A Light Differential Download Algorithm for Software Defined Radio Devices

Alessandro Brawerman
School of Electrical and Comp. Eng.
Georgia Institute of Technology
Atlanta, Georgia 30318
Email: ale@ece.gatech.edu

Douglas Blough
School of Electrical and Comp. Eng.
Georgia Institute of Technology
Atlanta, GA 30332
Email: doug.blough@ece.gatech.edu

Benny Bing
Georgia Tech Broadband Institute
Georgia Institute of Technology
Atlanta, GA 30318
Email: benny@ece.gatech.edu

***Abstract*— Radio configuration (R-CFG) files for software defined radio (SDR) devices can be downloaded over the air, allowing these devices to support multi-mode functionality using a single transceiver. The drawback of this is that the wireless link is constrained and downloading the entire R-CFG could take some time. In an effort to achieve efficiency, this paper presents a new algorithm for differential download, referred to as light differential download algorithm (LDDA). The LDDA is the first differential download algorithm specifically designed for SDR devices. It presents several new features that make not only R-CFG differential download a possibility, but also allows the update of any software by differential download. Experiments using Java 2 Micro Edition are performed to demonstrate the feasibility and superior performance of the LDDA when compared with other approaches.**

Keywords **- Software defined radio devices, radio configuration, wireless download and differential download for mobile computing.**

I. Introduction

Software defined radio (SDR) [1] allows multiple radio standards to operate on common radio frequency hardware, thereby ensuring compatibility among legacy, current, and evolving wireless systems.

One of the key issues involves dynamic wireless software download [2], which allows an SDR device to reconfigure its hardware parameters by downloading different software code of radio configurations (R-CFGs) over the air. Clearly, this flexibility will be a great advantage to the wireless industry, since with some R-CFGs stored in one SDR device, the customer is able to use the same device in different parts of the world, with different technology modes.

However, SDR devices are usually constrained devices, thus, storing several R-CFGs might not be the best solution. Another solution would be to download a new R-CFG version, to update the current R-CFG or to exchange modes, only when necessary. The drawback is that the wireless link is also constrained and downloading the entire R-CFG could take some time.

To achieve a better solution, the use of differential download is proposed. Since R-CFGs are similar in their basis, i.e., they usually have a similar set of commands, there is no need to download the entire R-CFG when updating to a new version of the same mode or exchanging to a different mode. Downloading the differences between the installed R-CFG and the one to be used is more efficient.

This paper presents a new algorithm for differential download, referred to as light differential download algorithm (LDDA). The LDDA is responsible for calculating a delta-set between an old R-CFG and a new R-CFG. The delta-set is the difference between those R-CFGs. With this application, the SDR device downloads the much smaller delta file instead of the entire R-CFG.

The LDDA is the first differential download algorithm specifically designed for SDR constrained devices. It presents several new features that make it useful for updating R-CFGs within the same mode, exchanging to a different mode, and updating any software by differential download. Some of the novel ideas of the LDDA are: optimization tailored for R-CFG files, integration of delta-set generation and data integrity check, efficient instruction logic, elimination of redundancy on the data file, simpler and smaller delta-sets, and independence of OS platform. Experiments using Java 2 Micro Edition (J2ME) [3] are performed to demonstrate the feasibility and superior performance of the LDDA when compared with other approaches.

II. Related Work

There are different algorithms that claim to use the techniques of differential download, also known as delta compression, to improve the update of a certain file. The most efficient algorithms are discussed in this section.

Rsync [4] takes a different approach to differential download. It allows a client to request changes to a file from the server without requiring the server to maintain any old versions. The server calculates the differences on the fly. This is a drawback, since extra time would be necessary when comparing with the LDDA. Besides, Rsync requires a large number of operations on the client side. Thus, it would present low performance if employed by SDR devices, which are by nature constrained and use a low bandwidth network.

The Xdelta algorithm [5] is based on the idea of block fingerprints introduced by Rsync. It also uses adler32 and MD5 checksums to generate fingerprints, but different from Rsync, it requires that the server has all the existent versions of the requested file. Thus, the differences can be generated

0-7803-8784-8/04/$20.00 © 2004 IEEE

off-line, a priori. An advantage of Xdelta is that it uses a split encoding that separates the sequence of instructions from the data output. The performance of Xdelta is also unsatisfactory for constrained SDR devices, since its logic depends on the use of computer intensive operations executed by a couple of Linux libraries, such as glib and zlib.

III. The LDDA Specification

Consider an old R-CFG currently installed and maintained by the Device Manager (DM) of a certain SDR device. Suppose the old R-CFG is subject to be upgraded to a new R-CFG, which might be or not of the same mode. Instead of downloading the entire new R-CFG, the idea is to download only a difference result set, called a delta-set, which represents the difference between the old R-CFG and the new R-CFG.

The LDDA delta-set comprises two files: delta file and data file. The data file contains the new data that appears on the new R-CFG and is not present on the old R-CFG, i.e., the new data to be inserted. The delta file is a list of commands used to update the SDR device. The commands indicate whether to copy data from the old R-CFG or to copy new data from the data file.

Since the volume of the delta-set is significantly smaller than that of the raw new R-CFG, the advantage is obvious, downloading the delta-set is a much faster and more efficient process than downloading the raw R-CFG. The smaller the delta-set is, the faster to transport it electronically and the less it will cost to update the software.

In order to carry out the update at the SDR device, the device's manufacturer should execute the LDDA to generate the delta-set and make it available for the download. The process involves:

- The LDDA server side, which compares the old R-CFG and the new R-CFG, and generates the delta-set.
- Communication between the manufacturer's server and the SDR-DM.
- The LDDA client side, which incorporates the differences into the old R-CFG, thereby generating the desired new R-CFG.

Figure 1 depicts the whole process. The manufacturer generates the delta-set. The SDR device connects to the manufacturer and downloads the most up-to-date delta-set. The SDR-DM applies the changes into the old R-CFG, generating the new R-CFG.

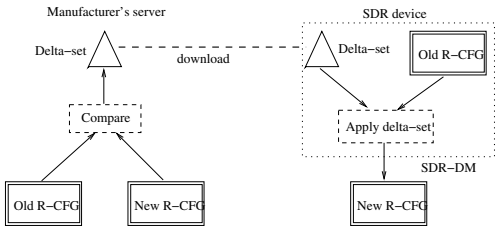

Fig. 1. SDR R-CFG Differential Download.

The LDDA is not responsible for installing the R-CFG on the client side. It assemblies the R-CFG and makes it available to the DM, so that it can be installed later.

A. LDDA's Basic

The nomenclature used to specify the algorithm is as follows:

R-CFG file - structured binary file that is applied to reconfigure the Field-Programmable Gate Array (FPGA) of an SDR device. It is formed by FPGA commands that are able to change radio parameters, power output, frequency range, etc.

Old R-CFG - file that contains the current R-CFG installed in the SDR device. Located on both server and client.

New R-CFG - file to replace the old R-CFG. Together with the old R-CFG, it is used to generate the delta-set on the server side. Located only on the server.

Updated R-CFG - file that is the same as the new R-CFG, but located on the client side. It is generated using the delta-set and the old R-CFG.

Data file - file that contains the new data for insertion. Located on the server and downloaded by the client.

Delta file - file containing a list of instructions to generate the updated R-CFG from the old R-CFG and data file. Located on the server and downloaded by the client. The following instructions can be found within a delta file:

1) copy X-Y: copies a block of commands from block X up to block Y, from the old R-CFG to the updated R-CFG. This instruction is used when there is a command or a block of commands that is the same in the old R-CFG and the new R-CFG.
2) insert X-Y: copies a block of commands from block X up to block Y, from the data file to the updated R-CFG. This data does not exist in the old R-CFG, only in the new R-CFG.

In reality, there are only copy instructions in the LDDA. The insert instruction is a copy instruction that refers to the data file instead of the old R-CFG. This instruction is left as insert to facilitate comprehension.

The LDDA is optimized for SDR R-CFG download, since it treats the R-CFG as a sequence of commands and not just an unstructured binary file. The authors of this paper are not aware of any other delta compression algorithm that interprets R-CFG files. Because of the R-CFG structure, LDDA presents features that other delta compression algorithms do not, e.g. efficient instruction logic and elimination of redundancy in the data file.

Efficient Instruction Logic

Different from previous algorithms, the LDDA presents efficient instruction logic, i.e., it tries to group in a single instruction as many FPGA commands as possible. Suppose that the last instruction in the delta file is: insert 2, which means copy from the data file the second FPGA command. Now, suppose the current instruction informs to copy the third FPGA command from the data file. The LDDA will edit the last instruction on the delta file to contain the third command also. Thus, the final instruction is: insert 2-3.

The efficient instruction logic makes the final delta file smaller and easier to be interpreted by the client and it has shown to improve overall performance.

Eliminating redundancy on the data file

The LDDA is able to eliminate command redundancy in the data file. If an FPGA command to be inserted (new data) occurs more than once in the new R-CFG, it will only appear one time in the data file, thus greatly reducing the overall delta-set size. To correctly assemble the updated R-CFG, the delta file will contain as many references to that command as it occurs in the new R-CFG.

B. Delta-set Creation and Update

The LDDA is divided into 2 parts: creation of the delta-set and update phase.

The delta-set creation occurs only on the server side. The server maintains all existent versions of each R-CFG, from the oldest one up to the most up-to-date. It also maintains the delta-set between each old version and the most up-to-date. Thus, when the client needs a certain delta-set, the server has already generated it and it is ready to be downloaded.

It is important to mention that since the delta-sets are generated a priori, and that the servers are not constrained devices, the time required to generate delta-sets is not a big concern. This is a disadvantage of the LDDA; the delta creation usually takes more time on the LDDA than the Xdelta. However, the LDDA concentrates on creating a clean and easy to interpret delta-set. Therefore, when the client reads the delta file it does not have to do many operations, i.e., the LDDA reduces the workload on the client side.

Together with the delta file, the server also creates a delta header that contains: name and version of the old R-CFG; name and version of the new R-CFG; fingerprint of the new R-CFG.

The name and version of the old R-CFG ensure that the client will use this delta-set for the correct old R-CFG file. The name and version of the new R-CFG generate the correct new R-CFG name on the client, and finally, the fingerprint is used to prevent the installation of an incorrect R-CFG. It is a common practice to calculate the fingerprint after the server has generated the delta-set. However, the LDDA generates the fingerprint concurrently with the delta-set.

The pseudo-code to create the delta-set is presented below:

```
delta_set_creation() {
   open old R-CFG;
   while (!EOF) {
       read X = command or block of commands;
       calculate fingerprint of X;
       input fingerprint and X index in a hash table; }
   close old R-CFG;
   open new R-CFG;
   while (!EOF) {
       read Y = command or block of commands;
       calculate fingerprint of Y;
       accumulate fingerprint to new R-CFG fingerprint;
       if (fingerprint is on hash table)
           efficient_instruction_logic(copy, Y);
       else efficient_instruction_logic(insert, Y);
   close new R-CFG; }
```

A hash table is used to discover whether an FPGA command occurs in both, the old R-CFG and the new R-CFG. Figure 2 depicts the hash table structure. The fingerprint field stores the fingerprint of a command or block. In this example, suppose the fingerprint is a hexadecimal number of 16 bits. The command index field shows the index of that command within the old R-CFG. For instance, the fingerprint for the second command and the forty-eighth command is $ABCD$, which means they are actually the same command.

The LDDA builds a hash table mapping fingerprint values to their offsets for blocks of size s in the source version. Hash collisions are unlike to happen for large blocks, such as blocks of 128 bits. This is the default value in the LDDA. For smaller blocks hash collisions can occur with a small probability, but, more important, the overall performance is decreased when employing small blocks.

Fingerprint	Command Index
ABCD	2 48
1234	190
1AD4	17

Fig. 2. Hash table structure.

Figure 3 shows how to create the delta-set having the old and new R-CFG. The example is in high-level language (the fingerprints in the parenthesis were not actually calculated). From the old R-CFG the hash table is created. Comparing the new R-CFG and the hash table the delta-set is created. Delta file contains the list of commands and data file the new data to be inserted.

That table also illustrates the efficient instruction logic and shows how the elimination of redundancy works. Notice that the command *"X = Y - X"* occurs more than once in the new R-CFG. The elimination of redundancy logic includes that command only once in the data file. The references to the data file are there in the delta file, and when following the instruction list, the updated R-CFG is assembled correctly.

Old R-CFG	New R-CFG
X = 3 (0014) Y = 5 (0023) S = X + Y (0139) Y = Y − 1 (0123) S = X + Y (0139)	X = 3 (0014) Y = 5 (0023) Z = 2 (0012) S = X + Y (0139) X = Y − X (0146) Y = S + Z (0158) X = Y − X (0146)
Hash Table	Fingerprint / Command 0014 / 1 0023 / 2 0139 / 3 / 5 0123 / 4
Data File	Z = 2 X = Y − X Y = S + Z
Delta File	Copy 1-2 Insert 1 Copy 3 Insert 2-3 Insert 2

Fig. 3. Creation of the delta file.

The update phase occurs on the client side and it has to be as simple and as efficient as possible, since the client is a constrained device. In the LDDA after having downloaded the appropriate delta-set, the client only needs to interpret the

delta file and copy blocks from either the old R-CFG or the data file to the updated R-CFG. This technique has shown to improve overall performance since, different from other differential download algorithms, there is only one type of instruction on the delta file.

Figure 4 shows an example of the update phase; it follows the example in Figure 3. From the delta-set plus old R-CFG the updated R-CFG is generated. As it can be seen from the delta file, the updated R-CFG contains blocks 1 and 2 from the old R-CFG, block 1 from the data file, followed by block 3 from the old R-CFG, and finally, blocks 2 and 3 from the data file.

Delta file	Data file	Old R-CFG
Copy 1-2	Z = 2 (1)	X = 3 (1)
Insert 1	X = Y − X (2)	Y = 5 (2)
Copy 3	Y = S + Z (3)	S = X + Y (3)
Insert 2-3		Y = Y − 1 (4)
Insert 2		S = X + Y (5)

Updated R-CFG = New R-CFG
X = 3
Y = 5
Z = 2
S = X + Y
X = Y − X
Y = S + Z
X = Y − X

Fig. 4. The update phase.

The pseudo-code for the update phase is presented bellow:

```
update_phase() {
 open delta file and data file;
 open old R-CFG; //get name & version from the header
 create updated R-CFG; //get name & version from the header
 for each instruction on delta file {
     if (instruction == copy)
        copy blocks from old R-CFG;
    else if (instruction == insert)
        copy blocks from data file;
    accumulate block fingerprint; }
 close data file and old R-CFG;
 compare final fingerprint with the one in the header;
 if (they are the same) completion;
 else error;
 close delta file and updated R-CFG; }
```

C. The LDDA's Adavantages

The advantages of LDDA are listed below:

1) Optimized for R-CFG download: the LDDA correctly interprets the R-CFG, i.e., each block of bytes read is one FPGA command or block of commands. Because of this fact, the LDDA is able to find how many times and where in the flow one command occurs in that specific R-CFG. This improves update time, since the data file is compressed to its minimal.
2) Data integrity check: the LDDA takes into account the fact that MD5 values are being calculated already to compare the new R-CFG and old R-CFG, so instead of calculating the MD5 value for the whole new R-CFG, it accumulates the individual MD5 values of each command. With the LDDA there is no need to analyze the new R-CFG again after the delta creation or the update phase has been completed.
3) Delta instruction list: the LDDA discovers when subsequent FPGA commands are being copied and creates only one instruction for those commands.
4) Delta-set: the LDDA delta-set is divided into 2 files to avoid reading unnecessary data during the update phase, making it easier and faster.
5) Elimination of redundancy on the data file: new data that appears more than once in the new R-CFG, will appear only once in the data file. With this feature the LDDA's delta-set is usually smaller than the ones generated by other algorithms.
6) Update phase: the LDDA updates the R-CFG faster since its delta file is simpler to be interpreted by a constrained device. $Updatephase = downloadtime + timetocreatethenewR - CFG$.
7) J2ME implementation: the LDDA is fully implemented using J2ME, so it is OS platform independent.
8) Updating other softwares: the LDDA can be used to update any kind of software, as long as the server has an old and a new version of the same software. However, the LDDA is optimized to be employed with R-CFGs, thus its performance when updating other softwares is not a worry.

IV. Practical Experiments

In this section the LDDA is compared with the Xdelta. The results obtained with the experiments show that the techniques employed by the LDDA generate better performance, regarding completion time.

For the first 4 experiments a PentiumIV 2.6GHz with 728MB RAM and Linux OS is used as the manufacturer's server and a PentiumIV 2.6GHz with 256MB RAM and Linux OS is used as the client.

Due to the fact that the Xdelta requires compute intensive operations executed by the Linux libraries, glib and zlib, it has not been ported to SDR devices yet. Thus, in the first 4 experiments, no SDR devices have been used. Even though this setup is not an optimized environment for the LDDA, it has already proven to be faster than the Xdelta. The LDDA performance on a constrained client is evaluated in experiment 5 - a complete experiment.

A. Experiment 1

In this experiment, the technique of integrating the delta-set creation and new R-CFG data integrity check is compared against the technique, usually used by other algorithms, of calculating the fingerprint after having created the delta-set.

The graph in Figure 5 shows the comparison between the LDDA integrated scheme against LDDA using a non integrated scheme. For this experiment, a 1MB R-CFG base (old R-CFG) is used and new data is inserted in the new R-CFG. Therefore, fields like 1024 + 128 mean that the old R-CFG has $1024KB = 1MB$ and 128KB more is inserted in the new R-CFG.

As it can be noticed, the LDDA technique of integrating the delta creation and the data integrity check performs better.

This is a small improvement by itself, but it will ultimately improve the overall algorithm's performance.

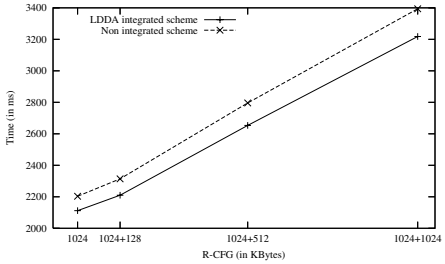

Fig. 5. Integrated scheme X Non integrated scheme. Old R-CFG = 1024KB.

B. Experiment 2

In this experiment, the goal is to show the time it takes for the update phase to complete, i.e., the time to transfer the delta-set is not computed. The graph in Figure 6 shows the time it takes for the client to generate the updated R-CFG when using the LDDA and the Xdelta. It can be seen that in all cases the LDDA presents better performance, about 50% faster to execute the update phase. This is due to the facts that the LDDA is optimized for SDR downloads, so it interprets the R-CFG as a list of FPGA commands; and it builds a simpler delta file, so the client does not need to do many operations to understand how to generate the new R-CFG.

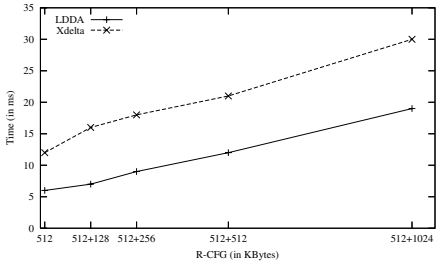

Fig. 6. Comparison of total time to update R-CFG. Old R-CFG = 512Kbytes.

C. Experiment 3

In this experiment, the LDDA is again compared with the Xdelta, but the goal here is to show the benefits of the elimination of redundancy logic, and the total time to generate the updated R-CFG on the SDR device (time to transfer delta-set over the wireless connection + update phase time).

Six different tests are executed in this experiment. First, the time to transfer the entire new R-CFG is computed. Second, the time to execute the LDDA with no code redundancy, i.e., there is not a single FPGA command repeated within the new data, is computed. In the third, 10% of the new data is code redundancy. For the fourth, 20% of the new data is code redundancy. In the fifth, 30% of the new data is code redundancy. Notice that, the older the current R-CFG version on the SDR device is, the greater the percentage of code redundancy. Finally, the time to execute the Xdelta is computed.

The graph in Figure 7 shows the results when transferring the entire new R-CFG and when executing LDDA (with no code redundancy). Notice that, the more up-to-date the current R-CFG version on the SDR is, the better the LDDA performs. For instance, in the case that 128KB are included in the new R-CFG (1024+128 case) the LDDA performs about 9 times faster, while in the case that 1MB is included (1024 + 1024 case) the LDDA performs about 2 times faster. This is due to the fact that the older the R-CFG on the SDR device, the more new data has to be transmitted over the wireless connection. The bottleneck in this case is the network, not the SDR device.

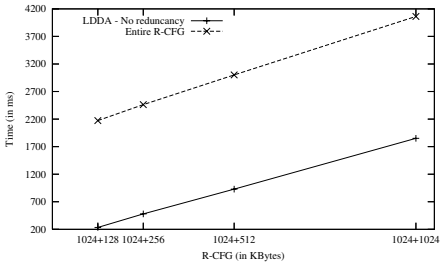

Fig. 7. Entire R-CFG against using LDDA.

The graph in Figure 8 shows the results when executing Xdelta and LDDA with the code redundancy feature. The case of 1024 + 128 is not shown on this graph for readability purposes. It can be seen, as expected, that for all cases LDDA with 30% code redundancy presents better performance. With the elimination of redundancy logic, the LDDA is able to create smaller delta-sets, therefore the time to transfer and complete the whole update process is even faster than when using Xdelta. A 10% to 25% improvement is achieved, by LDDA, when completing the whole process with no code redundancy, and 30% improvement is achieved if 30% of the new data is redundant.

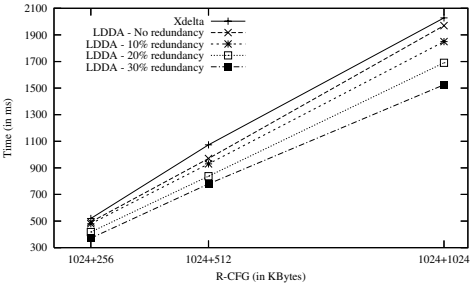

Fig. 8. Using the elimination of redundancy logic.

D. Experiment 4

The goal in this experiment is to show that the delta-set can be further compressed using binary compression algorithms; however, with the available algorithms it is more costly to compress the delta-set, transfer the compressed file over the

connection, decompress the delta-set and finally update the R-CFG on the SDR device than simply execute LDDA with no further compression.

The following common algorithms/models are employed to further compress R-CFGs delta-sets: Gzip [6], LZ [7], arithmetic coding based on the adaptive unigram (AU) model [8], and based on the prediction by partial matching (PPM) Model [9].

Gzip and LZ are not efficient for R-CFG delta-sets. They do not compress R-CFG delta-sets very well, and in fact, Gzip generates a larger file due to information that is included in its header. On the other hand, the AU and the PPM model do compress R-CFGs delta-sets. However, due to the intensive computational operations when executing those models, there is no real gain on performance.

Figure 9 shows the difference in performance when comparing the LDDA without code redundancy, the AU model, and the PPM model. For the LDDA, the time to transfer the delta-set and update the R-CFG on the device is computed, and for the AU and the PPM only the time to decompress the delta-set is computed. Notice that the time to decompress the delta-set is much higher than executing the LDDA. Therefore, those compression algorithms are not suitable for SDR constrained devices. The development of a light binary compression algorithm, capable of decompressing the delta-set in a constrained device without adding further delays, is a topic of future work.

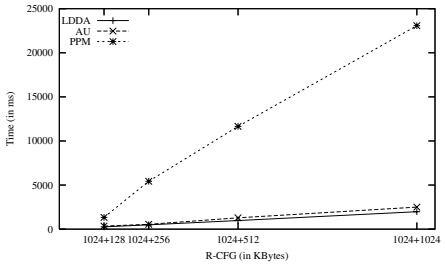

Fig. 9. Decompressing the delta-set.

E. A Complete Experiment in a Constrained Environment

The goal in this experiment is to evaluate performance of the LDDA in a constrained environment. The experiment setup comprises of: an SDR constrained device, in this case a Sharp Zaurus PDA SL-5600 with CPU speed of 400 MHz, 32MB SDRAM, Linux OS and J2ME support, connected through an 11Mbps wireless link, to a Pentium 4 2.6GHz server with 256MB RAM providing HTTP services.

The results in Figure 10 compare the LDDA against the method of transferring the entire R-CFG, both in a constrained and in a non-constrained environment. Note that the gain with the LDDA in a non-constrained environment falls from a range of 90% to 50% to a range of 50% to 25% in a constrained environment.

V. Conclusion

A new method for differential download, called light differential download algorithm (LDDA) is proposed. The new

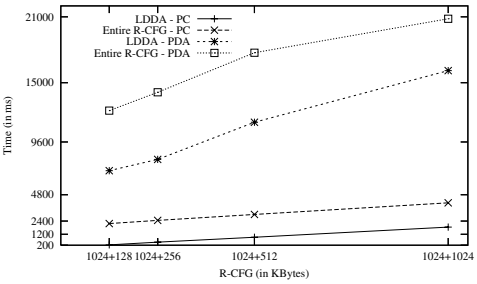

Fig. 10. Constrained environment.

algorithm is responsible for calculating a delta file provided an old R-CFG and a new R-CFG. With this application, an SDR device downloads the much smaller delta file instead of the entire R-CFG. The LDDA, which is the first differential download algorithm specifically designed for SDR download, presents several new features, such as: optimization tailored for R-CFG downloads, efficient instruction logic, elimination of redundancy logic, simpler and smaller delta-sets, and independence of OS platform.

Experiments comparing the LDDA and the Xdelta, another differential download algorithm, were presented. The results showed that the LDDA performs better than the Xdelta, even in an unconstrained environment. A 50% improvement is achieved by the LDDA when assembling the instructions to generate the new R-CFG. A 10% to 25% improvement is achieved by the LDDA when completing the whole process with no code redundancy, and 30% improvement is achieved with 30% code redundancy. Finally, the LDDA is evaluated in a constrained environment. The results show that the gain with the LDDA in a non-constrained environment falls from a range of 90% to 50% to a range of 50% to 25% in a constrained environment when comparing with a method of transferring the entire R-CFG. Future work includes the possibility to compress even further the delta-set by creating a new binary compression method specifically tailored to R-CFG files.

References

[1] Bing B. and Jayant N., "A cellphone for all standards," IEEE Spectrum, May, pp. 34-39, 2002.

[2] Cummings M. and Heath S., "Mode switching and software download for software defined radio: The SDR Forum approach," IEEE Communications Magazine, Aug., pp. 104-06, 1999.

[3] Java 2 Micro Edition Technology website. Available HTTP: http://www.wireless.java.sun.com/j2me

[4] Tridgell A. and Barker P., "The Rsync algorithm," Tech. Rep. TR-CS-96-05, The Australian National University, June 1996.

[5] MacDonald J., "File system support for delta compression," M.S. Thesis, UC Berkeley, May 2000.

[6] The Gzip website. Available HTTP: http://www.gzip.org

[7] Ziv J. and Lempel A. "A universal algorithm for sequential data compression," IEEE Transactions on Information Theory, IT-23, pp 337-343, 1977.

[8] Fenwick, P.M., "A new data structure for cumulative frequency tables," Software Practice and Experience 24(3), pp. 327-336, 1994.

[9] Cleary, J.G. and Witten I.H., "Data compression using adaptive coding and partial string matching," IEEE Transactions on Communications. 32(4), pp. 396-402, 1984.