Energy Impact of Secure Computation on a Handheld Device[*]

Zhiyuan Li [†] Rong Xu
Department of Computer Sciences, Purdue University
West Lafayette, IN 47907
{li,xur}@cs.purdue.edu

Abstract

Computation offloading is an important approach to save the energy consumption while improving performance for wireless networked handheld devices. With such an approach, computational tasks are offloaded from the handheld device to a server, depending on the tradeoff between the communication cost and the computation cost. Adding security to the wireless network changes the relative cost of computation and communication. In this paper, we measure the energy consumption characteristics of multimedia applications on a handheld device, supported by computation offloading though a wireless LAN which is secured with IPsec. The measurement indicates that despite the overhead of the security mechanism, offloading remains quite effective as a method to reduce program execution time and energy consumption.

1 Introduction

Handheld computing devices with wireless network connections have the potential to become powerful mobile tools to access information and software resources from anywhere at any time. However, the short battery life on these devices has been a major obstacle. For example, current high-end pocket PCs with wireless connection can stay active for just a few hours. Battery conservation is widely recognized as a key research issue for handheld and mobile computing. Many architectural and software efforts have been initiated to improve the energy efficiency of handheld devices [4, 5, 6].

Computation offloading is one of the most effective ways to save the energy consumption while improving performance for wireless networked handheld devices. By offloading, the handheld devices obtains the desired results without performing all the operations. The battery energy spent on computation is thus saved. However, there is an energy cost for data transfer via the wireless connection. Careful tradeoff must be made between energy consumed by network communication versus computation. In our recent work, we studied computation offloading techniques for reducing energy consumption and execution time. Our result showed substantial reduction in both execution time and energy consumption for a suite of benchmark programs.

Another critical issue is security. Despite the great benefits which wireless networks can provide, the security of wireless LANs remains a big concern. It is well known that IP protocol is inherently insecure. IP packets can be easily forged. It is easy to inspect and manipulate the addresses or contents of packets. For traditional wired network, this may not be a big problem, because one has to access the physical wire to break the security. In contrast, wireless technology has no inherent physical protection. Radio waves can travel freely through most physical barriers, easily spreading confidential data beyond the walls of an office or a home. If not handled properly, this potentially creates a major security hole in a network.

Most current wireless networks are based on widely adopted IEEE 802.11 standard. Unfortunately the security specification, called WEP, in this standard has been proven insecure and is thus inadequate for protecting a wireless network from eavesdropping or abuse [2]. As a result, users who have high security concerns turn to other protection measures. Secure Socket Layer (SSL), IP security protocol (IPsec) [3], for example, have been used to provide strong encryption and authentication for wireless connections.

In our previous work [9, 10], we presented algorithms for computation offloading based on models which do

[*]©IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

[†]The author names are listed in alphabetical order.

not take into account the performance and energy cost of the security mechanisms. The experiments were also performed on insecure wireless links. Adding security to the network affects offloading in several ways. First, it consumes extra energy to decrypt the inbound communication and encrypt the outbound communication. Considering the limited computation power of handheld devices, these encrypt/decrypt operations may incur delays to communication, which decreases the effective bandwidth and thus increases the energy for communication. Second, the encryption and authentication may increase the size of the IP packets, which in turn may lower the effective bandwidth and increase the communication cost. Third, running the secure protocol, including key exchanges, incurs extra network traffic.

Adding security to wireless network changes the relative cost of computation and communication. This, therefore, has an impact on the tradeoff between the computation cost and the communication cost. Recently, we have implemented IPsec on a handheld device and a proxy server for offloading. In this paper, we measure the energy consumption characteristics of the handheld device and reexamine the offloading technique in the wireless network environment which is secured with IPsec.

In the remaining sections of the paper, we first review our previous work of offloading in Section 2. We then briefly review IPsec (Section 3). In Section 4, we report and compare the results of offloading in the IPsec environment. This is followed by a discussion of related work and a conclusion.

2 Computation Offloading

Given a program, computation offloading scheme identifies the computation tasks which can be offloaded from the handheld device to a server such that the program execution time and energy consumption can be reduced. For many programs, for example multimedia programs, the energy tradeoff is nonobvious. We performed off-line profiling and used an energy model to estimate energy spent on executing individual procedures (on the handheld) and on data communication between procedures which are separated (between the server and the handheld). Based on an energy-minimization objective function, we apply a task-partition algorithm to determine the procedures that can be beneficially offloaded. We have considered the following two areas in which computation offloading can be useful.

All Data Local: This is the case in which both the input and the output are local to the handheld device, and inherently the application has no need to communicate with the external world. On the other hand, offloading the computation in part or as a whole requires sending data needed by the offloaded tasks and receiving the results back. Figure 1 illustrates this case.

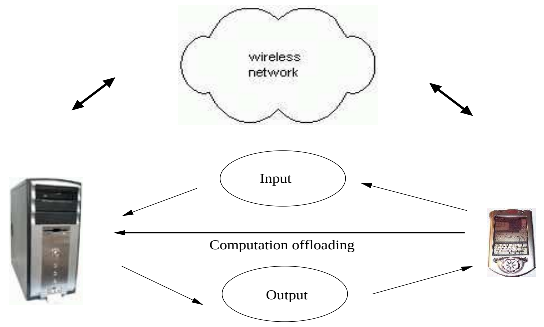

Figure 1. Computation of¤oading where all data are local.

Data Transferred with Heavy Processing: When the handheld device downloads data for processing (e.g. decompressing and replaying downloaded multimedia data) or uploading data after processing (e.g. sending multimedia data via email after compression), where the processing should take place (i.e. whether on the server or on the handheld) becomes an interesting question. The tradeoff can be nonobvious. Downloading data after decompression will save energy for computation but at the expense of spending more energy on communication. Performing decompression locally, on the other hand, will reverse the tradeoff argument. Figure 2 illustrates this case.

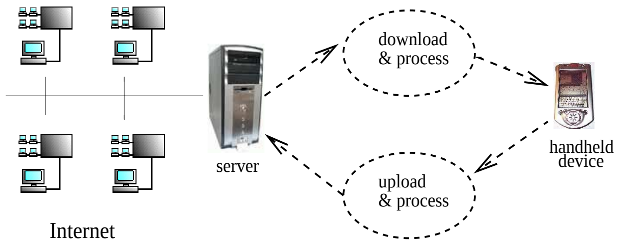

Figure 2. Computation of¤oading where data are transferred with heavy processing.

We find that a common energy consumption model can be developed for a class of application scenarios, including both cases mentioned above [9, 10]. Using a simplified data-sharing model, the task partition problem can be transformed to a *max-flow/min-cut* problem [10], for which low-degree polynomial algorithms exist. A more refined data-sharing model requires *Branch-and-Bound* for its solutions. Graph-pruning is performed to ensure fast partitioning [9].

3 IPsec

IPsec provides a viable solution for securing IP traffic. It uses strong encryption and authentication algorithms to protect the integrity and confidentiality of traffic at the IP level. Working at this level, IPsec is transparent to applications while protecting any traffic carried over IP, unlike other encryption mechanisms which generally protect only a particular higher-level protocol, for example, PGP for mail, SSH for remote shell, and SSL for TCP socket.

IPsec can be used on any machine which performs IP networking. IPsec is typically installed on gateway machines to protect traffic. It can also be installed on a variety of other nodes such as routers, firewall machines, application servers, and end-user desktop or laptop machines.

IPsec offers two services, *authentication* and *encryption*. These can be used separately but are often used together. Packet-level authentication assures the integrity of a packet. It ensures that a packet comes from an expected data origin and that its contents are not tampered. The encryption component encrypts contents of a packet to protect them from eavesdroppers.

Three protocols used in an IPsec implementation:

AH (Authentication Header) provides a packet-level authentication service but does not encrypt the packet.

ESP (Encapsulating Security Payload) provides encryption plus authentication for a packet.

IKE (Internet Key Exchange) negotiates connection parameters, including keys, for the other two protocols.

Establishing secured communication between end systems is a two-phase process. In phase one, the two gateways establish a security association (SA). After an SA is established, each end system possesses a set of keys that the systems use to encrypt the phase-two traffic. As part of phase one, each end system must identify and authenticate itself to the other. In phase two the gateways negotiate SAs and generate the keys that the end systems use to encrypt the IP traffic sent between them. The systems can reuse valid SAs from phase one to establish multiple SAs in phrase two. SAs in both phase one and phase two regularly expire and reestablish to ensure the integrity of IPsec communications.

IPsec has two modes of operations. The *transport mode* supports a host-to-host connection involving only two machines. In the *tunnel mode*, the IPsec machines act as gateways to carry traffic for any number of client machines. The *tunnel mode* is considered more expensive than *transport mode*.

The application of IPsec to IP traffic can result in a considerable increase in the size of IP datagrams. For example, in *tunnel* mode, a new IP header is created for each datagram. More over, there may exist multiple layers of IPsec protocols applied to a datagram. To reduce such extra data traffic, IPsec includes the IP Compression protocol (IPComp) [12, 13, 1]. IPComp utilizes lossless compression algorithms to reduce the size of the IP datagram. It compresses outbound IP datagrams and decompresses inbound datagrams. However, IPComp does not guarantee a decrease of the size of the datagram. As shown in the next section, sometime it results in an increase rather than a reduction.

4 Experimental Result

4.1 Experimentation Setup

The handheld device used in our experiments is a Compaq iPAQ 3650 which has a 206MHz Intel StrongArm SA1110 processor and 32MB RAM. The proxy server is a Dell Dimension 4100 which has a 1GHz P-III processor. Both machines run Linux operating systems and communicate through TCP socket calls. The wireless connection is through a Lucent Orinoco (WaveLAN) Golden PCMCIA card which follows the IEEE 802.11b standard. Under the 11 Mb/s nominal peak rate, the effective data rate of the WaveLAN card is measured as about 5 Mb/s. The bit rate (for both send and receive) can be adjusted downward in a few different ways, by changing the settings of the access point, by increasing the communication distance, or by increasing structure obstacles between the two antennas.

To measure the electrical current drawn by the handheld device, we connect it to an HP 3458a low-impedance (0.1 Ω) digital multi-meter which takes several hundred samples per second and automatically records maximum, minimum and average electrical current. In order to get a reliable and accurate reading, we disconnect the batteries from both the iPAQ and the extension pack, using an external 5V DC power supply instead. The start and finish of the meter reading is controlled by software, using the trigger mechanism built in the multi-meter. According to our measurement, the overhead associated with the triggering interrupts is less than 0.5% and the readings are consistent over repeated runs. We compute the energy consumption of system during the execution by the simple equation:

$$energy = voltage * current\_drawn * elapsed\_time$$

4.1.1 IPsec setup

The implementation of IPsec used in our experiments is Linux `FreeS/Wan 1.96`. We use ESP protocol which encrypts the payload by `3DES` and authenticates the datagram by `MD5`. The server and iPAQ are in the same subnet and we setup an IPsec tunnel between them. The two hosts use `RSA` digital signatures to authenticate each other. We find the compression protocol (IPComp) has a rather complex effect on the results of offloading. We will discuss the impact of IP compression on Section 4.4. For the other parts of this section, we do not activate IP compression.

4.1.2 Power parameters

We measure the electrical current drawn in different running modes. The reading is shown in Table 1. (All the numbers are measured with the screen turned off. When using the medium back-light, the current will increase about 140mA for each running mode. Throughout the experiments, the WaveLAN always remains connected.) The first column shows two functioning modes of the iPAQ: *idle* when it does nothing and *busy* when it performs computation. The *Send/Recv* column indicates whether there is intensive sending or receiving. During sending or receiving, the computer is neither idle nor computation-intensive, hence the '–' entry in the iPAQ column. The *IPsec* column indicates whether IPsec is deployed. We do not observe changes in the current due to the IPsec when there exists no network activities. Again, we put the '–' entry in IPsec column. The last column lists the electrical current. drawn from the external DC power.

We maintain a constant voltage power supply of 5V. Hence, with the current readings, we can easily obtain the power parameters needed in our algorithms.

The application of IPsec also decreases the data throughput over the network. Table 2 lists the changes in the average data throughput measured over extensive data communication test cases. These bandwidth parameters are used in the algorithms.

Table 1. Power parameters

iPAQ	Send/Recv	IPsec	Current(A)
idle	no	–	0.33
busy	no	–	0.48
-	send	no	0.44
-	recv	no	0.42
-	send	yes	0.52
-	recv	yes	0.45

Table 2. Effective bandwidth w/ and w/o IPsec

Send /Recv	IPsec	Effective bandwidth(Mbps)
send	no	5.0
recv	no	5.1
send	yes	2.2
recv	yes	2.4

4.1.3 Test programs

Table 3 lists the programs used in our experiments, their descriptions and their input parameters, including the input files we use, which can be found in the *Mediabench* web-site. We experiment with both modes (all-data-local and input-output-separated) which can be applied to the real world situations. A program may have rather different execution paths to provide different functionalities. We perform tests on those different paths, using different profile information to generate the cost graph. As a result, it may have different versions of transformed code.

4.2 Result with Input and Output Separated

We apply the same flow algorithm as in [10], but using the new power parameters, to the benchmark program with input and output data separated (one on the server and the other on the mobile device). Since it is the handheld device which initiates the data transfer, by default the application program would all process locally. For each program, we test either one of the following two cases as in [10]:

1. the data is transferred from the handheld to the server if the data get consistently decreased by processing.

2. the data is transferred from the server to the handheld if the data get consistently increased by processing.

Table 3. Test programs

Program Name	Description	Input Parameters
toast	GSM, voice transcoding	toast -fpl clinton.pcm
untoast	GSM, voice transcoding	untoast -fpl clinton.pcm
encode	G.721, voice compression	encode -4 -l <clinton.pcm >out.g721
decode	G.721, voice decompression	decode -4 -l <clinton.pcm.g721 >out.pcm
epic	EPIC, image compression	epic test_image -b 25
unepic	EPIC, image decompression	unepic test_image.E test_image.out
pgp/encrypt	PGP, encryption	pgp -fes Bill -zbillms -u Bill <pgptest.plain >pgptest.pgp
pgp/decrypt	PGP, decryption	pgp -fdb -zbillms <pgptest.pgp >pgptest.dec
rawcaudio/compress	ADPCM, speech compression	rawcaduio <clinton.pcm >out.adpcm
rawcaudio/decompress	ADPCM, speech decompression	rawcaduio <clinton.adpcm >out.pcm
ghostscript	Postscript interpreter	gs -sDEVICE=ppm -S OutputFile=test.ppm -dNOPAUSE -q –tiger.ps
cjpeg	JPEG, image compression	cjpeg -dct int -progressive -opt -outfile out.jpeg nasa.ppm
djpeg	JPEG, image decompression	djpeg -ppm -outfile out.ppm nasa.jpeg
pegwit/encrypt	PEGWIT, encryption	pegwit -e my.pub pgptest.plain pegwit.enc <encryption_junk
pegwit/decrypt	PEGWIT, decryption	pegwit -d pegwit.enc pegwit.dec < my.sec
mpeg2encode	MPEG, mpeg2 encoding	mpeg2encode test.par out.m2v
mpeg2decode	MPEG, mpeg2 decoding	mpeg2decode -b mei16v2.m2v -r -f -o0 rec%d

With the new power parameters, our algorithm predicts that for 13 of 17 test programs, energy will be saved by mapping the program, in part or on the whole, to the server. For the remaining 4 programs, namely, `cjpeg`, `djpeg`, `rawcaudio` and `rawdaudio` our algorithm finds that the default running mode is the most energy-efficient. Figure 3 plots the measured energy consumed by each program. The energy of processing on the handheld is normalized to 1. The "*default*" label in the figure indicates that processing is done completely on the handheld device and the "*algorithm*" label indicates that the computation is according to our algorithm. The data showed in the figure validates the decision of our algorithm. To verify the four no-benefit cases, we also tried to let the server completely process the program and then measured the energy consumption. We found that energy consumed by complete offloading to be larger than complete local processing on the handheld. The former is 2.0, 2.7, 2.9 and 2.5 times of `cjpeg`, `djpeg` , `rawcaudio` and `rawdaudio` respectively,

Comparing to our previous result for offloading from an insecure wireless LAN [10], the decision to offload or not remains the same on the secured environment in all the test cases. However, as expected, the benefit margin has narrowed. We obtained an average of 72% energy reduction compared with the default (processing locally). Now, we have 52% of energy reduction.

Figure 4 shows the average electrical current drawn during the execution of the program. To assess the IPsec overhead, we normalize the electrical current reading to the case of local processing without IPsec. From the "*default*" bar, we see the current increasing when IPsec runs. In contrast to previous data [10], where offloading lowers the current on the local device, here the current is usually a little higher (average 6%) than the default due to the cost of IPsec.

Figure 5 compares the running time of benchmark programs. Again, we normalize the data based on the time when running the program locally without IPsec. The "*default*" bar clearly shows the overhead of IPsec. In almost all the cases, the running time increases compared with no IPsec. For `ghostscript`, the running time doubles. The "*algorithm*" bar shows that offloading can obtain significant performance improvement. This is true for several programs even compared local processing without IPsec. On the other hand, compared with previous data [10], the gap of performance difference between *default* and *algorithm* decreases.

4.3 Results with All Data Local

We also reexamine the results of our offloading scheme with both input and output data local to the

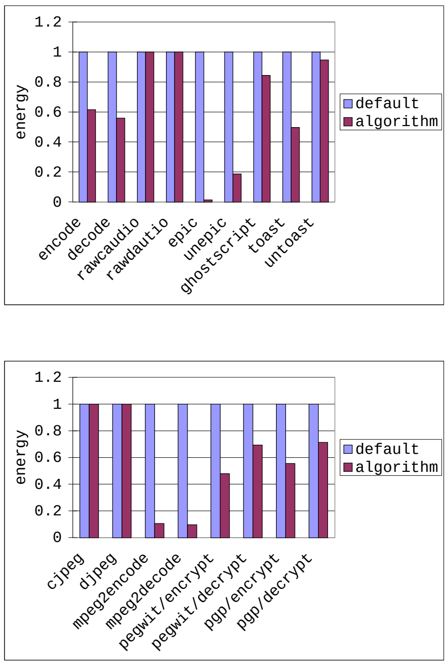

Figure 3. normalized energy consumption on the handheld device using IPsec, where the input and output are separated

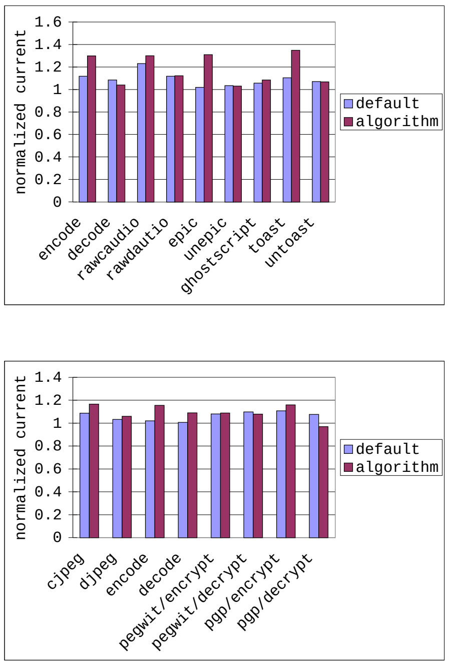

Figure 4. normalized current drawn on the handheld device using IPsec, where the input and output are separated

handheld device. We find that 10 of 17 programs can get better energy usage by offloading in the secure wireless environment. Previously there were 12 of them benefiting from offloading in the insecure environment. The decisions for `untoast` and `pegwit/encrypt` have changed from offloading to local processing due to the IPsec overhead.

Figure 6 plots the energy consumed by the handheld for each program. The "*default*" label indicates the program is executed entirely on the local device. The "*algorithm*" label indicates that our algorithm generates the task mapping. Compared with our previous data [9], we see that, as expected, the benefit of offloading decreases due to the overhead of IPsec. The average energy reduction rate (excluding the locally processed cases) decreases from 66% to 50%. Nevertheless, offloading still achieves significant energy-saving for the handheld device.

For `untoast` and `pegwit/encrypt`, if we continued to apply the same task partition as in our previous experiments, the energy consumption would have been 26% and 65% higher than that of running locally.

Figure 7 compares the average electrical current drawn during execution. The "*w/o IPsec*" label means the code generated by our algorithm in insecure wireless environment. The "*w/ IPsec*" label means the code is running under IPsec. The figure shows that, the computation and networking activities required for securing the wireless communication does increase the current by a nontrivial amount (a 18% increase on average).

Figure 8 shows the running time comparison between IPsec and insecure wireless. Due to variety of time for different programs, we normalize data relative to the baseline which is running on iPAQ. We can see that for those program which the algorithm decides to not running locally, the running time can increase significantly.

4.4 Impact of Compression

Running IPsec can increase the size of IP datagrams because of the longer headers. IP Compression protocol (IPComp) employs lossless compression algorithms to reduce the IP datagram size. If the content of the package has high redundancy, i.e a high compression radio, IPComp can reduce the network traffic significantly. IPComp compresses individual datagrams separately. A large datagram potentially yields a better compression ratio. On the other hand, data compres-

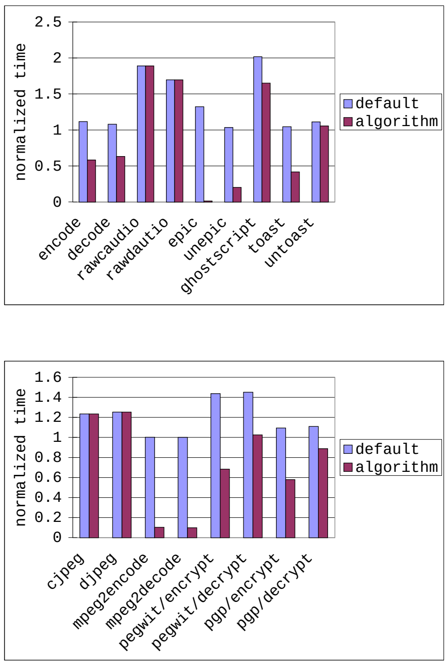

Figure 5. normalized running time on the handheld device using IPsec, where the input and output are separated

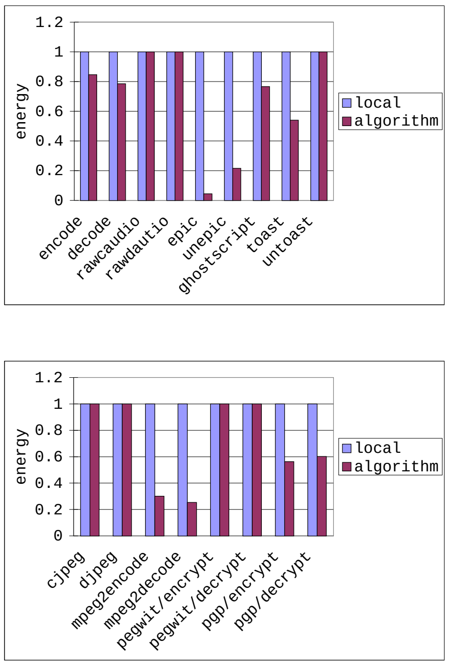

Figure 6. normalized energy consumption on the handheld device using IPsec, with all data local

sion does not guarantee energy saving even though the network traffic is reduced. The CPU energy used for compression and decompression should also be taken into account. The effect of IPComp therefor highly depends on the characteristics of programs.

In order to obtain the parameters needed by our task mapping algorithms in the IPsec environment using IPComp, we measure the data throughput and energy consumption when transmitting and receiving a range of data sets with different compression ratios. The compression factor ranges from 18 to 1. We also varies the datagram sizes. The effective data throughput for downloading data from the server varies from 2.3Mbps to 4.5Mbps and the electric current ranges from 460mA to 570mA. The data throughput for upload data from iPAQ to server varies from 2.0Mbps to 4.5Mbps and the electric current drawn ranges from 510mA to 580mA.

To evaluate the impact of IPComp on computation offloading, we remeasure the energy consumption under IPComp. Figure 9 shows the result of offloading with all data local. Figure 10 shows the result of offloading with input and output separated and with all data local respectively. The data are normalized against the case of local processing running IPsec without IPComp. The figures show that the overhead of compression decreases the benefit of offloading unless the data transferred are highly compressible data (for example, `ghostscript` program performs much better than no-compression cases.)

5 Related Work

The idea of offloading computation on mobile computers has been explored previously [14, 11, 8, 9, 10], and most of them assume all data to be local. To the best of our knowledge, our work is the first to measure the impact of securing the wireless LAN on the performance and battery life on handheld devices.

Ramesh Karri [7] examines the energy consumption characteristics of secure wireless session. Their experiments are performed on Symbol PT2800 Pocket PC running Windows CE. Their work compares energy consumption of different algorithms and different protocols in secure wireless session, but they do not consider computation offloading.

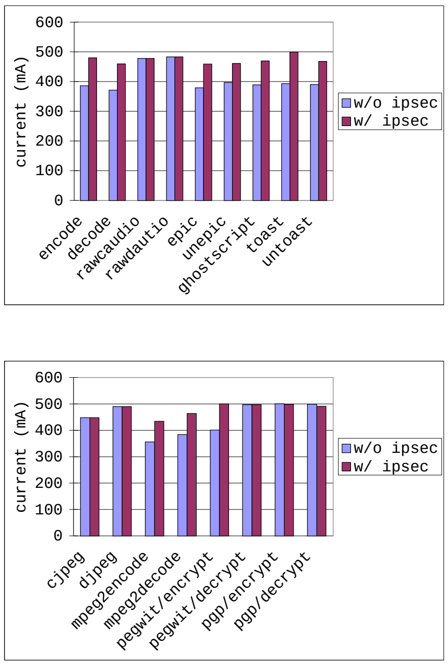

Figure 7. current drawn on the handheld device for programs using the algorithm, with all data local

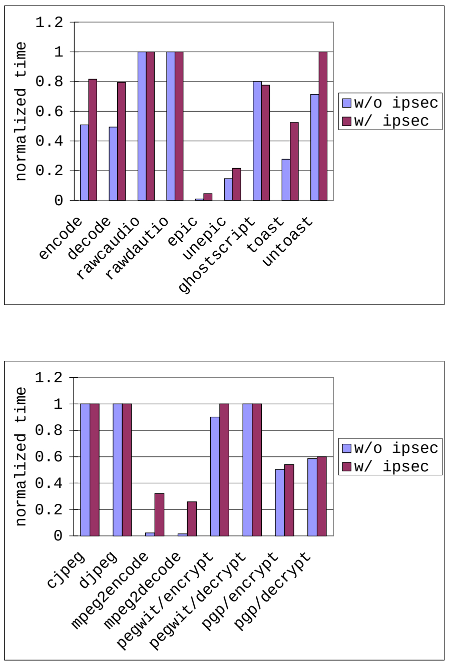

Figure 8. normalized running time on the handheld device for programs using the algorithm, with all data local

6 Conclusion

In this paper, we measure the energy consumption characteristics of offloading multimedia applications on a handheld computer in a wireless network LAN secured with IPsec. Our experiments show that among 25 test cases which previously benefited from offloading, 23 of them remain to benefit. The average energy saving of offloading is 51% which decreases from the average of 70% in a nonencrypted environment. The measurement indicates that, despite the overhead of the security mechanism, offloading remains quite effective as a method to reduce program execution time and energy consumption. In the future work, we shall further investigate the energy consumption attributed to different factors, such as communication, encryption/decryption, and compression/decompression.

References

[1] Shacham A, Monsour R, Pereira R., and Thomas M. IP payload compression protocol (IPComp). *RFC 2393*, December 1998.

[2] Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting mobile communications: the insecurity of 802.11. In *7th Annual International Conference on Mobile Computing and Networking*, Rome, Italy, July 2001.

[3] IP Security Protocol (IPSEC) Charter. http://www.itef.org/html.charters/ipsec-charter.html.

[4] James W. Davis. Power benchmark strategy for systems employing power management. *IEEE International Symposium on Electronics and Environment*, 1993.

[5] Kinshuk Govil, Edwin Chan, and Hal Wasserman. Comparing algorithms for dynamic speed-setting of a low-power cpu. *Proceedings of the First International Conference on Mobile Computing and Networking, MobiCom'95*, pages 13–25, November 1995.

[6] David P. Helmbold, Darrell D. E. Long, and Bruce Sherrod. A dynamic disk spin-down technique for mobile computing. *Proceedings of the second annual ACM International Conference on Mobile Computing and Networking*, November 1996.

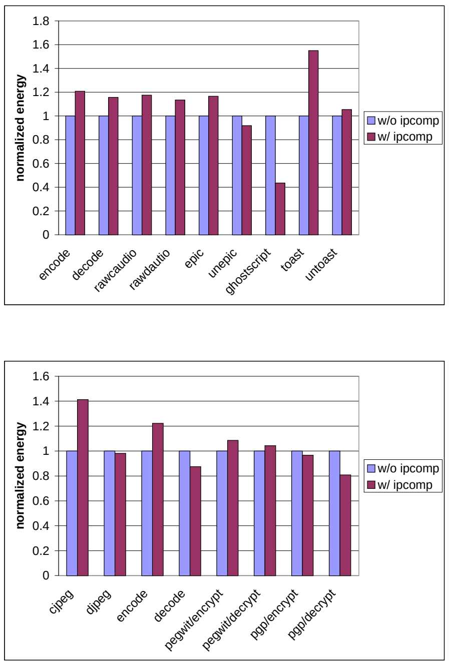

Figure 9. normalized energy consumption on the handheld device using IPsec w/ compression (all data are local)

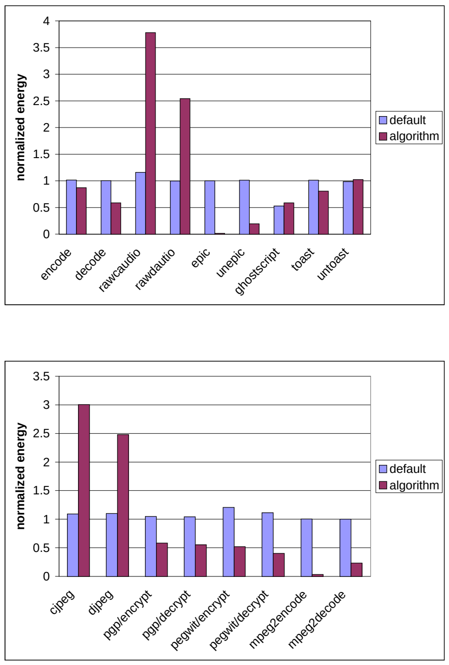

Figure 10. normalized energy consumption on the handheld device using IPsec w/ compression (input and output are separated)

[7] Ramesh Karri and Piyush Mishra. Minimization of energy consumption of secure wireless session with qos constraints. In *Proceedings of IEEE International Conference on Communications*, New York city, NY, April 2002.

[8] Ulrich Kermer, Jamey Hicks, and James M. Rehg. A compilation framework for power and energy management on mobile computers . In *14th International Workshop on Parallel Computing (LCPC'01)*, August 2001.

[9] Zhiyuan Li, Cheng Wang, and Rong Xu. Computation offloading to save energy on handheld devices: A partition scheme. In *International Conference on Compiler, Architecture and Synthesis for Embedded Systems*, pages 238–246, Atlanta, Georgia, November 2001. ACM Press.

[10] Zhiyuan Li, Cheng Wang, and Rong Xu. Task allocation for distributed multimedia processing on wirelessly networked handheld devices. In *International Parallel and Distributed Processing Symposium*, Fort Lauderdale, Florida, April 2002. IEEE Computer Society Press.

[11] Mazliza Othman and Stephen Hailes. Power conservation strategy for mobile computers using load sharing. *Mobile Computing and Communications Review*, 2(1):44–50, January 1998.

[12] Friend R and Timothy M. IP compression using LZS. *RFC 2395*, December 1998.

[13] Pereira R. IP payload compression using DEFLATE. *RFC 2394*, December 1998.

[14] Alexey Rudenko, Peter Reiher, Gerald J. Popek, and Geoffrey H. Kuenning. Saving portable computer battery power through remote process execution. *Mobile Computing and Communications Review*, 2(1):19–26, January 1998.