A REPORT

ON

INCREMENTAL UPDATE IN NAVIGATION DATA STANDARDS

By

Abhinav Kumar	2011C6PS819P	Information Systems

At

Nokia - Location & Commerce, Mumbai

A Practice School-II Station Of

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE,

PILANI

(July-September 2014)

A REPORT

ON

INCREMENTAL UPDATE IN NAVIGATION DATA STANDARDS

By

Abhinav Kumar 2011C6PS819P Information Systems

Prepared in complete fulfillment of the

Practice School – II

Course No.: BITS C412/BITS C413/BITS G639

At

Nokia - Location & Commerce, Mumbai

A Practice School-II Station Of

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE,

PILANI

(July-September 2014)

Acknowledgments

First and foremost, I would like to extend my heartfelt gratitude to my mentor, **Mr. Shripad S. Kulkarni**, without his support and guidance; these projects would not have been possible. I would like to thank him for helping me whenever I was in doubt, for encouraging me to come up with new solutions to the problems and giving me ample time to learn and implement. I have really learnt a lot under his mentorship.

I would also like to thank my former manager, **Mr. Prateek Shrivastava** and my present manager **Mr. Pfeifle Martin** for all their support.

I would also like to thank the HR representative **Ms. Diksha Dalvi** for always being there to clear any of our inhibitions or doubts.

Besides, I would like to thank the management of Nokia Location & Commerce for providing me with an environment conducive for learning & producing the best results. Also, I would like to take this opportunity to thank Birla Institute of Technology & Science for offering this program. It gave me the opportunity to enhance not only my theoretical but also my communication skills.

Last but not the least; I would also like to thank my Practice School instructors **Mr. Vijay Reddy** and **Ms. Anand Vijayalakshmi** for their continuous guidance throughout the program.

Birla Institute of Technology & Science
Pilani (Rajasthan)
Practice School Division

Station: Nokia Location & Commerce **Centre**: Mumbai

Duration: From: July 2014 **To:** December 2014

Date of Submission: 24th September2013

Title of Project: INCREMENTAL UPDATE IN NAVIGATION DATA STANDARDS

ID: 2011C6PS819P **Name**: Abhinav Kumar **Discipline**: Information Systems

Name of Mentor: Mr. Shripad S. Kulkarni **Designation:** Senior Engineer

Name of PS Faculty: Mr. Vijay Reddy / Ms. Anand Vijayalakshmi

Project Area(s): Data Compression

Abstract:

This project aims at developing a variant of the standard open source algorithm called BSDiff which takes out the binary differences between two versions of a file to create a smaller delta file. The aim of the project is to construct this delta file so small that it can perform operations faster on user's hardware (car dashboard). Optimization of the algorithm is required to enhance the performance of the algorithm from 50 hours to 10 hours of code execution on the user's machine.

Abhinav Kumar Vijay Reddy

22nd September 2014 22nd September 2014

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI (RAJASTHAN)

Practice School Division

Response Option Sheet

Station: Nokia Location & Commerce **Centre:** Mumbai

Students: Abhinav Kumar **ID:** 2011C6PS819P

Title of the Project: Incremental Update in Navigation Data Standards

S.No.	Response Options	Course No.(s) and Name
1	A new course can be designed out of this project.	NO
2	The project can help modification of the course content of some of the existing courses.	NO
3	The project can be used directly in some of the existing Compulsory Discipline Courses (CDC)/ Discipline courses other than Compulsory (DCOC)/ Emerging Area (EA) etc. courses.	NO
4	The project can be used in preparatory courses like Analysis and Application Oriented Courses (AAOC)/ Engineering Sciences (ES)/ Technical Art (TA) and core courses.	NO
5	This project cannot come under any of the above mentioned options as it relates to the professional work of the host organization.	YES

Signature of the Student Signature of Faculty

Table of Contents

PROJECT

INCREMENTAL UPDATE IN NAVIGATION DATA STANDARD

I. LIST OF ABBREVIATIONS AND SYMBOLS

Abbreviations

Abbrev.	Stands for	Description
IHU	Infotainment Head Unit	A hardware unit which will include navigation in the SPA project.
HDD	High Density Disc	Storage media inside the IHU.
NDS	Navigation Data Standard	A standard to organize map data. The standard is maintained by the *NDS Association*.
OTA	Over The Air	Data transfer of files from VCC cloud to the IHU.
UR	Update Region	A particular region to be updated

Definitions

Basemap A Basemap is a complete set of Update Regions (UR) which creates a complete map like Europe or North America.

Update Region An Update Region is a part of a complete Basemap, which can be updated separately, like Scandinavia and Benelux within the Europe map.

Map increment Map increments are used to update a single UR. The map increment is a file with a binary difference between data for an old UR and a new UR.

II. LIST OF ILLUSTRATIONS

1. Introduction

1.1 Nokia Corporation

NOKIA

- Today principal products are mobile telephones and portable IT devices. It also offers Internet services including applications, games, music, media and messaging, and free-of-charge digital map information and navigation services through its wholly owned subsidiary **HERE (formely NAVTEQ)**. Nokia also owns a company named **Nokia Solutions and Networks**, which provides telecommunication network equipment and services. As of 2012, Nokia employs 101,982 people across 120 countries, conducts sales in more than 150 countries, and reports annual revenues of around €30 billion.

- By 2012, it was the world's second-largest mobile phone maker in terms of unit sales, with a global market share of 18.0% in the fourth quarter of that year. Nokia is a public limited-liability company listed on the Helsinki Stock Exchange and New York Stock Exchange. It is the world's 274th-largest company measured by 2013 revenues according to the *Fortune Global 500*. Nokia was the world's largest vendor of mobile phones from 1998 to 2012. However, over the past five years its market share declined as a result of the growing use of touchscreen smartphones from other vendors—principally the iPhone, by Apple, and devices running on Android, an operating system created by Google — which Nokia chose not to adopt and compete with it instead. As a result, the corporation's share price fell from a high of US$40 in late 2007 to under US$2 in mid-2012. In a bid to recover, Nokia announced a strategic partnership with Microsoft in February 2011, leading to the replacement of Symbian with Microsoft's Windows Phone operating system in all Nokia smartphones.

- **Nokia Corporation** is a Finnish multinational communications and information technology corporation that is headquartered in Espoo, Finland. Over the past 150 years, Nokia has evolved from a riverside paper mill in south-western Finland to a global telecommunications leader connecting over 1.3 billion people. During that time, it has done everything from making rubber boots and car tyres to generating electricity to manufacturing TVs.

1.2 NAVTEQ

- NAVTEQ's underlying map database is based on first-hand observation of geographic features rather than relying on official government maps. It provides data used in a wide range of applications, including automotive navigation systems for many car makers, accounting for around 85% of market share. Most clients use Navteq to provide traffic reports in major metropolitan areas throughout North America.

- Nokia has numerous subsidiaries. The largest in terms of revenues is Navteq, a Chicago, Illinois-based provider of digital map data and location-based content and services for automotive navigation systems, mobile navigation devices, Internet-based mapping applications, and government and business solutions. Navteq was acquired by Nokia on 1 October 2007. Navteq's map data is part of the Nokia Maps online service where users can download maps, use voice- guided navigation and other context-aware web services. The company is a wholly owned subsidiary of Nokia but operates independently.

- NAVTEQ partners with third-party agencies and companies to provide its services for portable GPS devices made by Garmin, Lowrance, NDrive and web-based applications such as Yahoo! Maps, Bing Maps, Nokia Maps, and MapQuest. Microsoft's aviation game Flight Simulator X uses NAVTEQ data to achieve a high level of visual realism for automatic terrain generation. XM Satellite Radio and Sirius Satellite Radio use NAVTEQ data to show traffic information on navigation systems. NAVTEQ data has also been used for GPS- and GSM-based sex offender tracking systems in North Carolina and Georgia. NAVTEQ also provides graphics systems, information services, and personnel for TV and radio broadcasting via NAVTEQ Media Services.

- Its main competitors are Google and the Dutch company Tele Atlas.

1.3 Nokia Location & Commerce (HERE)

- Here captures location content such as road networks, buildings, parks and traffic patterns. It then sells or licenses that mapping content, along with navigation services and location solutions to other businesses such as Garmin, BMW, Oracle and Amazon.com.

- **Here**, formerly **Ovi Maps** (2007–2011) and **Nokia Maps** (2011–2012), is a Nokia business unit that brings together Nokia's mapping and location assets under one brand. The technology of Here is based on a cloud-computing model, in which location data and services are stored on remote servers so that users have access to it regardless of which device they use.

- In May 2011, **Ovi Maps** was renamed to **Nokia Maps** when Nokia streamlined its services offering under the head brand. It also merged NAVTEQ with itself to form Nokia –Location & Commerce.
- On 13 November 2012, Nokia announced that it would rebrand its location offering as **Here** to highlight its vision for the future of location based services and its belief in the importance of mapping.

- For more than a decade, Nokia has built its mapping and location business by acquiring location technology and know-how. It all began in 2001 as Smart2Go, a generic 3D-map interface for access to tourist information on mobile terminals. It was developed by an EU consortium named TellMaris. Nokia gained the rights to the software when it acquired Berlin-based route planning software company gate 5 in August 2006, which has become the cornerstone for the company's mapping business. It then made the Smart2Go application free to download

2. What is the purpose of this document?

The purpose of this document is to describe

1. Introduction to **Bsdiff**-Comparing by block approach
2. **Advantages** of block based approach over Standard Bsdiff approach.
3. Statistics and **graphs** showing advantages of the block based approach

3. BSDiff Standard approach

3.1 Standard Bsdiff – input and output

Bsdiff tool accepts any two versions of any file and creates the patch as shown in below diagram.

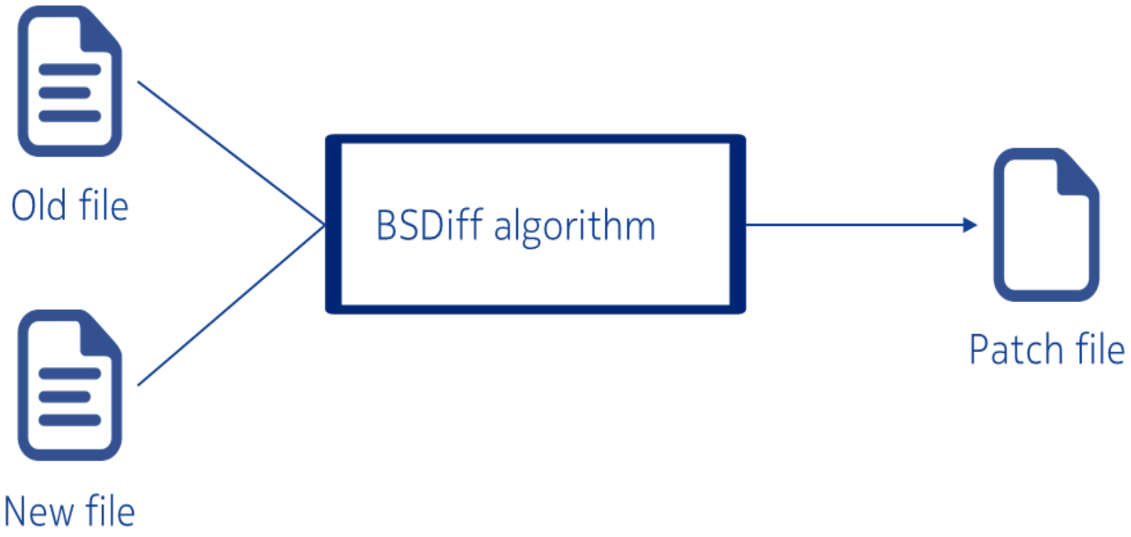


Figure 1 – BSDiff flowchart

3.2 Standard BSDiff algorithm overview

1. Takes old file, new file and patch file location as input.
2. Read whole old file in buffer.
3. Creates suffix tree for whole old file.
4. Reads new file byte by byte and tries to find best matching in old file. This will take help of suffix tree created in three and old file contents in buffer.
5. Finds approximately matching byte's differences and stores them in diff block. Newly added bytes in new file will be appended in extra block. Control block row (ctrl0, ctrl1, ctrl2) will be appended in such a way that Ctrl0 will be the length of uncompressed diff block (approximately matching bytes) Ctrl1 will be length of uncompressed extra block. (Newly added bytes in new file)
6. Control block, diff block and extra block will be compressed by using bzip2.
7. Repeat from step four to six till we process all new file.
8. Update header in patch file with correct values of control block size and diff size.
9. Close all files and exit

3.3 Patch format

Title	Length in bytes	Comment
Header	32	First 8 bytes have “BSDIFF40”, Next 8 bytes says how much is control block length, third 8 Bytes says diff block length, final 8 bytes will say what is new file size in bytes.
Control block	Length mentioned in header.	Number of 24 byte rows, each row will have ctrl0, ctrl1 and ctr2 each of 8 bytes. This block is compressed by using bzip2. **Ctrl0** represents length of diff block. These many bytes will be read from Diff block. **Ctrl1** represents length of extra block. These many bytes will be read from Extra block **Ctrl2** represents offset seek in old file for next row
Diff Block	Length mentioned in header.	Approximately matching bytes from old and new are subtracted and concatenated in this block. This block is also compressed by bzip2.
Extra Block	Length is not mentioned anywhere. So from end of diff block till end of file represents this block	Non-matching contents from new file will be appended in this block. Bzip2 is also used to compress this block though it will not succeed to reduce overall size as it succeeds for control block and diff block.

Table 1 – Patch File construction

3.4 Need to update Standard Bsdiff approach

Standard Bsdiff tries to find best matching of new file contents in whole old file. As a result the seek time (ctrl2) for each control block row can have very large values resulting in more seek in bspatch. This will not affect if we have whole old file in buffer. But unfortunately usually targets on which we execute bspatch have very limited memory available. As a result they try to read data chunk by chunk in buffer as and when required. So if we have more seek for few resulting new bytes it become time consuming to create New file by using bspatch and hence its required to update the standard Bsdiff. Below Diagram shows issue in standard Bsdiff in detail.

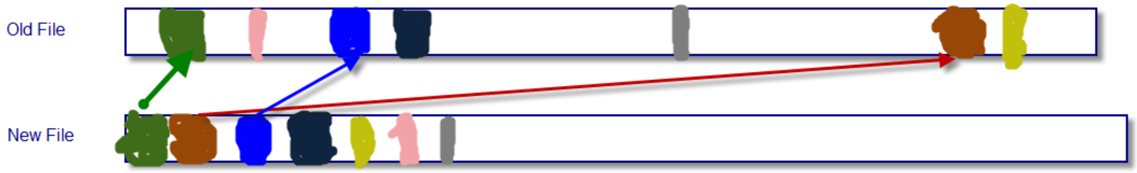


Figure 2 – Standard approach matching

Bspatch is keeping oldPos to point to old file location from where we need to read data from old file. And in each control block row Ctrl2 represents amount of seek it has to do to process next block.

So after reading green block, oldPos has to reach brown box. And after brown box it has toreach blue box. So ctrl2 will hold the offset difference in between these boxes as required. Finding so much scattered patterns for resulted small new block makes bspatch very slow if we are not holding whole old file in buffer. As discussed earlier it become difficult to avail more memory on targets where bspatch runs, so at the end bspatch will require much more time from expected. Hence to overcome this we come up with block based comparing approach.

4. BSDiff – Improved approach

4.1 Comparing Old and new files block by block

In this approach new file contents are only compared with its related locations in old file. By this way we come up with less seek time. It might be that we may result in little .

Let's understand this approach by example. Suppose Old file size is 19 MB and the new file size is 17 MB. We divide new file in 1MB blocks so as shown below we will have N1 to N17 new blocks. The new blocks are represented from (N1, N2…N16, N17) size of each block being 1 MB.

Now, we divide old file in two types of old blocks –

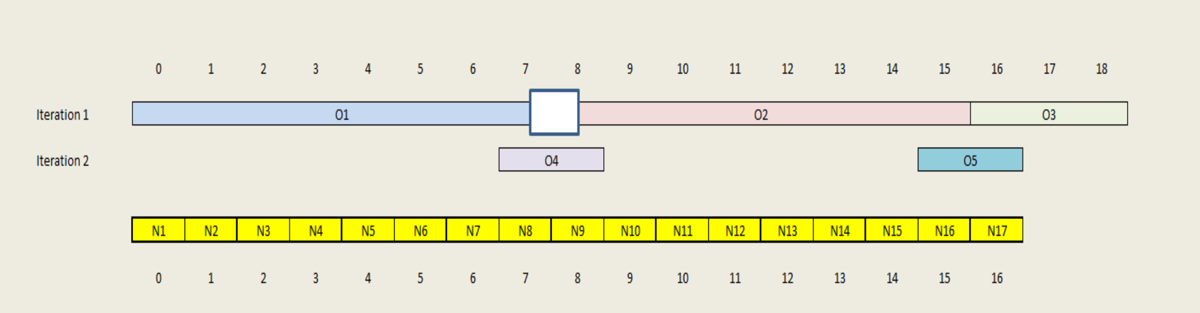


Figure 3 – Improved approach matching

- **Normal old blocks** – We divide the old file in few MB blocks usually 8 MB Blocks. So for given 19 MB old file, we will have O1, O2 and O3 each of 8, 8 and 3 MB respectively. Last block O3 is of 3 MB as we have only 3 MB remaining in Old file.   We can pass the normal old block size to Bsdiff; its value should be greater than 2 MB. For best results we use 8 MB as normal old block size.

- **Overlapping old blocks** – On borders of Normal block, we have these 2 MB blocks – size is exactly double of one new block size. So as shown above we will have two overlapping blocks for 8MB Normal old block size and 19MB old file size. If we have 'N' normal old blocks then we will have 'N-1' Overlapping old blocks. Last overlapping block might result in less than 2 MB if (old file size % 8 MB < 1 MB).

4.2 First case scenario for improved approach

Old blocks	size	Start Index
O1	8 MB	0
O2	8 MB	8
O3	3 MB(because of less data)	16
O4	2 MB	7
O5	2 MB	15

Table 2 – Improved case example

4.3 Increased possible matches in new files

So each old block will be compared with its related new blocks. That is each Normal block will be compared with 8 new blocks and overlapping block will be compared with 2 new blocks. And hence we asked each BB team to match respective locations in old and new file.
In interest of finding more matches we added LeftFocus and RightFocus. This will compare more new blocks with each Old block in addition to its related blocks.

- **LeftFocus**- this is value representing for each old block how many left New blocks we should compare with each old block. Usually we use value 8.
- **RightFocus**- this is value representing for each old block how many right New blocks we should compare with each old block. Usually we use value 8.

Given NormaOldBlockSize=8MB, LeftFocus=8MB and RightFocus=8MB; Each normal old block will be compared with 24 MB of new file and each overlapping old block will be compared with 18 MB of new MB; as shown for Normal old block in below diagram.

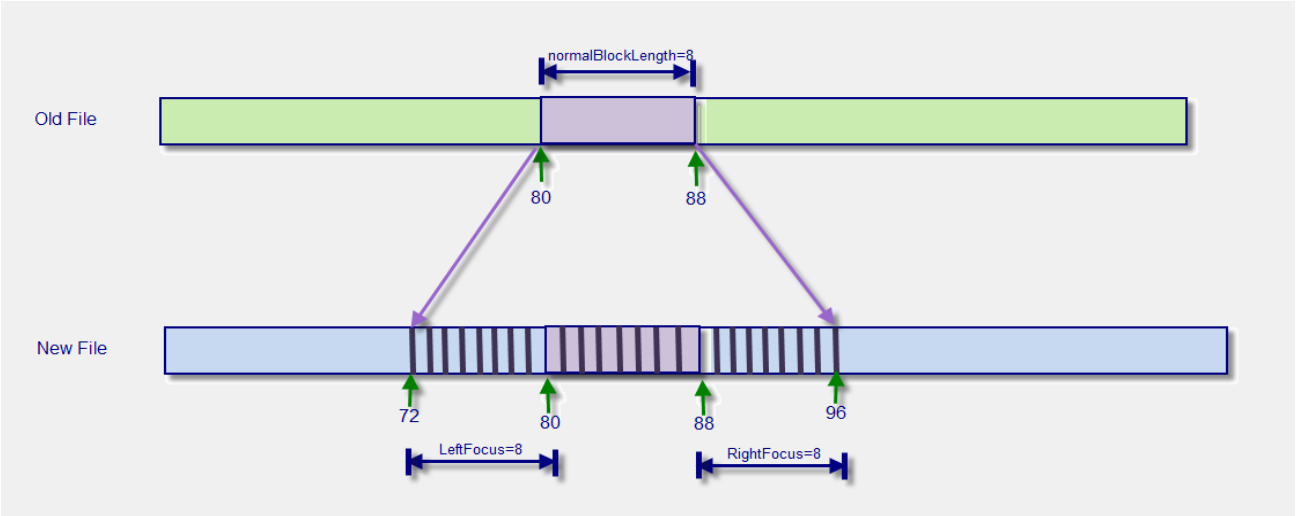


Figure 4 – Improved case extended version

4.4 Inference of the block based approach

- In simple words, one normal old block is compared with 24 (8(left) + 8(exactly related) + 8(right)) new blocks separately, resulting in 24 small patch files. This process will be continued from start to end in old file, for normal as well for overlapping blocks. Each overlapping block is of size 2MB and hence with 8 as left and right focus each overlapping block will be compared with 18 New blocks(8+2+8). Last old blocks could be compared with more or less new blocks as per old and new file sizes.

- For last Normal old block and for last overlapping block, we ensured that we cover up whole new file. For example if Old file size is 12 and new file size is 24; we have to ensure that last normal & overlapping block not only compares till 20 but till 24. Mostly this is rare case, and sometimes if this happened for most of files and patch sizes are not considerable then will suggest downloading whole next release and not using incremental update.

- So when we are covering all old blocks; it might happen that one new block gets compared with more than one old block. In this case we have to compare their respective patch sizes and chose the one with smaller patch size. In this exercise we have to maintain below three arrays. As we have 17 new blocks; size of these arrays will be 17.

5. Proposed algorithm

The algorithm proceeds by following the given in this specific order –

5.1 Iteration 1: Finding patterns in normal old blocks (old block size = 8 MB),

1. All entries in figure 2 will be initialized to -1. OldBlockIndex = 1. The program starts with iteration 1

2. The suffix tree for O[OldBlockIndex] is generated.

3. NewBlockIndex = 1.

4. By using suffix tree of O[OldBlockIndex], we create patch file (temp.patch) for N[NewBlockIndex].

5. If(Best matching patch size of [NewBlockIndex-1]== -1 OR Best matching patch size of [NewBlockIndex-1] > temp.patch size created in 3)
   then do 6
   else goto 7

6. The patch size is stored in Figure 2 in the cell marked by Best matching patch size of [NewBlockIndex-1].
   OldBlockIndex is noted in the cell marked by "Best matching old block number[NewBlockIndex-1]"
   The size of the old block **(8)** is noted in the cell below it.
   Rename temp.patch file as **temp_ NewBlockIndex.patch**

7. NewBlockIndex ++
   if NewBlockIndex is valid (value between 1 to 17) for given new file
   goto 4

8. OldBlockIndex ++;

9. If OldBlockIndex is valid (i.e value between 1 to 3) then goto 2.

5.2 Iteration 2: Finding patterns in overlapping old blocks (old block size = 2 MB),

Now we come to iteration 2

10. The suffix tree for O[OldBlockIndex] is generated.

11. NewBlockIndex = 1.

12. By using suffix tree of O[OldBlockIndex], we create patch file (temp.patch) for N[NewBlockIndex].

13. If(Best matching patch size of [NewBlockIndex-1]== -1 OR Best matching patch size of [NewBlockIndex-1] > temp.patch size created in 3)
    then do 14
    else goto 15

14. The patch size is stored in Figure 2 in the cell marked by Best matching patch size of [NewBlockIndex-1].
    OldBlockIndex is noted in the cell marked by "Best matching old block number[NewBlockIndex-1]"
    The size of the old block **(2)** is noted in the cell below it.
    Rename temp.patch file as **temp_ NewBlockIndex.patch**

15. NewBlockIndex ++
    if NewBlockIndex is valid (value between 1 to 17) for given new file
    goto 12

16. OldBlockIndex ++;

17. If OldBlockIndex is valid (i.e value between 4 and 5) then goto 10.

The output of iteration 1 and 2 will be in the contents of the table below and temp_XY.patch files where XY will have values from 1 to 17.

5.3 Final patch creation

By using table 2 data now we have to create final patch file

18. NewBlockIndex = 1.

19. Initilize final patch file *FP, temp diff file *DF and temp extra file *EF with normal file write.

20. Write 32 bit standard header in final patch file. 8 Bit “BSDIFF80” , next 8 bits control block size(now 0), next 8 bits diff block size(now 0) and final 8 bits with new file size.

21. **Add dummy control block entry with diff size = 0, extra size = 0 and properly adjusted offset by using table 2.**

22. Read control block from temp_NewBlockIndex.patch and append to *FP**. Last control block offset needs to be adjusted correctly by using table 2 above.**

23. Read diff block from temp_NewBlockIndex.patch and append to *DF

24. Read extra block from temp_NewBlockIndex.patch and append to *EF

25. NewBlockIndex++

26. If NewBlockIndex is valid(1 to 17) go to 21

27. Update header with final control block length

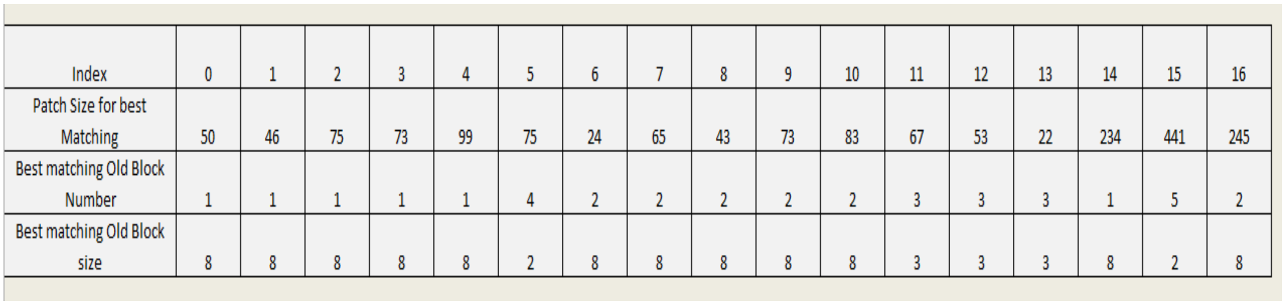
28. Append *DF data to *FP, and update final diff block data header.

29. Append *EF data to *FP

30. End

6. Final result obtained

6.1 Example dataset



Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Patch Size for best Matching	50	46	75	73	99	75	24	65	43	73	83	67	53	22	234	441	245
Best matching Old Block Number	1	1	1	1	1	4	2	2	2	2	2	3	3	3	1	5	2
Best matching Old Block size	8	8	8	8	8	2	8	8	8	8	8	3	3	3	8	2	8

Table 3 – Proposed Data Structure

1. Array holding best patch sizes for each new block,
2. Array holding best matching old block number,
3. Best matching old block size. This is not used for further processing but just kept for future use.

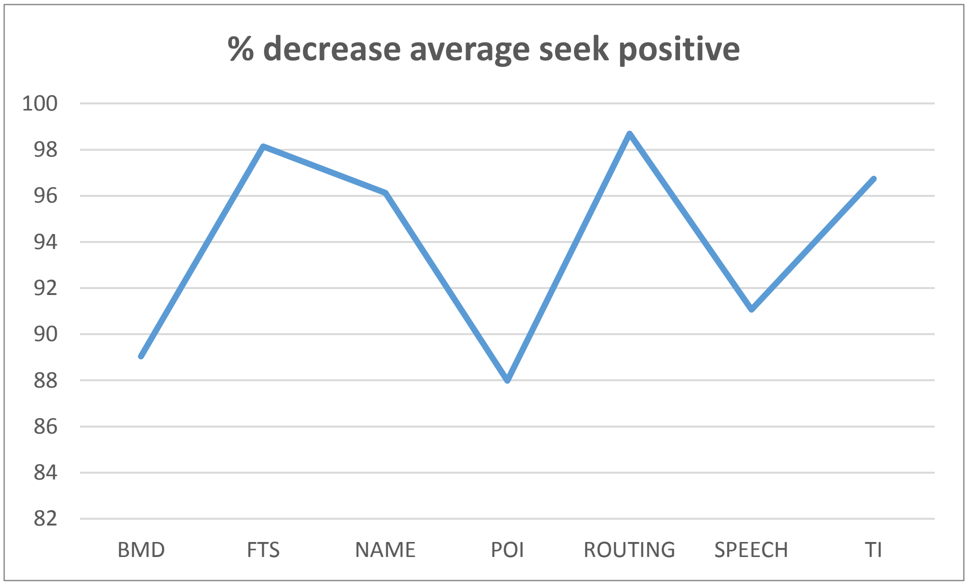
Also we keep best matching patch file for each new block with name temp_XXX.patch; where XXX represents new block number.

Once all data is processed, we will have above three arrays and patches for each new blocked. By using this we merge all these patch files to one final one by merging their respective control blocks, diff blocks and extra blocks. Important thing to do in merging is to manage offset value for last control block of each patch file; so that for each new block oldPos in bspatch will point to its best matching old block. Also only before first patch we add one additional control block row (ctrl0=0; ctrl1=0; ctrl2=offset); where offset will get value so that OldPos will point to that old block which is best matching with first new block. Finally header information is updated in final patch and here we complete the Bsdiff patch creation by comparing block approach.

The resultant patch with comparing block approach will be in same format as we had with standard Bsdiff; this will make sure that we are not required to make any changes in the bspatch algorithm to take advantage of comparing block approach.

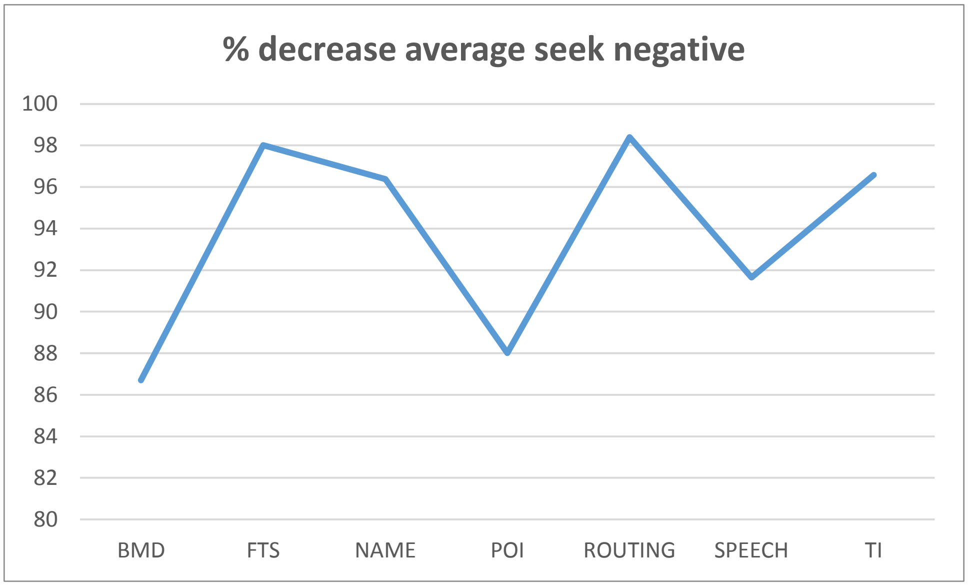
6.2 Advantages of comparing block approach over standard approach

- As per statistics; the seek value gets reduced drastically; which will ensure faster new file creation by using bspatch. From below statistics it shows that the seek value gets reduced by **96.50 %** for both positive and negative.



BB Name	Average seek (Positive)		% decrease
	Standard BSDIFF	**Comparing Blocks BSDIFF**	
BMD	7756615.81	849949.4	89.04226507
FTS	97943940.81	1818462.404	98.14336406
NAME	60300138.21	2333509.15	96.13017612
POI	8856924.932	1064551.897	87.98056995
ROUTING	99489506.56	1302219.747	98.6910984
SPEECH	28148511.43	2515141.42	91.06474448
TI	302495637.7	9883834.018	96.73256973

Table 4 – Average positive seek



BB Name	Average seek (Negative)		% decrease
	Standard BSDIFF	**Comparing Blocks BSDIFF**	
BMD	-13212756.56	-1757329.644	86.69975008
FTS	-100078470.3	-1992334.855	98.00922731
NAME	-62543518.14	-2265043.066	96.37845274
POI	-11323230.35	-1358321.365	88.00411788
ROUTING	-132897569.8	-2125998.033	98.40027321
SPEECH	-29020701.3	-2424415.365	91.64591048
TI	-349076246.5	-11923442.33	96.58428712

Table 5 – Average negative seek

- Patches generated by comparing block approach seems promising that it will help to create new files faster by using bspatch.

- This ensures that new file is created 1MB by 1MB so bspatch can use 1MB buffer size for holding new file contents.

- All data to create any 1 MB new file will totally reside in one old block, so after creating each 1MB bspatch should read next old block again to ensure that it will have all required data from old file to process next 1MB.

6.3 Disadvantages of comparing block approach over standard approach

- The patch size can be increased by using comparing block approach as compared with standard Bsdiff approach. From statistics we have (NDS IP 21 to 23 data) in worst case we had around 14.5 % patch increase and in best case around 2.5 % increase in size.

- This is because we are not finding patterns in whole file and only compare related file locations. That's why we might compromise best matching patterns scattered in whole file with matching patterns in related memory locations.

- This is the reason to use this block comparing Bsdiff approach; we recommend that the new and old files related memory locations should be best matching.

6.4 Planned Improvements

- Same Bsdiff exe cannot be used in parallel to create patch files. This is because of names if temp files and there location. They need to be placed in some folder whose name will be given by using timestamp and username and/or process name.

- These temp files/folder needs to be removed once final patch is created.

7. Conclusion

- Standard BSDiff patches work best if the whole file is in the main memory while applying bspatch.

- BSDiff comparing related block approach helps to **reduce seek value by 96.5 %.**

- This **solution has been accepted** and last three update delivery has been made with the proposed approach.

- BSDiff comparing related block approach will help bspatch to create new files much **faster as compared to standard BSDiff** approach.

- **Client results: This approach takes about 10 hours for same set of files where the standard approach takes about 50 hours.**

8. References

Ref. No	Title
1	***Incremental Update SPA Project_IP21_1.docx**, Full documentation*
2	***http://www.daemonology.net/bsdiff/,** Open source BSDiff algorithm*
3	***Oracle® Database Concepts**, 10g Release 2 (10.2) ,B14220-02,October 2005*
4	***Complete Reference Java**, Herbert Schildt, ISBN-10: 0070435928, June 2011*
5	***Linux Fundamentals**, Paul Cobbaut, June 2013*
6	***Pl/SQL Concepts**: http://plsql-tutorial.com/*

Table 6 - References

9. Appendix

9.1 Improved BSDiff code

```
/*$T indentinput.cpp GC 1.140 02/18/14 10:07:21 */

/*
 * Copyright 2003-2005 Colin Percival All rights reserved Redistribution and use
 * in source and binary forms, with or without modification, are permitted
 * providing that the following conditions are met: 1. Redistributions of source
 * code must retain the above copyright notice, this list of conditions and the
 * following disclaimer. 2. Redistributions in binary form must reproduce the
 * above copyright notice, this list of conditions and the following disclaimer in
 * the documentation and/or other materials provided with the distribution. THIS
 * SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
 * EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
 * OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 */
#include <stdlib.h>
#include "bzlib.h"
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <time.h>
/*
* include <err.h> ;
* #include <unistd.h>
*/
#include <io.h>
#include <fcntl.h>

/* include <sys/wait.h> */
#include <windows.h>
#include <process.h>
#include <sys/types.h>
typedef unsigned char       u_char;
typedef long                pid_t;

/*
===========================================================================================
===============================
===========================================================================================
===============================
*/
```

```
template<class T>
void err(int i, const char *str, T arg)
{
       /*~~~~~~~~~~~~~~~~~~~~~~~*/
       char   lastErrorTxt[1024];
       /*~~~~~~~~~~~~~~~~~~~~~~~*/

       FormatMessage
              (
              FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS,
              NULL,
              GetLastError(),
              0,
              lastErrorTxt,
              1024,
              NULL
              );
       printf("%s", lastErrorTxt);

       /*~~~~~~~~~~~~~*/
       printf(str, arg);
       exit(i);
       /*~~~~~~~~~~~~~*/
}

void displayTime()
{
       printf("Current time : %lld", clock());
}


/*
==========================================================================================
==============================
==========================================================================================
==============================
*/
void err(int i, const char *str)
{
       /*~~~~~~~~~~~~~~~~~~~~~~~*/
       char   lastErrorTxt[1024];
       /*~~~~~~~~~~~~~~~~~~~~~~~*/

       FormatMessage
              (
              FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS,
              NULL,
              GetLastError(),
              0,
              lastErrorTxt,
              1024,
              NULL
              );
       printf("%s", lastErrorTxt);
       if(str != NULL)
       {
              printf("%s", str);
```

```
        }

        exit(i);
}

/*
=========================================================================================
==============================
=========================================================================================
==============================
*/
template<class T>
void errx(int i, const char *str, T arg)
{
        /*~~~~~~~~~~~~~*/
        printf(str, arg);
        exit(i);
        /*~~~~~~~~~~~~~*/
}

/*
=========================================================================================
==============================
=========================================================================================
==============================
*/
void errx(int i, const char *str)
{
        printf("%s", str);
        exit(i);
}

#define MIN(x, y)    (((x) < (y)) ? (x) : (y))

/*
=========================================================================================
==============================
=========================================================================================
==============================
*/

static void split(off_t *I, off_t *V, off_t start, off_t len, off_t h)
{
        /*~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
        off_t  i, j, k, x, tmp, jj, kk;
        /*~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/

        if(len < 16)
        {
                for(k = start; k < start + len; k += j)
                {
                        j = 1;
                        x = V[I[k] + h];
                        for(i = 1; k + i < start + len; i++)
                        {
                                if(V[I[k + i] + h] < x)
                                {
                                        x = V[I[k + i] + h];
```

```
                j = 0;
            };
            if(V[I[k + i] + h] == x)
            {
                tmp = I[k + j];
                I[k + j] = I[k + i];
                I[k + i] = tmp;
                j++;
            };
        };
        for(i = 0; i < j; i++) V[I[k + i]] = k + j - 1;
        if(j == 1) I[k] = -1;
    };
    return;
};

x = V[I[start + len / 2] + h];
jj = 0;
kk = 0;
for(i = start; i < start + len; i++)
{
    if(V[I[i] + h] < x) jj++;
    if(V[I[i] + h] == x) kk++;
};
jj += start;
kk += jj;

i = start;
j = 0;
k = 0;
while(i < jj)
{
    if(V[I[i] + h] < x)
    {
        i++;
    }
    else if(V[I[i] + h] == x)
    {
        tmp = I[i];
        I[i] = I[jj + j];
        I[jj + j] = tmp;
        j++;
    }
    else
    {
        tmp = I[i];
        I[i] = I[kk + k];
        I[kk + k] = tmp;
        k++;
    };
};

while(jj + j < kk)
{
    if(V[I[jj + j] + h] == x)
    {
        j++;
    }
```

```
                else
                {
                        tmp = I[jj + j];
                        I[jj + j] = I[kk + k];
                        I[kk + k] = tmp;
                        k++;
                };
        };

        if(jj > start) split(I, V, start, jj - start, h);
        for(i = 0; i < kk - jj; i++) V[I[jj + i]] = kk - 1;
        if(jj == kk - 1) I[jj] = -1;
        if(start + len > kk) split(I, V, kk, start + len - kk, h);
}

/*
=========================================================================================
==============================
=========================================================================================
==============================
*/
static void qsufsort(off_t *I, off_t *V, u_char *old, off_t oldsize)
{
        /*~~~~~~~~~~~~~~~~~*/
        off_t  buckets[256];
        off_t  i, h, len;
        /*~~~~~~~~~~~~~~~~~*/

        for(i = 0; i < 256; i++) buckets[i] = 0;
        for(i = 0; i < oldsize; i++) buckets[old[i]]++;
        for(i = 1; i < 256; i++) buckets[i] += buckets[i - 1];
        for(i = 255; i > 0; i--) buckets[i] = buckets[i - 1];
        buckets[0] = 0;

        for(i = 0; i < oldsize; i++) I[++buckets[old[i]]] = i;
        I[0] = oldsize;
        for(i = 0; i < oldsize; i++) V[i] = buckets[old[i]];
        V[oldsize] = 0;
        for(i = 1; i < 256; i++)
                if(buckets[i] == buckets[i - 1] + 1) I[buckets[i]] = -1;
        I[0] = -1;

        for(h = 1; I[0] != -(oldsize + 1); h += h)
        {
                len = 0;
                for(i = 0; i < oldsize + 1;)
                {
                        if(I[i] < 0)
                        {
                                len -= I[i];
                                i -= I[i];
                        }
                        else
                        {
                                if(len) I[i - len] = -len;
                                len = V[I[i]] + 1 - i;
                                split(I, V, i, len, h);
                                i += len;
```

```
                                len = 0;
                        };
                };
                if(len) I[i - len] = -len;
        };

        for(i = 0; i < oldsize + 1; i++) I[V[i]] = i;
}

/*
=========================================================================================
==============================
=========================================================================================
==============================
*/
static off_t matchlen(u_char *old, off_t oldsize, u_char *_new, off_t newsize)
{
        /*~~~~~~*/
        off_t  i;
        /*~~~~~~*/

        for(i = 0; (i < oldsize) && (i < newsize); i++)
                if(old[i] != _new[i]) break;

        return i;
}

/*
=========================================================================================
==============================
=========================================================================================
==============================
*/
static off_t search(off_t *I, u_char *old, off_t oldsize, u_char *_new, off_t newsize,
off_t st, off_t en, off_t *pos)
{
        /*~~~~~~~~~*/
        off_t  x, y;
        /*~~~~~~~~~*/

        if(en - st < 2)
        {
                x = matchlen(old + I[st], oldsize - I[st], _new, newsize);
                y = matchlen(old + I[en], oldsize - I[en], _new, newsize);

                if(x > y)
                {
                        *pos = I[st];
                        return x;
                }
                else
                {
                        *pos = I[en];
                        return y;
                }
        };

        x = st + (en - st) / 2;
```

```
        if(memcmp(old + I[x], _new, MIN(oldsize - I[x], newsize)) < 0)
        {
                return search(I, old, oldsize, _new, newsize, x, en, pos);
        }
        else
        {
                return search(I, old, oldsize, _new, newsize, st, x, pos);
        };
}

/*
===========================================================================================
==============================
===========================================================================================
==============================
*/
static void offtout(off_t x, u_char *buf)
{
        /*~~~~~~*/
        off_t  y;
        /*~~~~~~*/

        if(x < 0)
                y = -x;
        else
                y = x;

        buf[0] = y % 256;
        y -= buf[0];
        y = y / 256;
        buf[1] = y % 256;
        y -= buf[1];
        y = y / 256;
        buf[2] = y % 256;
        y -= buf[2];
        y = y / 256;
        buf[3] = y % 256;
        y -= buf[3];
        y = y / 256;
        buf[4] = y % 256;
        y -= buf[4];
        y = y / 256;
        buf[5] = y % 256;
        y -= buf[5];
        y = y / 256;
        buf[6] = y % 256;
        y -= buf[6];
        y = y / 256;
        buf[7] = y % 256;

        if(x < 0) buf[7] |= 0x80;
}

char* to_string (long t)
{
        char ss[32];
        sprintf(ss,"%ld",t);
        return ss;
```

```
}

static off_t offtin(u_char *buf)
{
        off_t y;

        y=buf[7]&0x7F;
        y=y*256;y+=buf[6];
        y=y*256;y+=buf[5];
        y=y*256;y+=buf[4];
        y=y*256;y+=buf[3];
        y=y*256;y+=buf[2];
        y=y*256;y+=buf[1];
        y=y*256;y+=buf[0];

        if(buf[7]&0x80) y=-y;

        return y;
}

off_t getBlockStartPosition(long blockNumber, long numberOfNormalBlock, long
numberOfOverlappingBlocks, off_t maxNormalOldBlockSize, off_t newBlockSize)
{
        if(blockNumber <= numberOfNormalBlock)
        {
                return (blockNumber -1) * maxNormalOldBlockSize;
        }
        else if(blockNumber <= numberOfNormalBlock + numberOfOverlappingBlocks)
        {
                return maxNormalOldBlockSize * (blockNumber-numberOfNormalBlock) -
newBlockSize;
        }
        else
        {
                err(1, "Wrong block number %l", blockNumber);
                return -1;
        }
}

off_t createTemppatch(off_t *I, u_char    *old, u_char *_new, off_t newBlocksize, off_t
oldBlocksize, char *tempFileName)
{
        //=================================== Declare required variables
======================================
        u_char *db, *eb;
        off_t  i;
        off_t dblen = 0;
        off_t eblen = 0;
        u_char buf[8];
        off_t  scan, pos, len, lastscan, lastpos, lastoffset, oldscore, scsc;
        off_t  s, Sf, lenf, Sb, lenb;
        off_t  overlap, Ss, lens, diffsize;
        FILE   *tempPatch;
        BZFILE *pfbz2_tempPatch;
        int           bz2err;
        u_char header[32];
        //============================================================================
=====================
        // ========================= Allocate Memory for diff block and extra block
===========================
        if(((db = (u_char *) malloc(newBlocksize + 1)) == NULL) || ((eb = (u_char *)
malloc(newBlocksize + 1)) == NULL))
                err(1, "Unable to allocate memory for diff OR extra block");
        //================================================================================
=====================

        // ================================== Open patch file and write header and open in
bz2 mode ======================================
        if((tempPatch = fopen(tempFileName, "wb")) == NULL) err(1, "%s Unable to open
tempFileName in wb mode", tempFileName);

        memcpy(header, "BSDIFF40", 8);
        offtout(0, header + 8);
        offtout(0, header + 16);
        offtout(newBlocksize, header + 24);
        if(fwrite(header, 32, 1, tempPatch) != 1) err(1, "fwrite(%s)", tempFileName);

        if((pfbz2_tempPatch = BZ2_bzWriteOpen(&bz2err, tempPatch, 9, 0, 0)) == NULL)
errx(1, "BZ2_bzWriteOpen,tempFileName  bz2err = %d", bz2err);
        //================================================================================
=====================

        // ================================== Serach for patterns
======================================

        scan = 0;
        len = 0;
        lastscan = 0;
        lastpos = 0;
        lastoffset = 0;

        while(scan < newBlocksize)
        {
                oldscore = 0;

                for(scsc = scan += len; scan < newBlocksize; scan++)
                {
                        len = search(I, old, oldBlocksize, _new + scan, newBlocksize - scan,
0, oldBlocksize, &pos);

                        for(; scsc < scan + len; scsc++)
                                if((scsc + lastoffset < oldBlocksize) && (old[scsc +
lastoffset] == _new[scsc])) oldscore++;
                        if(((len == oldscore) && (len != 0)) || (len > oldscore + 8)) break;
                        if((scan + lastoffset < oldBlocksize) && (old[scan + lastoffset] ==
_new[scan])) oldscore--;
                };

                if((len != oldscore) || (scan == newBlocksize))
                {
                        s = 0;
                        Sf = 0;
                        lenf = 0;
                        for(i = 0; (lastscan + i < scan) && (lastpos + i < oldBlocksize);)
                        {
                                if(old[lastpos + i] == _new[lastscan + i]) s++;
                                i++;
                                if(s * 2 - i > Sf * 2 - lenf)
                                {
                                       Sf = s;
                                       lenf = i;
                                };
                        };

                        lenb = 0;
                        if(scan < newBlocksize)
                        {
                                s = 0;
                                Sb = 0;
                                for(i = 1; (scan >= lastscan + i) && (pos >= i); i++)
                                {
                                       if(old[pos - i] == _new[scan - i]) s++;
                                       if(s * 2 - i > Sb * 2 - lenb)
                                       {
                                              Sb = s;
                                              lenb = i;
                                       };
                                };
                        };

                        if(lastscan + lenf > scan - lenb)
                        {
                                overlap = (lastscan + lenf) - (scan - lenb);
                                s = 0;
                                Ss = 0;
                                lens = 0;
                                for(i = 0; i < overlap; i++)
                                {
                                       if(_new[lastscan + lenf - overlap + i] == old[lastpos +
lenf - overlap + i]) s++;
                                       if(_new[scan - lenb + i] == old[pos - lenb + i]) s--;
                                       if(s > Ss)
                                       {
                                              Ss = s;
                                              lens = i + 1;
                                       };
                                };

                                lenf += lens - overlap;
                                lenb -= lens;
                        };


        //==================================================================================
=====================

                        // ================== Create/write diff and extra block data
===========================================
                        for(i = 0; i < lenf; i++) db[dblen + i] = _new[lastscan + i] -
old[lastpos + i];
                        for(i = 0; i < (scan - lenb) - (lastscan + lenf); i++) eb[eblen + i]
= _new[lastscan + lenf + i];
                        dblen += lenf;
                        eblen += (scan - lenb) - (lastscan + lenf);

        //================================================================================
=====================

                        // ===================== Writing control block diff segment length
====================================
                        offtout(lenf, buf);
                        BZ2_bzWrite(&bz2err, pfbz2_tempPatch, buf, 8);
                        if(bz2err != BZ_OK) errx(1, "BZ2_bzWrite, bz2err = %d", bz2err);

        //================================================================================
=====================

                        // ===================== Writing control block extra segment length
====================================
                        offtout((scan - lenb) - (lastscan + lenf), buf);
                        BZ2_bzWrite(&bz2err, pfbz2_tempPatch, buf, 8);
                        if(bz2err != BZ_OK) errx(1, "BZ2_bzWrite, bz2err = %d", bz2err);

        //================================================================================
=====================

                        // ===================== Writing control block old offset
adjustments ====================================
                        offtout((pos - lenb) - (lastpos + lenf), buf);
                        BZ2_bzWrite(&bz2err, pfbz2_tempPatch, buf, 8);
                        if(bz2err != BZ_OK) errx(1, "BZ2_bzWrite, bz2err = %d", bz2err);

        //================================================================================
=====================

                        // ===================== Adjust pointers for next match
====================================
                        lastscan = scan - lenb;
                        lastpos = pos - lenb;
                        lastoffset = pos - scan;

        //================================================================================
=====================
              }
        }
        BZ2_bzWriteClose(&bz2err, pfbz2_tempPatch, 0, NULL, NULL);
        // ============================= Compute size of compressed ctrl data
==================================
        if((fseek(tempPatch, 0, SEEK_END)) )  err(1, "fwrite(%s)", tempFileName);
        if((len = ftell(tempPatch)) == -1) err(1, "ftello");
        offtout(len - 32, header + 8);
        if((pfbz2_tempPatch = BZ2_bzWriteOpen(&bz2err, tempPatch, 9, 0, 0)) == NULL)
errx(1, "BZ2_bzWriteOpen,tempFileName  bz2err = %d", bz2err);

        // ===================== Writing of Diff data ====================================
        BZ2_bzWrite(&bz2err, pfbz2_tempPatch, db, dblen);
        BZ2_bzWriteClose(&bz2err, pfbz2_tempPatch, 0, NULL, NULL);
        if(bz2err != BZ_OK) errx(1, "BZ2_bzWrite, bz2err = %d", bz2err);
        //------------ Compute size of compressed diff data
        if((fseek(tempPatch, 0, SEEK_END)) )  err(1, "fwrite(%s)", tempFileName);
        if((diffsize = ftell(tempPatch)) == -1) err(1, "ftello");
        offtout(diffsize - len, header + 16);
        if((pfbz2_tempPatch = BZ2_bzWriteOpen(&bz2err, tempPatch, 9, 0, 0)) == NULL)
 errx(1, "BZ2_bzWriteOpen,tempFileName  bz2err = %d", bz2err);
        //================================================================================
 =====================

        // ===================== Writing of extra data
 ====================================
        BZ2_bzWrite(&bz2err, pfbz2_tempPatch, eb, eblen);
        if(bz2err != BZ_OK) errx(1, "BZ2_bzWrite, bz2err = %d", bz2err);
        //================================================================================
 =====================

        // Close
        BZ2_bzWriteClose(&bz2err, pfbz2_tempPatch, 0, NULL, NULL);
        if(bz2err != BZ_OK) errx(1, "BZ2_bzWriteClose, bz2err = %d", bz2err);

        // ============================= Update header block
 ===================================================

        if(fseek(tempPatch, 0, SEEK_SET)) err(1, "fseeko");
        if(fwrite(header, 32, 1, tempPatch) != 1) err(1, "fwrite(%s)", tempFileName);
        if((fseek(tempPatch, 0, SEEK_END)) )  err(1, "fwrite(%s)", tempFileName);
        flushall();
        off_t tempSize;
        if((tempSize = ftell(tempPatch)) == -1) err(1, "ftello");

        if(fclose(tempPatch)) err(1, "fclose");

        /* Free the memory we used */
        free(db);
        free(eb);
        return tempSize;
 }



 int main(int argc, char *argv[])
 {
        displayTime();
        // PreCondition
        if(argc != 7) errx(1, "usage: %s oldfile newfile patchfile maxOldBlockSizeInKB
 leftFocus rightFocus\n", argv[0]);

        char *TEMP_FILE = "temp.patch";

        //================ Data to which hold best matching old bock and related patch
 size =========================
        off_t * bestMatchingPatchSize;
        off_t * bestMatchingOldBlockIndex;
        off_t * bestMatchingOldBlockSize; // TODO this can be removed by some other
 alternate code to save space

        //================ Read old normal block size =========================
        off_t maxOldBlockSizeInKB = atof(argv[4]);
        off_t normalOldBlocksize = maxOldBlockSizeInKB * 1024; // Converting KB to bytes
 so normalOldBlocksize holds number of bytes
        off_t  maxOldNormalBlockSize = normalOldBlocksize;
```

```
        off_t totalOldSize = -1;

        //================ Read new block size ========================
        off_t maxNewBlockSizeInKB = 1024; // Default value
        off_t newBlocksize = maxNewBlockSizeInKB * 1024; // Converting KB to bytes so
newBlocksize holds number of bytes
        off_t  maxNewBlockSize = newBlocksize;
        off_t totalNewSize = -1;

        //================ Read left focus ========================
        off_t leftFocusInMb = atof(argv[5]);
        off_t leftFocus = leftFocusInMb * 1024 * 1024;


        //================ Read right focus ========================
        off_t rightFocusInMb = atof(argv[6]);
        off_t rightFocus = rightFocusInMb * 1024 * 1024;

        //================ Read old overlapping block size ========================
        off_t overlappingOldBlocksize = maxNewBlockSize * 2; // Overlapping block size is
twice the size of new file block
        off_t  maxOldOverlappingBlockSize = overlappingOldBlocksize;

        if(maxOldOverlappingBlockSize >= maxOldNormalBlockSize)
        {
                errx(1, "usage: %s maxNewBlockSizeInKB should be at max half of
maxOldBlockSizeInKB\n", argv[0]);
        }

        //================ Initialize file pointer and other data for old File
===========================
        off_t fd_old;
        off_t numberOfOldNormalBlocks = 0;
        off_t numberOfOldOverlappingBlocks = 0;
        if
                (
                ((fd_old = open(argv[1], O_RDONLY | O_BINARY | O_NOINHERIT, 0)) < 0)
                ||      ((totalOldSize = lseek(fd_old, 0, SEEK_END)) == -1)
                ||      (lseek(fd_old, 0, SEEK_SET) != 0)
                ) err(1, "%s", argv[1]);

        if(normalOldBlocksize > totalOldSize)
        {
                normalOldBlocksize = totalOldSize;
        }

        numberOfOldNormalBlocks = ceil((double)totalOldSize/normalOldBlocksize);
        numberOfOldOverlappingBlocks = numberOfOldNormalBlocks - 1;
        printf("\nTotal number of normal blocks: %ld",numberOfOldNormalBlocks);
        printf("\n Total number of overlapping blocks: %ld",
numberOfOldOverlappingBlocks);
        displayTime();

        //================ Initialize file pointer and other data for new File
===========================
        off_t fd_new;
        off_t numberOfNewBlocks = 0;
        if
```

```
                (
                ((fd_new = open(argv[2], O_RDONLY | O_BINARY | O_NOINHERIT, 0)) < 0)
                ||      ((totalNewSize = lseek(fd_new, 0, SEEK_END)) == -1)
                ||      (lseek(fd_new, 0, SEEK_SET) != 0)
                ) err(1, "%s", argv[2]);

        if(newBlocksize > totalNewSize)
        {
                newBlocksize = totalNewSize;
        }

        numberOfNewBlocks = ceil((double)totalNewSize/newBlocksize);
        printf("\n Total number of new blocks %ld", numberOfNewBlocks);
        bestMatchingPatchSize = (off_t *) malloc(numberOfNewBlocks * sizeof(off_t));
        bestMatchingOldBlockIndex = (off_t *) malloc(numberOfNewBlocks * sizeof(off_t));
        bestMatchingOldBlockSize = (off_t *) malloc(numberOfNewBlocks * sizeof(off_t));

        // Iteration 1
        // 1.  All entries in figure 2 will be initialized to -1. The program starts with
iteration 1
        for(int index = 0; index<numberOfNewBlocks; index++)
        {
                bestMatchingPatchSize[index] = -1;
                bestMatchingOldBlockIndex[index] = -1;
                bestMatchingOldBlockSize[index] = -1;
        }

        // Allocate memory to read old and new blocks
        u_char *old, *_new;
        if((old = (u_char *) malloc(normalOldBlocksize + 1)) == NULL)
        {
                err(1, "%s Unable to allocate old block space", argv[1]);
        }


        // 2. OldBlockIndex = 1.
        // The suffix tree for O[OldBlockIndex] is generated.
        for(long normalOldBlockIndex = 1; normalOldBlockIndex<= numberOfOldNormalBlocks;
normalOldBlockIndex++)
                //for(long normalOldBlockIndex = 1; normalOldBlockIndex<= 0;
normalOldBlockIndex++)
        {
                if(totalNewSize < maxNewBlockSize)
                {
                        newBlocksize = totalNewSize;
                }
                else
                {
                        newBlocksize = maxNewBlockSize;
                }
                if((_new = (u_char *) malloc(newBlocksize + 1)) == NULL)
                {
                        err(1, "%s  Unable to allocate new block space", argv[2]);
                }

                // Memory allocation for last block
                if(normalOldBlockIndex == numberOfOldNormalBlocks)
                {
                     normalOldBlocksize = totalOldSize - ((normalOldBlockIndex -1) *
normalOldBlocksize);
                     free(old);
                     if((old = (u_char *) malloc(normalOldBlocksize + 1)) == NULL)
                     {
                            err(1, "%s Unable to allocate old block space", argv[1]);
                     }
              }

              // Read old block data in buffer old
              off_t  i;
              int r = normalOldBlocksize;
              while(r > 0 && (i = read(fd_old, old + normalOldBlocksize - r, r)) > 0) r -
= i;
              // Post Check of read: If r is equal to 0 then only we confirm that we have
read expected amount of data
              if(r > 0) err(1, "%s Unable to read the Old file", argv[1]);

              // Create suffix tree for the current old block
              off_t  *I, *V;
              if
                     (
                     ((I = (off_t *) malloc((normalOldBlocksize + 1) * sizeof(off_t))) ==
NULL)
                     ||     ((V = (off_t *) malloc((normalOldBlocksize + 1) *
sizeof(off_t))) == NULL)
                     ) err(1,  "%s unable to allocate memory for I OR V", argv[0]);

              qsufsort(I, V, old, normalOldBlocksize);
              free(V); // Free the memory of V as it is all -1 and not required

              // 3.  NewBlockIndex = 1.

              off_t startingNewBlock = (normalOldBlockIndex-1) *
(maxOldNormalBlockSize/(1024*1024)) - leftFocusInMb;
              if(startingNewBlock <1)
              {
                     startingNewBlock = 1;
              }

              off_t LastNewBlock = (normalOldBlockIndex) *
(maxOldNormalBlockSize/(1024*1024)) + rightFocusInMb;
              if(LastNewBlock > numberOfNewBlocks)
              {
                     LastNewBlock = numberOfNewBlocks;
              }
              if(normalOldBlockIndex == numberOfOldNormalBlocks)
              {
                     LastNewBlock = numberOfNewBlocks;
              }
              if((lseek(fd_new, (startingNewBlock-1) *1024 *1024, SEEK_SET) !=
(startingNewBlock -1) *1024 *1024)) err(1, "%s Unable to point to start of new file",
argv[1]);
              for(long newBlockIndex = startingNewBlock; newBlockIndex <= LastNewBlock;
newBlockIndex++)
              {
                     printf("\n Normal %ld and %ld",normalOldBlockIndex, newBlockIndex);
                     displayTime();
                     // Allocate only required memory for last block
                     if(newBlockIndex == numberOfNewBlocks)
                     {
                            newBlocksize =  (totalNewSize - (newBlockIndex -1) * 
newBlocksize);
                            free(_new);
                            if((_new = (u_char *) malloc(newBlocksize + 1)) == NULL)
                            {
                                   err(1, "%s Unable to allocate memory for new block", 
argv[2]);
                            }
                     }

                     // Read new file block in _new
                     r = newBlocksize;
                     while(r > 0 && (i = read(fd_new, _new + newBlocksize - r, r)) > 0) r 
-= i;
                     if(r > 0) err(1, "%s Unable to read new file", argv[2]);

                     // 4.  By using suffix tree of O[OldBlockIndex], we create patch file 
(temp.patch) for N[NewBlockIndex].
                     off_t tempPatchSize = createTemppatch(I, old, _new, newBlocksize, 
normalOldBlocksize, TEMP_FILE);

                     /*
                     5.     If(Best matching patch size of [NewBlockIndex-1]== -1 OR Best 
matching patch size of [NewBlockIndex-1] > temp.patch size created in 3)
                     then do 6
                     else goto 7
                     6.     The patch size is stored in Figure 2 in the cell marked by 
Best matching patch size of [NewBlockIndex-1].
                     OldBlockIndex is noted in the cell marked by “Best matching old 
block  number[NewBlockIndex-1]”
                     The size of the old block (8) is noted in the cell below it.
                     Rename temp.patch file as temp_ NewBlockIndex.patch
                     7.     NewBlockIndex ++
                     if NewBlockIndex  is valid (value between 1 to 17)  for given new 
file
                     goto 4

                     */
                     if(bestMatchingPatchSize[newBlockIndex-1] == -1 || 
bestMatchingPatchSize[newBlockIndex-1] > tempPatchSize)
                     {
                            bestMatchingPatchSize[newBlockIndex-1] = tempPatchSize;
                            bestMatchingOldBlockIndex[newBlockIndex-1] = 
normalOldBlockIndex;
                            bestMatchingOldBlockSize[newBlockIndex-1] = 
normalOldBlocksize;
                            char newTempFileName[25] = "temp_";
                            strcat(newTempFileName, to_string(newBlockIndex));
                            strcat(newTempFileName, ".patch");
                            if(remove(newTempFileName)!= 0)
                            {
                                   printf("Unable to remove %s",newTempFileName);
                            }
```

```
                if(rename(TEMP_FILE, newTempFileName) != 0)
                {
                    err(1, "unable to rename the temp file to new for new 
index %l", newBlockIndex);
                }
            }
        }
        free(I);
        free(_new);
        /*
        Below is done by for loop for old index
        8.     OldBlockIndex ++;
        9.     If OldBlockIndex is valid (i.e value between 1 to 3) then goto 2.
        */
    }
    free(old);
    close(fd_old);

    if
        (
        ((fd_old = open(argv[1], O_RDONLY | O_BINARY | O_NOINHERIT, 0)) < 0)
        ) err(1, "%s", argv[1]);
    close(fd_new);

    if
        (
        ((fd_new = open(argv[2], O_RDONLY | O_BINARY | O_NOINHERIT, 0)) < 0)

        ) err(1, "%s", argv[2]);

    // b.  Iteration 2: Finding patterns in overlapping old blocks (old block size = 2 
* new block size),
    if((old = (u_char *) malloc(overlappingOldBlocksize + 1)) == NULL)
    {
        err(1, "%s Unable to allocate old block space", argv[1]);
    }

    // The suffix tree for O[OldBlockIndex] is generated.
    for(long overlappingOldBlockIndex = 1; overlappingOldBlockIndex<= 
numberOfOldOverlappingBlocks; overlappingOldBlockIndex++)
    {
        if(totalNewSize < maxNewBlockSize)
        {
            newBlocksize = totalNewSize;
        }
        else
        {
            newBlocksize = maxNewBlockSize;
        }

        off_t startIndex = maxOldNormalBlockSize * overlappingOldBlockIndex - 
newBlocksize;
        long returnValue =  lseek(fd_old, startIndex, SEEK_SET);
        if(returnValue != startIndex )
        {
            err(1, "%s ", argv[1]);

        }
```

```
              if((_new = (u_char *) malloc(newBlocksize + 1)) == NULL)
              {
                     err(1, "%s  Unable to allocate new block space", argv[2]);
              }

              // Memory allocation for last block
              if(overlappingOldBlockIndex == numberOfOldOverlappingBlocks)
              {
                     overlappingOldBlocksize = totalOldSize - startIndex;
                     free(old);
                     if((old = (u_char *) malloc(overlappingOldBlocksize + 1)) == NULL)
                     {
                            err(1, "%s Unable to allocate old block space", argv[1]);
                     }
              }

              // Read old block data in buffer old
              off_t  i;
              int r = overlappingOldBlocksize;
              while(r > 0 && (i = read(fd_old, old + overlappingOldBlocksize - r, r)) >
0) r -= i;
              // Post Check of read: If r is equal to 0 then only we confirm that we have
read expected amount of data
              if(r > 0) err(1, "%s Unable to read the Old file in second iteration",
argv[1]);

              // Create suffix tree for the current old block
              off_t  *I, *V;
              if
                     (
                     ((I = (off_t *) malloc((overlappingOldBlocksize + 1) *
sizeof(off_t))) == NULL)
                     ||     ((V = (off_t *) malloc((overlappingOldBlocksize + 1) *
sizeof(off_t))) == NULL)
                     ) err(1,  "%s unable to allocate memory for I OR V", argv[0]);

              qsufsort(I, V, old, overlappingOldBlocksize);
              free(V); // Free the memory of V as it is all -1 and not required

              off_t startingNewBlock =  (startIndex/(1024*1024)) - leftFocusInMb;
              if(startingNewBlock <1)
              {
                     startingNewBlock = 1;
              }

              off_t LastNewBlock = (startIndex/(1024*1024)) + rightFocusInMb +2;
              if(LastNewBlock > numberOfNewBlocks)
              {
                     LastNewBlock = numberOfNewBlocks;
              }
              if((lseek(fd_new, (startingNewBlock-1) *1024 *1024, SEEK_SET) !=
(startingNewBlock-1) *1024 *1024)) err(1, "%s Unable to point to start of new file",
argv[1]);
              if(overlappingOldBlockIndex== numberOfOldOverlappingBlocks)
              {
                     LastNewBlock = numberOfNewBlocks;
              }
```

```
                for(long newBlockIndex = startingNewBlock; newBlockIndex <= LastNewBlock;
newBlockIndex++)
                {
                        printf("\n Overlapping %ld and %ld",overlappingOldBlockIndex,
newBlockIndex);
                        displayTime();
                        // Allocate only required memodry location for last block
                        if(newBlockIndex == numberOfNewBlocks)
                        {
                                newBlocksize =  (totalNewSize - (newBlockIndex -1) *
newBlocksize);
                                free(_new);
                                if((_new = (u_char *) malloc(newBlocksize + 1)) == NULL)
                                {
                                        err(1, "%s Unable to allocate memory for new block in
2nd iteration", argv[2]);
                                }
                        }

                        // Read new file block in _new
                        r = newBlocksize;
                        while(r > 0 && (i = read(fd_new, _new + newBlocksize - r, r)) > 0) r
-= i;
                        if(r > 0) err(1, "%s Unable to read new file", argv[2]);

                        off_t tempPatchSize = createTemppatch(I, old, _new, newBlocksize,
overlappingOldBlocksize, TEMP_FILE);

                        if(bestMatchingPatchSize[newBlockIndex-1] == -1 ||
bestMatchingPatchSize[newBlockIndex-1] > tempPatchSize)
                        {
                                bestMatchingPatchSize[newBlockIndex-1] = tempPatchSize;
                                bestMatchingOldBlockIndex[newBlockIndex-1] =
overlappingOldBlockIndex + numberOfOldNormalBlocks;
                                bestMatchingOldBlockSize[newBlockIndex-1] =
overlappingOldBlocksize;
                                char newTempFileName[25] = "temp_";
                                strcat(newTempFileName, to_string(newBlockIndex));
                                strcat(newTempFileName, ".patch");
                                if(remove(newTempFileName)!= 0)
                                {
                                        printf("Unable to remove %s",newTempFileName);
                                }
                                if(rename(TEMP_FILE, newTempFileName) != 0)
                                {
                                        err(1, "unable to rename the temp file to new for new
index %l", newBlockIndex);
                                }
                        }
                }
                free(I);
                free(_new);
        }
        free(old);
        close(fd_old);

        // c.  Final patch creation
        FILE *finalPatch, *tempDiff, *tempExtra;
```

```
       BZFILE *pfbz2_finalPatch, *pfbz2_tempDiff, *pfbz2_tempExtra;
       u_char header[32];
       u_char buf[8];
       int           bz2err;

       if((finalPatch = fopen(argv[3], "wb")) == NULL) err(1, "%s Unable to open final
patch in wb mode", argv[3]);
       if((tempDiff = fopen("tempDiff.patch", "wb")) == NULL) err(1, "%s Unable to open
final patch in wb mode", "tempDiff.patch");
       if((tempExtra = fopen("tempExtra.patch", "wb")) == NULL) err(1, "%s Unable to open
final patch in wb mode", "tempExtra.patch");

       memcpy(header, "BSDIFF40", 8);
       offtout(0, header + 8);
       offtout(0, header + 16);
       offtout(totalNewSize, header + 24);
       if(fwrite(header, 32, 1, finalPatch) != 1) err(1, "fwrite(%s)", argv[3]);
       fflush(finalPatch);
       if((pfbz2_finalPatch = BZ2_bzWriteOpen(&bz2err, finalPatch, 9, 0, 0)) == NULL)
errx(1, "BZ2_bzWriteOpen,tempFileName  bz2err = %d", bz2err);
       if((pfbz2_tempDiff = BZ2_bzWriteOpen(&bz2err, tempDiff, 9, 0, 0)) == NULL) errx(1,
"BZ2_bzWriteOpen,tempFileName  bz2err = %d", bz2err);
       if((pfbz2_tempExtra = BZ2_bzWriteOpen(&bz2err, tempExtra, 9, 0, 0)) == NULL)
errx(1, "BZ2_bzWriteOpen,tempFileName  bz2err = %d", bz2err);

       off_t preCtrl[3];
       off_t currentPosition;

       //21.  Add dummy control block entry with diff size = 0, extra size = 0 and
properly adjusted offset by using table 2.
       // ===================== Writing control block diff segment length
====================================
       preCtrl[0] = 0;
       //================================================================================
=====================

       // ===================== Writing control block extra segment length
====================================
       preCtrl[1] = 0;
       //================================================================================
=====================

       // ===================== Writing control block old offset adjustments
====================================
       off_t startOfBlock = getBlockStartPosition(bestMatchingOldBlockIndex[0],
numberOfOldNormalBlocks, numberOfOldOverlappingBlocks, maxOldNormalBlockSize,
maxNewBlockSize);
       preCtrl[2] = startOfBlock;
       currentPosition = 0;
       //================================================================================
=====================

       for(long newBlockIndex = 1; newBlockIndex <= numberOfNewBlocks; newBlockIndex++)
       {
              if(newBlockIndex == numberOfNewBlocks)
              {
                     printf("\nProcessing of final block started");
              }
              printf("\n Merging %ld in final patch ", newBlockIndex);
              displayTime();
              FILE * cpf, * dpf, * epf;
              BZFILE * cpfbz2, *dpfbz2, *epfbz2;
              FILE *tempControlBlock;
              FILE *tempDiffBlock;
              FILE *tempExtraBlock;

              int cbz2err, dbz2err, ebz2err;
              signed int bzctrllen,bzdatalen;

              if((tempControlBlock = fopen("tempControlBlock.patch", "wb")) == NULL)
err(1, "%s Unable to open final patch in wb mode", "tempControlBlock.patch");
              if((tempDiffBlock = fopen("tempDiffBlock.patch", "wb")) == NULL) err(1, "%s
Unable to open final patch in wb mode", "tempDiffBlock.patch");
              if((tempExtraBlock = fopen("tempExtraBlock.patch", "wb")) == NULL) err(1,
"%s Unable to open final patch in wb mode", "tempExtraBlock.patch");

              char newTempFileName[25] = "temp_";
              strcat(newTempFileName, to_string(newBlockIndex));
              strcat(newTempFileName, ".patch");

              off_t temp_patch;

              if
                     (
                     ((temp_patch = open(newTempFileName, O_RDONLY | O_BINARY |
O_NOINHERIT, 0)) < 0)
                     ) err(1, "%s", newTempFileName);

              /* Read header */
              if (read(temp_patch, header, 32) < 32) {
                     if (eof(temp_patch))
                            errx(1, "Corrupt patch\n");
                     err(1, "fread(%s)", newTempFileName);
              }

              /* Check for appropriate magic */
              if (memcmp(header, "BSDIFF40", 8) != 0)
                     errx(1, "Corrupt patch\n %s", newTempFileName);

              /* Read lengths from header */
              bzctrllen=offtin(header+8);
              bzdatalen=offtin(header+16);

              close(temp_patch);

              u_char *ctrlBlocks;
              ctrlBlocks = (u_char *) malloc(bzctrllen);
              /* Read All control blocks */
              if
                     (
                     ((temp_patch = open(newTempFileName, O_RDONLY | O_BINARY |
O_NOINHERIT, 0)) < 0)
                     ) err(1, "%s", newTempFileName);

              long returnValue =  lseek(temp_patch, 32, SEEK_SET);
              if(returnValue != 32)
```

```
        {
            errx(1, "Unable to seek in temp patch file\n");
        }

        long r = bzctrllen,i;
        while(r > 0 && (i = read(temp_patch, ctrlBlocks + bzctrllen - r, r)) > 0) r
-= i;
        if(r > 0) err(1, "%s Unable to read new file", newTempFileName);

        if(fwrite(ctrlBlocks, bzctrllen, 1, tempControlBlock) != 1) err(1,
"fwrite(%s)", newTempFileName);
        fflush(tempControlBlock);
        free(ctrlBlocks);
        fclose(tempControlBlock);



        if((cpf = fopen("tempControlBlock.patch", "rb")) == NULL) err(1,
"fopen(%s)", "tempControlBlock.patch");
        if(fseek(cpf, 0, SEEK_SET)) err(1, "fseeko(%s,0)",
"tempControlBlock.patch");
        if((cpfbz2 = BZ2_bzReadOpen(&cbz2err, cpf, 0, 0, NULL, 0)) == NULL)
            errx(1, "BZ2_bzReadOpen, bz2err = %d", cbz2err);

        // Read control blocks and right to file
        off_t ctrl[3];
        ctrl[0] = -1;
        ctrl[1] = -1;
        ctrl[2] = -1;
        while(true)
        {

            /* Read control data */
            int i;
            for(i=0;i<=2;i++)
            {
                int lenread = BZ2_bzRead(&cbz2err, cpfbz2, buf, 8);
                if ((lenread < 8) || ((cbz2err != BZ_OK) &&
                    (cbz2err != BZ_STREAM_END)))
                {
                    if(i==0 && lenread<1)
                    {
                        break;
                    }
                    else
                    {
                        errx(1, "Corrupt patch for %s \n",
newBlockIndex);
                    }
                }
                ctrl[i]=offtin(buf);
            }
            if(i==0)
            {
                break;
            }
            if(i!=3)
            {
                err(1, "currupt patch %s \n", newBlockIndex);
            }
            offtout(preCtrl[0], buf);
            BZ2_bzWrite(&bz2err, pfbz2_finalPatch, buf, 8);
            if(bz2err != BZ_OK) errx(1, "BZ2_bzWrite, bz2err = %d", bz2err);

            offtout(preCtrl[1], buf);
            BZ2_bzWrite(&bz2err, pfbz2_finalPatch, buf, 8);
            if(bz2err != BZ_OK) errx(1, "BZ2_bzWrite, bz2err = %d", bz2err);

            offtout(preCtrl[2], buf);
            BZ2_bzWrite(&bz2err, pfbz2_finalPatch, buf, 8);
            if(bz2err != BZ_OK) errx(1, "BZ2_bzWrite, bz2err = %d", bz2err);
            currentPosition += ctrl[0] + preCtrl[2];
            //printf("\nUpdated : %ld %ld %ld", 
preCtrl[0],preCtrl[1],preCtrl[2]);
            //printf("\nOriginal: %ld %ld %ld", ctrl[0],ctrl[1],ctrl[2]);

            // Lastly current becomes previous Not used for loop for 
optimization
            preCtrl[0] = ctrl[0];
            preCtrl[1] = ctrl[1];
            preCtrl[2] = ctrl[2];

        }

        off_t final_offset = preCtrl[2];

        if(newBlockIndex != numberOfNewBlocks)
        {
            off_t nextStartOfBlock = 
getBlockStartPosition(bestMatchingOldBlockIndex[newBlockIndex], numberOfOldNormalBlocks, 
numberOfOldOverlappingBlocks, maxOldNormalBlockSize, maxNewBlockSize);
            final_offset = nextStartOfBlock - currentPosition;
        }
        //  currentPosition = nextStartOfBlock;
        preCtrl[2] = final_offset;

        if(newBlockIndex == numberOfNewBlocks)
        {
            offtout(preCtrl[0], buf);
            BZ2_bzWrite(&bz2err, pfbz2_finalPatch, buf, 8);
            if(bz2err != BZ_OK) errx(1, "BZ2_bzWrite, bz2err = %d", bz2err);

            offtout(preCtrl[1], buf);
            BZ2_bzWrite(&bz2err, pfbz2_finalPatch, buf, 8);
            if(bz2err != BZ_OK) errx(1, "BZ2_bzWrite, bz2err = %d", bz2err);

            offtout(final_offset, buf);
            BZ2_bzWrite(&bz2err, pfbz2_finalPatch, buf, 8);
            if(bz2err != BZ_OK) errx(1, "BZ2_bzWrite, bz2err = %d", bz2err);
            printf("\nfinal %ld %ld %ld", preCtrl[0],preCtrl[1],preCtrl[2]);
        }
        BZ2_bzReadClose(&bz2err, cpfbz2);
        if(bz2err != BZ_OK) errx(1, "BZ2_bzReadClose cpfbz2 %d", bz2err);
        close(temp_patch);

        if
```

```
                     (
                     ((temp_patch = open(newTempFileName, O_RDONLY | O_BINARY |
O_NOINHERIT, 0)) < 0)
                     ) err(1, "%s", newTempFileName);

              returnValue =  lseek(temp_patch, 32 + bzctrllen, SEEK_SET);
              if(returnValue != 32+bzctrllen)
              {
                     errx(1, "Unable to seek in temp patch file\n");
              }

              // Now write current blocks diff block in temp diff
              u_char *diffBlocks;
              diffBlocks = (u_char *) malloc(bzdatalen);

              /* Read All diff blocks */

              r = bzdatalen,i;
              while(r > 0 && (i = read(temp_patch, diffBlocks + bzdatalen - r, r)) > 0) r
-= i;
              if(r > 0) err(1, "%s Unable to read new file", newTempFileName);

              if(fwrite(diffBlocks, bzdatalen, 1, tempDiffBlock) != 1) err(1, "fwrite(%s)
Diff write error %s", newTempFileName);
              fflush(tempDiffBlock);
              free(diffBlocks);
              fclose(tempDiffBlock);

              if((dpf = fopen("tempDiffBlock.patch", "rb")) == NULL) err(1, "fopen(%s)",
"tempDiffBlock.patch");

              if((dpfbz2 = BZ2_bzReadOpen(&cbz2err, dpf, 0, 0, NULL, 0)) == NULL)
                     errx(1, "BZ2_bzReadOpen, bz2err = %d", cbz2err);
              diffBlocks = (u_char *) malloc(1024);
              while(true)
              {
                     int lenread = BZ2_bzRead(&cbz2err, dpfbz2, diffBlocks, 1024);
                     if ((lenread < 1024) )
                     {
                            if(lenread <=0)
                            {
                                   break;
                            }
                     }
                     if ( ((cbz2err != BZ_OK) &&
                            (cbz2err != BZ_STREAM_END)))
                     {
                            err(1, "fwrite(%s) Error while reading %s", newTempFileName);
                            break;
                     }
                     BZ2_bzWrite(&bz2err, pfbz2_tempDiff, diffBlocks, lenread);

              }
              BZ2_bzReadClose(&bz2err, dpfbz2);
              if(bz2err != BZ_OK) errx(1, "BZ2_bzReadClose dpfbz2 %d", bz2err);


              close(temp_patch);
```

```
                if
                        (
                        ((temp_patch = open(newTempFileName, O_RDONLY | O_BINARY |
O_NOINHERIT, 0)) < 0)
                        ) err(1, "%s", newTempFileName);

                returnValue =  lseek(temp_patch, 32+bzctrllen+bzdatalen, SEEK_SET);
                if(returnValue != 32+bzctrllen+bzdatalen)
                {
                        errx(1, "Unable to seek in temp patch file\n");
                }

                //
                u_char *extraBlocks;
                extraBlocks = (u_char *) malloc(1024);
                /* Read extra blocks 1KB at a time*/
                while (true)
                {
                        long r = 1024,i;
                        r = read(temp_patch, extraBlocks, r);
                        if(r<=0)
                        {
                                break;
                        }
                        if(fwrite(extraBlocks, r, 1, tempExtraBlock) != 1) err(1,
"fwrite(%s) Diff write error %s", newTempFileName);
                        fflush(tempExtraBlock);
                }

                fclose(tempExtraBlock);
                if ((tempExtraBlock = fopen("tempExtraBlock.patch", "rb")) == NULL)
                {
                        err(1, "fopen(%s)", "tempExtraBlock.patch");
                }

                if ((epfbz2 = BZ2_bzReadOpen(&cbz2err, tempExtraBlock, 0, 0, NULL, 0)) ==
NULL)
                {
                        errx(1, "BZ2_bzReadOpen, bz2err = %d", cbz2err);
                }

                while(true)
                {
                        int lenread = BZ2_bzRead(&cbz2err, epfbz2, extraBlocks, 1024);
                        if ((lenread < 1024))
                        {
                                if(lenread <=0)
                                {
                                        break;
                                }
                        }
                        if (((cbz2err != BZ_OK) &&
                                (cbz2err != BZ_STREAM_END)))
                        {
                                err(1, "fwrite(%s) Error while reading %s", newTempFileName);
                                break;
                        }
                        BZ2_bzWrite(&bz2err, pfbz2_tempExtra, extraBlocks, lenread);
                }
                BZ2_bzReadClose(&bz2err, epfbz2);
                if(bz2err != BZ_OK) errx(1, "BZ2_bzReadClose epfbz2 %d", bz2err);
                free(extraBlocks);
                close(temp_patch);
                fclose(tempExtraBlock);
                fclose(dpf);
                fclose(cpf);
        }
        BZ2_bzWriteClose(&bz2err, pfbz2_tempDiff, 0, NULL, NULL);
        BZ2_bzWriteClose(&bz2err, pfbz2_tempExtra, 0, NULL, NULL);
        BZ2_bzWriteClose(&bz2err, pfbz2_finalPatch, 0, NULL, NULL);
        /* Compute final size of compressed ctrl data */
        long len = 0;
        if((len = ftell(finalPatch)) == -1) err(1, "ftello error in finding final ctr
len");
        offtout(len - 32, header + 8);
        //fflush(tempDiff);
        fflush(tempExtra);
        fclose(tempDiff);
        fclose(tempExtra);

        off_t temp_diff, temp_extra;
        if
                (
                ((temp_diff = open("tempDiff.patch", O_RDONLY | O_BINARY | O_NOINHERIT, 0))
< 0)
                ) err(1, "%s", "tempDiff.patch");
        if
                (
                ((temp_extra = open("tempExtra.patch", O_RDONLY | O_BINARY | O_NOINHERIT,
0)) < 0)
                ) err(1, "%s", "tempExtra.patch");

        // Now write all diff blocks in final patch
        if((fseek(finalPatch, 0, SEEK_END)) )  err(1, "fwrite(%s)", argv[3]);


        // ===================== Writing of Diff data ===================================


        if(bz2err != BZ_OK) errx(1, "BZ2_bzWrite, bz2err = %d", bz2err);
        u_char *buffer;
        buffer = (u_char *) malloc(1024);
        /* Read extra blocks 1KB at a time*/
        long counter = 0;
        while (true)
        {
                int r = read(temp_diff, buffer, 1024);
                if(r<=0)
                        break;
                if(fwrite(buffer, r, 1, finalPatch) != 1) err(1, "fwrite(%s) Diff write
error %s", "final diff");
                fflush(finalPatch);

                counter++;
        }
        printf("\nDiff 1024 counter = %ld", counter);

        /* Compute final size of diff data */
        long finalDiffSize = 0;
        if((fseek(finalPatch, 0, SEEK_END)) )  err(1, "fwrite(%s)", argv[3]);



        if((finalDiffSize = ftell(finalPatch)) == -1) err(1, "ftello error in finding
finalDiffSize");
        offtout(finalDiffSize - len, header + 16);

        // Now write all extra block in final patch

        /* Read extra blocks 1KB at a time*/
        counter = 0;
        while (true)
        {
                int r = read(temp_extra, buffer, 1024);
                if(r<=0)
                        break;
                if(fwrite(buffer, r, 1, finalPatch) != 1) err(1, "fwrite(%s) Diff write
error %s", "final diff");
                fflush(finalPatch);
                counter++;
        }
        printf("\Extra 1024 counter = %ld", counter);

        free(buffer);
        fclose(tempDiff);
        fclose(tempExtra);

        long finalPatchSize;
        if((fseek(finalPatch, 0, SEEK_SET)) )  err(1, "fwrite(%s)", argv[3]);
        if(fwrite(header, 24, 1, finalPatch) != 1) err(1, "fwrite(%s)", argv[3]);
        if((fseek(finalPatch, 0, SEEK_END)) )  err(1, "fwrite(%s)", argv[3]);
        if((finalPatchSize = ftell(finalPatch)) == -1) err(1, "ftello");
        fclose(finalPatch);
        printf("\n finalPatchSize: %d", finalPatchSize);
        for(long newBlockIndex = 0; newBlockIndex<numberOfNewBlocks; newBlockIndex++)
        {
                printf(" \n New %ld matches %ld old block", newBlockIndex+1,
bestMatchingOldBlockIndex[newBlockIndex]);
        }
        printf("\n Enter any char to exit...");
        int ch;
        scanf("%d", &ch);
}
```