**Exploiting Content Caching and Delivery Techniques for 5G Systems**

Prepared for:  Prof. Myung J. Lee

Prepared by: David Ungar

Rene Midouin

Department of Electrical Engineering

City College of New York

May 22nd, 2017

Table of Contents

[List of Figures iii](#_Toc483255936)

[Abstract 4](#_Toc483255937)

[1. Significance of the Problem 4](#_Toc483255938)

[2. Existing Approaches 5](#_Toc483255939)

[3. Contribution 6](#_Toc483255940)

[4. Results Achieved 10](#_Toc483255941)

[Conclusion 13](#_Toc483255942)

[References 14](#_Toc483255943)

[Appendix A 15](#_Toc483255944)

[Flow Chart of Simulation Code 15](#_Toc483255945)

[Simulation Code 16](#_Toc483255946)

# List of Figures

[Figure 1. Environment Architecture 7](#_Toc483255954)

[Figure 2. Data Structures 8](#_Toc483255955)

[Figure 3. Delay vs. Relative Cache Size. No caching - EPC+RAN - EPC+RAN+D2D 12](#_Toc483255956)

[Figure 4. Traffic vs. Relative Cache Size. No caching - EPC+RAN - EPC+RAN+D2D 12](#_Toc483255957)

[Figure 5. Flow Chart of Simulation Code 15](#_Toc483255958)

# Abstract

In the world of 5G, Extreme Mobile Broadband (xMBB) will be a reality, and the required data rates will skyrocket. Of particular note will be video content, as network users become increasingly accustomed to consuming massive amounts of ever higher quality audio-visual data. This creates a particularly tough challenge, as each user will be accessing the content of his or her own selection, leading to a situation where very large amounts of very personalized data are being sent to each specific user. This will create a tremendous strain on the network, and factors like latency will suffer substantially.

For this project, we analyzed various caching architectures and techniques. These include where to cache, like Edge Only and Edge + Cloud, as well as algorithms for what to cache, such as Least Recently Used (LRU) and Least Frequently Used (LFU). We coded in Java a computer simulation to generate the performance analysis, incorporating some of the mathematical models found in the attached papers. We built upon those models, and optimized them by adding the component of caching with D2D technology. The results show a clear advantage in the latency when caching with D2D is incorporated versus caching without D2D. We explored how significant of an advantage D2D provides when superimposed with caching, as well as the complexities involved in the actual real world implementation of D2D caching, addressing such issues as specific caching policies and algorithms.

# Significance of the Problem

When it comes to 5G, one of the most prominent issues is due to the centralized architecture of current cellular network where the wireless link capacity as well as the bandwidth of the radio access networks and the backhaul network cannot practically cope with the explosive growth in mobile traffic. Although caching contents at the Evolved Packet Core (EPC), the Radio Access Networks (RAN) and on User Equipment (UE) will help solve the problem and relax the strain on the network, without a good and robust caching policy, high performance and minimal latency is not guaranteed. By 2020, 50 billion things are expected to be connected to the internet. Multiple users will be requesting the same content that the network might not be able to cope with. Without D2D caching and good caching policy, 5G won't be as great and we will be doomed for we won't be able to watch our lovely cat videos.

# Existing Approaches

Rest assured, many solutions have been proposed to help solve that issue and to ensure that latency is at its lowest point. But when it comes to caching, all technical issues fall in the following three questions:

* **Where to cache?**
  + Evolve Packet Core (EPC)
    - URL Web-Based Web Caching
    - Prefix-Based Web Caching
    - Chunk-Level Redundancy Elimination
    - TCP-Level Redundancy Elimination
    - Packet-Level Redundancy Elimination
  + Radio Access Network (RAN)
    - Byte-caching
* **What to cache?**
  + Popular contents
    - Cat videos



* **How to cache?**
  + Least Frequently Used (LFU)
  + Least Recently Used (LRU)
  + First in First out (FIFO)

So, the idea is simply to cache at as many places as possible (EPC and RAN) where in our contribution we also highligthed caching with D2D, to cache mainly popular contents (what else should you cache besides cat videos. No ... seriously ...), and to use robust algorithms (LFU, LRU, FIFO) to determine which content you should bounce out first once one of your cache storage is full.

Based on the existing approaches, we quickly spotted room for improvement by showing how caching with D2D + EPC + RAN greatly improves latency compared to caching with EPC + RAN. Most importantly, good citizens that we are, we even tried our best to come up with our own caching policy, as we all know a bad caching policy is the essence of madness and drama.

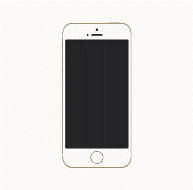
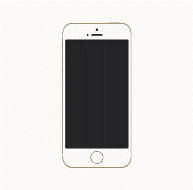
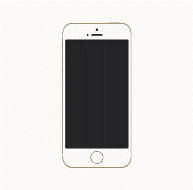
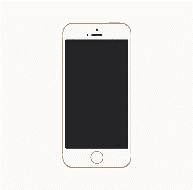
# Contribution

Not only will we present the obvious advantage of D2D caching in figure 3 and figure 4 (pg. 11), we also show the logic behind our caching policy. The caching policy that was implemented is a modified version of the Least Recently Used (LRU) algorithm. We explain thoroughly our contribution in the paragraphs below.

We begin with a set of diagrams that illustrate clearly the workings of the simulation. Figure 1 shows the architecture of the environment that was used for our simulation while figure 2 demonstrates how we leveraged Java's data structures to write optimal code. Following this we thoroughly explain our simulation, in the form of a detailed outline of the code flow and logic. This outline is followed by an example of an output of the running of a simulation, and finally those results are plotted in graph form.

RAN

INTERNET



**A**

**C**

**B**

EPC

20ms

20ms

5-10ms

Figure 1. Environment Architecture

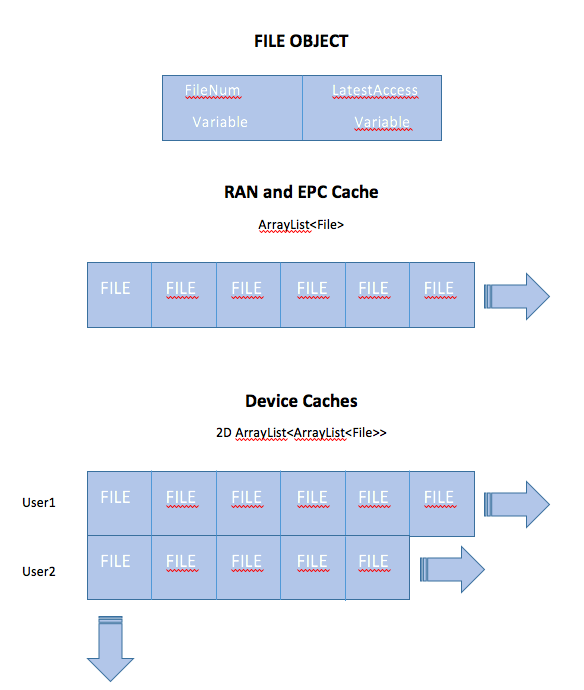


Figure 2. Data Structures

**Program Outline**

Note: When talking about CASE A, CASE B, and CASE C, we are referring to the corresponding functions from our simulation code in Appendix A.

**CASE A**

No caching.

**CASE B**

(Within the inner loop- this is done for every user hundreds of times):

-Choose file to request using random number generator

-First check if the requested file is in the user device itself

-If yes, no delay recorded. Update the time of latest access for the file.

-If not, check in RAN

-If in RAN, update the latest access and total delay variable accordingly (5-10ms)

-If not, check in EPC

-If in EPC, update the latest access and total delay variable accordingly (25-30ms)

-If not, must get from internet. Update delay (45-50ms) and increment “traffic”

-Since not found, must cache: First look for an empty spot (Starting with closest first:

Device itself, then RAN, then EPC). If empty spot found, cache the file there.

- If no empty spot found, use LRU (Least Recently Used) algorithm to decide which file to kick out of cache to make room for this one. Search through all the caches, device, RAN, and EPC, to find the one that has the most recent access the furthest in the past. This one is deleted and replaced with the current file.

**CASE C** (The differences over B are in bold)

(Within the inner loop- this is done for every user hundreds of times):

-Choose file to request using random number generator

-First check if the requested file is in the user device itself

-If yes, no delay recorded. Update the time of latest access for the file.

**-If not, check all the other user devices**

**-If found, update the total delay variable accordingly (5-10ms using random generator)**

**and latest access variable.**

-If not in device cache, check in RAN

-If in RAN, update the latest access and total delay variable accordingly (5-10ms)

-If not, check in EPC

-If in EPC, update the latest access and total delay variable accordingly (25-30ms)

-If not, must get from internet. Update delay (45-50ms) and increment “traffic”

-Since not found, must cache: First look for an empty spot (Starting with closest first: **devices**, then RAN, then EPC). If empty spot found, cache the file there.

-If no empty spot found, use LRU (Least Recently Used) algorithm to decide which file to kick out of cache to make room for this one. Search through all the caches, device, RAN, and EPC, to find the one that has the most recent access the furthest in the past. This one is deleted and replaced with the current file.

# Results Achieved

C:\Users\mdbmd\Desktop\Spring 17\Advanced Wireless\Project>java Cache

Case A has average delay of 47.48568 ms and traffic load of 1.0

Case B with max cache of 300 out of 3000 total files has average delay of 44.280

8 ms and traffic load of 0.8944

Case B with max cache of 600 out of 3000 total files has average delay of 41.320

28 ms and traffic load of 0.79636

Case B with max cache of 900 out of 3000 total files has average delay of 38.447

96 ms and traffic load of 0.70228

Case B with max cache of 1200 out of 3000 total files has average delay of 35.78

368 ms and traffic load of 0.61188

Case B with max cache of 1500 out of 3000 total files has average delay of 32.92

8 ms and traffic load of 0.51848

Case B with max cache of 1800 out of 3000 total files has average delay of 30.16

62 ms and traffic load of 0.42984

Case B with max cache of 3000 out of 3000 total files has average delay of 20.30

372 ms and traffic load of 0.12196

Case C with max cache of 300 out of 3000 total files has average delay of 44.129

96 ms and traffic load of 0.90014

Case C with max cache of 600 out of 3000 total files has average delay of 40.839

95 ms and traffic load of 0.80082

Case C with max cache of 900 out of 3000 total files has average delay of 37.528

98 ms and traffic load of 0.70134

Case C with max cache of 1200 out of 3000 total files has average delay of 34.21

32 ms and traffic load of 0.602235

Case C with max cache of 1500 out of 3000 total files has average delay of 30.90

7512 ms and traffic load of 0.502728

Case C with max cache of 1800 out of 3000 total files has average delay of 27.56

8886666666668 ms and traffic load of 0.40323

Case C with max cache of 3000 out of 3000 total files has average delay of 14.30

878 ms and traffic load of 0.006

**Figure 3.** Delay vs. Relative Cache Size.1) No caching 2) EPC+RAN 3) EPC+RAN+D2D

Figure 4. Traffic vs. Relative Cache Size. 1) No caching 2) EPC+RAN 3) EPC+RAN+D2D

# Conclusion

In conclusion, this project was eye-opening in many senses. Obviously, it showed the power of a simple idea and how it can have a profound impact on the performance of a network if implemented properly. More than that, since we opted to create our own simulation from scratch, we quickly discovered how even a simple concept has untold intricacies and complexities that have to be worked out and decided upon. Most of all, we felt truly excited to be working creatively in a cutting edge field upon which the future will be built.

# References

[1]  I. Cisco, “Cisco visual networking index: Forecast and methodology, 2011–2016,” *CISCO*

*White paper*, pp. 2011–2016, 2012.

[2]  D. Pompili, A. Hajisami, and T. X. Tran, “Elastic resource utilization framework for high

capacity and energy efficiency in Cloud RAN,” *Communications Magazine, IEEE*, vol. 54, no. 1, pp. 26–32, 2016.

[3]  E. Bastug, M. Bennis, and M. Debbah, “Living on the edge: The role of proactive caching in

5G wireless networks,” *IEEE Commun. Mag.*, vol. 52, no. 8, pp. 82–89, 2014.

# Appendix A

## Flow Chart of Simulation Code

Request a Random File

Check if the File is in Cache

-Access -Update Delay, etc.

-Access -Update -Cache (LRU)

YES

NO

Figure 5. Flow Chart of Simulation Code

## Simulation Code

**Java Code**

%=========================================================================%

% @author: David Ungar, Rene Midouin

% @date: May 17, 2017

% @title: Exploiting Content Caching and Delivery Techniques for 5G Systems

% @subject: Advanced Wireless Communication -- Project

%=========================================================================%

//I7500 Advanced Wireless

//D2D Caching Simulator

//David Ungar and Rene Midouin

import java.util.Random;

import java.util.ArrayList;

import java.io.PrintWriter;

public class Cache{

public static void main(String[] args){

CaseA();

CaseB(MAXB1);

CaseB(MAXB2);

CaseB(MAXB3);

CaseB(MAXB4);

CaseB(MAXB5);

CaseB(MAXB6);

CaseB(MAXB10);

CaseC(MAXC1);

CaseC(MAXC2);

CaseC(MAXC3);

CaseC(MAXC4);

CaseC(MAXC5);

CaseC(MAXC6);

CaseC(MAXC10);

}

//No Caching whatsoever.

public static void CaseA(){

int NUM\_USERS = 5;

Random rand = new Random();

int time = 0;

int delaySoFar = 0;

int traffic = 0;

//Simulate repeated random file requests from all the users by looping through all the users hundredes of times and having

//each request a random file. After each request check if the file is cached somewhere and record the delay and traffic load accordingly.

for (int i = 0; i < NUM\_SIMS; i++){

for (int j = 0; j < NUM\_USERS; j++){

time++;

int fileRequest = rand.nextInt(NUM\_FILES);

delaySoFar += (rand.nextInt(6) + 45);//delay btw 45-50 ms

traffic++;

}

}

double avgDelay = (double)delaySoFar/(NUM\_SIMS\*NUM\_USERS);

double trafficLoad = (double)traffic/(NUM\_SIMS\*NUM\_USERS);

System.out.println("Case A has average delay of " + avgDelay + " ms and traffic load of " + trafficLoad);

}

//Caching everywhere (EPC, RAN and user device) without D2D. Takes as argument the max # of files to be cached per node.

public static void CaseB(int Max){

int NUM\_USERS = 5;

//ArrayList of File Objects that are cached at the EPC

ArrayList<File> EPC = new ArrayList<File>();

//ArrayList of File Objects that are cached at the RAN

ArrayList<File> RAN = new ArrayList<File>();

//Create a 2D ArrayList to hold an ArrayList of Files cached at every user

ArrayList<ArrayList<File>> userArr = new ArrayList<ArrayList<File>>(NUM\_USERS);

for(int a = 0; a < NUM\_USERS; a++){

userArr.add(new ArrayList<File>());

}

Random rand = new Random();

int time = 0;

int delaySoFar = 0;

int traffic = 0;

//Simulate repeated random file requests from all the users by looping through all the users hundredes of times and having

//each one request a random file. After each request check if the file is cached somewhere and record the delay and traffic load accordingly.

for (int i = 0; i < NUM\_SIMS; i++){

for (int j = 0; j < NUM\_USERS; j++){

time++;

int fileRequest = rand.nextInt(NUM\_FILES);

int foundAt = -1;

//First check if the file is in the user device's cache

for (int k = 0; k < userArr.get(j).size(); k++){

if (userArr.get(j).get(k).getFileNum() == fileRequest){

foundAt = k;

break;

}

}

//If it is found in device cache...

if (foundAt != -1){

userArr.get(j).get(foundAt).setTime(time);//update the latest access of this file

}

//If it is not in device cache...

else{

//Check if the file is in the RAN cache

for (int k = 0; k < RAN.size(); k++){

if (RAN.get(k).getFileNum() == fileRequest){

foundAt = k;

break;

}

}

//If it is found in RAN cache...

if (foundAt != -1){

delaySoFar += (rand.nextInt(6) + 5);//between 5-10 ms of delay

RAN.get(foundAt).setTime(time);//update the latest access of this file

}

//If not in RAN cache either...

else{

//Check if the file is in the EPC cache

for (int k = 0; k < EPC.size(); k++){

if (EPC.get(k).getFileNum() == fileRequest){

foundAt = k;

break;

}

}

//If it is found in EPC cache...

if (foundAt != -1){

delaySoFar += (rand.nextInt(6) + 25);//between 25-30 ms of delay

EPC.get(foundAt).setTime(time);//update the latest access of this file

}

//If not in EPC either, it is not in cache at all

else{

traffic++;//Had to access from core

delaySoFar += (rand.nextInt(6) + 45);//between 45-50 ms of delay

//Since not cached, let's cache it! But where?

if(userArr.get(j).size() < MAX\_USER){//user device memory not full

userArr.get(j).add(new File(fileRequest, time));

}

else if(RAN.size() < Max){//RAN memory not full

RAN.add(new File(fileRequest, time));

}

else if(EPC.size() < Max){//EPC memory not full

EPC.add(new File(fileRequest, time));

}

else{//No empty slots so must use LRU to replace a cached file

int lruDevice = time + 1;

int deviceIndex = -1;

int lruRAN = time + 1;

int RANIndex = -1;

int lruEPC = time + 1;

int EPCIndex = -1;

for (int k = 0; k < userArr.get(j).size(); k++){//Find the lru in device cache

if (userArr.get(j).get(k).getTime() < lruDevice){

lruDevice = userArr.get(j).get(k).getTime();

deviceIndex = k;

}

}

for (int k = 0; k < RAN.size(); k++){//Find the lru in RAN cache

if (RAN.get(k).getTime() < lruRAN){

lruRAN = RAN.get(k).getTime();

RANIndex = k;

}

}

for (int k = 0; k < EPC.size(); k++){//Find the lru in EPC cache

if (EPC.get(k).getTime() < lruEPC){

lruEPC = EPC.get(k).getTime();

EPCIndex = k;

}

}

if (lruDevice < lruRAN && lruDevice < lruEPC){//device cache had the lru

userArr.get(j).remove(deviceIndex);

userArr.get(j).add(new File(fileRequest, time));

}

else if(lruRAN < lruDevice && lruRAN < lruEPC){//RAN had the lru

RAN.remove(RANIndex);

RAN.add(new File(fileRequest, time));

}

else{//EPC had the lru

EPC.remove(EPCIndex);

EPC.add(new File(fileRequest, time));

}

}

}

}

}

}

}

double avgDelay = (double)delaySoFar/(NUM\_SIMS\*NUM\_USERS);

double trafficLoad = (double)traffic/(NUM\_SIMS\*NUM\_USERS);

System.out.println("Case B with max cache of " + Max\*2 + " out of " + NUM\_FILES + " total files has average delay of " + avgDelay + " ms and traffic load of " + trafficLoad);

}

//Caching with D2D

public static void CaseC(int Max){

int NUM\_USERS = Max/10;

//ArrayList of File Objects that are cached at the EPC

ArrayList<File> EPC = new ArrayList<File>();

//ArrayList of File Objects that are cached at the RAN

ArrayList<File> RAN = new ArrayList<File>();

//Create a 2D ArrayList to hold an ArrayList of Files cached at every user

ArrayList<ArrayList<File>> userArr = new ArrayList<ArrayList<File>>(NUM\_USERS);

for(int a = 0; a < NUM\_USERS; a++){

userArr.add(new ArrayList<File>());

}

Random rand = new Random();

int time = 0;

int delaySoFar = 0;

int traffic = 0;

//Simulate repeated random file requests from all the users by looping through all the users hundredes of times and having

//each one request a random file. After each request check if the file is cached somewhere and record the delay and traffic load accordingly.

for (int i = 0; i < NUM\_SIMS; i++){

for (int j = 0; j < NUM\_USERS; j++){

time++;

int fileRequest = rand.nextInt(NUM\_FILES);

int foundAt = -1;

int userIndex = -1;

//First check if the file is in the device caches

for(int l = 0; l < userArr.size(); l++){

for (int k = 0; k < userArr.get(l).size(); k++){

if (userArr.get(l).get(k).getFileNum() == fileRequest){

foundAt = k;

userIndex = l;

//break;

}

}

}

//If it is found in device cache...

if (foundAt != -1){

userArr.get(userIndex).get(foundAt).setTime(time);//update the latest access of this file

if(userIndex != j){//add delay only if accessing another device's cache

delaySoFar += (rand.nextInt(6) + 5);//between 5-10 ms of delay

}

}

//If it is not in device caches...

else{

//Check if the file is in the RAN cache

for (int k = 0; k < RAN.size(); k++){

if (RAN.get(k).getFileNum() == fileRequest){

foundAt = k;

break;

}

}

//If it is found in RAN cache...

if (foundAt != -1){

delaySoFar += (rand.nextInt(6) + 5);//between 5-10 ms of delay

RAN.get(foundAt).setTime(time);//update the latest access of this file

}

//If not in RAN cache either...

else{

//Check if the file is in the EPC cache

for (int k = 0; k < EPC.size(); k++){

if (EPC.get(k).getFileNum() == fileRequest){

foundAt = k;

break;

}

}

//If it is found in EPC cache...

if (foundAt != -1){

delaySoFar += (rand.nextInt(6) + 25);//between 25-30 ms of delay

EPC.get(foundAt).setTime(time);//update the latest access of this file

}

//If not in EPC either, it is not in cache at all

else{

traffic++;//Had to access from core

delaySoFar += (rand.nextInt(6) + 45);//between 45-50 ms of delay

//Since not cached, let's cache it! But where?

int notFullUser = -1;

//Let's check if there's room in a device

for(int x = 0; x < userArr.size(); x++){

if(userArr.get(x).size() < MAX\_USER){

notFullUser = x;

}

}

if(notFullUser != -1){//device memory not full

userArr.get(notFullUser).add(new File(fileRequest, time));

}

else if(RAN.size() < Max){//RAN memory not full

RAN.add(new File(fileRequest, time));

}

else if(EPC.size() < Max){//EPC memory not full

EPC.add(new File(fileRequest, time));

}

else{//No empty slots so must use LRU to replace a cached file

int lruDevice = time + 1;

int whichUser = -1;

int deviceIndex = -1;

int lruRAN = time + 1;

int RANIndex = -1;

int lruEPC = time + 1;

int EPCIndex = -1;

for(int x = 0; x < userArr.size(); x++){//Find the lru in all device caches

for (int k = 0; k < userArr.get(x).size(); k++){

if (userArr.get(x).get(k).getTime() < lruDevice){

lruDevice = userArr.get(x).get(k).getTime();

whichUser = x;

deviceIndex = k;

}

}

}

for (int k = 0; k < RAN.size(); k++){//Find the lru in RAN cache

if (RAN.get(k).getTime() < lruRAN){

lruRAN = RAN.get(k).getTime();

RANIndex = k;

}

}

for (int k = 0; k < EPC.size(); k++){//Find the lru in EPC cache

if (EPC.get(k).getTime() < lruEPC){

lruEPC = EPC.get(k).getTime();

EPCIndex = k;

}

}

if (lruDevice < lruRAN && lruDevice < lruEPC){//device cache had the lru

userArr.get(whichUser).remove(deviceIndex);

userArr.get(whichUser).add(new File(fileRequest, time));

}

else if(lruRAN < lruDevice && lruRAN < lruEPC){//RAN had the lru

RAN.remove(RANIndex);

RAN.add(new File(fileRequest, time));

}

else{//EPC had the lru

EPC.remove(EPCIndex);

EPC.add(new File(fileRequest, time));

}

}

}

}

}

}

}

double avgDelay = (double)delaySoFar/(NUM\_SIMS\*NUM\_USERS);

double trafficLoad = (double)traffic/(NUM\_SIMS\*NUM\_USERS);

System.out.println("Case C with max cache of " + Max\*3 + " out of " + NUM\_FILES + " total files has average delay of " + avgDelay + " ms and traffic load of " + trafficLoad);

}

//Object to represent a file in cache

//Contains the file name and the latest time it was accessed

public static class File{

private int fileNum;

private int latestAccess;

//Constructor

public File(int fileNum, int time){

this.fileNum = fileNum;

latestAccess = time;

}

//Get the file number of this object

public int getFileNum(){

return fileNum;

}

//Get the most recent access time of this File Object

public int getTime(){

return latestAccess;

}

//Set (update) the most recent access time

public void setTime(int newTime){

latestAccess = newTime;

}

}

//Set the parameters for the simulation

public static final int NUM\_FILES = 3000;

public static final int NUM\_SIMS = 5000;

public static final int MAX\_USER = 10;

//Max files to be cached per node

//Simulations 1-6 represent cache sizes between 10 and 60 percent of the total # of files

public static final int MAXB1 = (NUM\_FILES/10)/2;//10% of the total files, evenly split between EPC and RAN

public static final int MAXC1 = (NUM\_FILES/10)/3;//10% of the total files, evenly split between EPC and RAN and user devices

public static final int MAXB2 = ((NUM\_FILES\*2)/10)/2;//20% of the total files, evenly split between EPC and RAN

public static final int MAXC2 = ((NUM\_FILES\*2)/10)/3;

public static final int MAXB3 = ((NUM\_FILES\*3)/10)/2;//30% of the total files, evenly split between EPC and RAN

public static final int MAXC3 = ((NUM\_FILES\*3)/10)/3;

public static final int MAXB4 = ((NUM\_FILES\*4)/10)/2;//40% of the total files, evenly split between EPC and RAN

public static final int MAXC4 = ((NUM\_FILES\*4)/10)/3;

public static final int MAXB5 = ((NUM\_FILES\*5)/10)/2;//50% of the total files, evenly split between EPC and RAN

public static final int MAXC5 = ((NUM\_FILES\*5)/10)/3;

public static final int MAXB6 = ((NUM\_FILES\*6)/10)/2;//60% of the total files, evenly split between EPC and RAN

public static final int MAXC6 = ((NUM\_FILES\*6)/10)/3;

public static final int MAXB10 = NUM\_FILES/2;//100% of the total files, evenly split between EPC and RAN

public static final int MAXC10 = NUM\_FILES/3;

}