Nikhil

BCSE3 A2

ROLL 001810501069

ASSIGNMENT 3

**TITLE**

In this assignment, you have to implement p-persistent CSMA with exponential backoff and

additive backoff. Measure the performance parameters like throughput (i.e., average amount

of data bits successfully transmitted per unit time) and forwarding delay (i.e., average end-to--

end delay, including the queuing delay and the transmission delay) experienced by the CSMA

frames (IEEE 802.3). Plot the comparison graphs for throughput and forwarding delay by

varying p. State your observations on the impact of different data rates for exponential/additive

backoff along with p-persistent CSMA

**DESIGN**

The main purpose of the assignment to understand and then implement the working multiple Access protocols which are used in networking.

My design of is as follow :

1. **Channel Class** – It represent the type of scheme we are using to simulate the process. Hence there are channel classes named –
2. OONECHANNEL to simulate the ONEPERSISTENT MODEL.
3. PCHANNEL to simulate the PPERISTENT MODEL.
4. NPCHANNEL to simulate the NON PERSISTENT MODEL.
5. **Persistent Classes** – It represent a particular type of station in the system. Hence they are also 3 in number.
6. ONEPERSISTENT this station will use ONEPERSISTENT technique to send the data.
7. NONPERSISTENT this station will use NONPERSISTENT technique to send the data.
8. PPERSISTENT this station will use PPERSISTENT technique to send the data.

**Main**

Here we have assumed that we have 5 station which will be sending data through the channel. With the logic we have implemented.

Each station will try to send the frame. But before sending the frame. Every station will sense the channel for its availability. Now Since we have 3 technique next step depends on the mechanism used.

In case of

One Persistent – If channel is free. It will send it immediately. Otherwise it will start sensing the channel. Till the time channel gets free and when channel is free it will send the frame

Station 5

Station 4

Station 3

Station 2

Station 1



Thread 1

Thread 4

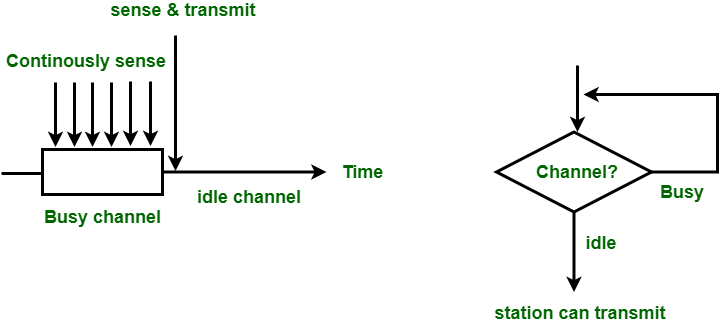
Thread 5

Thread 2

Thread 3

**IMPLEMENTATION/Code**

**One Persistent**

****

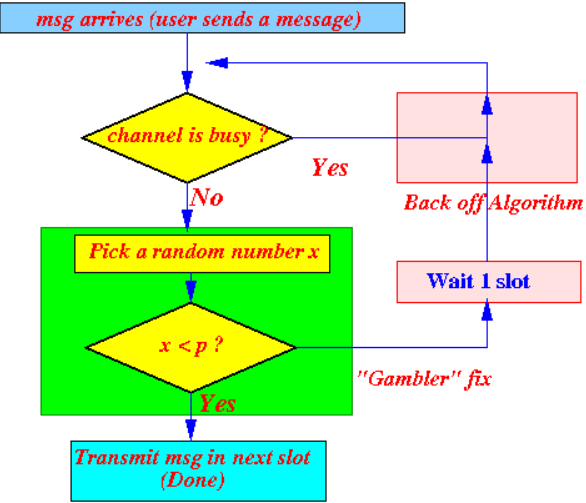
*package* onepersistent;  
  
*import* utilclasses.*ChannelConstants*;  
  
*import* java.io.FileWriter;  
*import* java.io.IOException;  
*import* java.util.Random;  
*import* java.util.concurrent.atomic.\*;  
  
*class* OneChannel *implements Runnable*, *ChannelConstants* {  
 String stationNumber;  
 Thread t;  
 *static int distance*;  
 *static int stat*=0;  
 *static int frame*;  
 *static int channelStatus*; *//Indicates if channel is being used  
 int* currentFrameNumber;  
 *int* maxFrameNumber;  
 *private final* AtomicBoolean checkIfSuccessfulTransmission;  
 *static int tfr*=50; *//Transmission time  
 private int* numberOfAttempts;  
 Random rand ;  
 *private static final* Object *distanceLock* = *new* Object();  
 *private static final* Object *channelUpdateLock* = *new* Object();  
 *private static final* Object *getStatLock* = *new* Object();  
 FileWriter fileWriter ;  
  
 OneChannel(String threadName, *int* maxFrameNumber, FileWriter fileWriter) {  
 stationNumber = threadName;  
 t = *new* Thread(*this*, stationNumber);  
 currentFrameNumber = 1;  
 *this*.maxFrameNumber =maxFrameNumber;  
 checkIfSuccessfulTransmission = *new* AtomicBoolean();  
 *this*.fileWriter = fileWriter;  
 t.start();  
 }  
 *public void* run() {  
 rand = *new* Random();  
 *while* (!checkIfSuccessfulTransmission.get()) {  
 numberOfAttempts++;  
 *while*(currentFrameNumber <= maxFrameNumber) {  
 OnePersistent.*updateTotalCount*();  
 *if* (numberOfAttempts < 15) { *//15 is the maximum number of attempts  
 try* {  
 *if* (*channelStatus* == *INUSE*) {  
 System.*out*.println(stationNumber + " is using 1 Persistent sensing, channel is busy");  
 *synchronized* (*this*){  
 *try* {  
 wait(); *//\*\*\*Waits forever.\*\*\** }  
 *catch* (InterruptedException e) {  
 System.*out*.println(("Interrupt"));  
 Thread.*currentThread*().interrupt();  
 System.*exit*(1);  
 }  
 }  
  
 }  
 *else* {  
 System.*out*.println(stationNumber + " is trying to transmit frame number : " + currentFrameNumber);  
  
 *if* (*channelStatus* == *FREE* && *distance* == 0) {*//Successful transmission  
 synchronized* (*getStatLock*){  
 *stat* = OneChannelThreads.*checking*(Thread.*currentThread*().getName());  
 *if*(*stat* == -1){  
 System.*out*.println("Error Setting up stat.");  
 System.*exit*(1);  
 }  
 *frame* = *this*.currentFrameNumber;  
 }  
 *synchronized* (*channelUpdateLock*){  
 *channelStatus* = *INUSE*;  
 }  
 *long* start = System.*nanoTime*();  
 *synchronized* (*distanceLock*){  
 *distanceLock*.wait(1000);  
 }  
 *long* end = System.*nanoTime*();  
 *synchronized* (*this*){  
 OnePersistent.*forwarding\_delay* += end-start;  
 }  
 OnePersistent.*updateDoneCount*();  
 fileWriter.write(stationNumber + " frame " + currentFrameNumber + " is successful.\n");  
 fileWriter.flush();  
 System.*out*.println(stationNumber + " frame " + currentFrameNumber + " is successful.");  
 checkIfSuccessfulTransmission.set(*true*);  
 currentFrameNumber++;  
 *synchronized* (*distanceLock*){  
 *distance* = 0;  
 }  
 *synchronized* (*channelUpdateLock*){  
 *channelStatus* = *FREE*;  
 }  
 *synchronized* (*this*){  
 *try*{  
 notifyAll(); */// \*\*\*Notify All doesnt works.\*\*\** } *catch* (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 *else* {*//Collision has occurred* System.*out*.println("Collision for frame " + currentFrameNumber + " of " +  
 stationNumber + " and frame " + *frame* + " of Station " + *stat*);  
  
 System.*out*.println("Retransmitting Station " + *stat* + "'s frame " + *frame*);  
 checkIfSuccessfulTransmission.set(*false*);  
 *synchronized* (*channelUpdateLock*){  
 *channelStatus* = *FREE*;  
 }  
  
 numberOfAttempts++;  
 */\*try {  
 int anInt = rand.nextInt((int) (Math.pow(2,(double) numberOfAttempts - 1)));  
 int backOffTime = anInt \* tfr;  
 Thread.sleep(backOffTime);  
  
 } catch (InterruptedException e) {  
 System.out.println("Interrupted");  
 Thread.currentThread().interrupt();  
 }\*/* }  
 *// Thread.sleep(1000);* }  
 } *catch* (InterruptedException | IOException e) {  
 System.*out*.println(stationNumber + "Main Interrupted");  
 Thread.*currentThread*().interrupt();  
 }  
 }  
 *else* {  
 checkIfSuccessfulTransmission.set(*true*);  
 System.*out*.println("Too many attempts for frame " + currentFrameNumber + "of " +  
 stationNumber + ". Transmission stopped");  
 }  
  
 }  
 }  
 }  
}

**UTILITY**

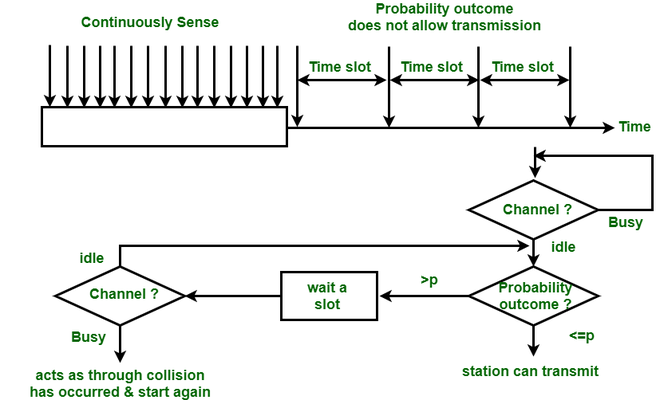
*package* onepersistent;  
  
*import* utilclasses.*ChannelConstants*;  
  
*/\*Class to check which station's frame the current transmission is colliding with.  
  
 \*/  
class* OneChannelThreads *implements ChannelConstants* {  
 *public static int* checking(String StationName)  
 { *int* stat;  
 *char* c = StationName.charAt(8);  
 stat = Integer.*parseInt*(c+"");  
 *return*(stat);  
 }  
}

ONE CHANNEL

*package* onepersistent;  
  
*import* utilclasses.*ChannelConstants*;  
  
*import* java.io.FileWriter;  
*import* java.io.IOException;  
*import* java.util.Random;  
*import* java.util.concurrent.atomic.\*;  
  
*class* OneChannel *implements Runnable*, *ChannelConstants* {  
 String stationNumber;  
 Thread t;  
 *static int distance*;  
 *static int stat*=0;  
 *static int frame*;  
 *static int channelStatus*; *//Indicates if channel is being used  
 int* currentFrameNumber;  
 *int* maxFrameNumber;  
 *private final* AtomicBoolean checkIfSuccessfulTransmission;  
 *static int tfr*=50; *//Transmission time  
 private int* numberOfAttempts;  
 Random rand ;  
 *private static final* Object *distanceLock* = *new* Object();  
 *private static final* Object *channelUpdateLock* = *new* Object();  
 *private static final* Object *getStatLock* = *new* Object();  
 FileWriter fileWriter ;  
  
 OneChannel(String threadName, *int* maxFrameNumber, FileWriter fileWriter) {  
 stationNumber = threadName;  
 t = *new* Thread(*this*, stationNumber);  
 currentFrameNumber = 1;  
 *this*.maxFrameNumber =maxFrameNumber;  
 checkIfSuccessfulTransmission = *new* AtomicBoolean();  
 *this*.fileWriter = fileWriter;  
 t.start();  
 }  
 *public void* run() {  
 rand = *new* Random();  
 *while* (!checkIfSuccessfulTransmission.get()) {  
 numberOfAttempts++;  
 *while*(currentFrameNumber <= maxFrameNumber) {  
 OnePersistent.*updateTotalCount*();  
 *if* (numberOfAttempts < 15) { *//15 is the maximum number of attempts  
 try* {  
 *if* (*channelStatus* == *INUSE*) {  
 System.*out*.println(stationNumber + " is using 1 Persistent sensing, channel is busy");  
 *synchronized* (*this*){  
 *try* {  
 wait(); *//\*\*\*Waits forever.\*\*\** }  
 *catch* (InterruptedException e) {  
 System.*out*.println(("Interrupt"));  
 Thread.*currentThread*().interrupt();  
 System.*exit*(1);  
 }  
 }  
  
 }  
 *else* {  
 System.*out*.println(stationNumber + " is trying to transmit frame number : " + currentFrameNumber);  
  
 *if* (*channelStatus* == *FREE* && *distance* == 0) {*//Successful transmission  
 synchronized* (*getStatLock*){  
 *stat* = OneChannelThreads.*checking*(Thread.*currentThread*().getName());  
 *if*(*stat* == -1){  
 System.*out*.println("Error Setting up stat.");  
 System.*exit*(1);  
 }  
 *frame* = *this*.currentFrameNumber;  
 }  
 *synchronized* (*channelUpdateLock*){  
 *channelStatus* = *INUSE*;  
 }  
 *long* start = System.*nanoTime*();  
 *synchronized* (*distanceLock*){  
 *distanceLock*.wait(1000);  
 }  
 *long* end = System.*nanoTime*();  
 *synchronized* (*this*){  
 OnePersistent.*forwarding\_delay* += end-start;  
 }  
 OnePersistent.*updateDoneCount*();  
 fileWriter.write(stationNumber + " frame " + currentFrameNumber + " is successful.\n");  
 fileWriter.flush();  
 System.*out*.println(stationNumber + " frame " + currentFrameNumber + " is successful.");  
 checkIfSuccessfulTransmission.set(*true*);  
 currentFrameNumber++;  
 *synchronized* (*distanceLock*){  
 *distance* = 0;  
 }  
 *synchronized* (*channelUpdateLock*){  
 *channelStatus* = *FREE*;  
 }  
 *synchronized* (*this*){  
 *try*{  
 notifyAll(); */// \*\*\*Notify All doesnt works.\*\*\** } *catch* (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 *else* {*//Collision has occurred* System.*out*.println("Collision for frame " + currentFrameNumber + " of " +  
 stationNumber + " and frame " + *frame* + " of Station " + *stat*);  
  
 System.*out*.println("Retransmitting Station " + *stat* + "'s frame " + *frame*);  
 checkIfSuccessfulTransmission.set(*false*);  
 *synchronized* (*channelUpdateLock*){  
 *channelStatus* = *FREE*;  
 }  
  
 numberOfAttempts++;  
 */\*try {  
 int anInt = rand.nextInt((int) (Math.pow(2,(double) numberOfAttempts - 1)));  
 int backOffTime = anInt \* tfr;  
 Thread.sleep(backOffTime);  
  
 } catch (InterruptedException e) {  
 System.out.println("Interrupted");  
 Thread.currentThread().interrupt();  
 }\*/* }  
 *// Thread.sleep(1000);* }  
 } *catch* (InterruptedException | IOException e) {  
 System.*out*.println(stationNumber + "Main Interrupted");  
 Thread.*currentThread*().interrupt();  
 }  
 }  
 *else* {  
 checkIfSuccessfulTransmission.set(*true*);  
 System.*out*.println("Too many attempts for frame " + currentFrameNumber + "of " +  
 stationNumber + ". Transmission stopped");  
 }  
  
 }  
 }  
 }  
}



**PPERSISTENT**

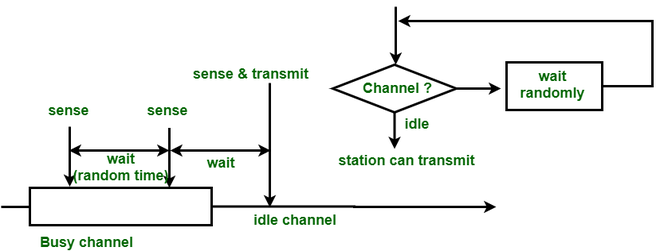
****

*package* ppersistent;  
  
*import* utilclasses.*ChannelConstants*;  
  
*import* java.io.FileWriter;  
*import* java.io.IOException;  
*import* java.util.Random;  
*import* java.util.concurrent.atomic.\*;  
  
*class* PChannel *implements Runnable*, *ChannelConstants* {  
 String stationNumber;  
 Thread t;  
 *static int distance*;  
 *static int stat*=0;  
 *static int frame*;  
 *static int channelStatus*; *//Indicates if channel is being used  
 int* currentFrameNumber;  
 *int* maxFrameNumber;  
 *private final* AtomicBoolean checkIfSuccessfulTransmission;  
 *static int tfr*=50; *//Transmission time  
 private int* numberOfAttempts;  
 Random rand ;  
 *private static final* Object *distanceLock* = *new* Object();  
 *private static final* Object *channelUpdateLock* = *new* Object();  
 *private static final* Object *getStatLock* = *new* Object();  
 FileWriter fileWriter ;  
 *private static final double prob* = 0.1;  
  
 PChannel(String threadName, *int* maxFrameNumber, FileWriter fileWriter) {  
 stationNumber = threadName;  
 t = *new* Thread(*this*, stationNumber);  
 currentFrameNumber = 1;  
 *this*.maxFrameNumber =maxFrameNumber;  
 checkIfSuccessfulTransmission = *new* AtomicBoolean();  
 *this*.fileWriter = fileWriter;  
 t.start();  
 }  
 *public void* run() {  
 rand = *new* Random();  
 *while* (!checkIfSuccessfulTransmission.get()) {  
 numberOfAttempts++;  
 *while*(currentFrameNumber <= maxFrameNumber) {  
 PPersistent.*updateTotalCount*();  
 *if* (numberOfAttempts < 15) { *//15 is the maximum number of attempts  
 try* {  
 *if* (*channelStatus* == *INUSE*) {  
 System.*out*.println(stationNumber + " is using 1 Persistent sensing, channel is busy");  
 *synchronized* (*this*){  
 *try* {  
 wait(); *//\*\*\*Waits forever.\*\*\** }  
 *catch* (InterruptedException e) {  
 System.*out*.println(("Interrupt"));  
 Thread.*currentThread*().interrupt();  
 System.*exit*(1);  
 }  
 }  
  
 }  
 *else* {  
 System.*out*.println(stationNumber + " is trying to transmit frame number : " + currentFrameNumber);  
  
 *if* (*channelStatus* == *FREE* && *distance* == 0) {*//Successful transmission  
 if*(rand.nextDouble() < *prob*){  
 *continue*;  
 }  
 *synchronized* (*getStatLock*){  
 *stat* = PChannelThreads.*checking*(Thread.*currentThread*().getName());  
 *if*(*stat* == -1){  
 System.*out*.println("Error Setting up stat.");  
 System.*exit*(1);  
 }  
 *frame* = *this*.currentFrameNumber;  
 }  
 *synchronized* (*channelUpdateLock*){  
 *channelStatus* = *INUSE*;  
 }  
 *long* start = System.*nanoTime*();  
 *synchronized* (*distanceLock*){  
 */\* for (; distance < 9000000; distance++)  
 for(int i =0;i<1000;i++){  
 //simulate transmission over some distance  
 }\*/  
 distanceLock*.wait(1000);  
 *//Thread.sleep(1000);* }  
 *long* end = System.*nanoTime*();  
 *synchronized* (*this*){  
 PPersistent.*forwarding\_delay* += end-start;  
 }  
 PPersistent.*updateDoneCount*();  
 fileWriter.write(stationNumber + " frame " + currentFrameNumber + " is successful.\n");  
 fileWriter.flush();  
 System.*out*.println(stationNumber + " frame " + currentFrameNumber + " is successful.");  
 checkIfSuccessfulTransmission.set(*true*);  
 currentFrameNumber++;  
 *synchronized* (*distanceLock*){  
 *distance* = 0;  
 }  
 *synchronized* (*channelUpdateLock*){  
 *channelStatus* = *FREE*;  
 }  
 *synchronized* (*this*){  
 *try*{  
 notifyAll(); */// \*\*\*Notify All doesnt works.\*\*\** } *catch* (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 *else* {*//Collision has occurred* System.*out*.println("Collision for frame " + currentFrameNumber + " of " +  
 stationNumber + " and frame " + *frame* + " of Station " + *stat*);  
  
 System.*out*.println("Retransmitting Station " + *stat* + "'s frame " + *frame*);  
 checkIfSuccessfulTransmission.set(*false*);  
 *synchronized* (*channelUpdateLock*){  
 *channelStatus* = *FREE*;  
 }  
  
 numberOfAttempts++;  
 */\*  
 try {  
 int anInt = rand.nextInt((int) (Math.pow(2,(double) numberOfAttempts - 1)));  
 int backOffTime = anInt \* tfr;  
 Thread.sleep(backOffTime);  
  
 } catch (InterruptedException e) {  
 System.out.println("Interrupted");  
 Thread.currentThread().interrupt();  
 }\*/* }  
 *// Thread.sleep(1000);* }  
 } *catch* (InterruptedException | IOException e) {  
 System.*out*.println(stationNumber + "Main Interrupted");  
 Thread.*currentThread*().interrupt();  
 }  
 }  
 *else* {  
 checkIfSuccessfulTransmission.set(*true*);  
 System.*out*.println("Too many attempts for frame " + currentFrameNumber + "of " +  
 stationNumber + ". Transmission stopped");  
 }  
  
 }  
 }  
 }  
}

*package* ppersistent;  
  
*import* utilclasses.ChannelConstants;  
  
*/\*Class to check which station's frame the current transmission is colliding with.  
  
 \*/  
public class* PChannelThreads *implements* ChannelConstants {  
 *public static int* checking(String StationName)  
 { *int* stat;  
 *char* c = StationName.charAt(8);  
 stat = Integer.parseInt(c+"");  
 *return*(stat);  
 }  
}

*package* ppersistent;  
  
*import* utilclasses.ChannelConstants;  
*import* java.io.FileWriter;  
*import* java.io.IOException;  
*import* java.util.Date;  
*import* java.util.Scanner;  
  
*class* PPersistent *implements* ChannelConstants {  
 *private static int* totalChannelAttempt = 0 ;  
 *private static int* totalDoneAttempt = 0 ;  
 *public static long* forwarding\_delay = 0 ;  
 *private static long* FRAME\_SIZE = 520 ;  
  
  
 *public static synchronized void* updateTotalCount(){  
 totalChannelAttempt++;  
 }  
 *public static synchronized void* updateDoneCount(){  
 totalDoneAttempt++;  
 }  
 *public static void* main(String[] args){  
 Scanner sc = *new* Scanner(System.in);  
 PChannel.channelStatus = FREE; *//initially channel is free* System.out.println("Enter number of stations");  
 *int* numberOfStations = sc.nextInt();  
 PChannel[] channels= *new* PChannel[numberOfStations+1];  
 *int*[] noOfFramesInChannel = *new int*[numberOfStations+1];  
 *for*(*int* i = 1;i<=numberOfStations;i++)  
 {  
 System.out.println("Enter number of frames for Station " + i);  
 noOfFramesInChannel[i] = sc.nextInt();  
  
 }  
 FileWriter fileWriter = *null*;  
 *try* {  
 fileWriter = *new* FileWriter("logs.txt",*true*);  
 fileWriter.write("Started Frame writing at "+*new* Date()+"\n");  
 fileWriter.flush();  
 } *catch* (IOException e) {  
 e.printStackTrace();  
 System.exit(1);  
 }  
 *long* start = System.nanoTime();  
 *for*(*int* i = 1;i<=numberOfStations;i++)  
 channels[i] = *new* PChannel("Station "+i,noOfFramesInChannel[i],fileWriter);  
  
 *try* {  
 *for*(*int* i=1;i<=numberOfStations;i++)  
 channels[i].t.join();  
 }  
 *catch* (InterruptedException e) {  
 System.out.println("Main Thread Interrupted");  
 Thread.currentThread().interrupt();  
 System.exit(1);  
 }  
 *long* end = System.nanoTime();  
 *long* totalExecution = end - start;  
 *int* count = 0 ;  
 *for*(*int* i = 1; i <=numberOfStations; ++i){  
 count += noOfFramesInChannel[i];  
 }  
 System.out.println("Time taken to send "+count+" frames of "+numberOfStations+" channel including wait and queue time is "+totalExecution+" NanoSeconds.");  
 *double* elapsedTimeInSecond = (*double*) forwarding\_delay/ 1\_000\_000\_000;  
 *long* noOfBytes= FRAME\_SIZE \* count;  
 *double* bps = noOfBytes/elapsedTimeInSecond;  
 System.out.println("Total no. of bytes sent are "+noOfBytes+" and total time taken in sending the bytes is "+elapsedTimeInSecond+"sec ("+forwarding\_delay+"ns)");  
 System.out.println("Hence speed of the system is "+bps+" Bytes/sec.");  
 *double* generated = totalChannelAttempt/elapsedTimeInSecond;  
 *double* sent = totalDoneAttempt/elapsedTimeInSecond;  
 System.out.println("No of frames generated in unit time are "+generated);  
 System.out.println("Successfully sent among the total generated are "+sent);  
 System.out.println("Efficiency of the system is "+((sent/generated)\*100));  
 *try* {  
 fileWriter.close();  
 } *catch* (IOException e) {  
 e.printStackTrace();  
 }  
 System.out.println("Transmission completed.");  
 }  
}

**NON PERSISTENT**

****

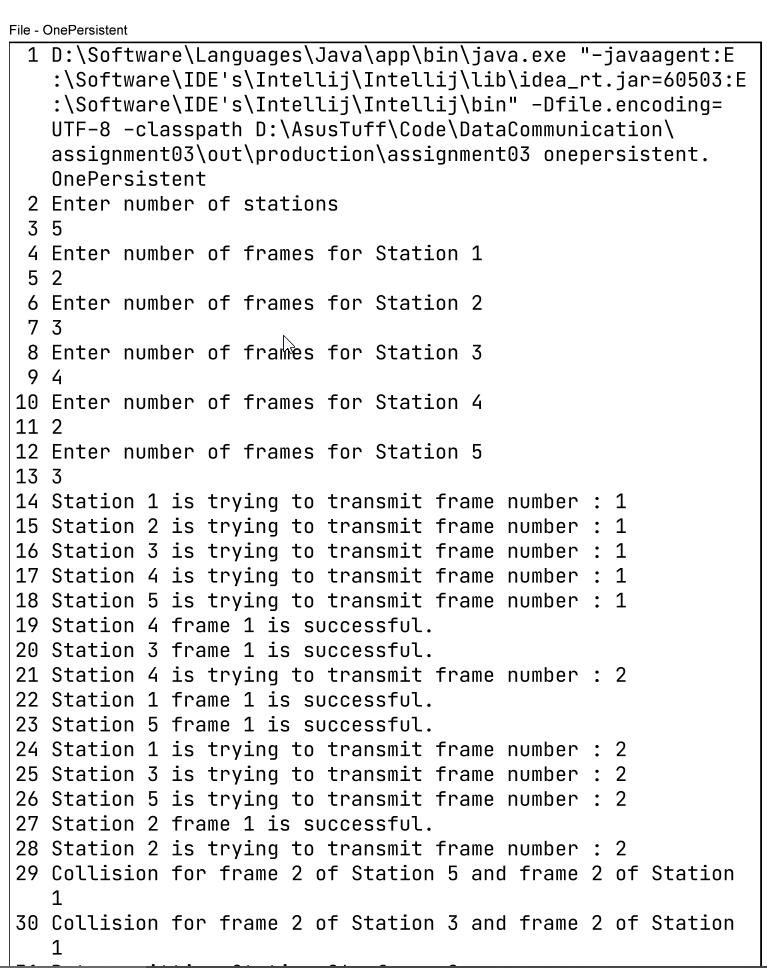
*package* nonpersistent;  
  
*import* utilclasses.*ChannelConstants*;  
*import* java.io.FileWriter;  
*import* java.io.IOException;  
*import* java.util.Date;  
*import* java.util.Scanner;  
  
*class* NonPersistent *implements ChannelConstants* {  
 *private static int totalChannelAttempt* = 0 ;  
 *private static int totalDoneAttempt* = 0 ;  
 *public static long forwarding\_delay* = 0 ;  
 *private static long FRAME\_SIZE* = 520 ;  
  
  
 *public static synchronized void* updateTotalCount(){  
 *totalChannelAttempt*++;  
 }  
 *public static synchronized void* updateDoneCount(){  
 *totalDoneAttempt*++;  
 }  
 *public static void* main(String[] args){  
 Scanner sc = *new* Scanner(System.*in*);  
 NPChannel.*channelStatus* = *FREE*; *//initially channel is free* System.*out*.println("Enter number of stations");  
 *int* numberOfStations = sc.nextInt();  
 NPChannel[] npChannels = *new* NPChannel[numberOfStations+1];  
 *int*[] noOfFramesInChannel = *new int*[numberOfStations+1];  
 *for*(*int* i = 1;i<=numberOfStations;i++)  
 {  
 System.*out*.println("Enter number of frames for Station " + i);  
 noOfFramesInChannel[i] = sc.nextInt();  
  
 }  
 FileWriter fileWriter = *null*;  
 *try* {  
 fileWriter = *new* FileWriter("logs.txt",*true*);  
 fileWriter.write("Started Frame writing at "+*new* Date()+"\n");  
 fileWriter.flush();  
 } *catch* (IOException e) {  
 e.printStackTrace();  
 System.*exit*(1);  
 }  
 *long* start = System.*nanoTime*();  
 *for*(*int* i = 1;i<=numberOfStations;i++)  
 npChannels[i] = *new* NPChannel("Station "+i,noOfFramesInChannel[i],fileWriter);  
  
 *try*{  
 *for*(*int* i=1;i<=numberOfStations;i++)  
 npChannels[i].t.join();  
 } *catch* (InterruptedException e) {  
 System.*out*.println("Main Thread Interrupted");  
 Thread.*currentThread*().interrupt();  
 System.*exit*(1);  
 }  
 *long* end = System.*nanoTime*();  
 *long* totalExecution = end - start;  
 *int* count = 0 ;  
 *for*(*int* i = 0 ; i < numberOfStations; ++i){  
 count += noOfFramesInChannel[i];  
 }  
 System.*out*.println("Time taken to send "+count+" frames of "+numberOfStations+" channel including wait and queue time is "+totalExecution+" NanoSeconds.");  
 *double* elapsedTimeInSecond = (*double*) *forwarding\_delay*/ 1\_000\_000\_000;  
 *long* noOfBytes= *FRAME\_SIZE* \* count;  
 *double* bps = noOfBytes/elapsedTimeInSecond;  
 System.*out*.println("Total no. of bytes sent are "+noOfBytes+" and total time taken in sending the bytes is "+elapsedTimeInSecond+"sec ("+*forwarding\_delay*+"ns)");  
 System.*out*.println("Hence speed of the system is "+bps+" Bytes/sec.");  
 *double* generated = *totalChannelAttempt*/elapsedTimeInSecond;  
 *double* sent = *totalDoneAttempt*/elapsedTimeInSecond;  
 System.*out*.println("No of frames generated in unit time are "+generated);  
 System.*out*.println("Successfully sent among the total generated are "+sent);  
 System.*out*.println("Efficiency of the system is "+((sent/generated)\*100));  
 *try* {  
 fileWriter.close();  
 } *catch* (IOException e) {  
 e.printStackTrace();  
 }  
 System.*out*.println("Transmission completed.");  
 }  
}

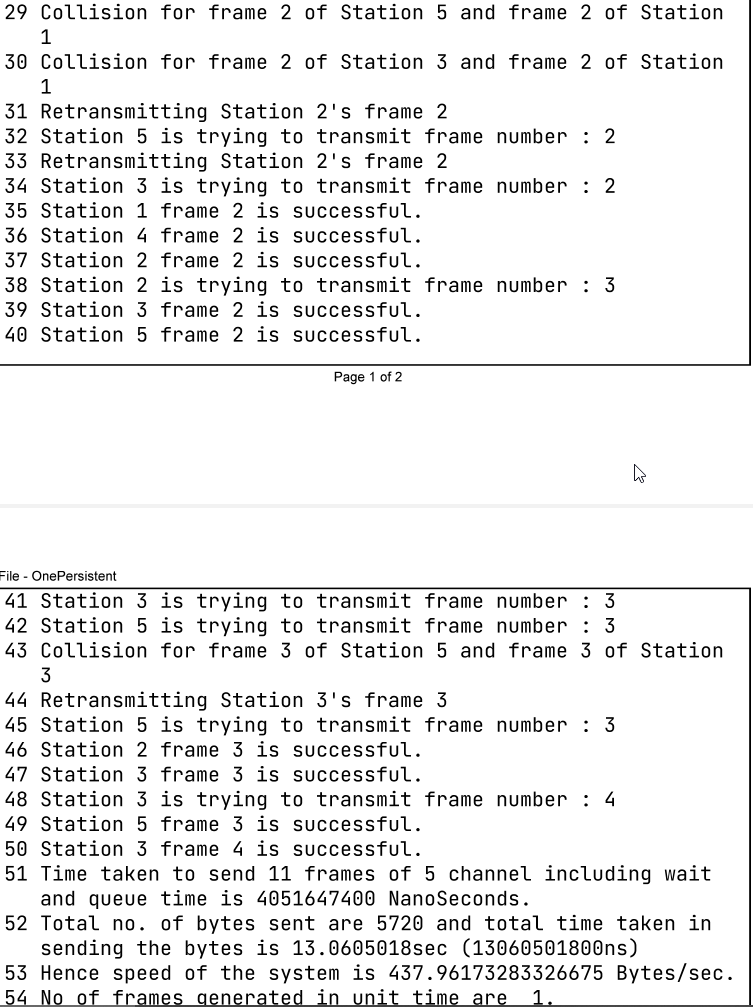
*package* nonpersistent;  
  
*import* utilclasses.*ChannelConstants*;  
  
*import* java.io.FileWriter;  
*import* java.io.IOException;  
*import* java.util.Random;  
*import* java.util.concurrent.atomic.\*;  
  
*class* NPChannel *implements Runnable*, *ChannelConstants* {  
 String stationNumber;  
 Thread t;  
 *static int distance*;  
 *static int stat*=0;  
 *static int frame*;  
 *static int channelStatus*; *//Indicates if channel is being used  
 int* currentFrameNumber;  
 *int* totalNoOfFrames;  
 *private final* AtomicBoolean checkIfSuccessfulTransmission;  
 *static int tfr*=50; *//Transmission time  
 private int* numberOfAttempts;  
 Random rand ;  
 *private static final* Object *distanceLock* = *new* Object();  
 *private static final* Object *channelUpdateLock* = *new* Object();  
 *private static final* Object *getStatLock* = *new* Object();  
 FileWriter fileWriter ;  
  
 NPChannel(String threadName, *int* totalNoOfFrames, FileWriter fileWriter) {  
 stationNumber = threadName;  
 t = *new* Thread(*this*, stationNumber);  
 currentFrameNumber = 1;  
 *this*.totalNoOfFrames =totalNoOfFrames;  
 checkIfSuccessfulTransmission = *new* AtomicBoolean();  
 *this*.fileWriter = fileWriter;  
 t.start();  
 }  
 *public void* run() {  
 rand = *new* Random();  
 *while* (!checkIfSuccessfulTransmission.get()){  
 numberOfAttempts++;  
 *while*(currentFrameNumber <= totalNoOfFrames) {  
 NonPersistent.*updateTotalCount*();  
 *if* (numberOfAttempts < 15) { *//15 is the maximum number of attempts  
 try* {  
 *if* (*channelStatus* == *INUSE*) {  
 System.*out*.println(stationNumber + " is using Non-Persistent sensing, channel is busy");  
 *try* {  
 *long* time =(*long*) rand.nextInt(50)+1000;  
 Thread.*sleep*(time);  
 }  
 *catch* (InterruptedException e) {  
 System.*out*.println(("Interrupt"));  
 Thread.*currentThread*().interrupt();  
 }  
 }  
 *else* {  
 System.*out*.println(stationNumber + " is trying to transmit frame number : " + currentFrameNumber);  
  
 *if* (*channelStatus* == *FREE* && *distance* == 0) {*//Successful transmission  
 synchronized* (*distanceLock*){  
 *distance*++;  
 }  
 *synchronized* (*getStatLock*){  
 *stat* = NPChannelThreads.*checking*(Thread.*currentThread*().getName());  
 *if*(*stat* == -1){  
 System.*out*.println("Error Setting up stat.");  
 System.*exit*(1);  
 }  
  
 *frame* = *this*.currentFrameNumber;  
 }  
 *synchronized* (*channelUpdateLock*){  
 *channelStatus* = *INUSE*;  
 }  
 *long* start = System.*nanoTime*();  
 *synchronized* (*distanceLock*){  
 *distanceLock*.wait(1000);  
 }  
 *long* end = System.*nanoTime*();  
 *synchronized* (*this*){  
 NonPersistent.*forwarding\_delay* += end-start;  
 }  
 NonPersistent.*updateDoneCount*();  
 fileWriter.write(stationNumber + " frame " + currentFrameNumber + " is successful.\n");  
 fileWriter.flush();  
 System.*out*.println(stationNumber + " frame " + currentFrameNumber + " is successful.");  
  
 checkIfSuccessfulTransmission.set(*true*);  
 currentFrameNumber++;  
 *synchronized* (*distanceLock*){  
 *distance* = 0;  
 }  
 *synchronized* (*channelUpdateLock*){  
 *channelStatus* = *FREE*;  
 }  
 }  
 *else* {*//Collision has occurred* System.*out*.println("Collision for frame " + currentFrameNumber + " of " +  
 stationNumber + " and frame " + *frame* + " of Station " + *stat*);  
  
 System.*out*.println("Retransmitting Station " + *stat* + "'s frame " + *frame*);  
 checkIfSuccessfulTransmission.set(*false*);  
 *synchronized* (*channelUpdateLock*){  
 *channelStatus* = *FREE*;  
 }  
 numberOfAttempts++;  
 *try* {  
 *int* anInt = rand.nextInt((*int*) (Math.*pow*(2,(*double*) numberOfAttempts - 1)));  
 *int* backOffTime = anInt \* *tfr*;  
 Thread.*sleep*(backOffTime);  
  
 } *catch* (InterruptedException e) {  
 System.*out*.println("Interrupted");  
 Thread.*currentThread*().interrupt();  
 }  
 }  
 Thread.*sleep*(1000);  
  
  
 }  
 } *catch* (InterruptedException | IOException e) {  
 System.*out*.println(stationNumber + "Main Interrupted");  
 Thread.*currentThread*().interrupt();  
 }  
 }  
 *else* {  
 checkIfSuccessfulTransmission.set(*true*);  
 System.*out*.println("Too many attempts for frame " + currentFrameNumber + "of " +  
 stationNumber + ". Transmission stopped");  
 }  
  
 }  
 }  
 }  
}

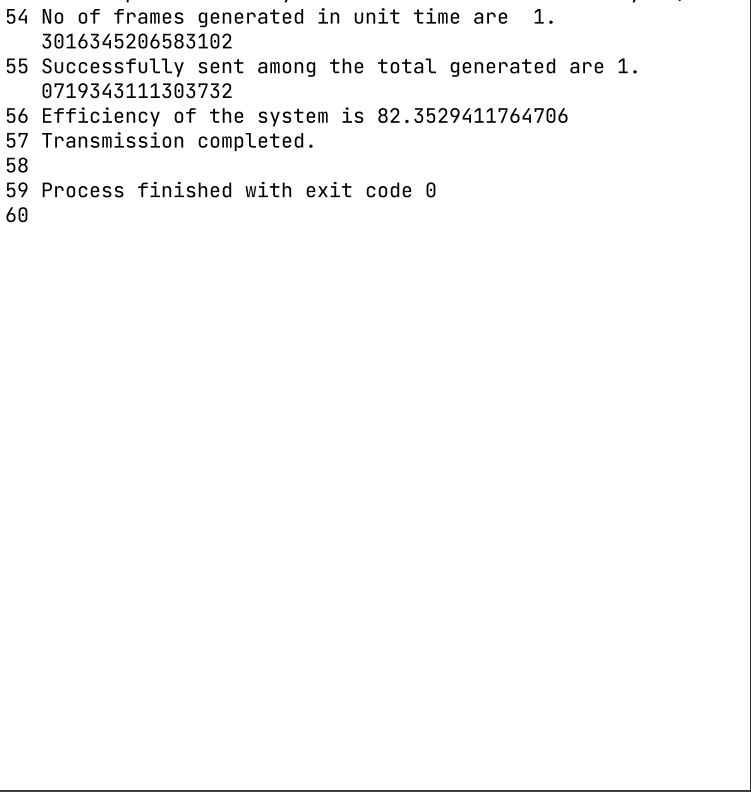
*package* nonpersistent;  
  
*/\*Class to check which station's frame the current transmission is colliding with.  
  
 \*/  
public class* NPChannelThreads {  
 *private* NPChannelThreads(){*throw new* UnsupportedOperationException("Util Class.");}  
  
 *public static int* checking(String name) {  
 *int* stat = -1 ;  
 *try*{  
 stat = Integer.parseInt(name.substring(8));  
 }*catch* (NumberFormatException e ){  
 *//Some Error.* System.out.println("NPChannel name I got is "+name);  
 }  
 *return* stat;  
 }  
  
}

**TEST CASES**

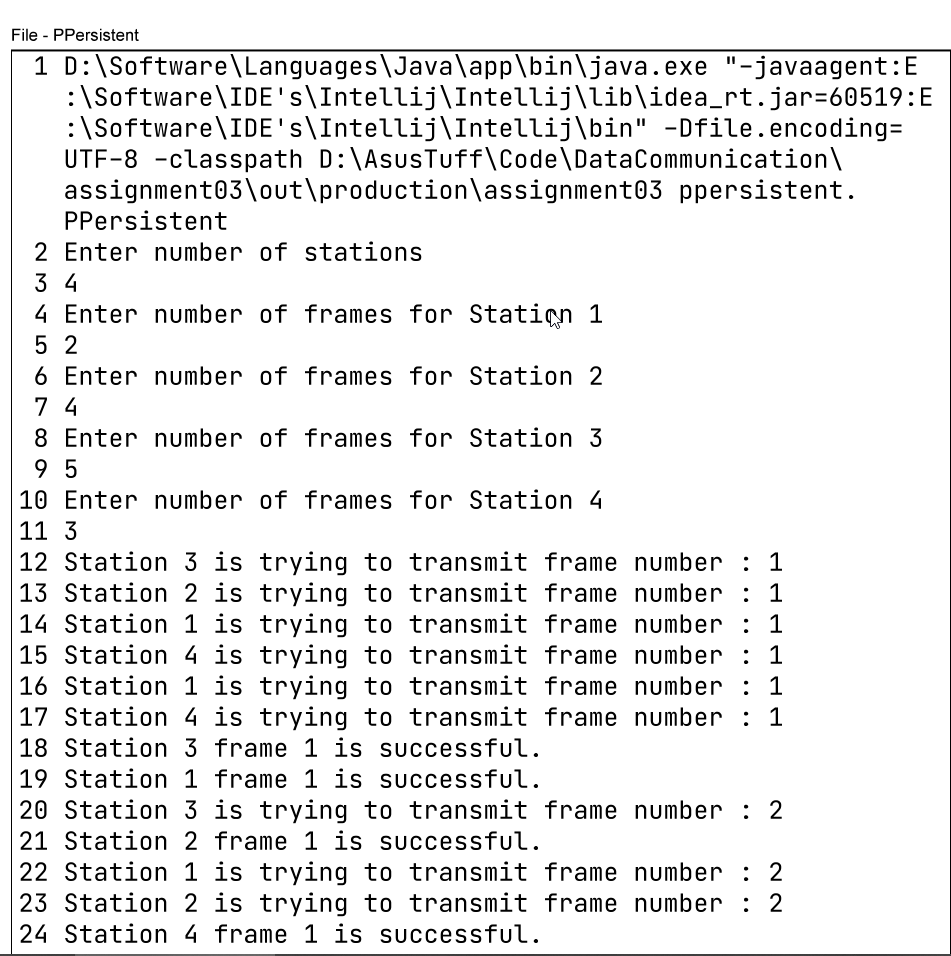
**ONEPERSISTENT**

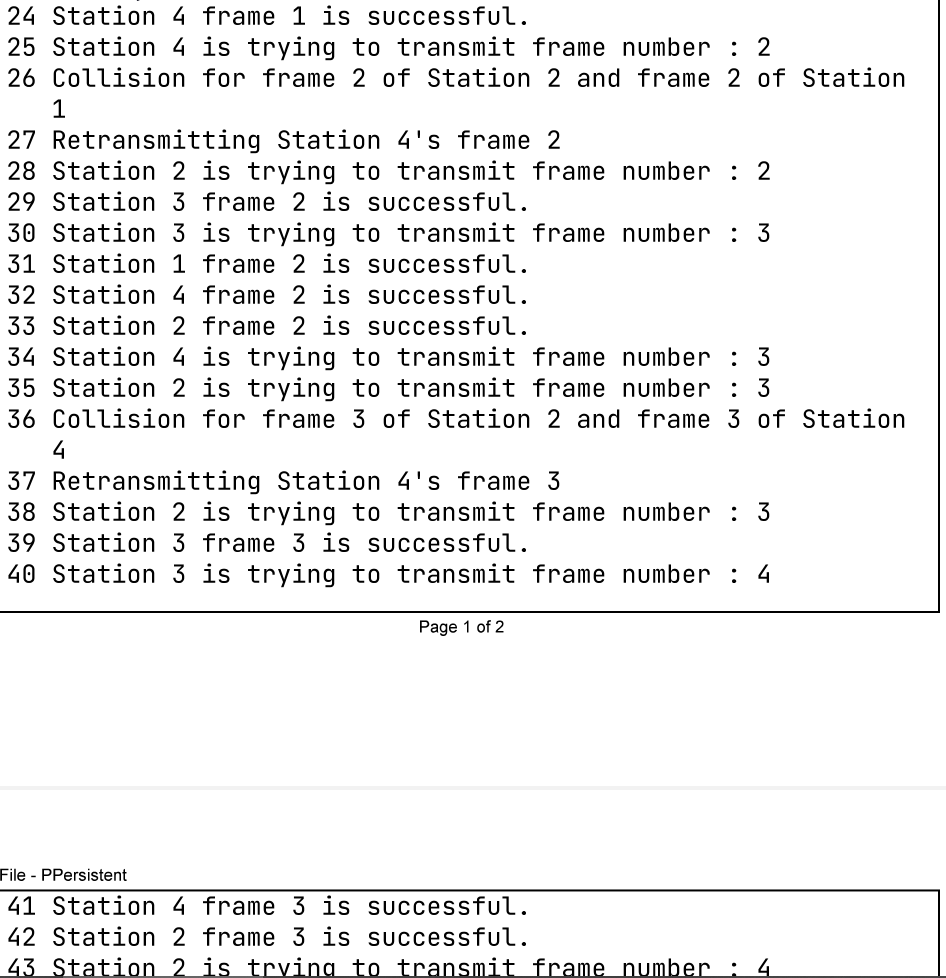


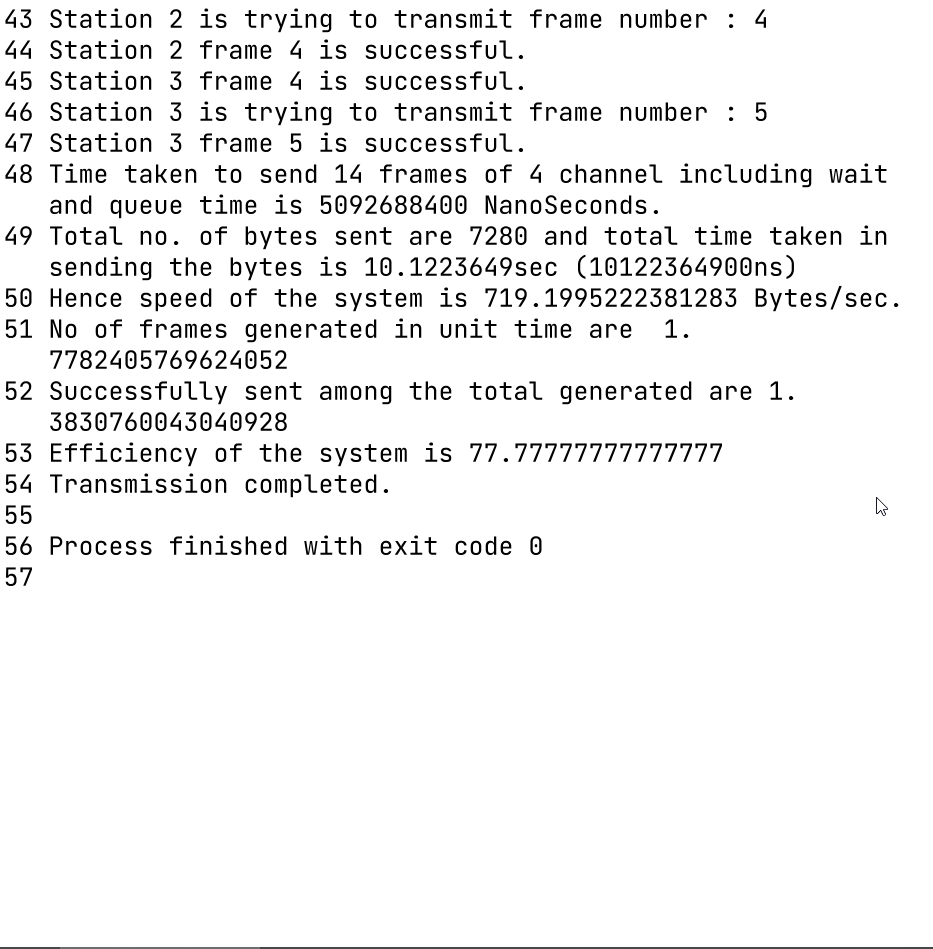




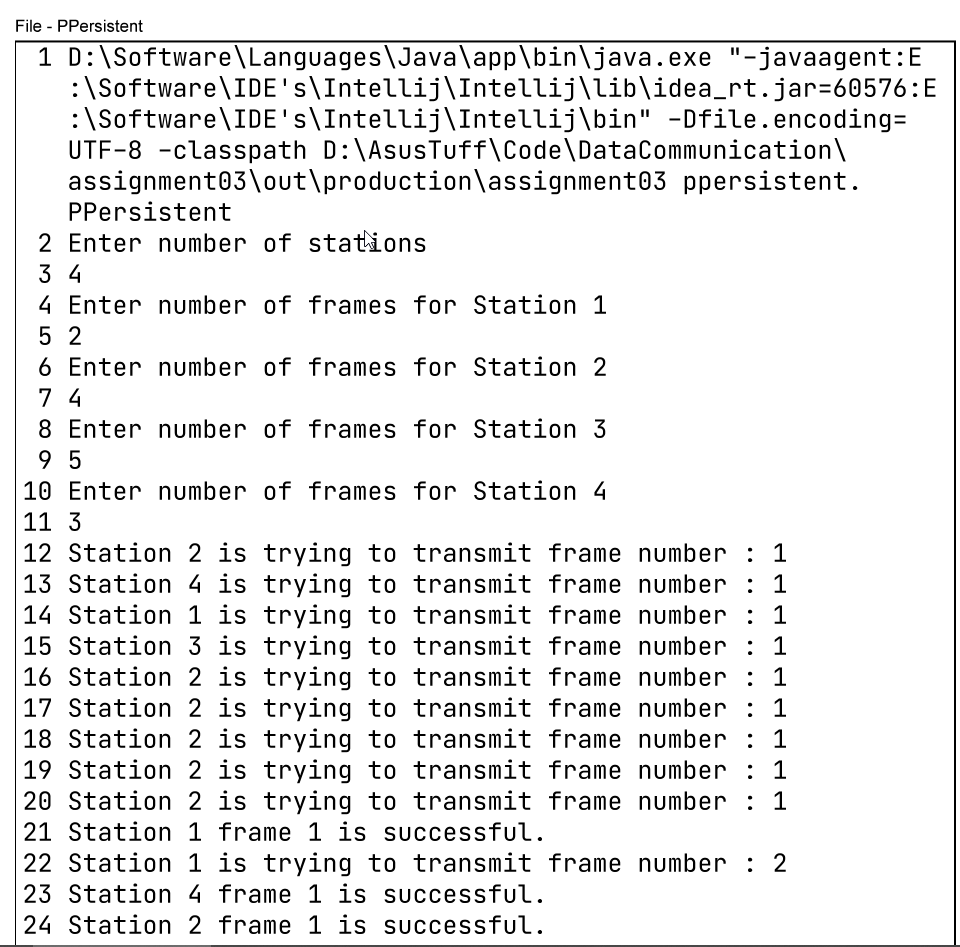
P PERSISTENT (p =0.1)

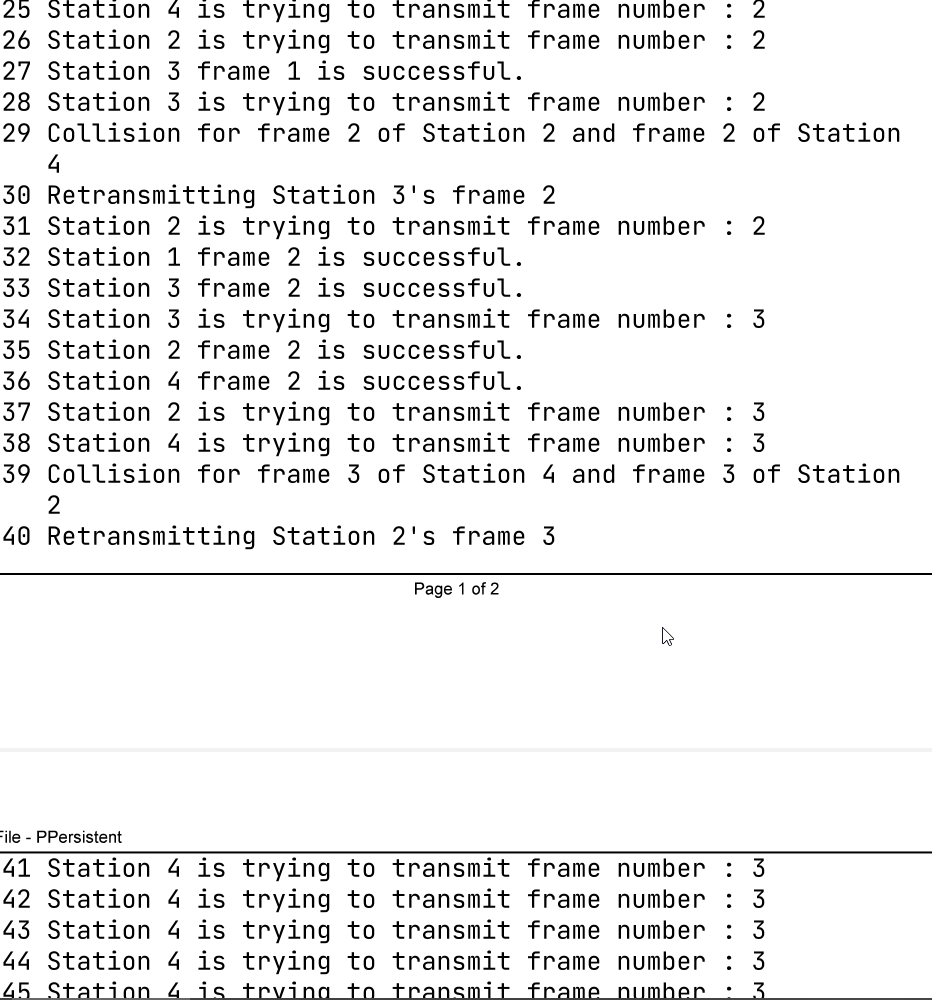


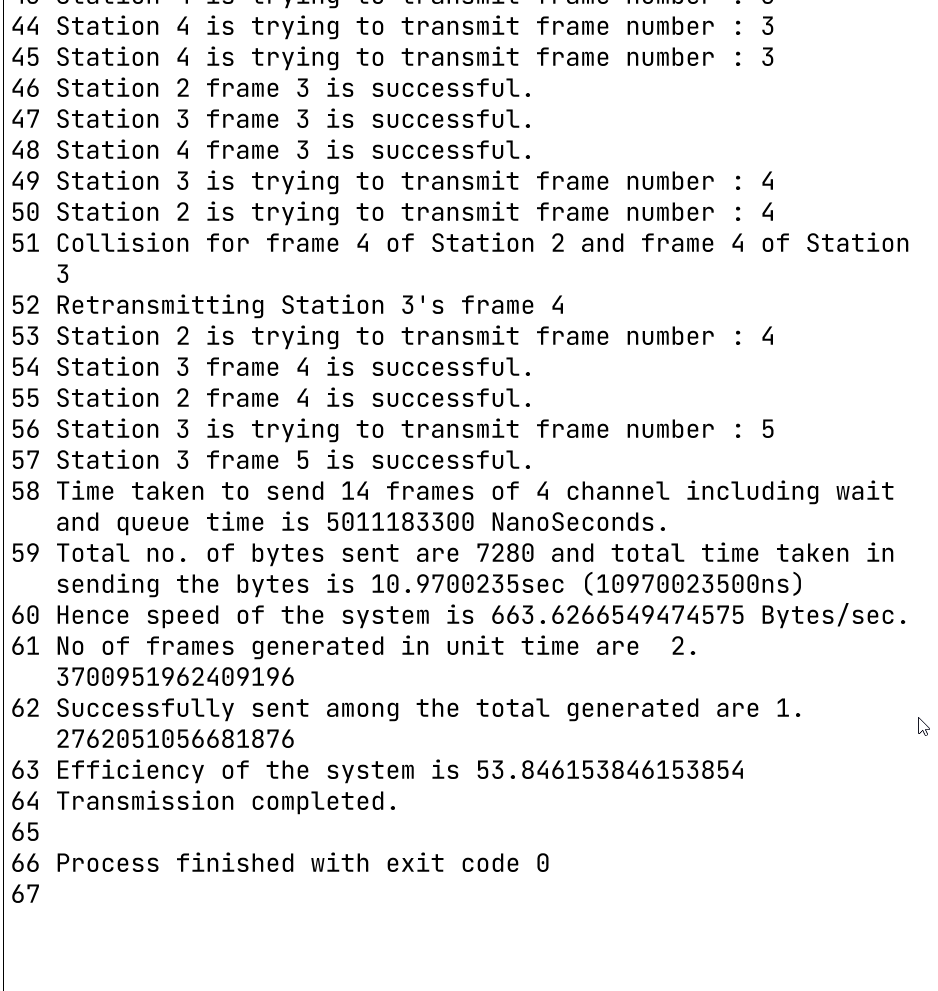




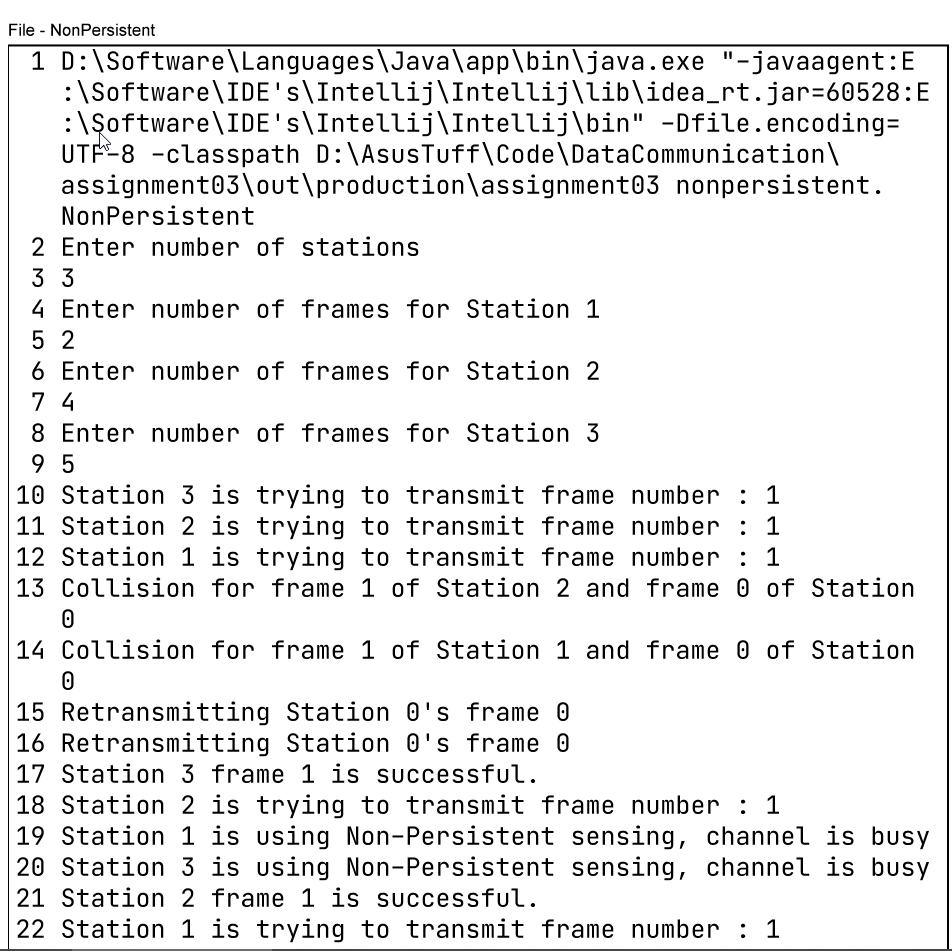
P PERSISTENT (p =0.4)

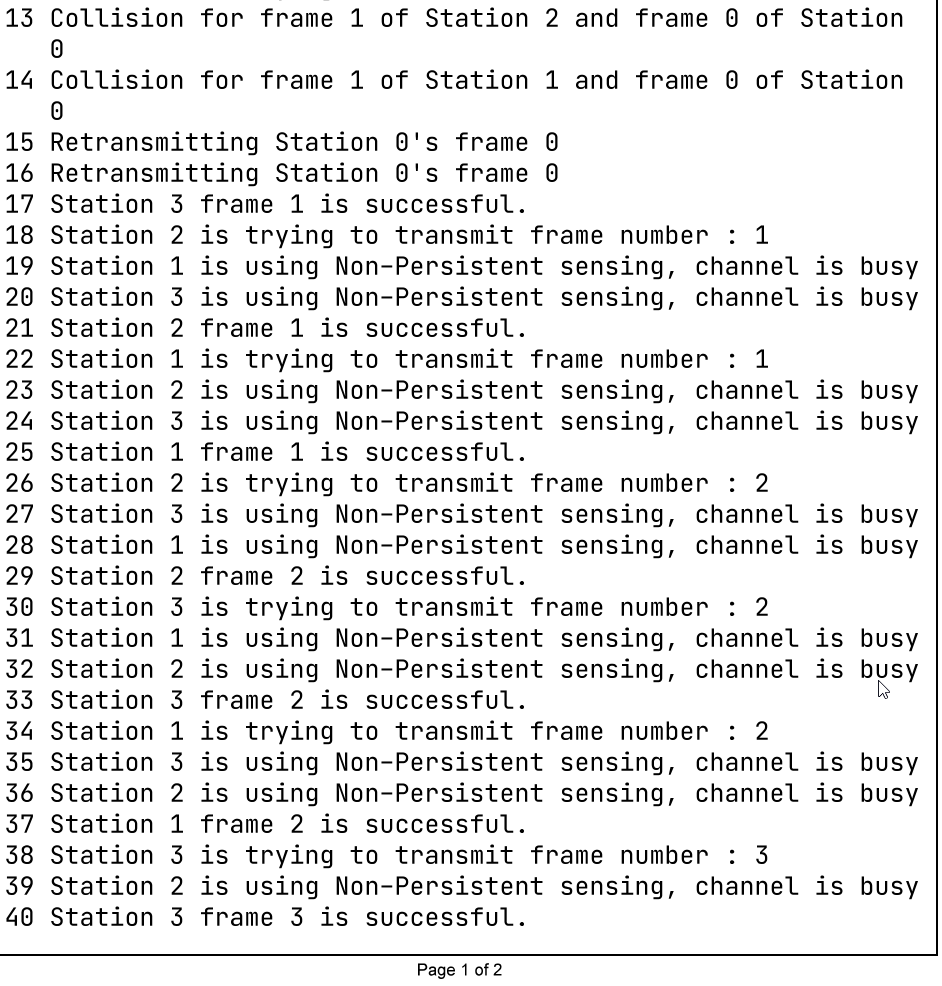


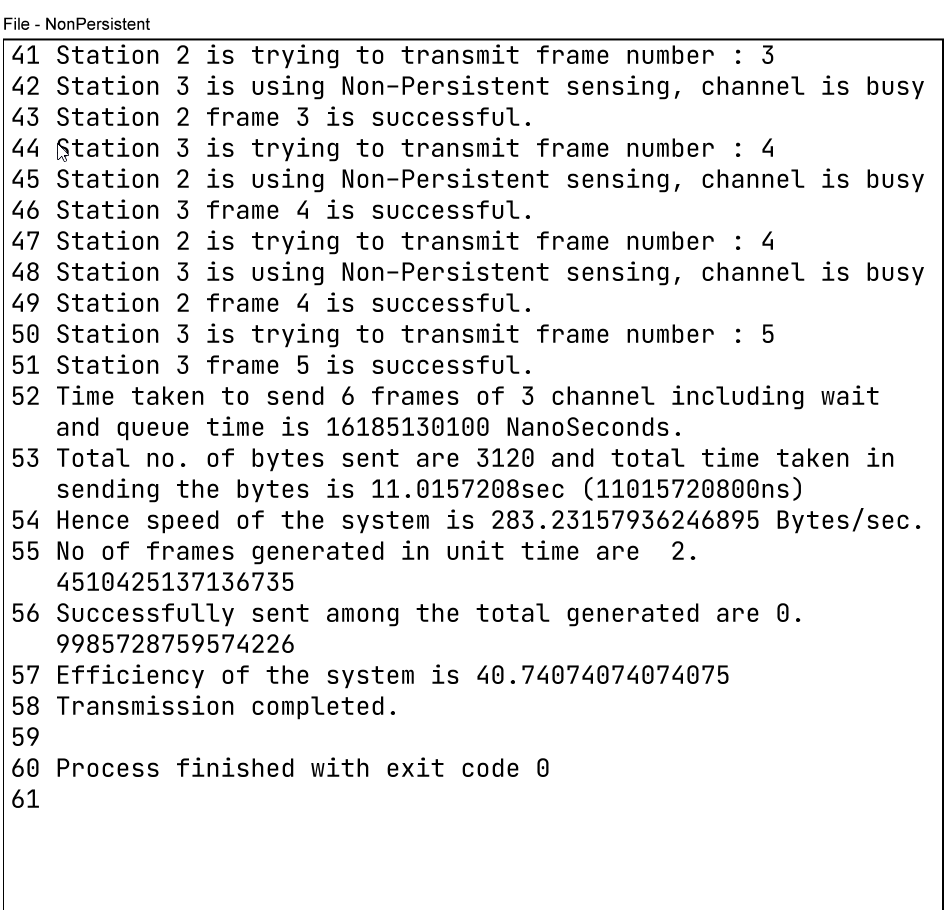




**NON PERSISTENT**



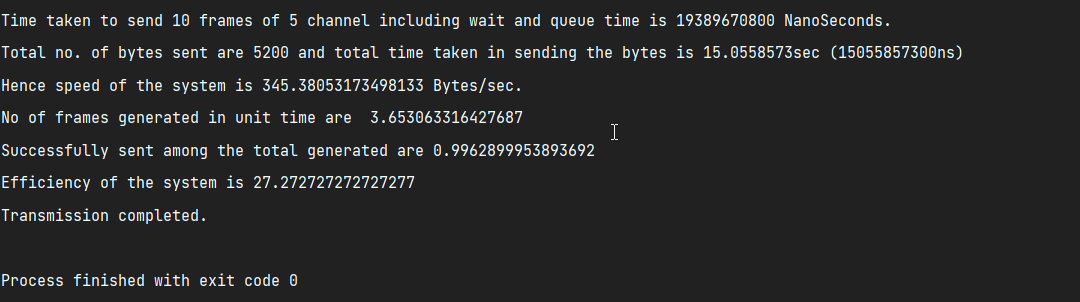




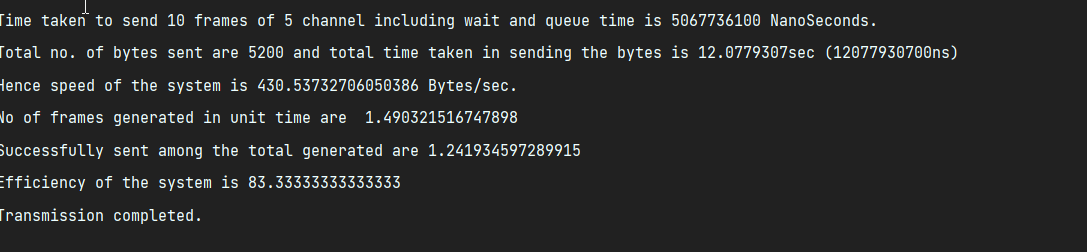
**RESULT**

|  |  |  |  |
| --- | --- | --- | --- |
| Scheme | No. of Station (total no. of frames) | No. Of Collisions | Total Time Taken |
| One Persistent | 5(15) | 6 | 13 Seconds |
| Non Persistent | 5(15) | 12 | 15 Seconds |
| P – Persistent (p = 0.1) | 5(15) | 2 | 9 Seconds |
| P – Persistent (p = 0.4) | 5(15) | 5 | 11 Seconds |

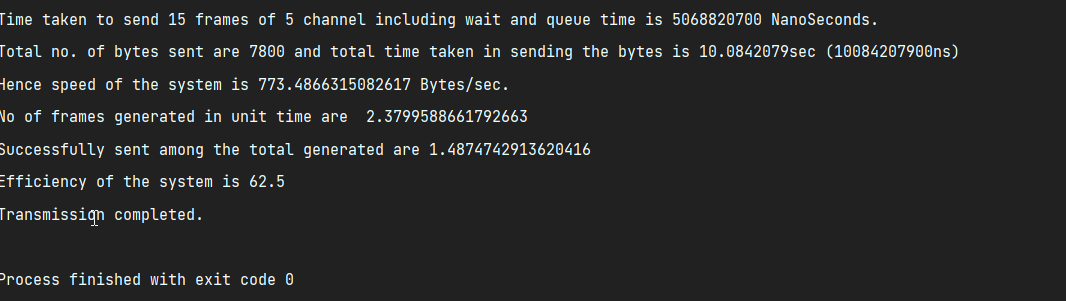
**Non Persistent**



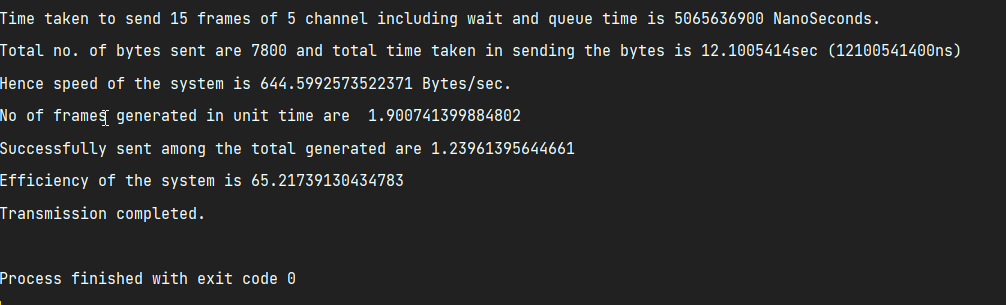
**One Persistent**



**p Persistent(p = 0.1)**

****

**p Persistent(p = 0.4)**

****

**P-persistent:-**

**P=0.2**

**Run Throughput Forwarding delay**

**Run1 252 2694**

**Run2 338 2008**

**Run3 230 2858**

**Average 271 bits/sec 2500 millisec**

**P-persistent:-**

**P=0.4**

**Run Throughput Forwarding delay**

**Run1 252 2694**

**Run2 338 2008**

**Run3 230 2858**

**Average 185 bits/sec 3500 millisec**

**P-persistent:-**

**P=0.6**

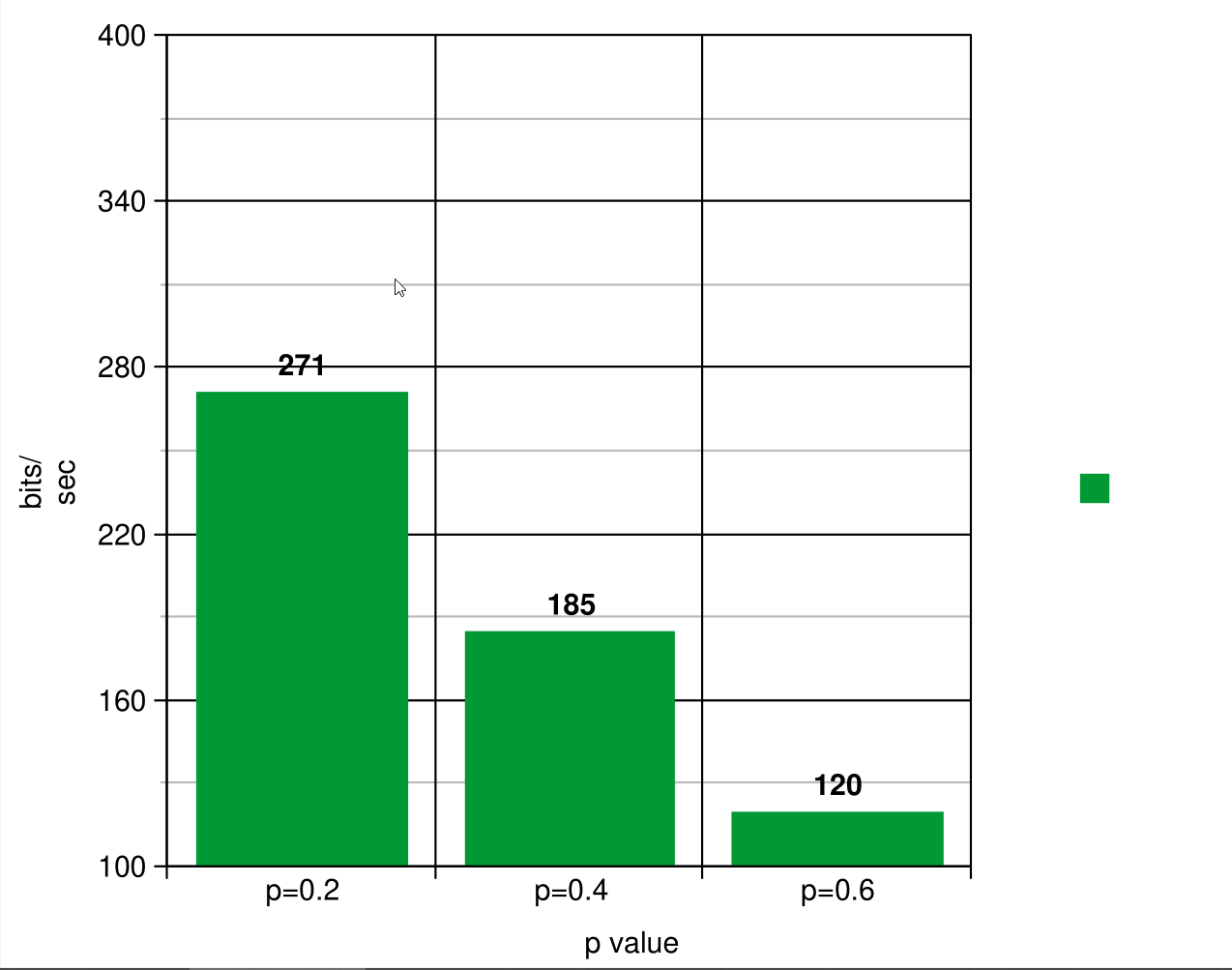
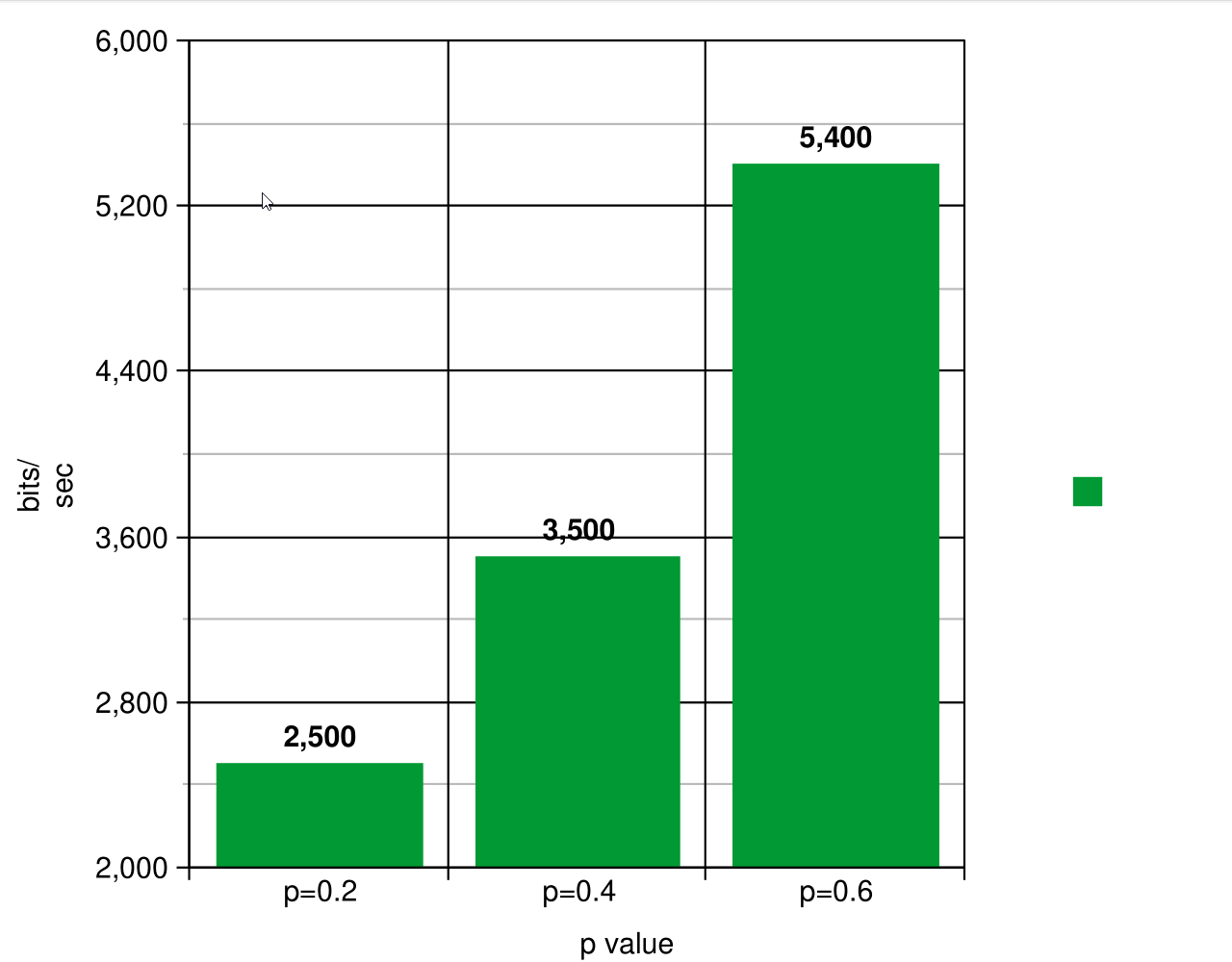
**Run Throughput Forwarding delay**

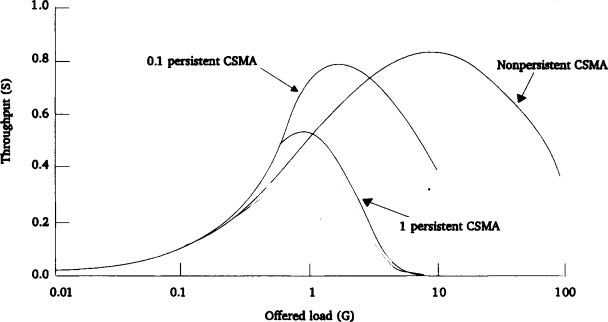
**Run1 123 5528**

**Run2 129 5263**

**Run3 120 5643**

**Average 120 bits/sec 5400 millisec**

****

****

**Analysis**

From the above table and graph we can draw the conclusion that P- persistent is a effective technique we can use to handle multiple access. But I found that this p-persistent depends darastically on the value of choose P. minor change in P can lead to great difference in performance.

Now Coming to the part how to choose the value of P ????

P must be set to a value such that :

* It is very likely that exactly one node will pick a random number x such that
  + x < p

i.e. exactly 1 node should transmit its message.

* Using probability theory
  + If n nodes are sensing, then the probability:

P = 1/n

will maximize the likelihood that exactly one node will transmit and (n-1)

nodes will back off.

EXAMPLE

* Consider that 2 nodes are waiting for the current transmission to finish then

We should choose

p = 0.5

* Now consider that in some case 3 nodes are waiting for the current transmission to

finish , then we should use

p = 0.33333333333(1/3)

Similarly if we have 4 nodes waiting then we should choose

P = 0.25

And so on……………….

**Conclusion**

P must be dynamic in nature.

Since it’s hard to know how many notes are busy. So a careful analysis should be done while picking the p.