported by the National Science Foundation through the grant CCF-1116565.

## References


[1] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the $9^{th}$ ACM SIGCOMM conference on Internet measurement conference (IMC '09)*, 2009.

[2] Cisco Market Trends. Cisco service provider: Wi-Fi: Offload mobile data and create new services, 2012.

[3] M. Cohen, H. S. Zhu, and S. E. E. ad Yu David Liu. Energy types. In *Proceedings of the 2012 ACM international conference on Object-oriented programming systems, languages, and applications*, Oct 2012.

[4] S. Colbert. Speech at the White House Correspondents' Association dinner. Transcript, April 26 2006.

[5] T. Do, S. Rawshdeh, and W. Shi. pTop: A process-level power profiling tool. In *Proceedings of the $2^{nd}$ Workshop on Power Aware Computing and Systems (HotPower'09)*, 2009.

[6] Gartner, Inc. Gartner highlights key predictions for IT organizations and users in 2010 and beyond, Jan. 2010.

[7] R. Gass and C. Diot. An experimental performance comparison of 3G and Wi-Fi. In *Proceedings of the $11^{th}$ international conference on passive and active measurement (PAM '10)*, 2010.

[8] R. Ge, X. Feng, S. Song, H. Chang, D. Li, and K. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Transactions on Parallel and Distributed Systems*, 21(5):658–671, 2010.

[9] J. Gibbons and S. Ruth. Municipal Wi-Fi: big wave or wipeout. *Internet Computing, IEEE*, 10(61):107–125, 2006.

[10] A. Gokhale and D. C. Schmidt. Measuring the performance of communication middleware on high-speed networks. In *Proceedings on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '96)*, 1996.

[11] A. Gupta and P. Mohapatra. Energy consumption and conservation in wifi based phones: A measurement-based study. In *Proceedings of the $4^{th}$ Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, 2007 (SECON '07)*, June 2007.

[12] B. Haumacher, T. Moschny, and M. Philippsen. The JavaParty project. `www.ipd.uka.de/JavaParty`, 2007.

[13] Y. Huang, S. Mohapatra, and N. Venkatasubramanian. An energy-efficient middleware for supporting multimedia services in mobile grid environments. In *Proceedings of International Conference on Information Technology: Coding and Computing, 2005.*, volume 2, April 2005.

[14] A. Ibrahim, Y. Jiao, E. Tilevich, and W. R. Cook. Remote batch invocation for compositional object services. In *The $23^{rd}$ European Conference on Object-Oriented Programming (ECOOP '09)*, July 2009.

[15] A. Kansal and F. Zhao. Fine-grained energy profiling for power-aware application design. *SIGMETRICS Perform. Eval. Rev.*, 36(2):26–31, Aug. 2008.

[16] Y.-W. Kwon and E. Tilevich. Energy-efficient and fault-tolerant distributed mobile execution. In *Proceedings of the $32^{nd}$ International Conference on Distributed Computing Systems (ICDCS '12)*, June 2012.

[17] Y.-W. Kwon, E. Tilevich, and W. Cook. Which middleware platform should you choose for your next remote service? *Service Oriented Computing and Applications*, 5:61–70, 2011.

[18] Microsoft Research. Network Emulator for Windows Toolkit (NEWT) version 2.1, 2010.

[19] S. Mohapatra, N. Dutt, A. Nicolau, and N. Venkatasubramanian. DYNAMO: A cross-layer framework for end-to-end QoS and energy optimization in mobile handheld devices. *Selected Areas in Communications, IEEE Journal on*, 25(4):722 –737, May 2007.

[20] R. Monson-Haefel and D. Chappell. *Java Message Service*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2000.

[21] A. Noureddine, A. Bourdon, R. Rouvoy, and L. Seinturier. A preliminary study of the impact of software engineering on greenIT. In *Proceedings of the First International Workshop on Green and Sustainable Software*, Jun 2012.

[22] A. Noureddine, A. Bourdon, R. Rouvoy, and L. Seinturier. Runtime monitoring of software energy hotspots. In *Proceedings of the* $27^{th}$ *IEEE/ACM International Conference on Automated Software Engineering (ASE '12)*, 2012.

[23] OSGi Alliance. OSGi service platform release 4.2 specification, 2011.

[24] M. Philippsen, B. Haumacher, and C. Nester. More efficient serialization and RMI for Java. *Concurrency: Practice and Experience*, 12(7):495–518, 2000.

[25] J. S. Rellermeyer, G. Alonso, and T. Roscoe. R-OSGi: Distributed applications through software modularization. In *Proceedings of the ACM/IFIP/USENIX* $8^{th}$ *International Middleware Conference*, November 2007.

[26] K. Sachs, S. Kounev, J. Bacon, and A. Buchmann. Performance evaluation of message-oriented middleware using the specjms2007 benchmark. *Performance Evaluation*, 66(8):410 – 434, 2009.

[27] C. Seo, S. Malek, and N. Medvidovic. An energy consumption framework for distributed Java-based systems. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering (ASE '07)*, 2007.

[28] J. W. Stamos and D. K. Gifford. Remote evaluation. *ACM Trans. Program. Lang. Syst.*, 12(4):537–564, 1990.

[29] M. Sullivan. PCWorld: 3G and 4G wireless speed showdown: Which networks are fastest? `http://www.pcworld.com/article/253808/3g_and_4g_wireless_speed_showdown_which_networks_are_fastest_.html`, April 2012.

[30] The Apache Software Foundation. ActiveMQ. `http://activemq.apache.org/`, 2010.

[31] The Apache Software Foundation. Apache CXF Distributed OSGi. `http://cxf.apache.org/distributed-osgi.html`, 2010.

[32] P. Tran, P. Greenfield, and I. Gorton. Behavior and performance of message-oriented middleware systems. In *Proceedings of the* $22^{nd}$ *International Conference on Distributed Computing Systems (ICDCS '02)*, 2002.

[33] Y. Xiao, R. Kalyanaraman, and A. Yla-Jaaski. Energy consumption of mobile YouTube: Quantitative measurement and analysis. In *Proceedings of the Second International Conference on Next Generation Mobile Applications, Services and Technologies, 2008 (NGMAST '08)*, pages 61 –69, Sept. 2008.